Bezpieczeństwo sieci w Pythonie

Wydanie II

Rozwiązywanie problemów
za pomocą skryptów i bibliotek

José Manuel Ortega

Helion

Tytuł oryginału: Mastering Python for Networking and Security: Leverage the scripts and libraries of Python version 3.7 and beyond to overcome networking and security issues, 2nd Edition

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-8255-8

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/bezsp2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Część II. Komunikacja sieciowa i pozyskiwanie informacji z sieci Tor

Część IV. Podatności serwerów na ataki i bezpieczeństwo modułów języka Python

ROZDZIAŁ 8

Skaner Nmap

Ten rozdział jest poświęcony modułowi `nmap`, służącemu do skanowania sieci, tj. uzyskiwania informacji o hostach, otwartych portach i uruchomionych usługach. Dlatego jego znajomość jest bardzo cenna. Moduł jest napisany w języku Python i wykorzystuje narzędzie Nmap.

Na początku rozdziału opisane jest wykrywanie za pomocą narzędzia Nmap portów otwartych, zamkniętych i blokowanych. W dalszej części opisano zastosowanie modułu `nmap` do synchronicznego i asynchronicznego skanowania portów oraz jego współpracę z modułami `os` i `subprocess`. Ostatnia część rozdziału jest poświęcona skryptom narzędzia Nmap wyszukującym podatności hostów na ataki.

W tym rozdziale opisane są następujące zagadnienia:

- skanowanie portów za pomocą narzędzia Nmap,
- skanowanie portów za pomocą modułu `nmap`,
- tryby skanowania,
- uruchamianie narzędzia Nmap za pomocą modułów `os` i `subprocess`,
- wykrywanie usług i ich podatności na ataki za pomocą skryptów narzędzia Nmap.

Wymagania techniczne

Aby w pełni wykorzystać materiał zawarty w tym rozdziale, niezbędna jest podstawowa wiedza o programowaniu w języku Python i o protokole HTTP. Użyta jest tu wersja języka Python 3.7, dostępna na stronie *www.python.org/downloads*.

Przykładowe kody wykorzystane w tym rozdziale są zapisane w katalogu *r08* w załączonych do książki plikach. Dodatkowo, na stronie *https://bit.ly/3l4uMWS* znajduje się film instruktażowy.

W tym rozdziale jest wymagana instalacja modułu `nmap`, którą można wykonać przy użyciu narzędzi dostępnych w systemie operacyjnym. Na przykład w systemie Debian należy użyć następującego polecenia:

```
$ pip install python-nmap
```

Skanowanie portów za pomocą narzędzia Nmap

Zacznijmy od zapoznania się z narzędziem Nmap, służącym do skanowania portów i wykrywania usług uruchomionych na wybranym hoście. W tym podrozdziale opisane są również różne tryby skanowania.

Komputery do przesyłania danych w sieci wykorzystują protokoły komunikacyjne i porty. Na jednym serwerze może działać wiele aplikacji i usług udostępnianych za pomocą różnych portów. Na przykład do przesyłania stron internetowych jest wykorzystywany protokół HTTP i domyślny port numer 80. Natomiast w usłudze FTP zazwyczaj stosuje się port nr 21, a w SMTP (ang. *Simple Mail Transfer Protocol*, prosty protokół przesyłania poczty) port numer 25.

Numer portu jest liczbą 16-bitową z zakresu od 1 do 65 535. Do nawiązania połączenia niezbędny jest adres IP hosta, rodzaj protokołu i numer portu.

Zatem, w jaki sposób skanuje się porty?

Techniki skanowania w narzędziu Nmap

Nmap (Network Mapper) jest otwartym, bezpłatnym narzędziem do wykrywania urządzeń w sieci i audytowania ich bezpieczeństwa. Narzędzie jest dostępne dla wszystkich najważniejszych systemów operacyjnych (Linux, Windows i macOS) na stronie *https://nmap.org/download.html*. Najczęściej stosuje się je do skanowania portów w określonym segmencie sieci. Uruchomione w wierszu poleceń wyświetla dostępne parametry, jak na rysunku 8.1.

```
Nmap 7.91 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sn: Ping Scan - disable port scan
  -Pn: Treat all hosts as online -- skip host discovery
  -PS/PA/PU/PY[portlist]: TCP SYN/ACK, UDP or SCTP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
  -PO[protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify custom DNS servers
  --system-dns: Use OS's DNS resolver
  --traceroute: Trace hop path to each host
SCAN TECHNIQUES:
  -sS/sT/sA/sW/sM: TCP SYN/Connect()/ACK/Window/Maimon scans
  -sU: UDP Scan
  -sN/sF/sX: TCP Null, FIN, and Xmas scans
  --scanflags <flags>: Customize TCP scan flags
  -sI <zombie host[:probeport]>: Idle scan
  -sY/sZ: SCTP INIT/COOKIE-ECHO scans
  -sO: IP protocol scan
  -b <FTP relay host>: FTP bounce scan
```

Rysunek 8.1. Parametry narzędzia Nmap

W sekcji SCAN TECHNIQUES są opisane parametry uruchamiające różne procedury skanowania:

- `-sT` (*TCP connect scan*) — identyfikowanie otwartych portów. Należy pamiętać, że technika ta jest wykrywana przez systemy IDS (ang. *Intrusion Detection System*, system wykrywania włamań). Port jest otwarty, jeżeli po wysłaniu do niego pakietu z ustawioną flagą SYN komputer odsyła pakiet z ustawionymi flagami SYN i ACK.
- `-sS` (*TCP SYN scan*) — technika podobna do powyższej, różniąca się od niej tym, że nie jest w niej nawiązywane pełne połączenie. Port jest uznawany za otwarty, jeżeli host na żądanie nawiązania połączenia odpowie pakietem zawierającym flagi SYN/ACK, a za zamknięty, jeżeli pakiet zawiera flagę RST.
- `-sU` (*UDP scan*) — technika skanowania wykorzystująca protokół UDP (ang. *User Datagram Protocol*, protokół przesyłania datagramów użytkownika). Port jest uznawany za otwarty, jeżeli host odeśle pakiet wykorzystując ten sam protokół. Jeżeli natomiast odpowie stosując protokół ICMP (ang. *Internet Control Message Protocol*, internetowy protokół przesyłania komunikatów kontrolnych) z komunikatem numer 3 (cel niedostępny), to znaczy, że port jest zamknięty.
- `-sA` (*TCP ACK scan*) — technika pozwalająca stwierdzić, czy host jest chroniony za pomocą zapory. Polega na wysłaniu pakietu zawierającego ustawioną flagę ACK. Jeżeli zostanie odebrany pakiet z ustawioną flagą RST, to znaczny, że port nie jest blokowany przez zaporę. Jeżeli odpowiedź nie nadejdzie lub będzie to pakiet ICMP, to znaczy, że dostęp do skanowanego portu jest blokowany przez zaporę.
- `-sN` (*TCP NULL scan*) — technika polegająca na wysłaniu do hosta pakietu TCP bez ustawionych jakichkolwiek flag. Jeżeli host odeśle poprawną odpowiedź, to znaczy, że dany port jest otwarty. Port jest uznawany za zamknięty, jeżeli host odeśle pakiet zawierający flagę RTS.
- `-sF` (*TCP FIN Scan*) — technika polegająca na wysłaniu do hosta pakietu TCP z ustawioną flagą FIN. Jeżeli host odeśle poprawną odpowiedź, to znaczy, że dany port jest otwarty. Port jest uznawany za zamknięty, jeżeli host odeśle pakiet zawierający flagę RTS.
- `-sX` (*TCP Xmas scan*) — technika polegająca na wysłaniu do hosta pakietu TCP z ustawionymi flagami PSH, FIN i URG. Jeżeli host odeśle poprawną odpowiedź, to znaczy, że dany port jest otwarty. Port jest uznawany za zamknięty, jeżeli host odeśle pakiet zawierający flagę RTS. Jeżeli natomiast odpowie pakietem ICMP, to znaczy, że dostęp do skanowanego portu jest blokowany przez zaporę.

Domyślnie stosowana technika zależy od uprawnień konta użytkownika i możliwości wysyłania różnego rodzaju pakietów. Ponadto wykorzystanie narzędzia ograniczają zapory sieciowe i systemy IDS, które mogą blokować opisane techniki skanowania.

> Dokładny opis technik skanowania oferowanych przez narzędzie Nmap znajduje się na stronie *https://nmap.org/book/man-port-scanning-techniques.html*.

Narzędzie Nmap jest również dostępne pod nazwą Zenmap (*https://nmap.org/zenmap*). Jest to wersja wyposażona w graficzny interfejs użytkownika, dzięki czemu jest prostsza w użyciu.

Domyślnie narzędzie Nmap skanuje najczęściej wykorzystywane porty i dostarcza informacje o stanie każdego z nich. Port może znajdować się w jednym z poniższych stanów:

- **otwartym** (*open*), oznaczającym, że jest dostępna usługa wykorzystująca dany port,
- **zamkniętym** (*closed*), oznaczającym, że żadna usługa nie wykorzystuje danego portu,
- **zablokowanym** (*filtered*), oznaczającym, że nie można określić możliwości nawiązania połączenia z danym portem,
- **niezablokowanym** (*unfiltered*), oznaczającym, że dany port jest dostępny, ale nie można określić możliwości nawiązania z nim połączenia.

Skaner portów można utworzyć wykorzystując w tym celu moduł `socket`, a następnie skanować każdy port w osobnym wątku. Przy użyciu powyższego modułu można jednak wykonywać tylko proste skanowanie. Na przykład nie jest możliwe stosowanie opisanych wyżej zaawansowanych technik wykorzystujących flagi ACK, FIN, PSH i URG. Dlatego pojawił się moduł `nmap`, umożliwiający programowe korzystanie z narzędzia Nmap.

Skanowanie portów przy użyciu modułu nmap

W tym podrozdziale jest opisany moduł `nmap`, umożliwiający skanowanie portów. Dowiesz się, jak ten moduł wykorzystuje narzędzie Nmap i jak dzięki niemu można optymalizować skanowanie hostów, domen, sieci i adresów IP.

Moduł `nmap` jest bardzo popularny. Wykorzystuje się go nie tylko do badania bezpieczeństwa sieci i wykonywania testów penetracyjnych. Jego głównym przeznaczeniem jest wykrywanie portów i usług dostępnych w danym hoście. Jest doskonałym narzędziem dla administratorów i specjalistów od bezpieczeństwa, chcących automatyzować procesy testowania infrastruktury.

Kod źródłowy modułu jest dostępny w repozytorium Bitbucket, pod adresem *https://bitbucket.org/xael/python-nmap*. Najnowszą wersję instalacyjną można pobrać ze strony *http://xael.org/pages/python-nmap-en.html*.

Teraz zaimportuj moduł `nmap`, sprawdź jego wersję i zapoznaj się z dostępnymi klasami. W tym celu uruchom interpreter języka Python i wpisz poniższe polecenia.

```
>>> import nmap
>>> nmap.__version__
'0.6.4'
>>> dir(nmap)
['ET', 'PortScanner', 'PortScannerAsync', 'PortScannerError',
'PortScannerHostDict', 'PortScannerTimeout', 'PortScannerYield', 'Process',
'__author__', '__builtins__', '__cached__', '__doc__', '__file__',
'__last_modification__', '__loader__', '__name__', '__package__', '__path__',
'__spec__', '__version__', 'convert_nmap_output_to_encoding', 'csv', 'io', 'nmap',
'os', 're', 'shlex', 'subprocess', 'sys']
```

Po sprawdzeniu poprawności instalacji modułu możesz rozpocząć skanowanie hostów. W tym celu utwórz instancję klasy `PortScanner`, a następnie użyj zawartej w niej metody `scan()`.

Dobrym sposobem dowiedzenia się, jak działa określony proces, jest użycie dostępnych w klasie metod, które można wyświetlić za pomocą funkcji `dir()`:

```
>>> port_scan = nmap.PortScanner()
>>> dir(port_scan)
['_PortScanner__process', '__class__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
'__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'_nmap_last_output', '_nmap_path', '_nmap_subversion_number',
'_nmap_version_number', '_scan_result', 'all_hosts', 'analyse_nmap_xml_scan',
'command_line', 'csv', 'get_nmap_last_output', 'has_host', 'listscan',
'nmap_version', 'scan', 'scaninfo', 'scanstats']
```

Powyższy wynik zawiera nazwy właściwości i metod klasy `PortScanner`.

Opis metody `scan()` można uzyskać za pomocą funkcji `help()`. Poniższy wynik pokazuje, że metoda ta ma pięć argumentów: adres hosta, numery portów, parametry narzędzia Nmap, opcję rozszerzenia uprawnień i maksymalny czas skanowania.

```
>>> help(port_scan.scan)
Help on method scan in module nmap.nmap:
scan(hosts='127.0.0.1', ports=None, arguments='-sV', sudo=False, timeout=0) method
↪of nmap.nmap.PortScanner instance
    Scan given hosts
    May raise PortScannerError exception if nmap output was not xml
    Test existance of the following key to know
    if something went wrong : ['nmap']['scaninfo']['error']
    If not present, everything was ok.
    :param hosts: string for hosts as nmap use it 'scanme.nmap.org' or '198.116.0-
    ↪255.1-127' or '216.163.128.20/20'
    :param ports: string for ports as nmap use it '22,53,110,143-4564'
    :param arguments: string of arguments for nmap '-sU -sX -sC'
    :param sudo: launch nmap with sudo if True
    :param timeout: int, if > zero, will terminate scan after seconds, otherwise
    ↪will wait indefintely
    :returns: scan_result as dictionnary
```

Aby przeprowadzić skanowanie, należy najpierw zaimportować moduł, a następnie utworzyć instancję klasy `PortScanner`. Proces skanowania uruchamia się, wywołując metodę `scan()`. Wszystkie argumenty metody są opcjonalne, jednak najczęściej określa się pierwsze dwa: adres hosta i numery portów.

Poniższy przykład ilustruje skanowanie portów 80 – 85 hosta o nazwie *scanme.nmap.org*. Argument `-sV` oznacza, że podczas skanowania będą rozpoznawane wersje usług.

```
>>> portScanner = nmap.PortScanner()
>>> results = portScanner.scan('scanme.nmap.org', '80-85', '-sV')
>>> results
```

```
{'nmap': {'command_line': 'nmap -oX - -p 80-85 -sV scanme.nmap.org', 'scaninfo':
{'tcp': {'method': 'syn', 'services': '80-85'}}, 'scanstats': {'timestr': 'Mon Jun
14 22:26:35 2021', 'elapsed': '19.45', 'uphosts': '1', 'downhosts': '0',
'totalhosts': '1'}}, 'scan': {'45.33.32.156': {'hostnames': [{'name':
'scanme.nmap.org', 'type': 'user'}, {'name': 'scanme.nmap.org', 'type': 'PTR'}],
'addresses': {'ipv4': '45.33.32.156'}, 'vendor': {}, 'status': {'state': 'up',
'reason': 'echo-reply'}, 'tcp': {80: {'state': 'open', 'reason': 'syn-ack',
'name': 'http', 'product': 'Apache httpd', 'version': '2.4.7', 'extrainfo':
'(Ubuntu)', 'conf': '10', 'cpe': 'cpe:/a:apache:http_server:2.4.7'}, 81: {'state':
'closed', 'reason': 'reset', 'name': 'hosts2-ns', 'product': '', 'version': '',
'extrainfo': '', 'conf': '3', 'cpe': ''}, 82: {'state': 'closed', 'reason':
'reset', 'name': 'xfer', 'product': '', 'version': '', 'extrainfo': '', 'conf':
'3', 'cpe': ''}, 83: {'state': 'closed', 'reason': 'reset', 'name': 'mit-ml-dev',
'product': '', 'version': '', 'extrainfo': '', 'conf': '3', 'cpe': ''}, 84:
{'state': 'closed', 'reason': 'reset', 'name': 'ctf', 'product': '', 'version':
'', 'extrainfo': '', 'conf': '3', 'cpe': ''}, 85: {'state': 'closed', 'reason':
'reset', 'name': 'mit-ml-dev', 'product': '', 'version': '', 'extrainfo': '',
'conf': '3', 'cpe': ''}}}}}
```

Powyższy wynik zawiera informację, że spośród wskazanych portów otwarty jest tylko jeden, o numerze 80. Poza tym są widoczne dane o wersji programu serwerowego. Wynik jest zwracany w formacie JSON i zawiera te same informacje, których dostarcza narzędzie Nmap użyte bezpośrednio w wierszu poleceń.

Wywołując metodę `command_line()` można sprawdzić, jakie polecenie zostało użyte do uruchomienia narzędzia Nmap:

```
>>> portScanner.command_line()
'nmap -oX - -p 80-85 -sV scanme.nmap.org'
```

Klasa `PortScanner` zawiera metodę `all_hosts()`, zwracającą listę przeskanowanych hostów. Można ją wykorzystać do uzyskania informacji o adresach IP i stanach hostów. Ilustruje to poniższy przykład.

```
>>> for host in portScanner.all_hosts():
...     print('Host: %s (%s)' % (host, portScanner[host].hostname()))
...     print('Stan: %s' % portScanner[host].state())
...
Host: 45.33.32.156 (scanme.nmap.org)
State: up
```

Za pomocą metody `scaninfo()` uzyskuje się informacje o zastosowanej technice skanowania oraz o sprawdzonych portach:

```
>>> portScanner.scaninfo()
{'tcp': {'method': 'syn', 'services': '80-85'}}
```

Poniższy skrypt wykorzystuje moduł `nmap` do skanowania wskazanego hosta. W argumentach metody `scan()` są umieszczone następujące informacje:

- skanowane porty: 21, 22, 23, 25 i 80,
- parametr `-n` oznaczający, że podczas skanowania nazwy nie będą odwzorowywane przy użyciu usługi DNS.

Kod jest zapisany w załączonym do książki pliku *skaner_portów.py*.

```
#!/usr/bin/env python3
import nmap

portScanner = nmap.PortScanner()
host_scan = input('Skanowany host: ')
portlist = "21,22,23,25,80"
portScanner.scan(hosts = host_scan, arguments = '-n -p' + portlist)
print(portScanner.command_line())
hosts_list = [(x, portScanner[x]['status']['state']) for x in portScanner.all_hosts()]
for host, status in hosts_list:
    print(host, status)
for protocol in portScanner[host].all_protocols():
    print('Protokół: %s' % protocol)
    listport = portScanner[host]['tcp'].keys()
    for port in listport:
        print('Port: %s, stan: %s' % (port,
portScanner[host][protocol][port]['state']))
```

Użyta w powyższym skrypcie metoda `all_protocols()` zwraca listę protokołów wykorzystanych podczas skanowania. Wynik działania skryptu jest następujący:

```
$ python skaner_portów.py
Skanowany host: scanme.nmap.org
nmap -oX - -n -p21,22,23,25,80 scanme.nmap.org
45.33.32.156 up
Protokół: tcp
Port: 21, stan: closed
Port: 22, stan: open
Port: 23, stan: closed
Port: 25, stan: filtered
Port: 80, stan: open
```

Wynik ten zawiera informacje o stanach wskazanych portów.

Teraz, gdy wiesz już, jak skanować określone porty za pomocą modułu `nmap`, poznaj dostępne tryby skanowania.

Tryby skanowania w module nmap

W tym podrozdziale przyjrzymy się trybom skanowania dostępnym w module `nmap`.

Automatyczne skanowanie może być wykonywane w dwóch trybach:

- **synchronicznym**, w którym skanowanie kolejnego portu rozpoczyna się po zakończeniu skanowania poprzedniego portu,
- **asynchronicznym**, w którym kilka portów jest skanowanych jednocześnie. W tym trybie definiuje się funkcję zwrotną, która jest wywoływana po zakończeniu skanowania portu. Funkcja ta może wykonywać dowolne operacje, na przykład sprawdzać stany portów lub skanować za pomocą narzędzia Nmap określoną usługę (HTTP, FTP, MySQL itp.).

Przyjrzyjmy się dokładniej obu trybom i zaimplementujmy je.

Implementacja skanowania synchronicznego

W poniższym skrypcie jest zdefiniowana klasa `NmapScanner`, służąca do skanowania zadanego hosta i portu. Adres IP hosta i listę portów przeznaczonych do przeskanowania należy podać w parametrach skryptu.

Kod jest zapisany w załączonym do książki pliku *NmapScanner.py*.

```
#!/usr/bin/env python3
import optparse
import nmap

class NmapScanner:
    def __init__(self):
        self.portScanner = nmap.PortScanner()

    def nmapScan(self, ip_address, port):
        self.portScanner.scan(ip_address, port)
        self.state = self.portScanner[ip_address]['tcp'][int(port)]['state']
        print(" [+] Wykonywane polecenie:", self.portScanner.command_line())
        print(" [+] " + ip_address + " tcp/" + port + " " + self.state)

def main():
    parser = optparse.OptionParser("%prog " +
             "--ip_address <adres hosta> --ports <lista portów>")
    parser.add_option('--ip_address', dest = 'ip_address', type = 'string',
                      help = 'Adres skanowanego hosta.')
    parser.add_option('--ports', dest = 'ports', type = 'string',
                      help = 'Lista portów oddzielonych przecinkami.')
    (options, args) = parser.parse_args()
    if (options.ip_address == None) | (options.ports == None):
        print('[-] Podaj adres IP hosta i listę portów.')
        exit(0)
    ip_address = options.ip_address
    ports = options.ports.split(',')
    for port in ports:
        NmapScanner().nmapScan(ip_address, port)

if __name__ == "__main__":
    main()
```

Na początku powyższego kodu jest zdefiniowana metoda `nmapScan()`, skanująca wskazanego w argumentach hosta i jego port. Funkcja `main()` najpierw przetwarza zadane parametry skryptu, a następnie wywołuje w pętli metodę `nmapScan()` w celu przeskanowania listy portów. Skrypt uruchomiony z parametrem `-h` wyświetla informacje o wymaganych parametrach:

```
$ python NmapScanner.py -h
Usage: NmapScanner.py --ip_address <adres hosta> --ports <lista portów>
Options:
  -h, --help            show this help message and exit
  --ip_address=IP_ADDRESS
                        Adres skanowanego hosta.
  --ports=PORTS         Lista portów oddzielonych przecinkami.
```

Jeśli po uruchomieniu skryptu podamy w parametrach adres hosta `45.33.32.156` (odpowiadający domenie *scanme.nmap.org*) i listę portów `21,22,23,25,80`, uzyskamy następujący wynik:

```
$ python NmapScanner.py --ip_address 45.33.32.156 --ports 21,22,23,25,80
 [+] Wykonywane polecenie: nmap -oX - -p 21 -sV 45.33.32.156
 [+] 45.33.32.156 tcp/21 closed
 [+] Wykonywane polecenie: nmap -oX - -p 22 -sV 45.33.32.156
 [+] 45.33.32.156 tcp/22 open
 [+] Wykonywane polecenie: nmap -oX - -p 23 -sV 45.33.32.156
 [+] 45.33.32.156 tcp/23 closed
 [+] Wykonywane polecenie: nmap -oX - -p 25 -sV 45.33.32.156
 [+] 45.33.32.156 tcp/25 filtered
 [+] Wykonywane polecenie: nmap -oX - -p 80 -sV 45.33.32.156
 [+] 45.33.32.156 tcp/80 open
```

Wyniki skanowania można nie tylko wyświetlać w terminalu, ale również zapisywać w pliku w formacie CSV. Poniższy kod jest zapisany w załączonym do książki pliku *NmapScannerCSV.py*.

```
#!/usr/bin/env python3
import optparse
import nmap
import csv

class NmapScannerCSV:
    def __init__(self):
        self.portScanner = nmap.PortScanner()

    def nmapScanCSV(self, host, ports):
        try:
            print("Skanowanie portów " + str(ports) + " ..........")
            self.portScanner.scan(host, arguments = '-n -p' + ports)
            print("[*] Wykonywane polecenie: %s" % self.portScanner.command_line())
            print(self.portScanner.csv())
            print("Podsumowanie hosta", host)
            with open('csv_file.csv', mode = 'w') as csv_file:
                csv_writer = csv.writer(csv_file, delimiter = ',')
                csv_writer.writerow(['Host', 'Protokół', 'Port', 'Stan'])
                for x in self.portScanner.csv().split("\n")[1:-1]:
                    splited_line = x.split(";")
                    host = splited_line[0]
                    protocol = splited_line[5]
                    port = splited_line[4]
                    state = splited_line[6]
                    print("Protokół: " + protocol + ", port: " + port +
                          ", stan: " + state)
                    csv_writer.writerow([host, protocol, port, state])
        except Exception as exception:
            print("Błąd połączenia z hostem " + host, exception)
```

W pierwszej części kodu jest wywoływana metoda `csv()` z klasy `PortScanner`, zwracająca dane w formacie CSV, pozwalającym na ich łatwą analizę. Każdy wiersz zawiera adres hosta, nazwę protokołu, numer portu i opis jego stanu. W drugiej części kodu jest zdefiniowana funkcja `main()`, przetwarzająca parametry skryptu:

```
def main():
    parser = optparse.OptionParser("%prog " +
             "--host <adres hosta> --ports <lista portów>")
    parser.add_option('--host', dest = 'host', type = 'string',
                      help = 'Adres skanowanego hosta.')
    parser.add_option('--ports', dest = 'ports', type = 'string',
                      help = 'Lista portów oddzielonych przecinkami.')
    (options, args) = parser.parse_args()
    if (options.host == None) | (options.ports == None):
        print('[-] Podaj adres IP hosta i listę portów.')
        exit(0)
    host = options.host
    ports = options.ports
    NmapScannerCSV().nmapScanCSV(host, ports)

if __name__ == "__main__":
    main()
```

Funkcja `main()` wywołuje również metodę `nmapScanCSV()`, w której argumentach umieszcza adres IP hosta i listę portów.

Wynik działania skryptu jest następujący:

```
$ python NmapScannerCSV.py --host 45.33.32.156 --ports 21,22,23,25,80 253
Skanowanie portów 21,22,23,25,80 ..........
[*] Wykonywane polecenie: nmap -oX - -n -p21,22,23,25,80 45.33.32.156
host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;
↪version;conf;cpe
45.33.32.156;;;tcp;21;ftp;closed;;;reset;;3;
45.33.32.156;;;tcp;22;ssh;open;;;syn-ack;;3;
45.33.32.156;;;tcp;23;telnet;closed;;;reset;;3;
45.33.32.156;;;tcp;25;smtp;filtered;;;no-response;;3;
45.33.32.156;;;tcp;80;http;open;;;syn-ack;;3;

Podsumowanie hosta 45.33.32.156
Protokół: ftp, port: 21, stan: closed
Protokół: ssh, port: 22, stan: open
Protokół: telnet, port: 23, stan: closed
Protokół: smtp, port: 25, stan: filtered
Protokół: http, port: 80, stan: open
```

Powyższy wynik zawiera polecenie `nmap` zwracające wyniki w formacie CSV. W każdym wierszu znajdują się adres hosta, nazwa protokołu, numer portu i opis jego stanu. Stan `conn-refused` oznacza, że port jest zamknięty, a `syn-ack`, że otwarty. Na końcu jest zamieszczone podsumowanie uzyskanych danych.

Poniższy skrypt wykorzystuje narzędzie Nmap nie tylko do uzyskania informacji o stanach portów wskazanego hosta, ale również o jego systemie operacyjnym. Poniższy kod jest zapisany w załączonym do książki pliku *system_operacyjny.py*.

```
import nmap, sys

command = "system_operacyjny.py <nazwa lub adres IP hosta>"
if len(sys.argv) == 1:
    print(command)
    sys.exit()
host = sys.argv[1]
portScanner = nmap.PortScanner()
open_ports_dict =  portScanner.scan(host, arguments = "-O -v")
if open_ports_dict is not None:
    open_ports_dict = open_ports_dict.get("scan").get(host).get("tcp")
    print("Port\tOpis")
    port_list = open_ports_dict.keys()
    for port in port_list:
        print(port, "\t", open_ports_dict.get(port)['name'])
    print("\n--------------Informacje o systemie opeacyjnym--------------------\n")
    print("Skanowany host:", portScanner[host]['osmatch'][0]['osclass'][0]['cpe'])
    print("Rodzina systemu operacyjnego:",
          portScanner[host]['osmatch'][0]['osclass'][0]['osfamily'])
    print("Typ systemu operacyjnego:",
          portScanner[host]['osmatch'][0]['osclass'][0]['type'])
    print("Wersja systemu operacyjnego:",
          portScanner[host]['osmatch'][0]['osclass'][0]['osgen'])
    print("Producent systemu operacyjnego:",
          portScanner[host]['osmatch'][0]['osclass'][0]['vendor'])
    print("Dokładność analizy:",
          portScanner[host]['osmatch'][0]['osclass'][0]['accuracy'])
```

W powyższym skrypcie jest wywoływana metoda `scan()` z klasy `PortScanner`. Argument `-O` oznacza, że metoda nie tylko skanuje porty hosta, ale również odczytuje informacje o jego systemie operacyjnym. Uzyskane dane są zapisywane w słowniku `portScanner[host]` i przypisywane do klucza `osmatch`.

Wynik uruchomienia powyższego skryptu jest następujący:

```
$ sudo python system_operacyjny.py 127.0.0.1
Port    Opis
22       ssh
631      ipp
--------------Informacje o systemie opeacyjnym--------------------
Skanowany host: ['cpe:/o:linux:linux_kernel:2.6.32']
Rodzina systemu operacyjnego: Linux
Typ systemu operacyjnego: general purpose
Wersja systemu operacyjnego: 2.6.X
Producent systemu operacyjnego: Linux
Dokładność analizy: 100
```

Wynik zawiera informacje o otwartych portach i systemie operacyjnym lokalnego komputera o adresie 127.0.0.1.

Powyższy skrypt należy uruchomić za pomocą polecenia `sudo`, ponieważ wymagany jest niskopoziomowy dostęp do gniazda sieciowego. Jeżeli po uruchomieniu pojawi się komunikat:

```
You requested a scan type which requires root privileges. QUITTING!
```
(Zażądałeś typu skanowania wymagającego uprawnień administratora. WYJŚCIE)

będzie to oznaczać, że należy użyć polecenia `sudo`.

Teraz, po zapoznaniu się z synchronicznym skanowaniem portów za pomocą modułu `nmap`, zajmijmy się trybem asynchronicznym, w którym można wykonywać wiele operacji skanowania jednocześnie.

Implementacja skanowania asynchronicznego

Do skanowania asynchronicznego służy klasa `PortScannerAsync`. W tego rodzaju skanowaniu niezbędne jest wskazanie funkcji zwrotnej, która ma być wywoływana po zakończeniu operacji.

Poniższy kod jest zapisany w załączonym do książki pliku *PortScannerAsync.py*.

```
import nmap
portScannerAsync = nmap.PortScannerAsync()

def callback_result(host, scan_result):
    print(host, scan_result)

portScannerAsync.scan(hosts = 'scanme.nmap.org', arguments = '-p 21',
                      callback = callback_result)
portScannerAsync.scan(hosts = 'scanme.nmap.org', arguments = '-p 22',
                      callback = callback_result)
portScannerAsync.scan(hosts = 'scanme.nmap.org', arguments = '-p 23',
                      callback = callback_result)
portScannerAsync.scan(hosts = 'scanme.nmap.org', arguments = '-p 80',
                      callback = callback_result)
while portScannerAsync.still_scanning():
    print("Skanowanie >>>")
    portScannerAsync.wait(5)
```

W skrypcie jest zdefiniowana funkcja `callback_result()`, wywoływana po zakończeniu skanowania zadanego hosta i portu. Pętla `while` jest wykonywana do chwili zakończenia ostatniego skanowania. Wynik działania skryptu jest następujący:

```
$ python PortScannerAsync.py
Skanowanie >>>
45.33.32.156 {'nmap': {'command_line': 'nmap -oX - -p 21 45.33.32.156',
'scaninfo': {'tcp': {'method': 'connect', 'services': '21'}}, 'scanstats':
{'timestr': 'Wed Jun 16 16:06:55 2021', 'elapsed': '0.38', 'uphosts': '1',
'downhosts': '0', 'totalhosts': '1'}}, 'scan': {'45.33.32.156': {'hostnames':
[{'name': 'scanme.nmap.org', 'type': 'PTR'}], 'addresses': {'ipv4':
'45.33.32.156'}, 'vendor': {}, 'status': {'state': 'up', 'reason': 'conn-
```

```
refused'}, 'tcp': {21: {'state': 'closed', 'reason': 'conn-refused', 'name':
'ftp', 'product': '', 'version': '', 'extrainfo': '', 'conf': '3', 'cpe': ''}}}}}
45.33.32.156 {'nmap': {'command_line': 'nmap -oX - -p 23 45.33.32.156',
'scaninfo': {'tcp': {'method': 'connect', 'services': '23'}}, 'scanstats':
{'timestr': 'Wed Jun 16 16:06:55 2021', 'elapsed': '0.38', 'uphosts': '1',
'downhosts': '0', 'totalhosts': '1'}}, 'scan': {'45.33.32.156': {'hostnames':
[{'name': 'scanme.nmap.org', 'type': 'PTR'}], 'addresses': {'ipv4':
'45.33.32.156'}, 'vendor': {}, 'status': {'state': 'up', 'reason': 'syn-ack'},
'tcp': {23: {'state': 'closed', 'reason': 'conn-refused', 'name': 'telnet',
'product': '', 'version': '', 'extrainfo': '', 'conf': '3', 'cpe': ''}}}}}
```

Jak widać, wyniki skanowania poszczególnych portów nie są zwracane w oryginalnej kolejności.

W poniższym przykładzie jest zdefiniowana klasa `NmapScannerAsync`, asynchronicznie skanująca hosta i porty wskazane w argumentach skryptu. Kod jest zapisany w załączonym do książki pliku *NmapScannerAsync.py*.

```
#!/usr/bin/env python3
import nmap
import argparse

def callbackResult(host, scan_result):
    port_state = scan_result['scan'][host]['tcp']
    print("Polecenie: " + scan_result['nmap']['command_line'])
    for key, value in port_state.items():
        print('Port {0} --> {1}'.format(key, value))
```

W pierwszej części skryptu jest zdefiniowana funkcja `callback_result()`, wywoływana po zakończeniu skanowania jednego portu. Funkcja wyświetla informacje o użytym poleceniu i stanie portu.

W następnej części jest zaimplementowana klasa `NmapScannerAsync`, zawierająca konstruktor `__init__()`, inicjujący właściwość `portScannerAsync`. Zdefiniowana jest również metoda `scanning()`, wywoływana po rozpoczęciu skanowania, oraz metoda `nmapScanAsync()`, wykonująca właściwe skanowanie.

```
class NmapScannerAsync:
    def __init__(self):
        self.portScannerAsync = nmap.PortScannerAsync()

    def scanning(self):
        while self.portScannerAsync.still_scanning():
            print("Skanowanie >>>")
            self.portScannerAsync.wait(5)

    def nmapScanAsync(self, hostname, port):
        try:
            print("Skanowanie portu " + port + " ..........")
            self.portScannerAsync.scan(hostname, arguments = "-A -sV -p" +
                                       port, callback = callbackResult)
            self.scanning()
        except Exception as exception:
            print("Błąd połączenia z hostem " + hostname, str(exception))
```

Metoda `nmapScanAsync()` skanuje wskazany w argumencie port, a po zakończeniu operacji wywołuje funkcję `callbackResult()`.

W trzeciej części skryptu przetwarzane są adres hosta i numery portów podane w parametrach skryptu, a następnie dla każdego portu jest wywoływana metoda `nmapScanAsync()`:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description = 'Asynchroniczny skaner portów')
    parser.add_argument("--host", dest = "host", help = "nazwa lub adres hosta",
                        required = True)
    parser.add_argument("-ports", dest = "ports",
                help = "lista portów oddzielonych przecinkami (domyślnie 80,8080)",
                default = "80,8080")
    parsed_args = parser.parse_args()
    port_list = parsed_args.ports.split(',')
    host = parsed_args.host
    for port in port_list:
        NmapScannerAsync().nmapScanAsync(host, port)
```

Wynik działania skryptu jest następujący:

```
$ python NmapScannerAsync.py --host scanme.nmap.org -ports 21,22,23,25,80
Skanowanie portu 21 ..........
Skanowanie portu 22 ..........
Skanowanie >>>
Skanowanie >>>
Polecenie: nmap -oX - -A -sV -p22 45.33.32.156
Port 22 --> {'state': 'open', 'reason': 'syn-ack', 'name': 'ssh', 'product':
'OpenSSH', 'version': '6.6.1p1 Ubuntu 2ubuntu2.13', 'extrainfo': 'Ubuntu Linux;
protocol 2.0', 'conf': '10', 'cpe': 'cpe:/o:linux:linux_kernel', 'script': {'ssh-
hostkey': '\n  1024 ac:00:a0:1a:82:ff:cc:55:99:dc:67:2b:34:97:6b:75 (DSA)\n  2048
20:3d:2d:44:62:2a:b0:5a:9d:b5:b3:05:14:c2:a6:b2 (RSA)\n  256
96:02:bb:5e:57:54:1c:4e:45:2f:56:4c:4a:24:b2:57 (ECDSA)\n  256
33:fa:91:0f:e0:e1:7b:1f:6d:05:a2:b0:f1:54:41:56 (EdDSA)'}}
Skanowanie portu 23 ..........
Skanowanie portu 25 ..........
Skanowanie >>>
Polecenie: nmap -oX - -A -sV -p25 45.33.32.156
Port 25 --> {'state': 'closed', 'reason': 'conn-refused', 'name': 'smtp',
'product': '', 'version': '', 'extrainfo': '', 'conf': '3', 'cpe': ''}
Skanowanie portu 80 ..........
Skanowanie >>>
Polecenie: nmap -oX - -A -sV -p80 45.33.32.156
Port 80 --> {'state': 'open', 'reason': 'syn-ack', 'name': 'http', 'product':
'Apache httpd', 'version': '2.4.7', 'extrainfo': '(Ubuntu)', 'conf': '10', 'cpe':
'cpe:/a:apache:http_server:2.4.7', 'script': {'http-server-header': 'Apache/2.4.7
(Ubuntu)', 'http-title': 'Go ahead and ScanMe!'}}
```

W powyższym wyniku są widoczne porty podane w parametrach skryptu, wykorzystywane polecenia oraz rezultaty skanowania zwrócone w postaci słownika.

Jak widać, porty o numerach 22 i 80 są otwarte, natomiast z kluczem `extrainfo` są skojarzone dodatkowe informacje o serwerze i usługach wykorzystujących skanowane porty.

W skanowaniu asynchronicznym wyniki nie zawsze są zwracane w kolejności skanowania portów, jak to ma miejsce podczas skanowania synchronicznego.

Po zapoznaniu się z trybami skanowania dostępnymi w module `nmap` zajmijmy się uruchamianiem narzędzia Nmap za pomocą modułów `os` i `subprocess`.

Uruchamianie narzędzia Nmap za pomocą modułów os i subprocess

W tym podrozdziale dowiesz się, jak uruchamiać narzędzie Nmap za pomocą modułów `os` i `subprocess` bez instalowania dodatkowych zależności.

Narzędzie Nmap najprościej uruchamia się z użyciem powłoki systemu operacyjnego. Poniższy kod jest zapisany w załączonym do książki pliku *nmap_os.py*.

```
import os

nmap_command = "nmap -sT 127.0.0.1"
os.system(nmap_command)
```

Po uruchomieniu skryptu otrzymujemy listę otwartych portów lokalnego komputera:

```
$ sudo python3 nmap_os.py
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000092s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
631/tcp open  ipp
```

Ten sam efekt można uzyskać wykorzystując moduł `subprocess` zawierający klasę `Popen`. W pierwszym argumencie konstruktora klasy należy umieścić tablicę zawierającą polecenie `nmap` i dodatkowe parametry. Klasa posiada również właściwości `stdout` i `stderr`, ułatwiające przetwarzanie informacji wysyłanych przez polecenie do standardowego strumienia wyjściowego i strumienia błędów.

Poniższy kod jest zapisany w załączonym do książki pliku *nmap_subprocess.py*.

```
from subprocess import Popen, PIPE

process = Popen(['nmap', '-O', '127.0.0.1'], stdout = PIPE, stderr = PIPE)
stdout, stderr = process.communicate()
print(stdout.decode())
```

W kodzie tym klasa `Popen` jest wykorzystywana do uruchomienia narzędzia Nmap z parametrem `-O` w celu uzyskania informacji o otwartych portach i o systemie operacyjnym lokalnego komputera. Wynik działania skryptu jest następujący:

```
$ sudo python nmap_subprocess.py
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000022s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
631/tcp open  ipp
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.32
OS details: Linux 2.6.32
Network Distance: 0 hops
OS detection performed. Please report any incorrect results at
https://nmap.org/submit/
```

Powyższy skrypt skanuje 1000 najczęściej stosowanych portów. Dla każdego z nich wyświetla informacje o wykorzystywanych protokole i usłudze. Uzyskany wynik zawiera również dane systemu operacyjnego lokalnego komputera.

Teraz, gdy wiesz już, jak korzystać z narzędzia Nmap za pomocą modułów `os` i `subprocess`, zajmijmy się wykrywaniem usług i ich podatności na ataki.

Wykrywanie usług i ich podatności na ataki za pomocą skryptów narzędzia Nmap

W tym podrozdziale dowiesz się, jak wykonywać zaawansowane wykrywanie usług oraz uzyskiwać informacje o ich podatnościach na ataki.

Uruchamianie skryptów narzędzia Nmap

Narzędzie Nmap doskonale nadaje się do skanowania sieci i usług. Jedną z jego wyjątkowych funkcjonalności jest silnik NSE (ang. *Nmap Scripting Engine*, silnik skryptowy Nmap), dostarczający zasób dodatkowych informacji o stanach usług, jak również o ich podatnościach na ataki, na przykład ShellShock, Poodle czy HeartBleed.

Narzędzie Nmap wykorzystuje do wykrywania podatności wydajny język skryptowy Lua. Dzięki temu może wykonywać zaawansowane operacje i uzyskiwać informacje o wskazanym hoście.

Skrypty uruchamia się za pomocą parametru `--script` i jednej lub kilku nazw poniższych kategorii:

- `auth` — skrypty wykrywające podatności stosowanego w usłudze mechanizmu uwierzytelnienia,
- `default` — podstawowe, domyślne skrypty,

- `discovery` — skrypty dostarczające informacje o zadanym hoście,
- `external` — skrypty wykorzystujące zewnętrzne zasoby,
- `intrusive` — skrypty, które mogą ingerować w działanie hosta,
- `malware` — skrypty sprawdzające, czy host jest zainfekowany szkodliwym kodem lub ma luki w zabezpieczeniach,
- `safe` — skrypty bezpieczne dla hosta,
- `vuln` — skrypty wykrywające popularne podatności,
- `all` — wszystkie powyższe skrypty.

W systemie Linux skrypty są zapisane w katalogu */usr/share/nmap/scripts*. Ich szczegółowe opisy są dostępne na stronach *http://nmap.org/book/man-nse.html* i *https://nmap.org/nsedoc/scripts*.

Poniżej jest pokazany przykład użycia polecenia `nmap` z parametrem `--script banner`. Polecenie to uruchamia skrypt odczytujący baner serwera zawierający informacje o uruchomionych usługach. Szczegółowy opis skryptu znajduje się na stronie *https://nmap.org/nsedoc/scripts/banner.html*.

```
$ sudo nmap -sSV --script banner scanme.nmap.org
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.18s latency).
Other addresses for scanme.nmap.org (not scanned):
2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 961 closed ports, 33 filtered ports
PORT      STATE SERVICE    VERSION
22/tcp    open  ssh        OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13
(Ubuntu Linux; protocol 2.0)
|_banner: SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.13
80/tcp    open  http       Apache httpd 2.4.7 ((Ubuntu))
|_http-server-header: Apache/2.4.7 (Ubuntu)
2000/tcp  open  tcpwrapped
5060/tcp  open  tcpwrapped
9929/tcp  open  nping-echo Nping echo
| banner: \x01\x01\x00\x18>\x95}\xA4_\x18d\xED\x00\x00\x00\x00\xD5\xBA\x8
|_6s\x97%\x17\xC2\x81\x01\xA5R\xF7\x89\xF4x\x02\xBAm\xCCA\xE3\xAD{\xBA...
31337/tcp open  tcpwrapped
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Powyższy wynik zawiera informacje o otwartych portach, wersjach usług wykorzystujących te porty, a także o systemie operacyjnym.

Innym ciekawym skryptem jest `discovery`, który dostarcza jeszcze więcej informacji o usługach działających na serwerze. Przykład jego użycia jest następujący:

```
$ sudo nmap --script discovery scanme.nmap.org
Pre-scan script results:
| targets-asn:
|_  targets-asn.asn is a mandatory parameter
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.17s latency).
Other addresses for scanme.nmap.org (not scanned):
2600:3c01::f03c:91ff:fe18:bb2f
All 1000 scanned ports on scanme.nmap.org (45.33.32.156) are filtered
```

```
Host script results:
| asn-query:
| BGP: 45.33.32.0/24 and 45.33.32.0/19 | Country: US
|   Origin AS: 63949 - LINODE-AP Linode, LLC, US
|_    Peer AS: 1299 2914 3257
| dns-brute:
|   DNS Brute-force hostnames:
|     ipv6.nmap.org - 2600:3c01:0:0:f03c:91ff:fe70:d085
|     chat.nmap.org - 45.33.32.156
|     chat.nmap.org - 2600:3c01:0:0:f03c:91ff:fe18:bb2f
|     *AAAA: 2600:3c01:0:0:f03c:91ff:fe98:ff4e
|_    *A: 45.33.49.119
...
```

Zaprezentowany tu wynik zawiera informację o uruchomieniu procesu `dns-brute` w celu uzyskania informacji o poddomenach i ich adresach IP.

Za pomocą skryptów można również uzyskiwać informacje o kluczach publicznych i algorytmach szyfrowania wykorzystywanych w usłudze SSH. Ilustruje to poniższy przykład:

```
$ sudo nmap -sSV -p22 --script ssh-hostkey scanme.nmap.org
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   1024 ac:00:a0:1a:82:ff:cc:55:99:dc:67:2b:34:97:6b:75 (DSA)
|   2048 20:3d:2d:44:62:2a:b0:5a:9d:b5:b3:05:14:c2:a6:b2 (RSA)
|   256 96:02:bb:5e:57:54:1c:4e:45:2f:56:4c:4a:24:b2:57 (ECDSA)
|_  256 33:fa:91:0f:e0:e1:7b:1f:6d:05:a2:b0:f1:54:41:56 (EdDSA)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
$ sudo nmap -sSV -p22 --script ssh2-enum-algos scanme.nmap.org
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu
Linux; protocol 2.0)
| ssh2-enum-algos:
|   kex_algorithms: (8)
|       curve25519-sha256@libssh.org
|       ecdh-sha2-nistp256
|       ecdh-sha2-nistp384
|       ecdh-sha2-nistp521
|       diffie-hellman-group-exchange-sha256
|       diffie-hellman-group-exchange-sha1
|       diffie-hellman-group14-sha1
|       diffie-hellman-group1-sha1
|   server_host_key_algorithms: (4)
|       ssh-rsa
|       ssh-dss
|       ecdsa-sha2-nistp256
|       ssh-ed25519
...
```

Powyższy wynik zawiera informacje o algorytmach szyfrowania danych stosowanych w usłudze SSH, oferowanej przez serwer *scanme.nmap.org domain* na porcie numer 22.

Po zapoznaniu się ze skryptami narzędzia Nmap wykrywającymi usługi i dostarczającymi dodatkowe informacje, zajmijmy się identyfikowaniem podatności usług na ataki.

Wykrywanie podatności usług na ataki

Narzędzie Nmap jest wyposażone w skrypty wykrywające podatności na ataki usługi FTP wykorzystującej port 21. Wśród nich jest skrypt `ftp-anon`, sprawdzający, czy jest możliwy anonimowy dostęp do usługi — bez podawania nazwy użytkownika i hasła. Użycie skryptu ilustruje poniższy przykład:

```
$ sudo nmap -sSV -p21 --script ftp-anon ftp.be.debian.org
PORT   STATE SERVICE VERSION
21/tcp open  ftp     ProFTPD
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
| lrwxrwxrwx   1 ftp      ftp            16 May 14  2011 backports.org ->
  ↪/backports.org/debian-backports
| drwxr-xr-x   9 ftp      ftp          4096 Jul 22 14:47 debian
| drwxr-sr-x   5 ftp      ftp          4096 Mar 13  2016 debian-backports
| drwxr-xr-x   5 ftp      ftp          4096 Jul 19 01:21 debian-cd
| drwxr-xr-x   7 ftp      ftp          4096 Jul 22 12:32 debian-security
| drwxr-sr-x   5 ftp      ftp          4096 Jan  5  2012 debian-volatile
| drwxr-xr-x   5 ftp      ftp          4096 Oct 13  2006 ftp.irc.org
| -rw-r--r--   1 ftp      ftp           419 Nov 17  2017 HEADER.html
| drwxr-xr-x  10 ftp      ftp          4096 Jul 22 14:05 pub
| drwxr-xr-x  20 ftp      ftp          4096 Jul 22 15:14 video.fosdem.org
|_-rw-r--r--   1 ftp      ftp           377 Nov 17  2017 welcome.msg
```

Kod ten asynchronicznie uruchamia różne skrypty testujące wskazany port usługi. Po wykonaniu każdego skryptu jest wywoływana funkcja wyświetlająca dodatkowe informacje o usłudze. Kod jest zapisany w załączonym do książki pliku *NmapScannerAsyncFTP.py*.

```
#!/usr/bin/env python3
import nmap
import argparse

def callbackFTP(host, result):
    try:
        script = result['scan'][host]['tcp'][21]['script']
        print("Polecenie: " + result['nmap']['command_line'])
        for key, value in script.items():
            print('Skrypt {0} --> {1}'.format(key, value))
    except KeyError:
        pass

class NmapScannerAsyncFTP:
    def __init__(self):
        self.portScanner = nmap.PortScanner()
        self.portScannerAsync = nmap.PortScannerAsync()

    def scanning(self):
        while self.portScannerAsync.still_scanning():
            print("Skanowanie >>>")
            self.portScannerAsync.wait(10)
```

W powyższym kodzie jest zdefiniowana funkcja `callbackFTP()`, wywoływana po zakończeniu działania każdego skryptu. Funkcja ta wyświetla szczegółowe informacje o danym skrypcie.

Zaprezentowana poniżej metoda asynchronicznie uruchamia skrypty sprawdzające port usługi wskazany w argumencie. Jako pierwszy jest uruchamiany skrypt *ftp-anon.nse*, sprawdzający możliwość anonimowego zalogowania się do usługi (jeżeli w argumencie jest podany port nr 21 i jest on otwarty).

```
def nmapScanAsync(self, hostname, port):
    try:
        print("Skanowanie portu " + port + " ..........")
        self.portScanner.scan(hostname, port)
        self.state = self.portScanner[hostname]['tcp'][int(port)]['state']
        print(" [+] "+ hostname + " tcp/" + port + " " + self.state)
        if (port == '21') and
            self.portScanner[hostname]['tcp'][int(port)]['state'] == 'open':
            print('Sprawdzanie usługi FTP za pomocą skryptów Nmap......')
            print('Skrypt ftp-anon.nse .....')
            self.portScannerAsync.scan(hostname, arguments =
                "-A -sV -p21 --script ftp-anon.nse", callback = callbackFTP)
            self.scanning()
```

W następnej części kodu są uruchamiane skrypty *ftp-bounce.nse*, *ftp-libopie.nse*, *ftp-proftpd-backdoor.nse* i *ftp-vsftpd-backdoor.nse*, identyfikujące podatności właściwe dla różnych wersji usługi FTP.

```
            print('Skrypt ftp-bounce.nse .....')
            self.portScannerAsync.scan(hostname, arguments =
                "-A -sV -p21 --script ftp-bounce.nse", callback = callbackFTP)
            self.scanning()
            print('Skrypt ftp-libopie.nse .....')
            self.portScannerAsync.scan(hostname, arguments =
                "-A -sV -p21 --script ftp-libopie.nse", callback = callbackFTP)
            self.scanning()
            print('Skrypt ftp-proftpd-backdoor.nse .....')
            self.portScannerAsync.scan(hostname, arguments =
                "-A -sV -p21 --script ftp-proftpd-backdoor.nse", callback = callbackFTP)
            self.scanning()
            print('Skrypt ftp-vsftpd-backdoor.nse .....')
            self.portScannerAsync.scan(hostname, arguments =
                "-A -sV -p21 --script ftp-vsftpd-backdoor.nse", callback = callbackFTP)
            self.scanning()
    except Exception as exception:
        print("Błąd połączenia z hostem " + hostname, str(exception))
```

Poniżej jest przedstawiony przykład użycia opisanego skryptu do zbadania serwera *ftp.be.debian.org*:

```
$ python NmapScannerAsyncFTP.py --host 195.234.45.114
Skanowanie portu 21 ..........
 [+] 195.234.45.114 tcp/21 open
Sprawdzanie usługi FTP za pomocą skryptów Nmap......
Skrypt ftp-anon.nse .....
Skanowanie >>>
Skanowanie >>>
Polecenie: nmap -oX - -A -sV -p21 --script ftp-anon.nse 195.234.45.114
```

```
Skrypt ftp-anon --> Anonymous FTP login allowed (FTP code 230)
lrwxrwxrwx   1 ftp      ftp            16 May 14  2011 backports.org ->
↪/backports.org/debian-backports
drwxr-xr-x   9 ftp      ftp          4096 Oct  1 14:44 debian
drwxr-sr-x   5 ftp      ftp          4096 Mar 13  2016 debian-backports
drwxr-xr-x   5 ftp      ftp          4096 Sep 27 06:17 debian-cd
drwxr-xr-x   7 ftp      ftp          4096 Oct  1 16:32 debian-security
drwxr-sr-x   5 ftp      ftp          4096 Jan  5  2012 debian-volatile
drwxr-xr-x   5 ftp      ftp          4096 Oct 13  2006 ftp.irc.org
-rw-r--r--   1 ftp      ftp           419 Nov 17  2017 HEADER.html
drwxr-xr-x  10 ftp      ftp          4096 Oct  1 16:06 pub
drwxr-xr-x  20 ftp      ftp          4096 Oct  1 17:14 video.fosdem.org
-rw-r--r--   1 ftp      ftp           377 Nov 17  2017 welcome.msg
Skrypt ftp-bounce.nse  .....
```

Zwrócony wynik zawiera informacje dotyczące usługi FTP wykorzystującej port numer 21, uzyskane za pomocą skryptów narzędzia Nmap. Wyniki można wykorzystać w kolejnym procesie identyfikowania zagrożeń usługi.

Podsumowanie

Ten rozdział jest poświęcony modułom umożliwiającym skanowanie portów wybranego serwera. Jednym z najlepszych tego typu modułów jest `nmap`, wykorzystujący narzędzie Nmap, dostarczające zasób informacji o otwartych portach i dostępnych usługach. Jedną z najważniejszych funkcjonalności narzędzia jest silnik NSE, będący prawdziwym skanerem podatności usług na ataki.

Wiedzę zawartą w tym rozdziale i opisane tu narzędzia możesz wykorzystywać do wykonywania testów penetracyjnych portów i usług udostępnianych przez serwery, a także do wykrywania ich podatności na ataki.

W następnym rozdziale są opisane otwarte skanery podatności Nessus i OpenVAS. Dowiesz się, jak ich używać w skryptach i uzyskiwać dzięki nim informacje o podatnościach serwerów i usług internetowych na ataki.

Pytania

Poniższa lista pytań pozwoli Ci sprawdzić wiedzę nabytą podczas lektury tego rozdziału. Odpowiedzi znajdziesz w dodatku „Odpowiedzi”.

1. Jaka metoda klasy `PortScanner` wykonuje skanowanie synchroniczne?
2. Jaka metoda klasy `PortScanner` wykonuje skanowanie asynchroniczne?
3. W jaki sposób za pomocą metody `scan()` asynchronicznie skanuje się porty i wywołuje funkcję zwrotną po zakończeniu skanowania?

4. W jaki sposób wykonuje się skanowanie synchroniczne wybranego hosta i portu po utworzeniu obiektu za pomocą instrukcji `self.portScanner = nmap.PortScanner()`?
5. Jaką funkcję należy zdefiniować w celu wykonania asynchronicznego skanowania za pomocą klasy `PortScannerAsync`?

Dalsza lektura

Na podanych niżej stronach znajdziesz dokumentację narzędzi i modułów opisanych w tym rozdziale, a także dodatkowe informacje o nich.

- Moduł `nmap`: *http://xael.org/pages/python-nmap-en.html*.
- Skrypty narzędzia Nmap: *https://nmap.org/nsedoc/scripts*.
- Napisane w języku Python narzędzie SPARTA (*https://sparta.secforce.com*), służące do skanowania portów i wykonywania testów penetracyjnych usług. Narzędzie jest zintegrowane z Nmap, umożliwia wprowadzenie zakresu adresów IP skanowanych hostów. Po zakończeniu skanowania wyświetla informacje o hostach, otwartych portach i uruchomionych usługach.

Python w sieci: najlepsza ochrona!

Popularność Pythona wynika z jego wszechstronności, prostoty, a także ze zwięzłości i z łatwości pisania kodu. Rozbudowywana z każdą aktualizacją kolekcja narzędzi i bibliotek pozwala na używanie Pythona do coraz bardziej specjalistycznych zadań, takich jak zabezpieczanie sieci. O tym, że skuteczna ochrona sieci ma krytyczne znaczenie dla organizacji, świadczą powtarzające się przypadki cyberataków i utraty cennych danych. Warto więc wykorzystać możliwości Pythona do wykrywania zagrożeń i rozwiązywania różnych problemów związanych z siecią.

Tę książkę docenią specjaliści do spraw bezpieczeństwa i inżynierowie sieci. Dzięki niej zapoznasz się z najnowszymi pakietami i bibliotekami Pythona i nauczysz się pisać skrypty, które pozwolą Ci zabezpieczyć sieć na wielu poziomach. Dowiesz się, w jaki sposób przesyłać dane i korzystać z sieci Tor. Nauczysz się też identyfikować podatności systemu na ataki, aby tym skuteczniej zapewnić mu bezpieczeństwo. W naturalny sposób przyswoisz wiedzę, która pozwoli Ci tworzyć w Pythonie bezpieczne aplikacje, zaczniesz również stosować techniki kryptograficzne i steganograficzne. Znajdziesz tu także wskazówki, jak rozwiązywać różne problemy sieciowe, pisać skrypty do wykrywania zagrożeń sieci i stron internetowych, zabezpieczać urządzenia końcowe, pozyskiwać metadane i pisać skrypty kryptograficzne.

Najważniejsze zagadnienia:

- skrypty automatyzujące procedury bezpieczeństwa i testy penetracyjne
- narzędzia programistyczne służące do zabezpieczania sieci
- automatyczna analiza serwerów
- wykrywanie podatności na ataki i analiza bezpieczeństwa
- praca z siecią Tor
- stosowanie narzędzi do analizy śledczej

José Manuel Ortega jest inżynierem oprogramowania i analitykiem bezpieczeństwa. Specjalizuje się w nowych technologiach i testowaniu otwartego oprogramowania. Współpracował z uczelniami informatycznymi, publikował artykuły i organizował konferencje. Był także prelegentem na różnych krajowych i międzynarodowych spotkaniach.

Helion
helion.pl
0 801 339900
0 601 339900
INFORMATYKA W NAJLEPSZYM WYDANIU

ISBN 978-83-283-8255-8
9 788328 382558
Cena: 79,00 zł
Packt