1. **介绍**

本章内容 ：

1.建立Python开发环境

2.Python语言简介

3.变量，字符串，列表，字典介绍

4.使用用网络，迭代器，异常处理，模块等

5.写第一个Python程序，字典密码破解器

6.写第二个Python程序，压缩文件密码暴力破解

对我来说，武术的非凡之处在于它的简单。简单是最美的，而武术也没有什么特别之处；以无法为有法，以有限为无限，是为武术最高境界！

——截拳道宗师 李小龙

**引文：用python 进行的一次渗透测试**

最近，我的一个朋友对一家世界财富500强公司的计算机安全系统进行了渗透测试。虽然该公司已建立和保持一个了优秀的安全机制，但他最终还是发现了一个存在漏洞而未打补丁的服务器。几分钟之内，他用开源工具入侵了这个系统并获得管理权。然后，他扫描了剩下的服务器以及客户机，并没有发现任何额外的漏洞。  
 从这一点看，他的测试似乎结束了，但是真正的渗透测试才刚刚开始。

他打开了自己常用的文本编辑器，写下了一个Python测试脚本，利用这个脚本发现了其余存在漏洞的服务器，几分钟后，他获得了网络上超过一千台机器的管理权，然而，在这样做时，他随后产生了一个难以管理的问题。 他知道，系统管理员会注意到他的攻击并拒绝再让他访问。所以，他赶紧想办法在自己已经控制的服务器上，安装永久的后门。

检查了一下自己渗透测试用到的文件后，我的朋友意识到他的这台客户机存在着很重要的域控制器。以此得知，管理员使用了一个完全独立的管理账户登陆域控制器，我的朋友写了一个小脚本检查1000台机器上已经登录的用户，过了一会，我的朋友被告知，域管理员登录到了一个机器。他的监测基本完成，我的朋友现在知道在哪里继续他的攻击了。

我朋友的迅速反应和他在压力下能创造性的思考的能力，促使他成为了一个渗透测试者。他为了成功入侵这个世界500强公司，自己写了脚本工具。

一个小的Python脚本帮助他入侵了一千多个工作站。另一个小脚本允许他在管理员发现前成功triage。一个真正的渗透测试者会编写自己的工具来解决所遇到的问题。

所以，让我们以安装开发环境为开始，学习如何打造自己的工具吧！

**建立开发环境**

Python的下载网站（http://www.python.org/download/）提供了Python在Windows，Mac OS X和Linux上的安装包。如果您运行的是Mac OS X或Linux，Python的解释器已经预先安装在了系统上。安装包为程序开发者提供了Python解释器，标准库和几个内置模块。 Python标准库和内置模块提供的功能范围广泛，包括内建的数据类型，异常处理，数字和数学模块，文件处理功能，如  
加密服务，与操作系统互操作性，网络数据处理，并与IP协议交互，还包括许多其他有用模块。同时，程序开发者可以很容易地安装任何第三方软件包。  
第三方软件包的完整列表可在（<http://pypi.python.org/pypi/>） 上看到

**安装第三方库**  
在第二章中，我们将利用python的python-nmap包来处理的NMAP的结果。下面的例子描述了如何下载和安装python-nmap包（或其他任何包，真的）。一旦我们已经保存了包到本地，我们解压这个包，并进入压缩后的目录中。在工录中，我们执行python setup.py命令来安装python-nmap包。安装大多数第三方包将遵循下载，解压，执行python setup.py命令进行安装的相同的步骤。

programmer:∼# wget http://xael.org/norman/python/python-nmap/python-

nmap-0.2.4.tar.gz-On map.tar.gz

--2012-04-24 15:51:51--http://xael.org/norman/python/python-nmap/

python-nmap-0.2.4.tar.gz

Resolving xael.org... 194.36.166.10

Connecting to xael.org|194.36.166.10|:80... connected.

HTTP request sent, awaiting response... 200 OK

Length: 29620 (29K) [application/x-gzip]

Saving to: 'nmap.tar.gz'

100%[==================================================

===================================================

=============>] 29,620 60.8K/s in 0.5s

2012-04-24 15:51:52 (60.8 KB/s) - 'nmap.tar.gz' saved [29620/29620]

programmer:∼# tar -xzf nmap.tar.gz

programmer:∼# cd python-nmap-0.2.4/

programmer:∼/python-nmap-0.2.4# python setup.py install

running install

running build

running build\_py

creating build

creating build/lib.linux-x86\_64-2.6

creating build/lib.linux-x86\_64-2.6/nmap

copying nmap/\_\_init\_\_.py -> build/lib.linux-x86\_64-2.6/nmap

copying nmap/example.py -> build/lib.linux-x86\_64-2.6/nmap

copying nmap/nmap.py -> build/lib.linux-x86\_64-2.6/nmap

running install\_lib

creating /usr/local/lib/python2.6/dist-packages/nmap

copying build/lib.linux-x86\_64-2.6/nmap/\_\_init\_\_.py -> /usr/local/lib/

python2.6/dist-packages/nmap

copying build/lib.linux-x86\_64-2.6/nmap/example.py -> /usr/local/lib/

python2.6/dist-packages/nmap

copying build/lib.linux-x86\_64-2.6/nmap/nmap.py -> /usr/local/lib/

python2.6/dist-packages/nmap

byte-compiling /usr/local/lib/python2.6/dist-packages/nmap/\_\_init\_\_.py

to \_\_init\_\_.pyc

byte-compiling /usr/local/lib/python2.6/dist-packages/nmap/example.py

to example.pyc

byte-compiling /usr/local/lib/python2.6/dist-packages/nmap/nmap.py to

nmap.pyc

running install\_egg\_info

Writing /usr/local/lib/python2.6/dist-packages/python\_nmap-0.2.4.egg-

info

为了能够更简单的安装python的包，python提供了easy\_install模块。运行这个简单的安装程序，程序将会在python库中寻找这个包，如果发现则下载它并自动安装

programmer:∼ # easy\_install python-nmap

Searching for python-nmap

Readinghttp://pypi.python.org/simple/python-nmap/

Readinghttp://xael.org/norman/python/python-nmap/

Best match: python-nmap 0.2.4

Downloadinghttp://xael.org/norman/python/python-nmap/python-nmap-

0.2.4.tar.gz

Processing python-nmap-0.2.4.tar.gz

Running python-nmap-0.2.4/setup.py -q bdist\_egg --dist-dir /tmp/easy\_

install-rtyUSS/python-nmap-0.2.4/egg-dist-tmp-EOPENs

zip\_safe flag not set; analyzing archive contents...

Adding python-nmap 0.2.4 to easy-install.pth file

Installed /usr/local/lib/python2.6/dist-packages/python\_nmap-0.2.4-

py2.6.egg

Processing dependencies for python-nmap

Finished processing dependencies for python-nmap

为了快速建立一个开发环境，我们建议您从http://www.backtrack-linux.org/downloads/下载最新的BackTrack Linux的渗透测试专版的复制版。他提供了丰富的渗透测试工具，例如forensic，Web，网络分析和无线攻击。之后的几个例子中。可能会用到一些早已内置在BackTrack的工具或库。当在本书的例子中，需要用到标准库和内置模块之外的第三方包的时候，文章将会提供包的下载网站。

设置一个开发环境时，提前下载好所有的这些第三方模块会是有用的。在BackTrack上，您可以通过执行easy\_install命令来安装额外需要的库，这将会在Linux下，下载大多数例子中用到的库。

programmer:∼ # easy\_install pyPdf python-nmap pygeoip mechanize

BeautifulSoup4

第五章用到了一些明确的不能从easy\_install下载的的蓝牙库。您可以使用包管理器下载并安装这些库。

attacker# apt-get install python-bluez bluetooth python-obexftp

Reading package lists... Done

Building dependency tree

Reading state information... Done

<..SNIPPED..>

Unpacking bluetooth (from .../bluetooth\_4.60-0ubuntu8\_all.deb)

Selecting previously deselected package python-bluez.

Unpacking python-bluez (from .../python-bluez\_0.18-1\_amd64.deb)

Setting up bluetooth (4.60-0ubuntu8) ...

Setting up python-bluez (0.18-1) ...

Processing triggers for python-central

此外，第五章和第七章中的几个例子需要一个Windows版的Python下载器。最新的Windows版的Python下载器，请访问http://[www.python.org/getit/](http://www.python.org/getit/)

最近几年python源代码已经延伸成了2.x和3.x两个分支。Python的原作者Guido van Rossum试图清理代码使语言变得更一致，这个行为打破了python 2.x版本与之后版本的兼容性，例如作者对prin语句的更改。在本书出版时，BackTrack 5 R2把Python 2.6.5作为稳定的python版本。

programmer# python -V

Python 2.6.5

**解释型python VS** **交互型python**

与其他脚本语言类似，Python是一种解释型语言。在运行时，解释器处理代码并执行他，为了演示python解释器的使用，我们写一个.py文件来打印“Hello World”。

为了解释这个程序，我们调用python解释器创建一个新的脚本。

programmer# echo print \"Hello World\" > hello.py

programmer# python hello.py

Hello World

此外，python具有交互能力，程序设计师可以调用python解释器，并直接与解释器“交流”。要启动解释器，程序开发者要先不带参数的执行python，接着解释器会呈现一个>>>来提示程序设计师，他可以接收命令了。在这里，程序设计师输入print “Hello World”。按下回车后，python交互解释器会立即执行该语句。

programmer# python

Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)

[GCC 4.4.3] on linux2

>>>

>>> print "Hello World"

Hello World

为了初步了解语言背后的含义，本章偶尔会用到python解释器的交互能力。你可以通过找寻>>>提示，来发现样例中对解释器的操作。

因为我们要解释下面章节中的样例，所以我们将通过数个被称为方法或函数的、有一定功能的代码块，来建立我们的脚本。每当我们完成一个脚本，我们将展示如何重新组合这些方法（函数）来让他们在main()中被调用。试图运行一个只有孤立的函数定义而不去调用他们的脚本是毫无意义的。大多数情况下，你都可以认出完整的脚本，因为他们都有定义好的main()函数。

在我们开始写我们的第一个程序前，我们将说明一些python标准库的重要组成部分。

**Python语言**

在下面的内容中，我们会讲解变量，数据类型，字符串，复杂的数据结构，网络，选择，循环，文件处理，异常处理，与操作系统进行交互。为了显示这一点，我们将构建一个简单的TCP类型的漏洞扫描器，读取来自服务的提示消息，并把他们与已知的存在漏洞的服务版本做比较，作为一个有经验的程序设计师，你可能会发现一些最初的示例代码的设计非常难看，事实上，我们希望你能在我们的代码基础上进行发展，使他变得优雅。

那么，让我们从任何编程语言的基础——变量开始吧！

**变量**

在python中，变量对应的数据存储在内存中，这种在内存中的位置可以存储不同的值，如整型，实数，布尔值，字符串，或更复杂的数据结构，例如列表或字典。在下面的代码中，我们定义一个存储整形的变量和一个存储字符串的提示消息，为了把这两个变量连接到一个字符串中，我们必须用str()函数。

>>> port = 21

>>> banner = "FreeFloat FTP Server"

>>> print "[+] Checking for "+banner+" on port "+str(port)

[+] Checking for FreeFloat FTP Server on port 21

当程序设计师声明变量后，python为这些变量保存了内存空间。程序设计师不必声明变量的类型，相反，python解释器决定了变量类型何在内存中为他保留的空间的大小。思考下面的例子，我们正确的声明了一个字符串，一个整数，一个列表和一个布尔值，解释器都自动的正确的识别了每个变量的类型。

>>> banner = "FreeFloat FTP Server" # A string

>>> type(banner)

<type 'str'>

>>> port = 21 # An integer

>>> type(port)

<type 'int'>

>>> portList=[21,22,80,110] # A list

>>> type(portList)

<type 'list'>

>>> portOpen = True # A boolean

>>> type(portOpen)

<type 'bool'>

**字符串**

在python中字符串模块提供了一系列非常强大的字符串操作方法。阅读http://docs.python.org/library/string.html上的用法列表的python文档。

让我们来看几个常用的函数。思考下面这些函数的用法,upper()方法将字符串中的小写字母转为大写字母，lower()方法转换字符串中所有大写字母为小写,replace(old,new)方法把字符串中的old(旧字符串) 替换成 new(新字符串)，find()方法检测字符串中是否包含指定的子字符串。

>>> banner = "FreeFloat FTP Server"

>>> print banner.upper()

FREEFLOAT FTP SERVER

>>> print banner.lower()

freefloat ftp server

>>> print banner.replace('FreeFloat','Ability')

Ability FTP Server

>>> print banner.find('FTP')

10

**列表**

Python的数据结构——列表，提供了一种存储一组数据的方式。程序设计师可以构建任何数据类型的列表。另外，有一些内置的操作列表的方法，例如添加，删除，插入，弹出，获取索引，排序，计数，排序和反转。请看下面的例子，一个程序通过使用append()添加元素来建立一个列表，打印项目，然后在再次输出前给他们排序。程序设计师可以找到特殊元素的索引（例如样例中的80），此外，指定的元素也可以被移动。(例如样例中的443)

>>> portList = []

>>> portList.append(21)

>>> portList.append(80)

>>> portList.append(443)

>>> portList.append(25)

>>> print portList

[21, 80, 443, 25]

>>> portList.sort()

>>> print portList

[21, 25, 80, 443]

>>> pos = portList.index(80)

>>> print "[+] There are "+str(pos)+" ports to scan before 80."

[+] There are 2 ports to scan before 80.

>>> portList.remove(443)

>>> print portList

[21, 25, 80]

>>> cnt = len(portList)

>>> print "[+] Scanning "+str(cnt)+" Total Ports."

[+] Scanning 3 Total Ports.

**字典**

Python的数据结构——字典，提供了一个可以存储任何数量python对象的哈希表。字典的元素由键和值组成，让我们继续用我们的漏洞扫描器的例子来讲解python的字典。当扫描指定的TCP端口是，用字典包含每个端口对应的常见的服务名会很有用。建立一个字典，我们能查找像ftp这样的键并返回端口关联的值21。

当我们建立一个字典时，每一个键和他的用被冒号隔开，同时，我们用逗号分隔元素。注意，.keys()这个方法将返回字典的所有键的列表，.items()这个方法将返回字典的元素的一系列列表。接下来，我们验证字典是否包含了指定的键(ftp)，伴随着键，值21返回了。

>>> services = {'ftp':21,'ssh':22,'smtp':25,'http':80}

>>> services.keys()

['ftp', 'smtp', 'ssh', 'http']

>>> services.items()

[('ftp', 21), ('smtp', 25), ('ssh', 22), ('http', 80)]

>>> services.has\_key('ftp')

True

>>> services['ftp']

21

>>> print "[+] Found vuln with FTP on port "+str(services['ftp'])

[+] Found vuln with FTP on port 21

**网络**

套接字模块提供了一个可以使python建立网络连接的库。让我们快速的编写一个获取提示信息的脚本，连接到特定IP地址和端口后，我们的脚本将打印提示信息，之后，我们使用connect()函数连接到IP地址和端口。一旦连接成功，就可以通过套接字进行读写。这种recv(1024)的方法将读取之后在套接字中1024字节的数据。我们把这种方式的结果存到一个变量中，然后打印到服务器。

>>> import socket

>>> socket.setdefaulttimeout(2)

>>> s = socket.socket()

>>> s.connect(("192.168.95.148",21))

>>> ans = s.recv(1024)

>>> print ans

220 FreeFloat Ftp Server (Version 1.00).

**选择**

像大多数编程语言一样，python提供了条件选择的方式，通过if语句，计算一个逻辑表达式来判断选择的结果。继续写我们的脚本，我们想知道，是否指定的FTP服务器是容易受到攻击的。要做到这一点，我们要拿我们的结果和已知的易受攻击的FTP服务器版本作比较。

>>> import socket

>>> socket.setdefaulttimeout(2)

>>> s = socket.socket()

>>> s.connect(("192.168.95.148",21))

>>> ans = s.recv(1024)

>>> if ("FreeFloat Ftp Server (Version 1.00)" in ans):

... print "[+] FreeFloat FTP Server is vulnerable."

...elif ("3Com 3CDaemon FTP Server Version 2.0" in banner):

... print "[+] 3CDaemon FTP Server is vulnerable."

... elif ("Ability Server 2.34" in banner):

... print "[+] Ability FTP Server is vulnerable."

... elif ("Sami FTP Server 2.0.2" in banner):

... print "[+] Sami FTP Server is vulnerable."

... else:

... print "[-] FTP Server is not vulnerable."

...

[+] FreeFloat FTP Server is vulnerable."

**异常处理**

即使一个程序设计师编写的程序语法正确，该程序仍然可能在运行或执行时发生错误。考虑经典的一种运行错误——除以零。因为零不能做除数，所以python解释器显示一条消息，把错误信息告诉程序设计师：该错误使程序停止执行。 >>> print 1337/0

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ZeroDivisionError: integer division or modulo by ze

如果我们想在我们预设的范围内处理错误，会对运行的程序产生什么影响呢？python语言提供的异常处理能力就可以这样做。让我们来更新前面的例子，我们使用try/except进行异常处理。现在程序试图除以零。当错误发生时，我们的异常处理捕获错误并把错误信息打印到屏幕上。

>>> try:

... print "[+] 1337/0 = "+str(1337/0)

... except:

... print "[-] Error. "

...

[-] Error

>>>

不幸的是，这给了我们非常少的关于错误的异常处理的信息。但在对待特殊错误时，这可能很有用，要做到这一点，我们将存储异常信息到一个变量中，来打印出异常信息。

>>> try:

... print "[+] 1337/0 = "+str(1337/0)

... except Exception, e:

... print "[-] Error = "+str(e)

...

[-] Error = integer division or modulo by zero

>>>

现在，让我们用异常处理来更新我们的脚本，我们用异常处理把网络连接代码包装起来，接下来，我们连接到一台围在TCP端口21上开放FTP服务的机器。如果我们等待连接超时，我们将看到一条信息来表明网络连接操作超时。然后，我们的程序可以继续运行。

>>> import socket

>>> socket.setdefaulttimeout(2)

>>> s = socket.socket()

>>> try:

... s.connect(("192.168.95.149",21))

... except Exception, e:

... print "[-] Error = "+str(e)

...

[-] Error = Operation timed out

在本书中，让我们为你提供一个与异常处理有关的警告，为了清楚的说明各种各样的概念，在下面的内容中，我们已经在最小的地方都添加了异常处理，但我们仍然欢迎你更新这新脚本，并把强化的异常处理代码分享到配到网站上。

**函数**

在python中，函数提供了组建好的，可反复使用的代码片段。通常，这允许程序设计师写代码来执行单独或关联的行为。

尽管python提供了许多内置函数，程序设计师仍然可以创建自定义的函数。关键字def()开始了一个函数，程序设计师可以把任何变量放到括号里。这些变量随后被传递，这意味着在函数内部对这些变量的任何变化，都将影响调用的函数的值。继续以我们的FTP漏洞扫描器为例，让我们创建一个函数来执行只连接到FTP服务器的操作并返回提示信息

import socket

def retBanner(ip, port):

try:

socket.setdefaulttimeout(2)

s = socket.socket()

s.connect((ip, port))

banner = s.recv(1024)

return banner

except:

return

def main():

ip1 = '192.168.95.148'

ip2 = '192.168.95.149'

port = 21

banner1 = retBanner(ip1, port)

if banner1:

print '[+] ' + ip1 + ': ' + banner1

banner2 = retBanner(ip2, port)

if banner2:

print '[+] ' + ip2 + ': ' + banner2

if \_\_name\_\_ == '\_\_main\_\_':

main()

在返回信息后，我们的脚本需要与已知存在漏洞的程序进行核对。这也反映了函数的单一性和相关性。该函数checkVulns()用获得的信息来对服务器存在的漏洞进行判断。

import socket

def retBanner(ip, port):

try:

socket.setdefaulttimeout(2)

s = socket.socket()

s.connect((ip, port))

banner = s.recv(1024)

return banner

except:

return

def checkVulns(banner):

if 'FreeFloat Ftp Server (Version 1.00)' in banner:

print '[+] FreeFloat FTP Server is vulnerable.'

elif '3Com 3CDaemon FTP Server Version 2.0' in banner:

print '[+] 3CDaemon FTP Server is vulnerable.'

elif 'Ability Server 2.34' in banner:

print '[+] Ability FTP Server is vulnerable.'

elif 'Sami FTP Server 2.0.2' in banner:

print '[+] Sami FTP Server is vulnerable.'

else:

print '[-] FTP Server is not vulnerable.'

return

def main():

ip1 = '192.168.95.148'

ip2 = '192.168.95.149'

ip3 = '192.168.95.150'

port = 21

banner1 = retBanner(ip1, port)

if banner1:

print '[+] ' + ip1 + ': ' + banner1.strip('\n’)

checkVulns(banner1)

banner2 = retBanner(ip2, port)

if banner2:

print '[+] ' + ip2 + ': ' + banner2.strip('\n')

checkVulns(banner2)

banner3 = retBanner(ip3, port)

if banner3:

print '[+] ' + ip3 + ': ' + banner3.strip('\n')

checkVulns(banner3)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**迭代**

上一章中，你可能会发现我们几乎重复三次写了相同的代码，来检测三个不同的IP地址。

代替反复做一件事，使用for循环便利多个元素会更加容易。举个例子：如果我们想便利整个整个IP地址从192.168.98.1到192.168.95.254的子网，我们要用一个for循环从1到255进行遍历，来打印出子网内的信息。

>>> for x in range(1,255):

... print "192.168.95."+str(x)

...

192.168.95.1

192.168.95.2

192.168.95.3

192.168.95.4

192.168.95.5

192.168.95.6

... <SNIPPED> ...

192.168.95.253

192.168.95.254

同样，我们可能需要遍历已知的端口列表来检查漏洞。代替一系列的数字，我们可以通过一个元素列表遍历他们。

>>> portList = [21,22,25,80,110]

>>> for port in portList:

... print port

...

21

22

25

80

110

嵌套了两个for循环，现在我们可以打印出每个IP地址和端口了。

>>> for x in range(1,255):

... for port in portList:

... print "[+] Checking 192.168.95."\

+str(x)+": "+str(port)

...

[+] Checking 192.168.95.1:21

[+] Checking 192.168.95.1:22

[+] Checking 192.168.95.1:25

[+] Checking 192.168.95.1:80

[+] Checking 192.168.95.1:110

[+] Checking 192.168.95.2:21

[+] Checking 192.168.95.2:22

[+] Checking 192.168.95.2:25

[+] Checking 192.168.95.2:80

[+] Checking 192.168.95.2:110

<... SNIPPED ...>

随着程序有了遍历IP和端口的能力，我们也将个更新我们的漏洞检测脚本，现在，我们的脚本将测试全部254个IP地址所提供的telnet, SSH, smtp, http,

imap, and https服务。

import socket

def retBanner(ip, port):

try:

socket.setdefaulttimeout(2)

s = socket.socket()

s.connect((ip, port))

banner = s.recv(1024)

return banner

except:

return

def checkVulns(banner):

if 'FreeFloat Ftp Server (Version 1.00)' in banner:

print '[+] FreeFloat FTP Server is vulnerable.'

elif '3Com 3CDaemon FTP Server Version 2.0' in banner:

print '[+] 3CDaemon FTP Server is vulnerable.'

elif 'Ability Server 2.34' in banner:

print '[+] Ability FTP Server is vulnerable.'

elif 'Sami FTP Server 2.0.2' in banner:

print '[+] Sami FTP Server is vulnerable.'

else:

print '[-] FTP Server is not vulnerable.'

return

def main():

portList = [21,22,25,80,110,443]

for x in range(1, 255):

ip = '192.168.95.' + str(x)

for port in portList:

banner = retBanner(ip, port)

if banner:

print '[+] ' + ip + ': ' + banner

checkVulns(banner)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**文件I/O**

虽然我们的脚本已有了一些能帮助检测漏洞信息的if语句，但加进一个漏洞列表会更好，举个例子，假设我们有一个叫做vuln\_banners.txt的文本文件。在每一行该文件列出了具体的服务版本和已知的之前的漏洞，我们不需要构建一个庞大的if语句，让我们读取这个文本文件，并用他来判断是否我们的提示信息存在漏洞。

programmer$ cat vuln\_banners.txt

3Com 3CDaemon FTP Server Version 2.0

Ability Server 2.34

CCProxy Telnet Service Ready

ESMTP TABS Mail Server for Windows NT

FreeFloat Ftp Server (Version 1.00)

IMAP4rev1 MDaemon 9.6.4 ready

MailEnable Service, Version: 0-1.54

NetDecision-HTTP-Server 1.0

PSO Proxy 0.9

SAMBAR

Sami FTP Server 2.0.2

Spipe 1.0

TelSrv 1.5

WDaemon 6.8.5

WinGate 6.1.1

Xitami

YahooPOPs! Simple Mail Transfer Service Ready

我们将会把我们更新后的代码放到函数checkVulns()中。在这里我们将用只读模式(’r’)打开文本文件。然后使用函数readlines()遍历文件的每一行，对每一行，我们把他与我们的提示信息作比较，注意我们必须用方法.strip(‘\r’)去掉每行的回车符，如果发现一对匹配了，我们打印出有漏洞的服务信息。

def checkVulns(banner):

f = open("vuln\_banners.txt",'r')

for line in f.readlines():

if line.strip('\n') in banner:

print "[+] Server is vulnerable: "+banner.strip('\n')

**SYS模块**

内置的sys模块提供访问和维护python解释器的能力。这包括了提示信息，版本，整数的最大值，可用模块，路径钩子，标准错误，标准输入输出的定位和解释器调用的命令行参数。你能够在python的在线模块文档上找到更多与此相关的信息（http://docs.python.org/library/sys）。在创建python脚本时与sys模块交互会十分有用。我们可以，例如，想在程序运行时解析命令行参数。思考下我们的漏洞扫描器，

如果我们想要把文本文件的名字作为命令行参数传递会怎么样呢？领标sys.argv包含了全部的命令含参数。第一个索引sys.argv[0]包含了python脚本解释器的名称。列表中剩余的元素包含了以下全部的命令行参数。因此，如果我们只想传递附加的参数，sys.argv应该包含两个元素。

import sys

if len(sys.argv)==2:

filename = sys.argv[1]

print "[+] Reading Vulnerabilities From: "+filename

运行我们的代码片段，我们看到代码成功的解析了命令行参数并把他打印到了屏幕上。你可以花时间来学习下全部的sys模块提供给程序设计师的丰富的功能。

programmer$ python vuln-scanner.py vuln-banners.txt

[+] Reading Vulnerabilities From: vuln-banners.tx

**OS模块**

内置的OS模块提供了丰富的与MAC,NT,Posix等操作系统进行交互的能力。这个模块允许程序独立的与操作系统环境。文件系统，用户数据库和权限进行交互。思考一下，比如，上一章中，用户把文件名作为命令行参数来传递。他可以验证文件是否存在以及当前用户是否有权限都这个文件。如果失败，他将显示一条信息，来显示一个适当的错误信息给用户。

import sys

import os

if len(sys.argv) == 2:

filename = sys.argv[1]

if not os.path.isfile(filename):

print '[-] ' + filename + ' does not exist.'

exit(0)

if not os.access(filename, os.R\_OK):

print '[-] ' + filename + ' access denied.'

exit(0)

print '[+] Reading Vulnerabilities From: ' + filename

为了验证我们的代码，我们尝试读取一个不存在的文件，该文件使我们的程序打印出了错误信息，接下来，我们创建这个文件，我们的脚本成功的读取了他。最后我们限制了权限，我们的脚本正确的打印了拒绝访问的消息。

programmer$ python test.py vuln-banners.txt

[-] vuln-banners.txt does not exist.

programmer$ touch vuln-banners.txt

programmer$ python test.py vuln-banners.txt

[+] Reading Vulnerabilities From: vuln-banners.txt

programmer$ chmod 000 vuln-banners.txt

programmer$ python test.py vuln-banners.txt

[-] vuln-banners.txt access denied

现在我们可以重新组合漏洞扫描程序的各个零件。不用担心他会错误终止或是在执行时缺少使用线程的能力或是更好的分析命令行的能力，我们将会在后面的章节继续改进这个脚本

Import socket

import os

import sys

def retBanner(ip, port):

try:

socket.setdefaulttimeout(2)

s = socket.socket()

s.connect((ip, port))

banner = s.recv(1024)

return banner

except:

return

def checkVulns(banner, filename):

f = open(filename, 'r')

for line in f.readlines():

if line.strip('\n') in banner:

print '[+] Server is vulnerable: ' +\

banner.strip('\n')

def main():

if len(sys.argv) == 2:

filename = sys.argv[1]

if not os.path.isfile(filename):

print '[-] ' + filename +\

' does not exist.'

exit(0)

if not os.access(filename, os.R\_OK):

print '[-] ' + filename +\

' access denied.'

exit(0)

else:

print '[-] Usage: ' + str(sys.argv[0]) +\

' <vuln filename>'

exit(0)

portList = [21,22,25,80,110,443]

for x in range(147, 150):

ip = '192.168.95.' + str(x)

for port in portList:

banner = retBanner(ip, port)

if banner:

print '[+] ' + ip + ': ' + banner

checkVulns(banner, filename)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**你的的第一个Python程序**

随着了解了如何构建python脚本，让我们开始写我们的第一个程序。在我们向前迈进前，我们将描述一些轶闻轶事，强调我们的脚本的需要。

为你的第一个程序设立个平台：

**杜鹃蛋**

C. Stoll的《杜鹃蛋》(1989)堪称新派武侠的开山之作。它第一次把黑客活动与国家安全联系在一起。黑客极具破坏性的黑暗面也浮出海面，并且永远改变了黑客的形象。迄今仍是经久不衰的畅销书。 　　Stoll是劳伦斯伯克利实验室的天文学家和系统管理员。1986年夏，一个区区75美分的帐目错误引起了他的警觉，在追查这次未经授权的入侵过程中，他开始卷入一个错综复杂的电脑间谍案。神秘的入侵者是西德混沌俱乐部的成员。他们潜入美国，窃取敏感的军事和安全情报。出售给克格勃，以换取现金及可卡因。一场网络跨国大搜索开始了，并牵涉出FBI、CIA、克格勃、西德邮电部等。《杜鹃蛋》为后来的黑客作品奠定了一个主题：追捕与反追捕的惊险故事。而且也开始了新模式：一个坚韧和智慧的孤胆英雄，成为国家安全力量的化身，与狡猾的对手展开传奇的较量。

（该故事已经有经典的翻译版本，可以直接参考）

下载地址：[http://pan.baidu.com/s/1kTCNwMF 密码ug42](http://pan.baidu.com/s/1kTCNwMF%20密码ug42)

**你的第一个程序，一个UNIX密码破解器！**

我们只需要用标准库中的crypt模块的crypt()函数。传入密码和盐即可。

让我们赶快试一试用crypt()函数哈希一个密码试试，我们输入密码”egg”和盐”HX”，返回的哈希密码值是”HX9LLTdc/jiDE”，现在我们可以遍历整个字典，试图用常用的盐来匹配破解哈希密码！

>>>import crypt

>>>crypt.crypt(‘egg’, ‘HX’)

“HX9LLTdc/jiDE”

>>>

注意：哈希密码的前两位就是盐的前两位，这里我们假设盐只有两位。

程序分两部分，一部分是打开字典，另一部分是哈希匹配密码，代码如下：

# coding=UTF-8  
"""  
暴力破解UNIX的密码，需要输入字典文件和UNIX的密码文件  
"""  
**import** crypt  
**def testPass**(cryptPass):  
 salt = cryptPass[0:2]  
 dictfile = open('dictionary.txt', 'r') #打开字典文件  
 **for** word **in** dictfile.readlines():  
 word = word.strip('\n') #保留原始的字符，不去空格  
 cryptWord = crypt.crypt(word, salt)  
 **if** cryptPass == cryptWord:  
 **print**('Found passed : ', word)  
 **return  
 print**('Password not found !')  
 **return  
def main**():  
 passfile = open('passwords.txt', 'r') #读取密码文件  
 **for** line **in** passfile.readlines():  
 user = line.split(':')[0]  
 cryptPass = line.split(':')[1].strip('')  
 **print**("Cracking Password For :", user)  
 testPass(cryptPass)  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

但是现代的×NIX系统将密码存储在/etc/shadow文件中，提供了个更安全的哈希散列算法SHA-512算法，Python的标准库中hashlib模块提供了此算法，我们可以更新我们的脚本，破解SHA-512哈希散列加密算法的密码。

root@DJ-PC:/home/dj# cat /etc/shadow | grep root

root:$6$t0dy7TXs$mJxj1Ydfx83Eg0b7ry1etUQA8g7GliedT2DlnlLhiEunizJ1AAzSzQLfzV5J17D0MsZVwUVjP/0KHGV5Ue33F1:16411:0:99999:7:::

**你的第二个程序：ZIP文件密码破解**

Python的标准库提供了ZIP文件的提取压缩模块zipfile，现在让我们试着用这个模块，暴力破解出加密的ZIP文件！

我们可以用extractall()这个函数抽取文件，密码正确则返回正确，密码错误测抛出异常。

现在我们可以增加一些功能，将上面的单线程程序变成多线程的程序，来提高破解速度。

两个程序代码如下，注释处为单线程代码：

# coding=UTF-8  
"""  
用字典暴力破解ZIP压缩文件密码  
"""  
**import** zipfile

**import** threading  
**def extractFile**(zFile, password):  
 **try**:  
 zFile.extractall(pwd=password)

**print**("Found Passwd : ", password)  
 **return** password  
 **except**:  
 **pass  
def main**():  
 zFile = zipfile.ZipFile('unzip.zip')  
 passFile = open('dictionary.txt')  
 **for** line **in** passFile.readlines():  
 password = line.strip('\n')  
 t = threading.Thread(target=extractFile, args=(zFile, password))  
 t.start()  
 """  
 guess = extractFile(zFile, password)  
 if guess:  
 print('Password = ', password)  
 return  
 else:  
 print("can't find password")  
 return  
 """  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

现在，我们想用户可以指定要破解的文件和字典，我们需要借助Python标准库中的optparse模块来指定参数，具体的讲解将在下一章讲解，这里我们只提供本例的代码：

# coding=UTF-8  
"""  
ZIP压缩文件破解程序加强版，用户可以自己指定想要破解的文件和破解字典，多线程破解  
"""  
**import** zipfile  
**import** threading  
**import** optparse  
**def extractFile**(zFile, password):  
 **try**:  
 zFile.extractall(pwd=password)  
 **print**("Found Passwd : ", password)  
 **except**:  
 **pass  
def main**():  
 parser = optparse.OptionParser('usage%prog -f <zipfile> -d <dictionary>')  
 parser.add\_option('-f', dest='zname', type='string', help='specify zip file')  
 parser.add\_option('-d', dest='dname', type='string', help='specify dictionary file')  
 options, args = parser.parse\_args()  
 **if** options.zname == None | options.dname == None:  
 **print**(parser.usage)  
 exit(0)  
 **else**:  
 zname = options.zname  
 dname = options.dname  
 zFile = zipfile.ZipFile(zname)  
 dFile = open(dname, 'r')  
 **for** line **in** dFile.readlines():  
 password = line.strip('\n')  
 t = threading.Thread(target=extractFile, args=(zFile, password))  
 t.start()  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

**本章总结**

本章我们就认识了Python的基本用法，写了一个UNIX的密码破解器和ZIP文件密码破解器，下一章我们将用Python做进一步的渗透测试！