



FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA
TECNÓLOGO EM DEFESA CIBERNÉTICA

Relatório Técnico:

Global Solution - 1º Semestre

Túlio Tavares Deládio Silva

Belo Horizonte, MG

2024

SUMÁRIO

LISTA DE FIGURAS	II
1 INTRODUÇÃO	1
2 DESENVOLVIMENTO DOS CTFs	2
2.1 URL SUSPEITA	2
2.2 SOMANDO	3
2.3 TOP USER-AGENT	4
2.4 TOP IP	5
2.5 O SEXTO HOST	5
2.6 USUÁRIO DESTRUIDOR	6
2.7 VAZAMENTO	7
2.8 CRIPTOGRAFIA ALIENÍGENA	8
2.9 OBSOLETO	9
2.10 CONVERSA SUSPEITA	10
2.11 SOMENTE NÚMEROS	12
2.12 BLUE OF DEATH	13
2.13 TEMPO E ESPAÇO	14
2.14 CABO PERDIDO	15
2.15 COMANDO BÁSICO	17
3 CONCLUSÃO	19

LISTA DE FIGURAS

1	Título dos desafios	1
2	Identificando padrão hexadecimal	2
3	Decodificando para ASCII	3
4	Contagem de linhas	3
5	Adicionando atributo como coluna	4
6	Agrupamento e resposta	4
7	Agrupamento IP e resposta	5
8	Endpoints da importação e análise de pacotes	6
9	Visualizador de Eventos do Windows	7
10	Analise do pacote com conteúdo suspeito	7
11	Base64 para String	8
12	Substituição e Aplicação da Cifra de César	9
13	Importação e descriptografia dos pacotes usando Aircrack	10
14	Mensagem suspeita	11
15	Base64 para String	11
16	Crackeando um PDF com HashCat	12
17	Senhas da memória	13
18	Programa comentado	14
19	Output do programa criado	15
20	Criando Script para obter os dadas do payload	16
21	Base 64 to String	16
22	MD5 da flag	17
23	Pesquisa básica	18

1 INTRODUÇÃO

O Global Solutions é um método de avaliação da FIAP que traz desafios elaborados para o aprendizado dos alunos, utilizando os de CTFs (Capture The Flag). Nesse formato, os alunos, através de autonomia e estudo, precisam encontrar uma ”flag”(bandeira), ou seja, descobrir algo escondido utilizando as técnicas aprendidas durante o curso até o momento.

Nos últimos três dias, foram disponibilizados 15 desafios de diversos tipos, abordando os principais temas de análise de pacotes, sistemas operacionais Windows e Linux, arquivos de dump de memória, programação reversa e criptografia como é possível ver superficialmente nos títulos na plataforma onde foram feitos:

Figura 1: Título dos desafios



Fonte: WARGAMES - 1TDCOB

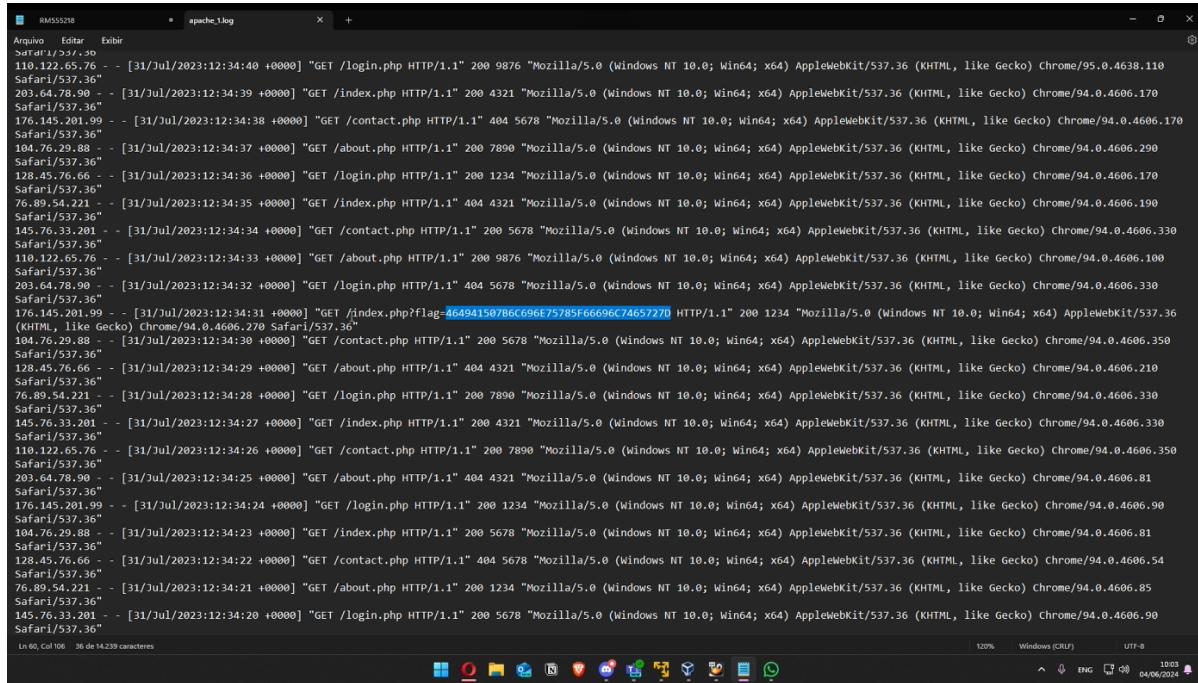
O presente documento tem como finalidade detalhar de maneira técnica os desafios encontrados, os processos empregados e os aprendizados adquiridos ao enfrentar cada desafio proposto. Ao longo do relatório, será realizada uma descrição técnica das estratégias utilizadas para capturar a Flag, fornecendo evidências relacionadas ao número de matrícula do aluno. Serão exploradas as dificuldades enfrentadas em cada etapa, destacando as soluções desenvolvidas e os conhecimentos adquiridos ao longo do processo de resolução dos desafios propostos.

2 DESENVOLVIMENTO DOS CTFs

2.1 URL SUSPEITA

O primeiro passo é sempre analisar o enunciado, contendo uma descrição deixando claro a presença de uma “URL SUSPEITA”, então já sabemos que devemos procurar por links em meio ao arquivo “apache_01.logs”, que também destaca ser um log de um servidor web. Ao abrir o documento, percebemos a presença de logs no mesmo formato, sempre seguindo a estrutura de IP no qual fez a solicitação, data, user agent e sistema operacional:

Figura 2: Identificando padrão hexadecimal

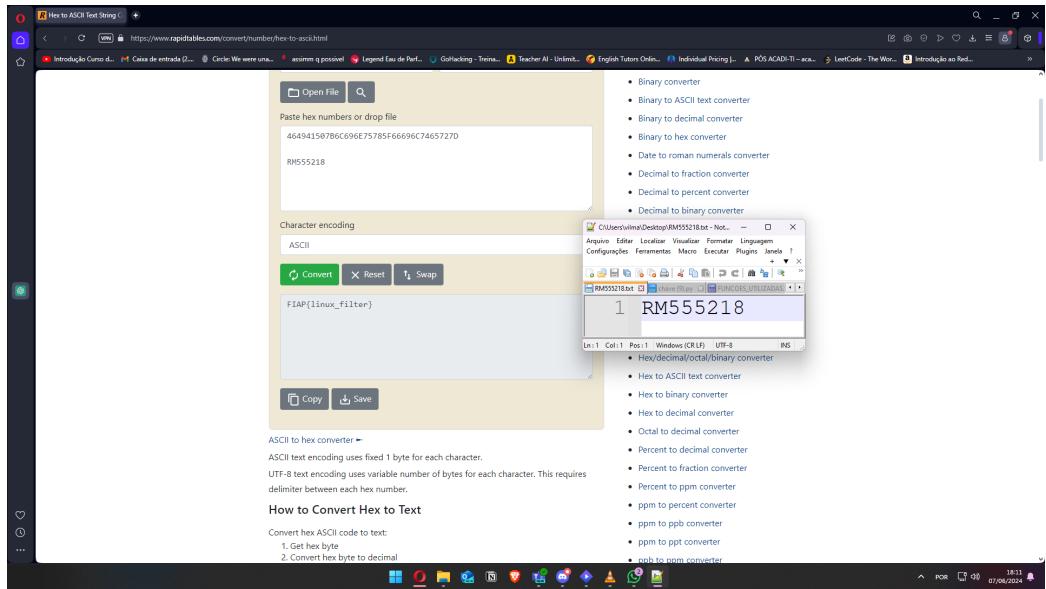


```
RMS55218 apache_01.log
Arquivo Editar Exibir
Safari/537.36" - [31/Jul/2023:12:34:40 +0000] "GET /login.php HTTP/1.1" 200 9876 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.110
Safari/537.36" 293.64.78.90 - [31/Jul/2023:12:34:39 +0000] "GET /index.php HTTP/1.1" 200 4321 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.170
Safari/537.36" 176.145.201.99 - [31/Jul/2023:12:34:38 +0000] "GET /contact.php HTTP/1.1" 404 5678 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.170
Safari/537.36" 184.76.29.88 - [31/Jul/2023:12:34:37 +0000] "GET /about.php HTTP/1.1" 200 7890 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.170
Safari/537.36" 128.45.76.66 - [31/Jul/2023:12:34:36 +0000] "GET /login.php HTTP/1.1" 200 1234 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.170
Safari/537.36" 76.89.54.221 - [31/Jul/2023:12:34:35 +0000] "GET /index.php HTTP/1.1" 404 4321 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.190
Safari/537.36" 184.76.29.88 - [31/Jul/2023:12:34:34 +0000] "GET /contact.php HTTP/1.1" 200 5678 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.190
Safari/537.36" 110.122.65.76 - [31/Jul/2023:12:34:33 +0000] "GET /about.php HTTP/1.1" 200 9876 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.100
Safari/537.36" 293.64.78.90 - [31/Jul/2023:12:34:32 +0000] "GET /login.php HTTP/1.1" 404 5678 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.330
Safari/537.36" 176.145.201.99 - [31/Jul/2023:12:34:31 +0000] "GET /index.php?flag=46494150786C696E75785F66696C74657270 HTTP/1.1" 200 1234 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.270
Safari/537.36" 184.76.29.88 - [31/Jul/2023:12:34:30 +0000] "GET /contact.php HTTP/1.1" 200 5678 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.350
Safari/537.36" 128.45.76.66 - [31/Jul/2023:12:34:29 +0000] "GET /about.php HTTP/1.1" 404 4321 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.210
Safari/537.36" 76.89.54.221 - [31/Jul/2023:12:34:28 +0000] "GET /login.php HTTP/1.1" 200 7890 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.330
Safari/537.36" 145.76.33.201 - [31/Jul/2023:12:34:27 +0000] "GET /index.php HTTP/1.1" 200 4321 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.330
Safari/537.36" 110.122.65.76 - [31/Jul/2023:12:34:26 +0000] "GET /contact.php HTTP/1.1" 200 7890 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.350
Safari/537.36" 293.64.78.90 - [31/Jul/2023:12:34:25 +0000] "GET /about.php HTTP/1.1" 404 4321 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.81
Safari/537.36" 176.145.201.99 - [31/Jul/2023:12:34:24 +0000] "GET /login.php HTTP/1.1" 200 1234 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.90
Safari/537.36" 184.76.29.88 - [31/Jul/2023:12:34:23 +0000] "GET /index.php HTTP/1.1" 200 5678 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.81
Safari/537.36" 128.45.76.66 - [31/Jul/2023:12:34:22 +0000] "GET /contact.php HTTP/1.1" 404 5678 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.54
Safari/537.36" 76.89.54.221 - [31/Jul/2023:12:34:21 +0000] "GET /about.php HTTP/1.1" 200 1234 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.85
Safari/537.36" 145.76.33.201 - [31/Jul/2023:12:34:20 +0000] "GET /login.php HTTP/1.1" 200 5678 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.90
Safari/537.36"
```

Fonte: WARGAMES - 1TDCOB - RM555218

Navegando para baixo, uma URL se destaca, contendo um parâmetro passado, típico da linguagem PHP. Imediatamente, identifiquei que se tratava de um código hexadecimal, já que consistia apenas de caracteres até o F. Utilizando um site de conversão de hexadecimal para texto (Rapid Tables), conseguindo obter a primeira Flag.

Figura 3: Decodificando para ASCII



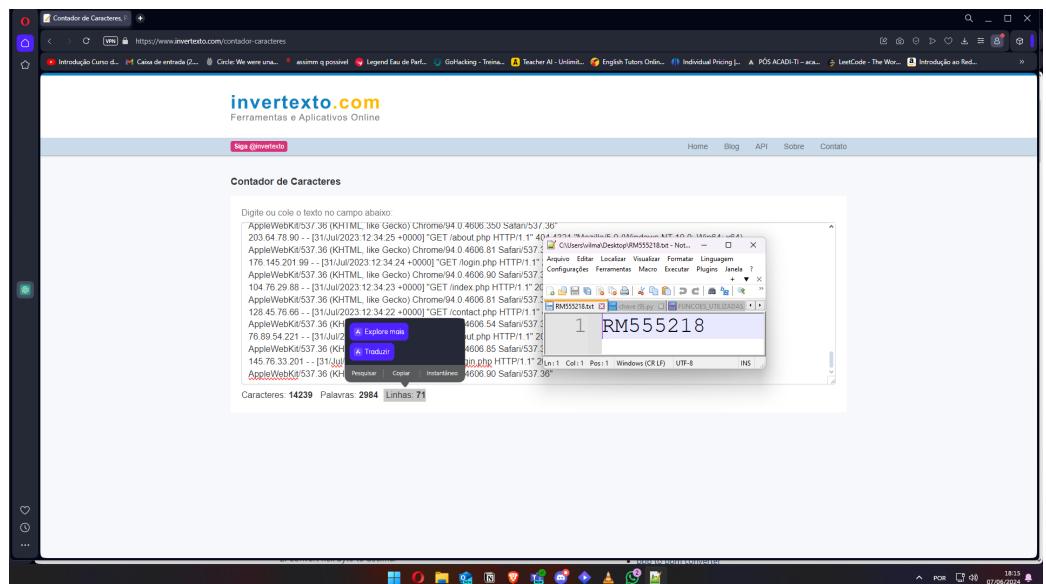
Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{linux_filter}

2.2 SOMANDO

Esse desafio se baseia na contagem de requisições 404 e 200. Ao perceber que todas essas requisições são feitas em linha, joguei os logs em um contador de linhas simples:

Figura 4: Contagem de linhas



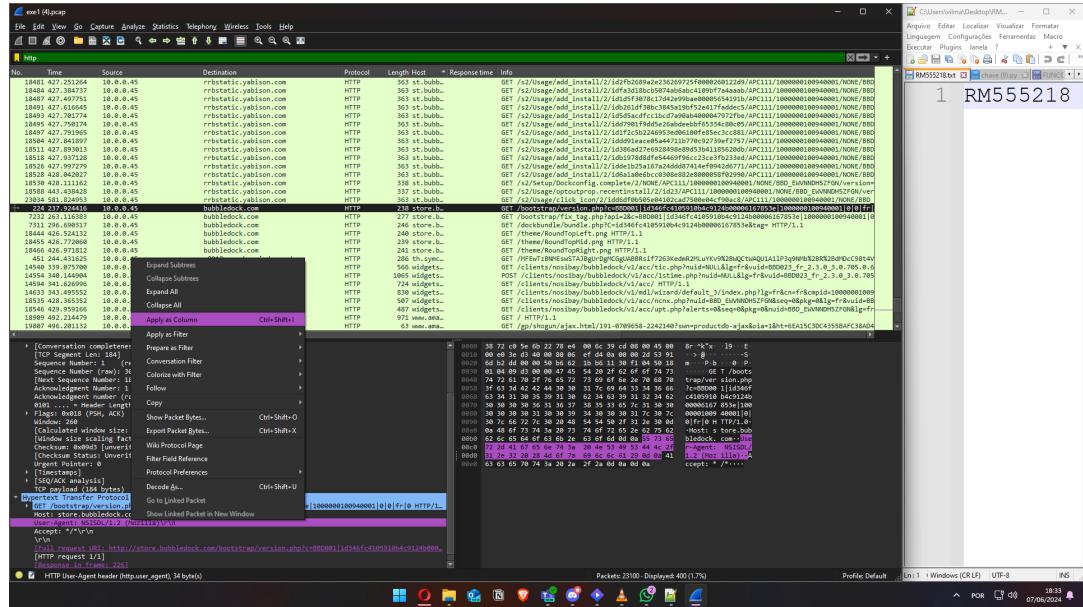
Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{71}

2.3 TOP USER-AGENT

Para resolver esse exercício, primeiramente abri o arquivo no Wireshark e filtrei os pacotes pelo tipo HTTP. Em seguida, adicionei o atributo "User-Agent" como uma coluna para cada pacote filtrado:

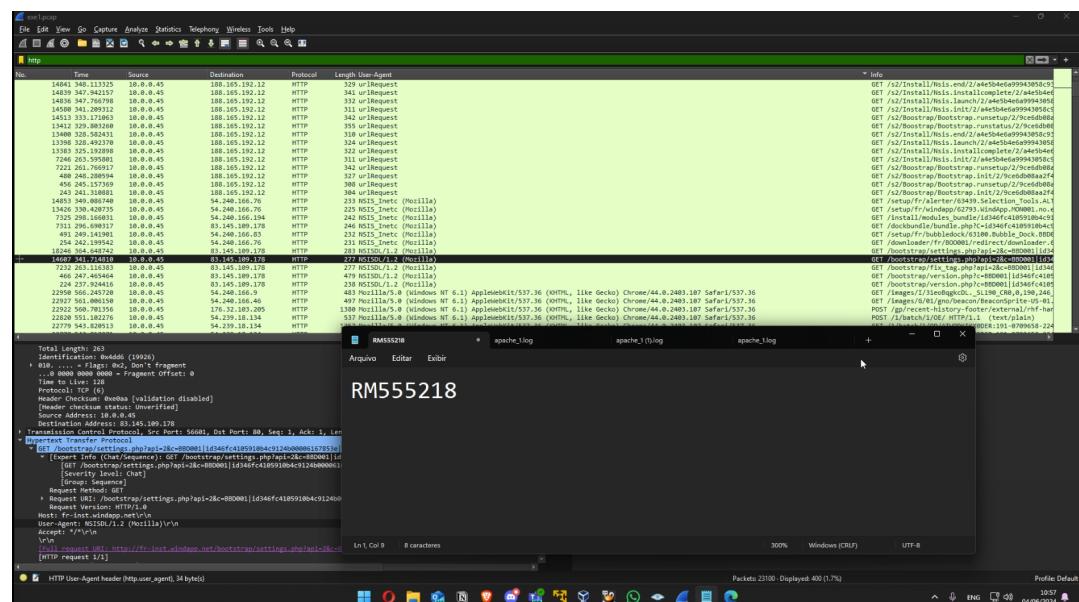
Figura 5: Adicionando atributo como coluna



Fonte: WARGAMES - 1TDCOB - RM555218

Após, foi possível ordenar e agrupar os pacotes com o mesmo User-Agent, permitindo identificar visualmente o User-Agent na posição 6 em maior número de requisições:

Figura 6: Agrupamento e resposta



Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{NSISDL/1.2 (Mozilla)}

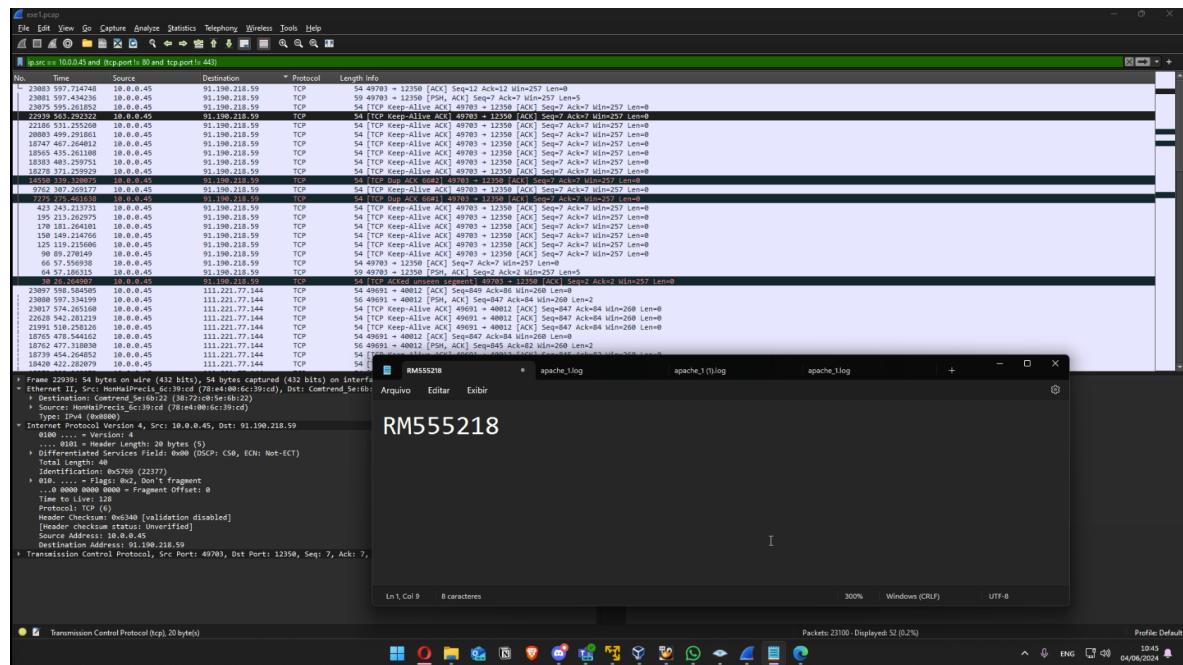
2.4 TOP IP

Um desafio clássico de filtros que deixa claro a necessidade de filtrar pela origem sendo o IP 10.0.0.43, e também, a necessidade de não ser HTTP nas portas padrão, 443 e 80. Dessa forma, criamos o seguinte filtro para o Wireshark:

```
1 ip.src == 10.0.0.43 and tcp.port != 80 and tcp.port != 443
```

Sobrando apenas duas alternativas de IPs após a aplicação do filtro, considerando ser o sexto, a Flag será o valor com menor quantidade de pacotes no agrupamento:

Figura 7: Agrupamento IP e resposta



Fonte: WARGAMES - 1TDCOB - RM555218

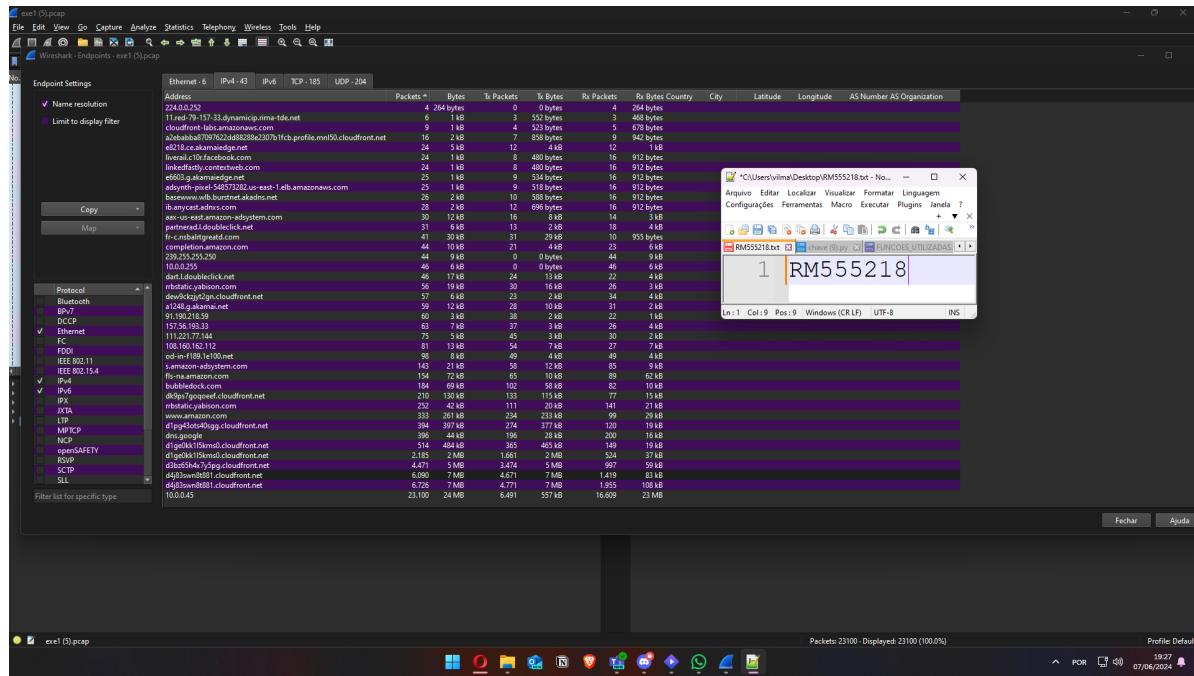
FIAP{91.190.218.59}

2.5 O SEXTO HOST

Com uma grande complexidade de enunciado, pois o antigo não mencionava o IP desejado, apenas o ”sexto host”, foi possível resolver o mesmo. No entanto, era necessário um referencial para identificar o sexto host. A solução foi importar o arquivo e, após

muita análise, utilizar a aba "Endpoints" do Wireshark para encontrar endereços com 6000 pacotes e nomear o host utilizando a opção "Name Resolution" na coluna da esquerda:

Figura 8: Endpoints da importação e análise de pacotes



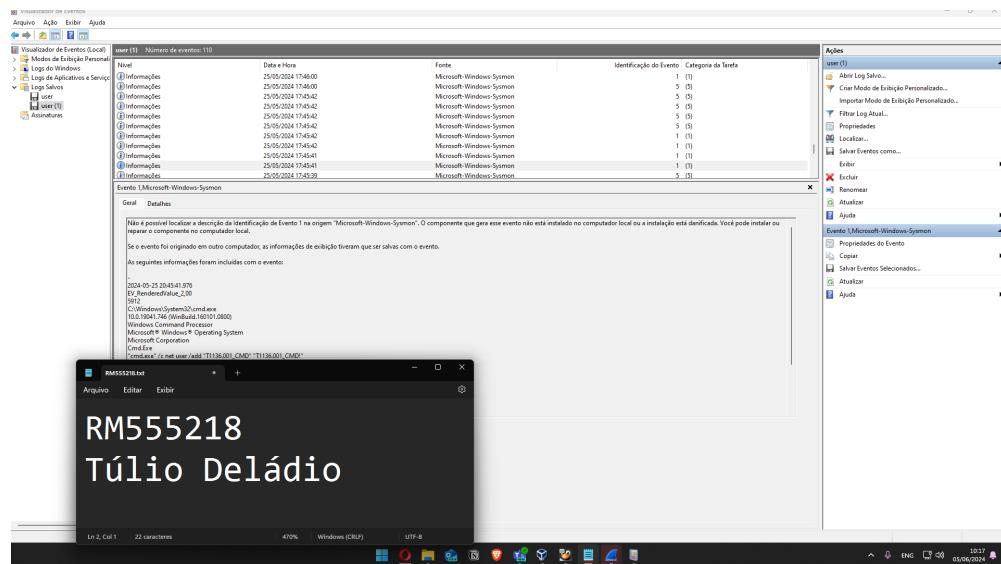
Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{d4j83swn8t881.cloudfront.net}

2.6 USUÁRIO DESTRUIDOR

Primeiramente, abrimos o arquivo .evtx disponibilizado utilizando o Visualizador de Eventos do Windows, automaticamente definido como o aplicativo padrão para tal formato. Descendo nos logs e prestando atenção nos comandos executados para adicionar usuários, como o “net user /add”, manualmente encontrei o usuário malicioso adicionado. Poderia ter feito de maneira mais rápida utilizando algum filtro simples, mas a análise manual foi suficiente para identificar o usuário suspeito:

Figura 9: Visualizador de Eventos do Windows



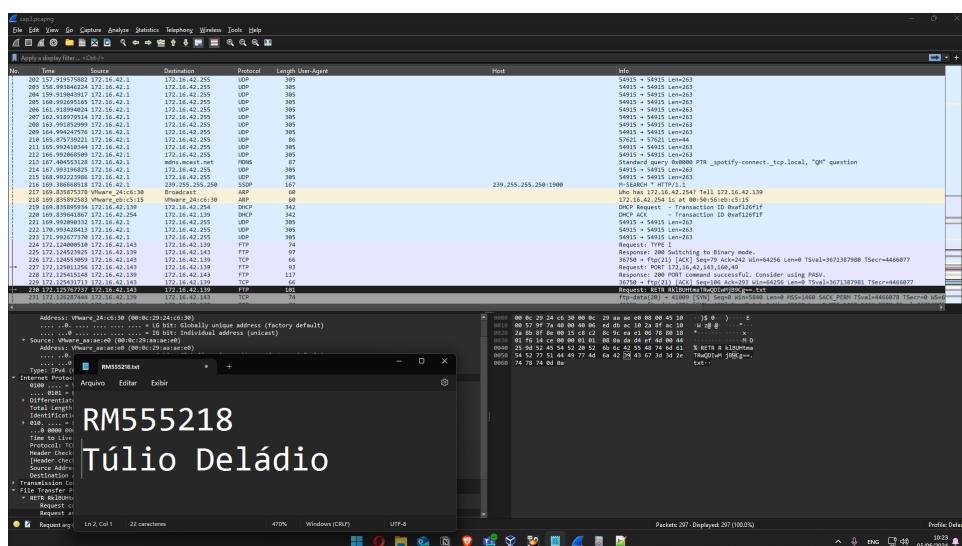
Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{T1136.001_CMD}

2.7 VAZAMENTO

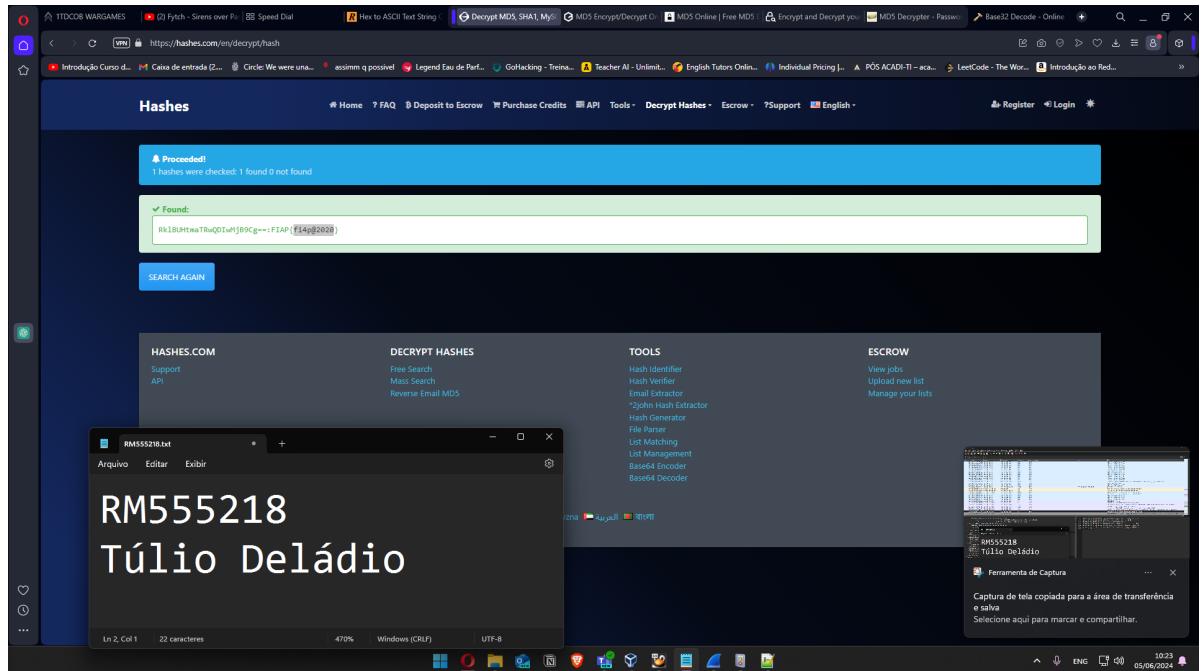
Este desafio também explorou técnicas avançadas de análise de pacotes. Durante a análise manual, ao examinar cuidadosamente o conteúdo Raw (ou payload) dos pacotes, identifiquei a presença de strings que pareciam estar codificadas em base64, indicadas pela ocorrência de dois iguais no final. Reconhecendo esse padrão, rapidamente utilizei um decodificador base64 para recuperar o conteúdo original e obtive a Flag:

Figura 10: Analise do pacote com conteúdo suspeito



Fonte: WARGAMES - 1TDCOB - RM555218

Figura 11: Base64 para String



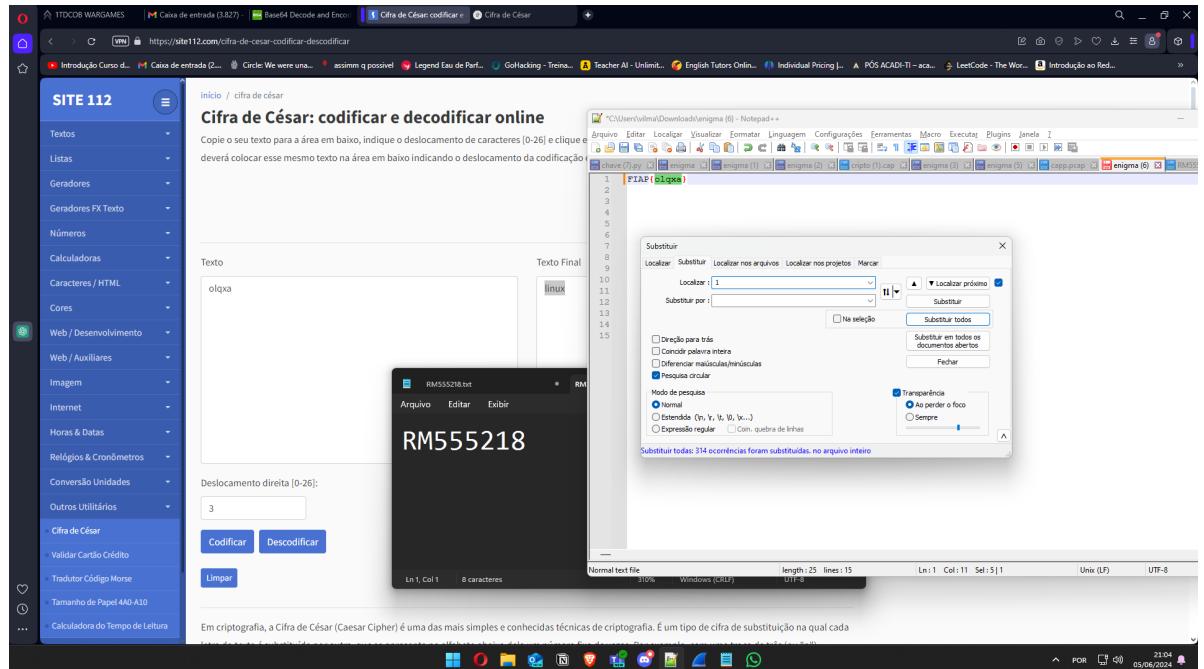
Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{fi4p2020}

2.8 CRIPTOGRAFIA ALIENÍGENA

Navegando rapidamente pelo arquivo usando o Notepad++, observei a presença de caracteres diferentes do padrão no meio da sequência binária, considerando a visualização inicial do arquivo seriam apenas 0s e 1s. Após tentativas de descriptografia via binário e Morse, sem sucesso, substitui todos os 0s e 1s por nada para obter as diferenças, resultando em uma sequência de caracteres que sobraram, formando uma flag. Ao tentar usar essa sequência como uma flag, não obtive êxito. Então, considerei a possibilidade de que fossem anagramas. Após várias tentativas com um script em Python para permutar os caracteres, sem sucesso, tentei diversas cifras até chegar à cifra de César, com uma rotação à direita de 3 caracteres. Essa abordagem finalmente mostrou a flag correta.

Figura 12: Substituição e Aplicação da Cifra de César



Fonte: WARGAMES - 1TDCOB - RM555218

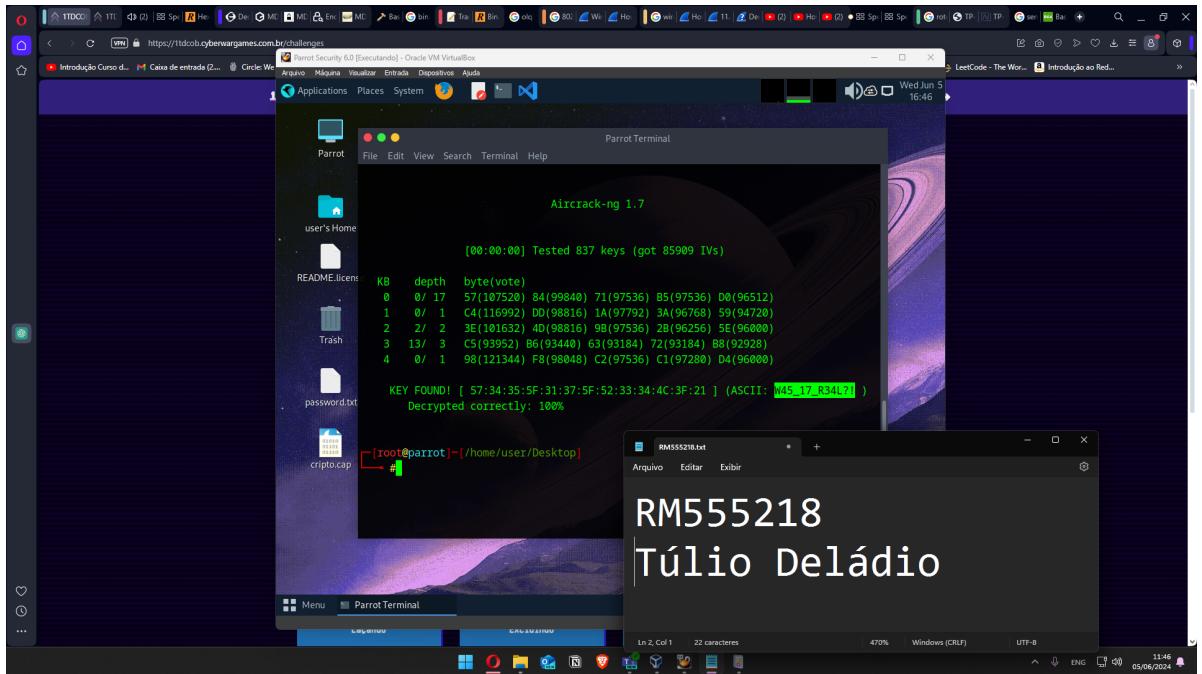
FIAP{linux}

2.9 OBSOLETO

Lendo o enunciado, rapidamente associei a situação a um protocolo antigo de redes Wi-Fi, que poderia permitir a descriptografia de pacotes. Ao abrir o arquivo com o Wireshark, tentei usar endereços MAC e sequências encontradas na análise manual como possíveis senhas, pois o Wireshark possui uma funcionalidade dedicada à descriptografia de pacotes de Wi-Fi com protocolos antigos. No entanto, sem sucesso.

Então, lembrei-me de uma ferramenta que já havia utilizado no passado para aprendizado sobre redes, o Aircrack-ng. Esta ferramenta é capaz de realizar ataques de força bruta ou ataques de dicionário para quebrar chaves de criptografia em redes Wi-Fi. O processo básico para usar o Aircrack-ng envolveu importar o arquivo capturado e selecionar o ponto de acesso com o SSID suspeito. Após isso, a ferramenta inicia o processo de cracking, tentando várias combinações de senhas até encontrar a correta, conseguindo descriptografar a senha em menos de 1 minuto de cracking, o que me permitiu obter a flag com sucesso e alcançando uma medalha.

Figura 13: Importação e descriptografia dos pacotes usando Aircrack



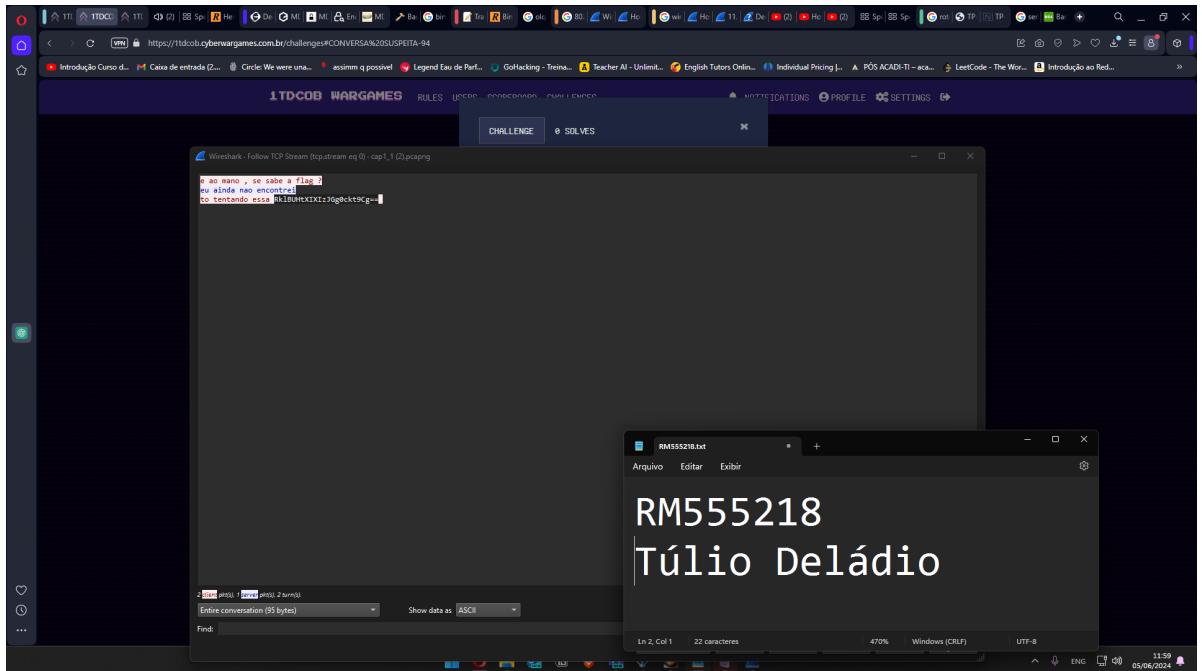
Fonte: WARGAMES - 1TDCOB - RM555218

FIAPI{W45_17_R34L?!"}

2.10 CONVERSA SUSPEITA

Ao abrir o pacote no Wireshark e identificar uma conversa suspeita com o exposto no enunciado, em que se esperava a troca de dados, foi necessário buscar por pacotes que transportavam mensagens. Isso geralmente é feito analisando os campos relevantes dos pacotes, como o payload em pacotes TCP, onde a mensagem estaria contida. Após uma análise manual dos pacotes e formatos, foram identificados pacotes TCP que continham alguma mensagem. Para examinar o conteúdo desses pacotes, utilizou-se a funcionalidade "Follow TCP Stream" do Wireshark, que organiza as mensagens em ordem cronológica seguindo uma conexão TCP.

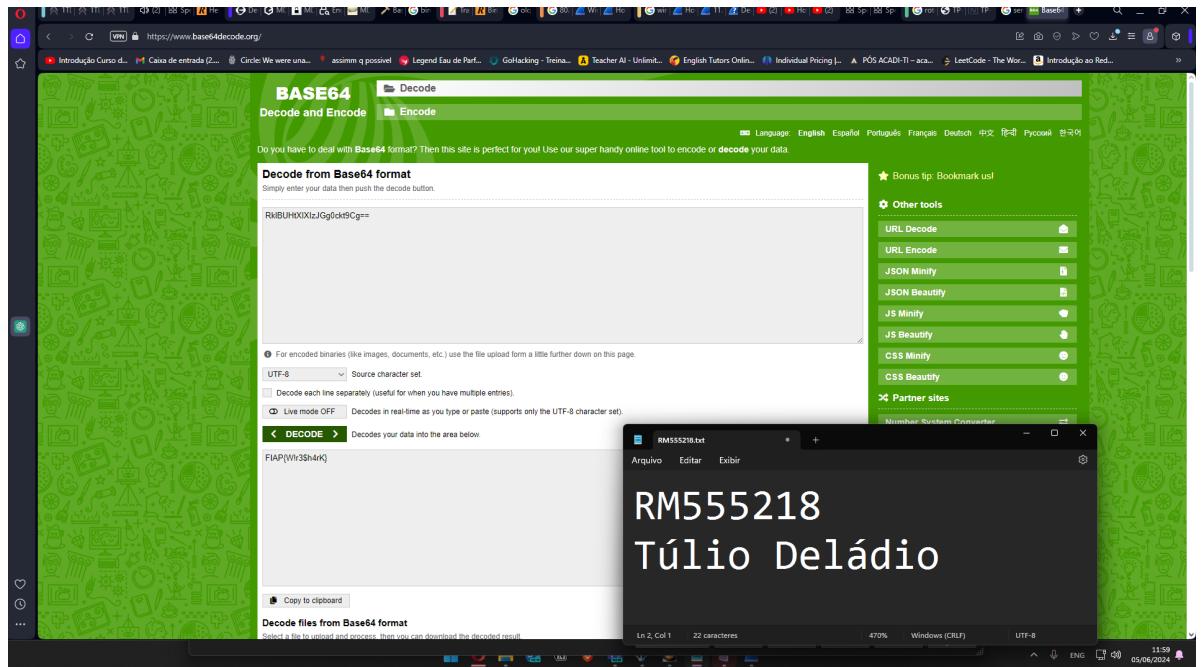
Figura 14: Mensagem suspeita



Fonte: WARGAMES - 1TDCOB - RM555218

Ao seguir o fluxo da conversa, foi possível encontrar novamente uma sequência de caracteres suspeita, que parecia estar codificada em base64. Essa sequência foi então decodificada, resultando na obtenção da flag:

Figura 15: Base64 para String



Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{W!r3\$h4rK}

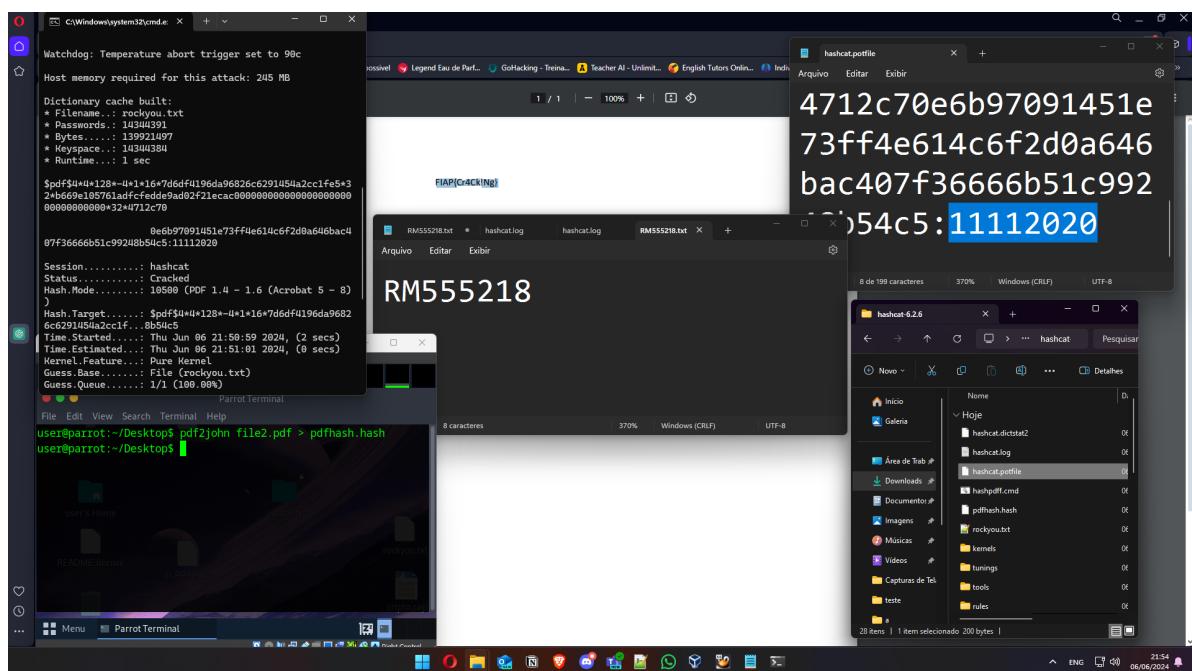
2.11 SOMENTE NÚMEROS

Esse exercício foi uma oportunidade de aprendizado valiosa para desenvolver métodos eficientes de brute force. Em busca de aprimorar minha compreensão, optei por utilizar ferramentas altamente performáticas para quebrar o hash do PDF disponibilizado. Primeiramente, utilizei o ParrotOS com a ferramenta pdf2john para extrair o hash do PDF. Em seguida, desenvolvi um arquivo .cmd para iniciar o Hashcat no Windows, com os parâmetros adequados para a tarefa em questão.

```
1 hashcat.exe -m 10500 -a0 pdfhash.hash rockyou.txt --force  
2 pause
```

Esse comando utiliza o Hashcat, para tentar encontrar a senha de um hash de PDF. Ele especifica o modo de ataque, o tipo de ataque (neste caso, um ataque de dicionário), o arquivo que contém o hash do PDF, o arquivo de lista de palavras a ser usado, neste caso a lista de palavras (rockyou.txt), para encontrar a senha correspondente, utilizando uma técnica de força bruta. Ao fim, como é possível ver na esquerda superior, o programa conseguiu crackear, assim utilizando a senha para abrir o pdf e obter as senhas:

Figura 16: Crakeando um PDF com HashCat



Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{Cr4Ck!Ng}

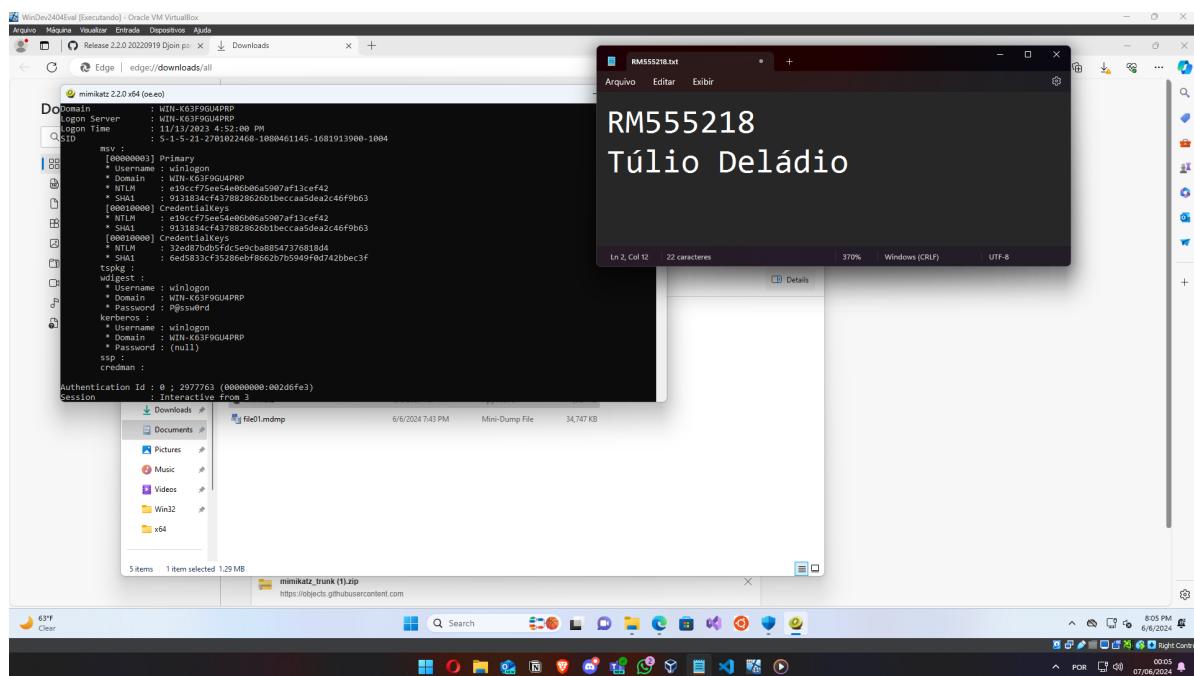
2.12 BLUE OF DEATH

Nesse desafio, foi fornecido um arquivo de dump de memória, possivelmente originado de um crash ou corrupção de memória, contendo hashes de senhas que precisavam ser extraídos para recuperar o acesso ao servidor. Inicialmente, tentei usar um editor hexadecimal para buscar as senhas, mas sem sucesso após horas de tentativa. Em seguida, recorri ao WinDbg, uma ferramenta de debug da Microsoft, mas também não obtive sucesso. Finalmente, descobri a ferramenta Mimikatz, que já havia ouvido falar, mas não sabia que poderia ajudar nesse desafio. Dentro do meu Windows virtualizado via VMWare, executei o Mimikatz e carreguei o arquivo de dump, da seguinte forma

```
1 mimikatz.exe #Inicia
2 sekurlsa::minidump dump.dmp #Carrega arquivo de dump
3 sekurlsa::logonPasswords #Expõe os dados encontrados
```

Ao acabar de processar, obteve êxito ao extrair diversas senhas. Buscando pelo usuário do Windows, encontrei o "winlogon" e a senha que era a flag desejada.

Figura 17: Senhas da memória

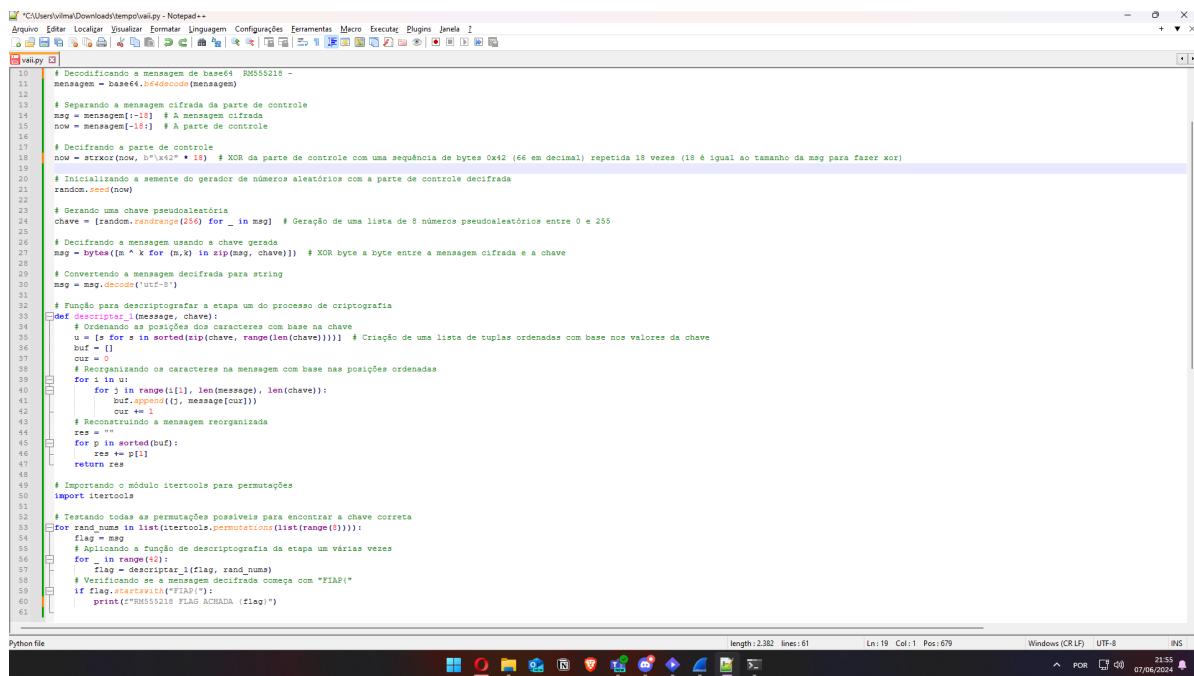


Fonte: WARGAMES - 1TDCOB - RM555218

2.13 TEMPO E ESPAÇO

Nesse desafio, a vulnerabilidade identificada na criptografia está relacionada à geração da chave pseudo aleatória a partir de uma parte da mensagem cifrada, conhecida como "parte de controle". Essa parte é usada como semente para inicializar o gerador de números aleatórios do Python através da função random.seed(). O problema principal é que a semente não é verdadeiramente aleatória, pois é derivada da própria mensagem cifrada. Isso pode permitir que um atacante, ao identificar padrões na mensagem cifrada ou prever a parte de controle, influencie ou deduza a chave gerada. Com essa falha no código analisado, foi criado o seguinte script em python para abusar da mesma, explicada de maneira comentada em cada linha:

Figura 18: Programa comentado



```

1# *C:\Users\vinicius\Downloads\tempo\valipy - Notepad++
2
3# Decodificando a mensagem de base64 RM555218 -
4# mensagen = base64.b64decode(mensagen)
5
6# Separando a mensagem cifrada da parte de controle
7# msg = mensagem[:-18] # A mensagem cifrada
8# now = mensagem[-18:] # A parte de controle
9
10# Decifrando a parte de controle
11now = atrxor(now, b'\x42' * 18) # XOR da parte de controle com uma sequência de bytes 0x42 (66 em decimal) repetida 18 vezes (18 é igual ao tamanho da msg para fazer xor)
12
13# Inicializando a semente do gerador de números aleatórios com a parte de controle decifrada
14random.seed(now)
15
16# Gerando uma chave pseudoaleatória
17chave = [random.randrange(256) for _ in msg] # Geração de uma lista de 8 números pseudoaleatórios entre 0 e 255
18
19# Decifrando a mensagem usando a chave gerada
20msg = bytex([m ^ k for (m,k) in zip(msg, chave)]) # XOR byte a byte entre a mensagem cifrada e a chave
21
22# Convertendo a mensagem decifrada para string
23msg = msg.decode('utf-8')
24
25# Função para descriptografar a etapa um do processo de criptografia
26def descripter_1(mensagem, chave):
27    # Ordenando as posições das caracteres com base na chave
28    # [(0, 1), (2, 3) ...] in sorted(zip(chave, range(len(chave)))) # Criação de uma lista de tuplas ordenadas com base nos valores da chave
29    buf = []
30    cur = 0
31    # Reorganizando os caracteres na mensagem com base nas posições ordenadas
32    for i in range(len(chave)):
33        for j in range(i+1, len(mensagem), len(chave)):
34            buf.append((j, mensagem[cur]))
35            cur += 1
36    # Reconstruindo a mensagem reorganizada
37    res = ''
38    for p in sorted(buf):
39        res += p[1]
40    return res
41
42# Importando o módulo itertools para permutações
43import itertools
44
45# Testando todas as permutações possíveis para encontrar a chave correta
46for rand_nums in list(itertools.permutations(list(range(9)))):
47    flag = msg
48    # Aplicando a função de descriptografia da etapa um várias vezes
49    for i in range(10):
50        flag = descripter_1(flag, rand_nums)
51    # Verificando se a mensagem decifrada começa com "FIAP"
52    if flag.startswith("FIAP"):
53        print("PERMUTAÇÃO ENCONTRADA: ", flag)
54
55# *C:\Users\vinicius\Downloads\tempo\valipy - Notepad++
56# Length: 2.382, lines: 61   Ln: 19 Col: 1 Pos: 679   Windows (CR LF)   UTF-8   INS
57# ^ POR 21:55 07/06/2024

```

Fonte: WARGAMES - 1TDCOB - RM555218

Figura 19: Output do programa criado

The screenshot shows a Windows desktop environment. In the foreground, there is a Notepad++ window titled "vaii.py" containing Python code. The code is a script for cracking a ciphered message using a known-plaintext attack. It includes functions for decoding messages, generating random lists, and performing byte-wise XOR operations. It also imports the `itertools` module for permutations and defines a function to decrypt messages using a given key. In the background, a PowerShell window titled "Windows PowerShell" is open with the command `PS C:\Users\viima\Downloads\tempo> python.exe .\vaii.py` running. The output of the script is displayed in the PowerShell window, showing the recovered flag: `FLAG ACHADA{Tim_15_pr3C10u5_s0_Enj0y!}`.

```
# Descodificando as mensagens de base64 -  
mensagem = base64.b64decode(mensagem)  
  
# Separando a mensagem cifrada da parte de controle  
msg = mensagem[:16] # A mensagem cifrada  
now = mensagem[16:] # A parte de controle  
  
# Decifrando a parte de controle  
now = strxor(now, b"\x02\x02\x02\x02") # XOR da parte de controle com uma sequência de zeros  
  
# Inicializando a semente do gerador de números aleatórios com a parte de controle  
random.seed(now)  
  
# Gerando uma chave pseudocaleatoria  
chave = [random.randrange(256) for _ in msg] # Geração de uma lista de 8 bytes  
  
# Decifrando a mensagem usando a chave gerada  
msg = bytes([m ^ k for (m,k) in zip(msg, chave)]) # XOR byte a byte entre a mensagem e a chave  
  
# Convertendo a mensagem decifrada para string  
msg = msg.decode("utf-8")  
  
# Função para descriptografar a etapa um do processo de criptografia  
  
def descriptar_1(message, chave):  
    # Recuperando os caracteres da mensagem com base nas posições ordenadas  
    buf = [0] * len(message)  
    for s in sorted(zip(chave, range(len(chave)))):
        buf[s] = message[s]
    buf = [s for s in sorted(buf)]
    cur = 0  
  
    # Recalculando os caracteres na mensagem com base nas posições ordenadas  
    for i in range(len(message)):
        for j in range(i+1, len(message), len(chave)):
            if buf[j] <= buf[i]:
                buf.append(message[j])
                cur += 1
    res = ""  
    # Reconstruindo a mensagem reorganizada  
    for p in sorted(buf):
        res += p[1]
    return res  
  
# Importando o módulo itertools para permutações  
import itertools  
  
# Testando todas as permutações possíveis para encontrar a chave correta  
for rand_nums in list(itertools.permutations(list(range(256)))):  
    flag = msg  
    flag = descriptar_1(flag, rand_nums)  
    # Verifica se a mensagem decifrada começa com "FIAP"  
    if flag.startswith("FIAP"):  
        print("RODRIGOZIS FLAG ACHADA: " + flag)
```

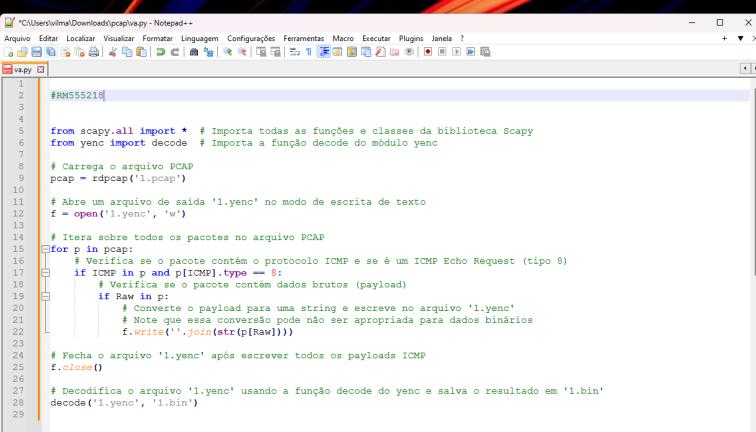
Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{T1m3_15_pr3C10u5_s0_Enj0y}

2.14 CABO PERDIDO

Esse CTF é baseado em um hexdump obtido por meio do primeiro pacote disponibilizado para análise (Considerado por ser maior e suspeito devido a sequencia na análise pelo Wireshark). Nele, é possível fazer um script em Python usando a biblioteca Scapy para conseguir os binários e decodificá-los na biblioteca YAML. Assim, usando um decodificador básico e tirando as linhas de quebra, conseguimos a seguinte flag. Dessa maneira, seguindo o que a flag indica, é necessário fazer o MD5 do texto da flag para obter a flag real:

Figura 20: Criando Script para obter os dados do payload



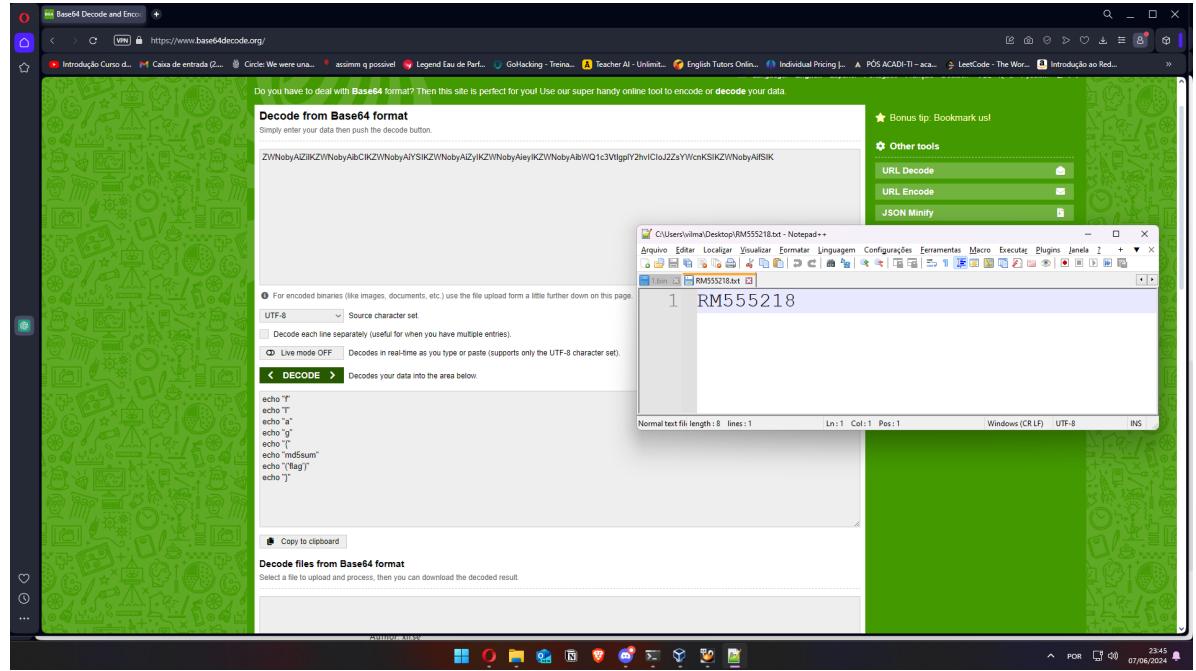
```
*C:\Users\vlma\Downloads\pcapiv.py - Notepad++
Arquivo Editar Localizar Visualizar Formatar Linguagem Configurações Ferramentas Macro Executar Plugins Janela ?
File Edit View Insert Format Language Options Tools Help Plugins Windows
vappy.py

1 #!/usr/bin/python
2
3
4 from scapy.all import * # Importa todas as funções e classes da biblioteca Scapy
5 from yenc import decode # Importa a função decode do módulo yenc
6
7
8 # Carrega o arquivo PCAP
9 pcap = rdpcap('1.pcap')
10
11 # Abre um arquivo de saída '1.yenc' no modo de escrita de texto
12 f = open('1.yenc', 'w')
13
14 # Itera sobre todos os pacotes no arquivo PCAP
15 for p in pcap:
16     # Verifica se o pacote contém o protocolo ICMP e se é um ICMP Echo Request (tipo 8)
17     if ICMP in p and p[ICMP].type == 8:
18         # Verifica se o pacote contém dados brutos (payload)
19         if Raw in p:
20             # Remove o payload
21             # Converte o payload para uma string e escreve no arquivo '1.yenc'
22             # Note que essa conversão pode não ser apropriada para dados binários
23             f.write(''.join(str(p[Raw])))
24
25     # Fecha o arquivo '1.yenc' após escrever todos os payloads ICMP
26 f.close()
27
28 # Decodifica o arquivo '1.yenc' usando a função decode do yenc e salva o resultado em '1.bin'
29 decode('1.yenc', '1.bin')

Python file length:1.015 lines:29 lin:2 Col:10 Pos:12 Windows (CR LF) UTF-8 INS
```

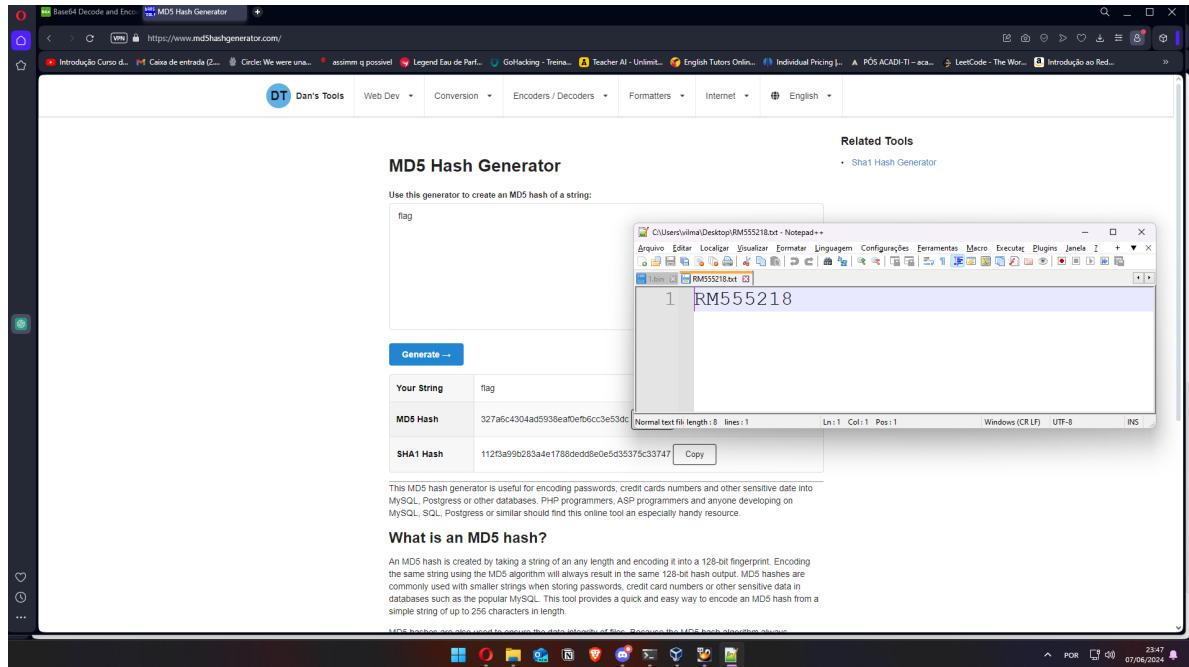
Fonte: WARGAMES - 1TDCOB - RM555218

Figura 21: Base 64 to String



Fonte: WARGAMES - 1TDCOB - RM555218

Figura 22: MD5 da flag



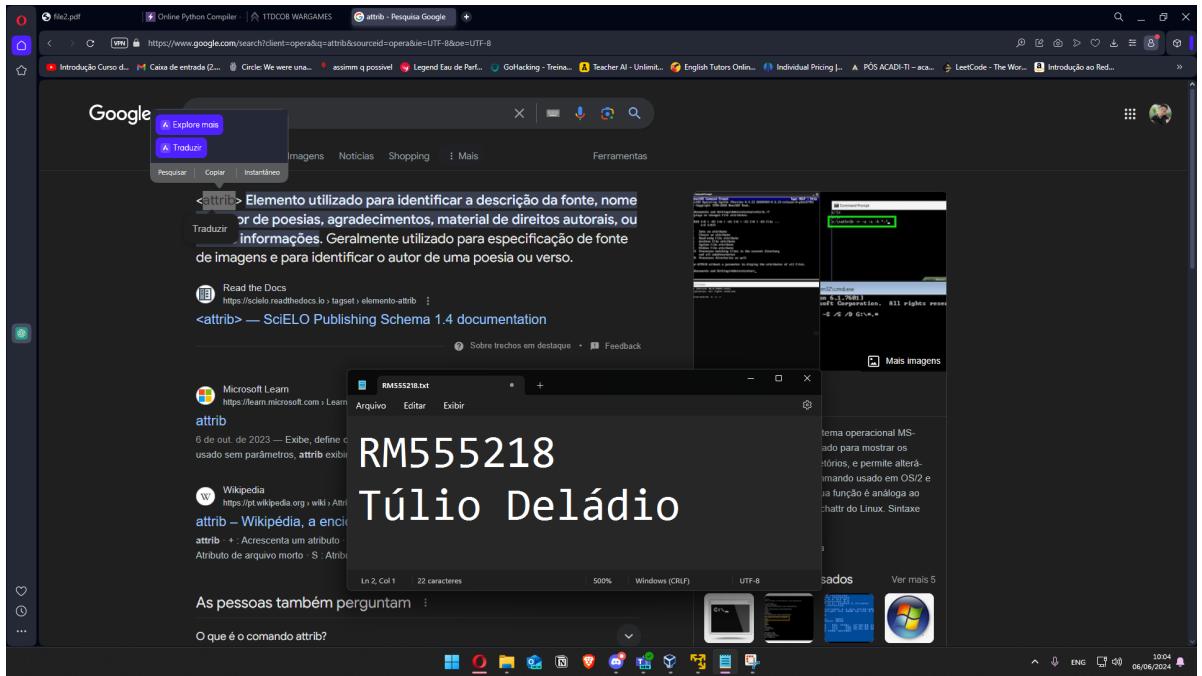
Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{327a6c4304ad5938eaf0efb6cc3e53dc}

2.15 COMANDO BÁSICO

O comando attrib é uma ferramenta versátil para visualizar e alterar os atributos de arquivos e diretórios no Windows. Ele permite definir atributos como leitura, sistema e oculto, proporcionando aos usuários controle sobre a visibilidade dos arquivos. Com minha experiência prévia em outros CTFs, identifiquei rapidamente o output desejado, obtendo a flag e alcançando uma medalha

Figura 23: Pesquisa básica



Fonte: WARGAMES - 1TDCOB - RM555218

FIAP{attrib}

3 CONCLUSÃO

No decorrer dos desafios do CTF, pude explorar uma variedade de técnicas e ferramentas para resolver problemas complexos de segurança da informação. Desde a análise de logs de servidores web até a extração de senhas de arquivos de dump de memória, cada desafio apresentou novos obstáculos e oportunidades de aprendizado.

A análise de pacotes no Wireshark foi essencial para identificar padrões e anomalias, como no desafio do User-Agent mais utilizado. A necessidade de filtrar e analisar tráfego de rede me levou a explorar técnicas de filtragem avançadas, como a busca por IPs específicos em portas não padrão. Além disso, o uso de ferramentas como o Mimikatz revelou-se crucial para extrair senhas de arquivos de dump de memória, demonstrando a importância de estar familiarizado com uma variedade de ferramentas de segurança.

Desafios como a decodificação de mensagens criptografadas e a quebra de hashes também ampliaram meu conhecimento em criptografia e técnicas de ataque. A resolução desses desafios exigiu criatividade, incentivando-me a explorar diferentes abordagens até encontrar a solução correta, além de referencias com outros grandes CTFs já realizados pelo mundo.