Conspiração Cibernética

Você, codinome “CH1M3R4”, especialista em segurança cibernética e programação, conhecido na comunidade pela habilidade em resolver complexos desafios tecnológicos. Em “Conspiração Cibernética“, você é convocado para investigar uma série de incidentes misteriosos que ameaçam a estabilidade mundial.

A história se desenrola em um futuro próximo, onde a tecnologia é onipresente e os sistemas automatizados são responsáveis por manter o funcionamento de importantes infraestruturas. No entanto, uma organização secreta chamada “Serpent Council” está sabotando esses sistemas, comprometendo a segurança e a privacidade de milhões de pessoas.

Você é acompanhado por uma equipe de especialistas, juntos vocês formam o Tech Sentinels:

- Dra. Amelia Lawson – uma cientista renomada em segurança de dados, especializada em criptografia. Sua inteligência e conhecimento técnico são inestimáveis.
- Agent Jake Reynolds – um ex-militar com experiência em operações táticas. Sua força e habilidades de combate são cruciais durante as missões perigosas.
- Chloe Ramirez – uma hacker habilidosa e engenhosa. Sua perícia em invasões de sistemas e conhecimento de redes clandestinas são indispensáveis na luta contra a Serpent Council.
- Emily Sullivan – uma analista de inteligência com um talento para investigação. Sua capacidade de conexão de pontos e habilidades de análise são vitais para descobrir os planos ocultos da organização.

Conforme você avança na trama, você e sua equipe se deparam com uma série de desafios de segurança cibernética que exigem suas habilidades em programação Python. Os programas em Python mencionados fazem parte do enredo, proporcionando interações diretas com a história.

Ao longo da história, você enfrentará reviravoltas emocionantes, confrontos perigosos e momentos de tensão enquanto tenta desmantelar a conspiração da Serpent Council. A trama mantém você engajado, enquanto aprende sobre *cyber security* e programação Python, aplicando seus conhecimentos para superar os desafios e resolver os enigmas.

Prepare-se para embarcar nessa emocionante aventura, onde sua capacidade de programação e segurança cibernética pode salvar o mundo da ameaça da Serpent Council e garantir a paz digital para todos. A contagem regressiva começou. O destino da humanidade está em suas mãos.

1º ATO - Codinome: A César o que é de César.

Durante a missão de combate ao **Serpent Council**, você e sua equipe de agentes secretos precisam garantir que suas comunicações sejam seguras e livres de interceptações. Para isso, decidem utilizar a **Cifra de César**, um método de criptografia simples e eficaz.

A **Cifra de César** é um sistema de criptografia por substituição, onde cada letra da mensagem original é substituída por outra letra que se encontra um número fixo de posições à frente no alfabeto. Esse número de posições é conhecido como "chave" e pode variar conforme o nível de segurança necessário.
Por exemplo, se escolhermos uma chave de 3, a letra "A" seria substituída pela letra "D", "B" pela letra "E", e assim por diante. Dessa forma, a palavra **"SEGURANCA"** seria criptografada como **"VHJXUDQFD"**.
Esse método recebe o nome de **Cifra de César** porque foi utilizado pelo imperador romano Júlio César para garantir a segurança de suas comunicações militares.

Na sua missão, cada agente possui uma chave de criptografia única. Isso significa que, para se comunicarem entre si, eles devem utilizar a mesma chave tanto para criptografar quanto para descriptografar as mensagens. Essa medida impede que interceptadores consigam entender facilmente o conteúdo das mensagens interceptadas.

A equipe está infiltrada em uma instalação altamente protegida da **Serpent Council**, com o objetivo de obter informações cruciais sobre o próximo plano de sabotagem. Enquanto você e Chloe investigam os sistemas de segurança, você decide cifrar uma mensagem para enviar ao restante da equipe.

Aqui está o código Python que você usa para garantir a segurança da comunicação:

Criptografando com a Cifra de César:

```
# Função para criptografar a mensagem usando a Cifra de César
def crypt_cesar(message, shift):
    encrypted_message = ""
    for char in message:
        if char.isalpha():
            # Calcula o deslocamento da letra baseado na chave (shift)
            shifted_char = chr((ord(char.upper()) - ord('A') + shift) % 26 + ord('A'))
            # Mantém o caso original (minúscula ou maiúscula)
            if char.islower():
                shifted_char = shifted_char.lower()
            encrypted_message += shifted_char
        else:
            encrypted_message += char
    return encrypted_message

# Exemplo de uso
mensagem = "Equipe, encontrei o local onde estão escondendo os arquivos confidenciais.
Preciso de ajuda para desativar as defesas."
chave = 3

mensagem_criptografada = crypt_cesar(mensagem, chave)
print("Mensagem criptografada: " + mensagem_criptografada)
```

Ao rodar esse programa, a seguinte mensagem será gerada:

Mensagem criptografada:

```
Htxlsh, hqfrqwuhl r orfdo rqgh hvwdr hvfrqghqgr rv duitxlyrv frqilghqfldlv. suhflvr gh dmduda sdud
ghdadwlydu dv ghihvdv.
```

Com a mensagem pronta, você a envia para os membros da equipe.

Descriptografando a Mensagem Recebida:

Pouco tempo depois, você recebe uma resposta do Agente Jak

```
uhfhelgr. ydprv qrv lqilowudu h ghvdelolwdu dv ghihvdv. pdqwhqkd-qrv lqirupdgrv vreuh r surjuhvvr.
```

Para decifrar a mensagem, escreva o código Python, aplicando a **Cifra de César** inversa com a mesma chave:

```

```

Ao rodar o código, a mensagem será:

Mensagem decifrada:

```

```

Agora que a mensagem foi decifrada, você tem todas as informações necessárias para prosseguir com a missão. Sua equipe está coordenada e pronta para agir, graças ao uso estratégico da criptografia para manter as comunicações seguras.

A Cifra de César pode parecer simples, mas foi uma ferramenta eficaz para proteger informações sensíveis durante séculos, e ainda hoje demonstra sua utilidade em situações onde a simplicidade e a segurança são essenciais.

Nesta missão, este método de cifragem garantiu que suas comunicações não fossem interceptadas ou compreendidas pelo inimigo, protegendo informações críticas e permitindo que sua equipe agisse com confiança.

Com a equipe em sintonia e a comunicação segura, vocês estão prontos para enfrentar o próximo desafio e vencer a **Serpent Council**.

2º ATO - Codinome: BASE64

Depois de semanas de silêncio, o alarme ressoou no laboratório subterrâneo. Dra. Amelia Lawson olhou para a tela, onde uma nova mensagem pulsava em vermelho. 'Interceptação confirmada', o sistema notificava. O tempo estava se esgotando—ela sabia que os Tech Sentinels tinham pouco tempo para decodificar aquela sequência enigmática antes que a Serpent Council colocasse em ação o próximo movimento devastador.

Amelia observava a sequência interminável de caracteres na tela. Mesmo sendo uma especialista em criptografia, algo na mensagem a inquietava. As marcas de codificação BASE64 eram óbvias, mas havia uma perturbação—como se a mensagem tivesse sido manipulada, camuflando segredos mais profundos. Cada caractere parecia um enigma, uma provocação direta da Serpent Council.

```
Vm0wd2QyUXlVWGxXYTJoV1YwZG9WbGx0ZUV0V01WbDNXa1JTV0ZKdGVGWlZNbmhQVmpBeFYySkVUbGhoTVVwVVZtc
EdZV1JIVmtsaVJtUk9ZV3hhZVZadGVGWmxSbGw1Vkd0V1VtSkdXbGhaYkZWM1pVWmFjVkZ0UmxSTmJFcEpWbTEwYzJGc1
NuVlJiR2hYWVd0R00xcFZXbXRXTVZwMFVteFNUbUY2UlRCV01uUnZWakpHVjFOdVVtaFNlbXhXVm1wT2IxTXhjRmhsUjNS
WVVqRktTVlZ0ZUZOVWJVWTJVbFJDVjFaRmEzaFZha1poWkVaT2NtRkdXbWxTTW1oWFZtMTBWMlF5VWxkaUtaHNVakJhY1
ZsclpEQk9iR3hXVjJzNVZXSkZjRWhXTVdoclZqRmFSbUl6WkZwV1JWcHlWVEJhVDJOdFNrZFRiV3hUVFRKb1dWWnJXbGRaV
m14WFZXdGtXR0V5VWxsWmJGWmhZMVpzY2xwRVFrOWlSM2hYVmpKek5WWlhTbFpYVkVwV1lrWktSRlpxUm1GU2JVVjZ
ZVVphYUdFeGNHOVhhMVpoVkRKT2MyTkZaR2hTTW5odlZGVm9RMWRzV25KWGJHUmFWbTE0V0ZaWGRHdGhiRXB6WTB
ac1dtSkdXbWhXYTFwVFZqRmtkVnBIZUdsU2JYY3hWa1phVTFVeFduSk5XRXBxVWxoQ1YxWnFUbTlsYkZweFVWaG9hMVpz
V2pGV01uaHJWakZLV1ZGcmJGZFdNMEpJVmtSS1RtVkdaSFZVYlVaVFRXNW9WVlpHWTNoaU1XUkhWMjVTVGxOSGFGQlZ
ha1pIVGtaa2NsWnRkRmRpVlhCNlZUSTFUMVp0U2xWV2ExSmFaV3RhYUZreFdrdGtSa3B6Vld4T2FWTkZTa3RXTW5oWFZtc
zFXRkpyWkZoaWF6VnhWVEJvUTFsV1VsWlhibVJyWWtad2VGVnRkREJoYXpGeVRsVndWMDF1YUhKV2FrWkxWakpPU0dGR
2FHbFNiSEJ2VjFaU1MxUXlUWGhhU0ZaVllrWmFjRlpxU205VUscFlaRWRHV2xZeFNucFdNalZUVkd4a1NGVnNWbFZXYkhC
TVdsWmFVMk14WkhSa1JtUk9ZVEZaTVZac1pEUmhNV1IwVWxob1dHRnJOVmhWYTFaaFpXeHJlV1ZlUm1waVZrcElWbGQ0
VDFSc1dsZGhNMnhYVFZkTmVGcEhNVkpsVmxwMVUyczFWMUpVVmxOV2JYUlRVakZLUjFkc2JGcGxiWGhTVm10a2JtUXlU
a2hWVkRGQ1dsRTlQVFJOVDg=u8
```

Chloe Ramirez, debruçada sobre o teclado ao lado de Amelia, soltou um suspiro de frustração. 'Esses caras jogam pesado', murmurou, os dedos voando sobre o código enquanto tentava encontrar uma pista. Jake Reynolds permanecia em pé atrás delas, observando com a frieza de um soldado, mas o suor em sua testa traía sua preocupação. 'Estamos correndo contra o tempo', alertou ele, a voz grave ecoando pelo ambiente.

Missão do Desafio: Decodificar a Mensagem

Instruções: Você, como parte da equipe **Tech Sentinels**, deve examinar a mensagem enviada à Dra. Amelia. Há traços de que a mensagem foi codificada utilizando **BASE64**, possivelmente com caracteres aleatórios inseridos para dificultar a decodificação direta.

Seu objetivo é: Decodificar a mensagem em **BASE64** e identificar a informação oculta.

A mensagem pode ter sido submetida a múltiplas codificações, com adição de caracteres em cada etapa. Prepare-se para múltiplos níveis de decodificação.

Você terá que implementar um script em Python que decodifica a mensagem, removendo os caracteres aleatórios adicionados e revertendo a sequência para a mensagem original.

Exemplo de código de decodificação:

O código abaixo fornece uma referência de como começar a decodificar uma mensagem, mas você terá que descobrir com os desafios inseridos. Nosso objetivo é tornar o desafio mais interessante e permitir que você tenha uma direção clara na busca da solução.

```
import base64

def decode_base64_with_hint(encoded_string):
```

```
    """
    Função de referência para decodificar uma mensagem codificada em BASE64.
    Serve como base para o processo de decodificação.
    """
    try:
        # Adicionando padding se necessário
        encoded_string = encoded_string.rstrip("=").ljust((len(encoded_string) + 3) // 4 * 4, "=")

        # Decodificando BASE64
        decoded_bytes = base64.b64decode(encoded_string)
        decoded_string = decoded_bytes.decode('utf-8')
        return decoded_string
    except Exception as e:
        return f"Erro ao decodificar: {e}"

# Exemplo de mensagem codificada em BASE64
mensagem_codificada =
'Vm0wd2QyUXlVWGxXYTJoV1YwZG9WbGx0ZUV0V01WbDNXa1JTV0ZKdGVGWlZNbmhQVmpBeFYySkVUbGhoTVVwVVZtcEdZV1JIV
mtsaVJtUk9ZV3hhZVZadGVGWmxSbGw1Vkd0V1VtSkdXbGhaYkZWM1pVWmFjVkZ0UmxSTmJFcEpWbTEwYzJGc1NuVlJiR2hYWVd
0R00xcFZXbXRXTVZwMFVteFNUbUY2UlRCV01uUnZWakpHVjFOdVVtaFNlbXhXVm1wT2IxTXhjRmhsUjNSWVVqRktTVlZ0ZUZOV
WJVWTJVbFJDVjFaRmEzaFZha1poWkVaT2NtRkdXbWxTTW1oWFZtMTBWMlF5VWxkalJtaHNVakJhY1ZsclpEQk9iR3hXVjJzNVZ
XSkZjRWhXTVdoclZqRmFSbUl6WkZwV1JWcHlWVEJhVDJOdFNrZFRiV3hUVFRKb1dWWnJXbGRaVm14WFZXdGtXR0V5VWxsWmJGW
mhZMVpzY2xwRVFrOWlSM2hYVmpKek5WWlhTbFpYVkVwV1lrWktSRlpxUm1GU2JVVjZZVVphYUdFeGNHOVhhMVpoVkRKT2MyTkZ
aR2hTTW5odlZGVm9RMWRzV25KWGJHUmFWbTE0V0ZaWGRHdGhiRXB6WTBac1dtSkdXbWhXYTFwVFZqRmtkVnBIZUdsU2JYY3hWa
1phVTFVeFduSk5XRXBxVWxoQ1YxWnFUbTlsYkZweFVWaG9hMVpzV2pGV01uaHJWakZLV1ZGcmJGZFdNMEpJVmtSS1RtVkdaSFZ
VYlVaVFRXNW9WVlpHWTNoaU1XUkhWMjVTVGxOSGFGQlZha1pIVGtaa2NsWnRkRmRpVlhCNlZUSTFUMVp0U2xWV2ExSmFaV3RhY
UZreFdrdGtSa3B6Vld4T2FWTkZTa3RXTW5oWFZtczFXRkpyWkZoaWF6VnhWVEJvUTFsV1VsWlhibVJyWWtad2VGVnRkREJoYXp
GeVRsVndWMDF1YUhKV2FrWkxWakpPU0dGR2FHbFNiSEJ2VjFaU1MxUXlUWGhhU0ZaVllrWmFjRlpxU205VlJscFlaRWRHV2xZe
FNucFdNalZUVkd4a1NGVnNWbFZXYkhCTVdsWmFVMk14WkhSa1JtUk9ZVEZaTVZac1pEUmhNV1IwVWxob1dHRnJOVmhWYTFaaFp
XeHJlV1ZIUm1waVZrcElWbGQ0VDFSc1dsZGhNMnhYVFZkTmVGcEhNVkpsVmxwMVUyczFWMUpVVmxOV2JYUlRVakZLUjFkc2JGc
GxiWGhTVm10a2JtUXlUa2hWVkRGQ1dsRTlQVFJOVDg=u8'

# Referência de decodificação
mensagem_decodificada = decode_base64_with_hint(mensagem_codificada)
print(f"Mensagem decodificada (referência): {mensagem_decodificada}")
```

Dra. Amelia enviou o desafio para a equipe, como membro do time, agora cabe a você decodificar a mensagem criptografada. Desvende o que está oculto por trás da codificação e ajude os **Tech Sentinels** a obter informações valiosas para deter os planos do **Serpent Council**.

Boa sorte!

Mensagem:

3º ATO – Codinome: DUALIDADE

A tensão no quartel-general dos Tech Sentinels era palpável. A Serpent Council havia lançado sua jogada final — uma nova mensagem interceptada trazia indícios claros de que o plano de criação de um governo paralelo estava perto da execução. O futuro da ordem global, tanto no mundo físico quanto digital, pendia por um fio.

Após a última mensagem ser decifrada, o Serpent Council ficou em alerta. Eles introduziram uma nova barreira, alternando entre BASE64 e BASE32, acrescentando camadas de caos ao codificar suas mensagens. Sabiam que a Tech Sentinels estaria observando. E agora, cada falha poderia significar um passo mais próximo da destruição.

Uma mensagem crucial interceptada revela que o **Serpent Council** está nos estágios finais de criar um **governo paralelo**. A equipe **Tech Sentinels**, liderada pela Dra. Amelia Lawson, sabe que esta informação é de vital importância e que precisam decifrá-la o quanto antes para confirmar as informações.

```
S01ZVVVUQ1dJVkNURVVaUkpaRUZDMlNLSTVJVk1WU1VLSlZUQ1UyU0tWTkVZVEpRSlpHRkdNSzJKUkxFS1RKUUtFWUhR
VENPS1ZIRktVSlJMSktGT1JMVUs1SldXVlNLS1pWRTRUS1ROTllGRVZUTEtaSEZDTUsyS0ZKR1c2Q0dLTVlEQ1RDTk5OWlRD
VlJRS1pIRktNQ09MRkpUQ1NTTEtKV0VNVlNYTlJERVNVVExMSkxWSVZTS0pWSlZLM0NMS1JER0lXQ1dLWkhGTVUzMktaR1
ZHUktHSzVHV1lUU0lLRldHSVNDUkdGTkVZVlNWTVJCRTRSUzJJSkxHV1NTVEtJWVZVU0NYSVU0VkNVM0tLWkdGSTNDS0t
KSUZJTUJaS0JNRkU1S1ZNTT09PT09PWtKLJ
```

Missão do Desafio: Você deve decodificar a mensagem enviada pelo **Serpent Council**, que foi codificada e identificar o conteúdo oculto. A cada etapa da codificação, foram adicionados elementos tornando o processo de decodificação mais desafiador.

Você precisará implementar um script que reveja as múltiplas iterações de codificação para descobrir o que está oculto.

> Esse terceiro ato introduz um desafio ainda mais complexo, com a alternância de métodos de codificação e a adição de caracteres aleatórios, aumentando a dificuldade e o nível de estratégia necessário para desvendar a mensagem oculta.

Após a descoberta dos métodos da **Tech Sentinels**, o **Serpent Council** cria um novo sistema de criptografia alternada. Agora, cabe a você, como parte da equipe **Tech Sentinels**, decifrar a mensagem oculta e impedir que o conselho consiga criar um governo paralelo. O destino do mundo digital está em suas mãos.

Mensagem:

4º ATO – Codinome: Secret eye

Os Tech Sentinels estavam acostumados a lidar com criptografias de texto, mas essa nova jogada do Serpent Council os deixou perplexos. Na sala de operações, a tela brilhava com um longo texto aparentemente inofensivo, mas todos sabiam que por trás daquela fachada se escondia algo muito mais perigoso. O inimigo havia mudado de estratégia. O que antes era um jogo simples de decodificação de texto havia se transformado em um campo no qual até mesmo a equipe mais experiente poderia se perder.

'Que texto é esse', Chloe murmurou, seus olhos fixos no monitor. 'Isso é muito mais complicado do que parece. Podemos estar lidando com esteganografia, compressão codificada... e quem sabe o que mais.' Amelia sabia que o tempo estava correndo. 'Precisamos descobrir o que está escondido aqui, e rápido. Tudo depende de conseguirmos quebrar esse novo código antes que eles deem o próximo passo'.

Missão: A equipe de interceptação descobriu um repositório no github onde um arquivo compactado parece conter dados da organização criminosa, há indícios de que ela contém informações ocultas. O time de inteligência descobriu que o **Serpent Council** vem aplicando sequência de técnicas de compressão, codificação hexadecimal, e o uso das já conhecidas camadas de BASE64 para esconder a verdadeira informação.

A sua missão é descompactar, decodificar e revelar o que está oculto.

https://github.com/elderofz1on/Cyber-Conspiracy/blob/main/resources/sugar.zip

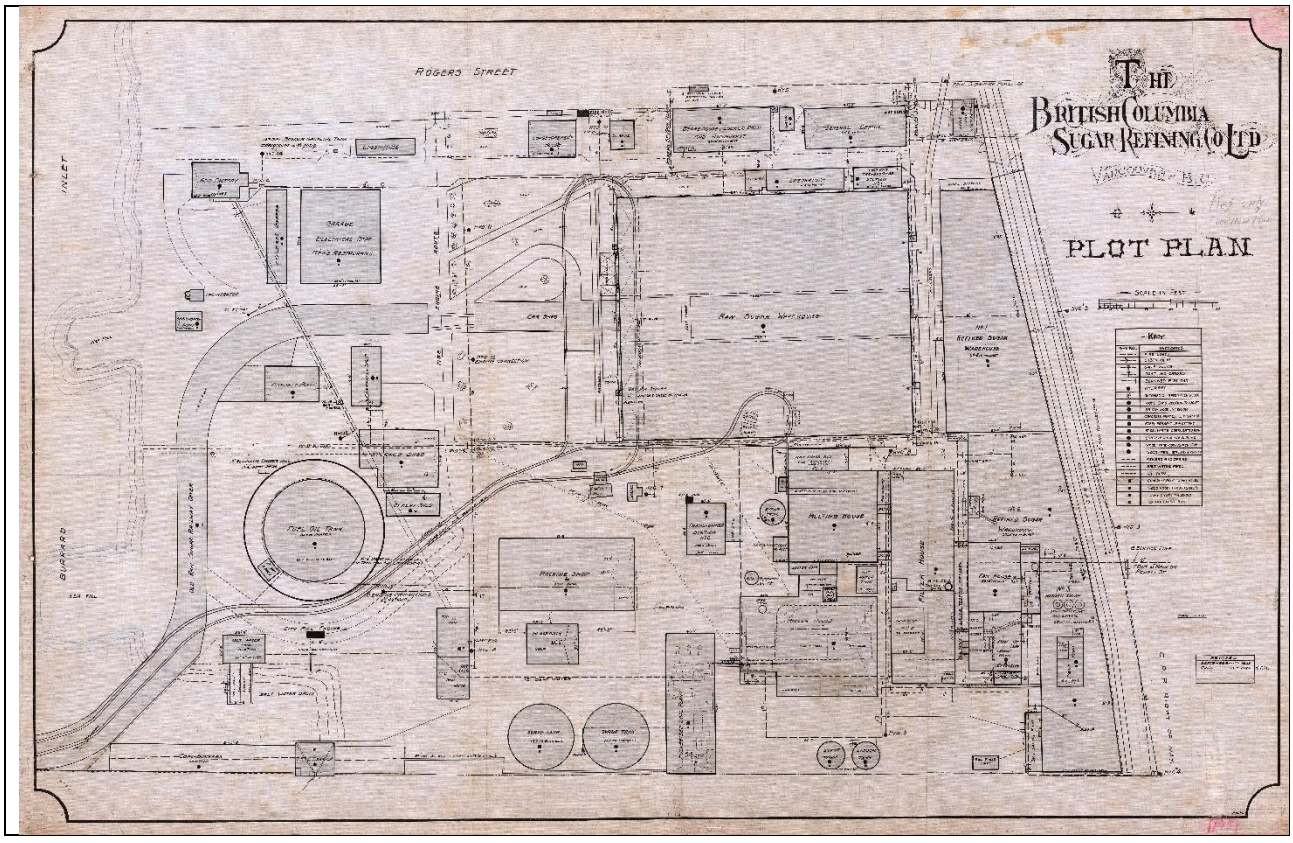
Após horas de tentativa, o texto revela seus segredos ocultos. Chloe conseguiu extrair uma sequência. 'Temos algo, mas... há mais por aqui. Eles estão jogando com a gente. O verdadeiro segredo está além dessa imagem.' Amelia sabia que estavam perto, mas também que o tempo estava acabando. 'Isso é só o começo', ela murmurou. O verdadeiro teste ainda estava por vir."

5º ATO – Esteganografia

O silêncio no laboratório dos Tech Sentinels era interrompido apenas pelo som das teclas digitadas rapidamente por Chloe. A imagem que tinham em mãos parecia inofensiva — apenas uma foto comum. Mas Chloe sabia que o Serpent Council havia dado o próximo passo. Agora, o jogo não era mais apenas sobre textos ocultos; eles estavam escondendo segredos dentro das próprias imagens. A equipe estava a um passo de descobrir a peça-chave para os planos deles. Mas será que conseguiríamos a tempo?

Chloe, após examinar a imagem interceptada (sugar.jpg), percebeu padrões incomuns dentro do arquivo. Sua análise preliminar sugere que a imagem foi manipulada usando esteganografia avançada, onde uma mensagem codificada em BASE64 está oculta na imagem.

Você, membro dos **Tech Sentinels**, é encarregado de extrair essa informação vital oculta dentro da imagem e decodificá-la para revelar o que o Serpent Council está planejando.



SOLUÇÕES

1º ATO - A César o que é de César.

Criptografando com a Cifra de César:

```
# Função para criptografar a mensagem usando a Cifra de César
def crypt_cesar(message, shift):
    encrypted_message = ""
    for char in message:
        if char.isalpha():
            # Calcula o deslocamento da letra baseado na chave (shift)
            shifted_char = chr((ord(char.upper()) - ord('A') + shift) % 26 + ord('A'))
            # Mantém o caso original (minúscula ou maiúscula)
            if char.islower():
                shifted_char = shifted_char.lower()
            encrypted_message += shifted_char
        else:
            encrypted_message += char
    return encrypted_message

# Exemplo de uso
mensagem = "Equipe, encontrei o local onde estão escondendo os arquivos confidenciais.
Preciso de ajuda para desativar as defesas."
chave = 3

mensagem_criptografada = crypt_cesar(mensagem, chave)
print("Mensagem criptografada: " + mensagem_criptografada)
```

Mensagem criptografada:

```
Htxlsh, hqfrqwuhl r orfdo rqgh hvwdr hvfrqghqgr rv duitxlyrv frqilghqfldlv. suhflvr gh dmduda sdud
ghdadwlydu dv ghihvdv.
```

Descriptografando a Mensagem Recebida:

```
uhfhelgr. ydprv qrv lqilowudu h ghvdelolwdu dv ghihvdv. pdqwhqkd-qrv lqirupdgrv vreuh r surjuhvvr.
```

```
# Função para descriptografar a mensagem usando a Cifra de César
def decrypt_cesar(message, shift):
    decrypted_message = ""
    for char in message:
        if char.isalpha():
            # Calcula o deslocamento inverso da letra baseado na chave (shift)
            shifted_char = chr((ord(char.upper()) - ord('A') - shift) % 26 + ord('A'))
            # Mantém o caso original (minúscula ou maiúscula)
            if char.islower():
                shifted_char = shifted_char.lower()
            decrypted_message += shifted_char
        else:
            decrypted_message += char
    return decrypted_message

# Exemplo de uso
mensagem_recebida = "uhfhelgr. ydprv qrv lqilowudu h ghvdelolwdu dv ghihvdv. pdqwhqkd-
qrv lqirupdgrv vreuh r surjuhvvr."
chave = 3

mensagem_decifrada = decrypt_cesar(mensagem_recebida, chave)
print("Mensagem decifrada: " + mensagem_decifrada)
```

Mensagem decifrada:

```
secretario. vamos nos infiltrar e desabilitar as defesas. mantenha-nos informados
sobre o progresso.
```

2º ATO - Codinome: BASE64

Mensagem recebida:

```
Vm0wd2QyUXlVWGxXYTJoV1YwZG9WbGx0ZUV0V01WbDNXa1JTV0ZKdGVGWlZNbmhQVmpBeFYySkVUbGhoTVVwVVZtc
EdZV1JIVmtsaVJtUk9ZV3hhZVZadGVGWmxSbGw1Vkd0V1VtSkdXbGhaYkZWM1pVWmFjVkZ0UmxSTmJFcEpWbTEwYzJGc1
NuVlJiR2hYWVd0R00xcFZXbXRXTVZwMFVteFNUbUY2UlRCV01uUnZWakpHVjFOdVVtaFNlbXhXVm1wT2IxTXhjRmhsUjNS
WVVqRktTVlZ0ZUZOVWJVWTJVbFJDVjFaRmEzaFZha1poWkVaT2NtRkdXbWxTTW1oWFZtMTBWMlF5VWxkalJtaHNVakJhY1
ZsclpEQk9iR3hXVjJzNVZXSkZjRWhXTVdoclZqRmFSbUl6WkZwV1JWcHlWVEJhVDJOdFNrZFRiV3hUVFRKb1dWWnJXbGRaV
m14WFZXdGtXR0V5VWxsWmJGWmhZMVpzY2xwRVFrOWlSM2hYVmpKek5WWlhTbFpYVkVwV1lrWktSRlpxUm1GU2JVVjZ
ZVVphYUdFeGNHOVhhMVpoVkRKT2MyTkZaR2hTTW5odlZGVm9RMWRzV25KWGJHUmFWbTE0V0ZaWGRHdGhiRXB6WTB
ac1dtSkdXbWhXYTFwVFZqRmtkVnBIZUdsU2JYY3hWa1phVTFVeFduSk5XRXBxVWxoQ1YxWnFUbTlsYkZweFVWaG9hMVpz
V2pGV01uaHJWakZLV1ZGcmJGZFdNMEpJVmtSS1RtVkdaSFZVYlVaVFRXNW9WVlpHWTNoaU1XUkhWMjVTVGxOSGFGQlZ
ha1pIVGtaa2NsWnRkRmRpVlhCNlZUSTFUMVp0U2xWV2ExSmFaV3RhYUZreFdrdGtSa3B6Vld4T2FWTkZTa3RXTW5oWFZtc
zFXRkpyWkZoaWF6VnhWVEJvUTFsV1VsWlhibVJyWWtad2VGVnRkREJoYXpGeVRsVndWMDF1YUhKV2FrWkxWakpPU0dGR
2FHbFNiSEJ2VjFaU1MxUXlUWGhhU0ZaVllrWmFjRlpxU205VlJscFlaRWRHV2xZeFNucFdNalZUVkd4a1NGVnNWbFZXYkhC
TVdsWmFVMk14WkhSa1JtUk9ZVEZaTVZac1pEUmhNV1IwVWxob1dHRnJOVmhWYTFaaFpXeHJlV1ZlUm1waVZrcElWbGQ0
VDFSc1dsZGhNMnhYVFZkTmVGcEhNVkpsVmxwMVUyczFWMUpVVmxOV2JYUlRVakZLUjFkc2JGcGxiWGhTVm10a2JtUXlU
a2hWVkRGQ1dsRTlQVFJOVDg=u8
```

Possível solução:

```
import base64

def decode_base64_with_fixed_chars(encoded_string, iterations=12):
    # Realiza o processo de reversão por 'iterations' vezes
    for i in range(iterations):
        # Remove os últimos 2 caracteres (adicionados aleatoriamente)
        encoded_string = encoded_string[:-2]
        # Decodifica a string base64
        encoded_string = base64.b64decode(encoded_string).decode()
        # Apresenta o valor da string a cada iteração
        print(f"Iteração {i+1}: {encoded_string}")
    return encoded_string

# Resultado final gerado no processo anterior
final_encoded_string =
'Vm0wd2QyUXlVWGxXYTJoV1YwZG9WbGx0ZUV0V01WbDNXa1JTV0ZKdGVGWlZNbmhQVmpBeFYySkVUbGhoTVVwVVZtcEdZV1JIV
mtsaVJtUk9ZV3hhZVZadGVGWmxSbGw1Vkd0V1VtSkdXbGhaYkZWM1pVWmFjVkZ0UmxSTmJFcEpWbTEwYzJGc1NuVlJiR2hYWVd
0R00xcFZXbXRXTVZwMFVteFNUbUY2UlRCV01uUnZWakpHVjFOdVVtaFNlbXhXVm1wT2IxTXhjRmhsUjNSWVVqRktTVlZ0ZUZOV
WJVWTJVbFJDVjFaRmEzaFZha1poWkVaT2NtRkdXbWxTTW1oWFZtMTBWMlF5VWxkalJtaHNVakJhY1ZsclpEQk9iR3hXVjJzNVZ
XSkZjRWhXTVdoclZqRmFSbUl6WkZwV1JWcHlWVEJhVDJOdFNrZFRiV3hUVFRKb1dWWnJXbGRaVm14WFZXdGtXR0V5VWxsWmJGW
mhZMVpzY2xwRVFrOWlSM2hYVmpKek5WWlhTbFpYVkVwV1lrWktSRlpxUm1GU2JVVjZZVVphYUdFeGNHOVhhMVpoVkRKT2MyTkZ
aR2hTTW5odlZGVm9RMWRzV25KWGJHUmFWbTE0V0ZaWGRHdGhiRXB6WTBac1dtSkdXbWhXYTFwVFZqRmtkVnBIZUdsU2JYY3hWa
1phVTFVeFduSk5XRXBxVWxoQ1YxWnFUbTlsYkZweFVWaG9hMVpzV2pGV01uaHJWakZLV1ZGcmJGZFdNMEpJVmtSS1RtVkdaSFZ
VYlVaVFRXNW9WVlpHWTNoaU1XUkhWMjVTVGxOSGFGQlZha1pIVGtaa2NsWnRkRmRpVlhCNlZUSTFUMVp0U2xWV2ExSmFaV3RhY
UZreFdrdGtSa3B6Vld4T2FWTkZTa3RXTW5oWFZtczFXRkpyWkZoaWF6VnhWVEJvUTFsV1VsWlhibVJyWWtad2VGVnRkREJoYXp
GeVRsVndWMDF1YUhKV2FrWkxWakpPU0dGR2FHbFNiSEJ2VjFaU1MxUXlUWGhhU0ZaVllrWmFjRlpxU205VlJscFlaRWRHV2xZe
FNucFdNalZUVkd4a1NGVnNWbFZXYkhCTVdsWmFVMk14WkhSa1JtUk9ZVEZaTVZac1pEUmhNV1IwVWxob1dHRnJOVmhWYTFaaFp
XeHJlV1ZIUm1waVZrcElWbGQ0VDFSc1dsZGhNMnhYVFZkTmVGcEhNVkpsVmxwMVUyczFWMUpVVmxOV2JYUlRVakZLUjFkc2JGc
GxiWGhTVm10a2JtUXlUa2hWVkRGQ1dsRTlQVFJOVDg=u8'

# Aplicando a função de reversão para retornar ao hash MD5 original
original_md5 = decode_base64_with_fixed_chars(final_encoded_string)
print(f"\nMD5 original: {original_md5}")
```

O hash final obtido é:

```
MD5: fcf1eed8596699624167416a1e7e122e

Hash Cracker
https://crackstation.net/

octopus
```

PoC – Gerando o desafio:

```
import base64
```

```
import random

def encode_base64_with_fixed_chars(input_string, iterations=12):
    base64_iterations = []
    # Realiza o processo por 'iterations' vezes
    for _ in range(iterations):
        # Codifica a string original ou modificada em base64
        base64_encoded = base64.b64encode(input_string.encode()).decode()
        # Gera sempre 2 caracteres aleatórios (pode ser letras ou números)
        random_chars =
''.join(random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', k=2))
        # Adiciona os caracteres aleatórios à string codificada
        input_string = base64_encoded + random_chars
        # Armazena o BASE64 gerado a cada iteração
        base64_iterations.append(base64_encoded)
    return base64_iterations, input_string

# Hash MD5 inicial
md5_hash = 'fcf1eed8596699624167416a1e7e122e'
# Aplicando a função para encadear BASE64 com 2 caracteres aleatórios fixos
iterations_base64, result = encode_base64_with_fixed_chars(md5_hash)

# Apresentando o BASE64 gerado em cada iteração e o resultado final
print("BASE64 gerado a cada iteração:")
for i, val in enumerate(iterations_base64):
    print(f"Iteração {i+1}: {val}")

print(f"\nResultado final: {result}")
```

3º ATO – Dualidade

Mensagem recebida:

```
S01ZVVVUQ1dJVkNURVVaUkpaRUZDMlNLSTVJVk1WU1VLSlZUQ1UyU0tWTkVZVEpRSlpHRkdNSzJKUkxFS1RKUUtFWUhR
VENPS1ZIRktVSlJMSktGT1JMVUs1SldXVlNLS1pWRTRUS1ROTllGRVZUTEtaSEZDTUsyS0ZKR1c2Q0dLTVlEQ1RDTk5OWlRD
VlJRS1pIRktNQ09MRkpUQ1NTTEtKV0VNVlNYTUERVNVVExMSkxWSVZTS0pWSlZLM0NMS1JER0lXQ1dLWkhGTVUzMktaR1
ZHUktHSzVHV1lUU0lLRldHSVNDUkdGTkVZVlNWTVJCRTRSUzJJSkxHV1NTVEtONUZNUzJXS1ZIRkNVM0tLWkhVNFZTS0lKS
UZJTUJaS0JLR0kySlFPND09PT09PXNQor
```

Possível solução:

```
import base64
import re
import binascii

# Função para corrigir o padding da string BASE64
def add_base64_padding(encoded_string):
    missing_padding = len(encoded_string) % 4
    if missing_padding:
        encoded_string += '=' * (4 - missing_padding)
    return encoded_string

# Função para corrigir o padding da string BASE32
def add_base32_padding(encoded_string):
    missing_padding = len(encoded_string) % 8
    if missing_padding:
        encoded_string += '=' * (8 - missing_padding)
    return encoded_string

# Função para remover caracteres inválidos para BASE32
def clean_base32_string(encoded_string):
    """Remove caracteres inválidos para garantir que apenas caracteres BASE32 válidos
permaneçam."""
    return re.sub(r'[^A-Z2-7=]', '', encoded_string)

def decode_alternating_base32_base64(encoded_string):
    # Trabalhar com a string como bytes
    encoded_bytes = encoded_string.encode()

    # Itera 6 vezes, alternando entre BASE32 e BASE64, começando de trás para frente
    for i in range(5, -1, -1):
        # Remove os dois caracteres aleatórios no final
        encoded_bytes = encoded_bytes[:-2]

        # Decodificar alternadamente entre BASE32 e BASE64
        if i % 2 == 0:  # Decodifica BASE32 nas iterações pares
            encoded_str = encoded_bytes.decode()
            encoded_str = clean_base32_string(encoded_str)  # Limpa a string BASE32
            encoded_str = add_base32_padding(encoded_str)  # Corrige o padding
            try:
                encoded_bytes = base64.b32decode(encoded_str)
            except binascii.Error as e:
                print(f"Erro na iteração {6 - i} ao decodificar BASE32: {e}")
                print(f"String com erro: {encoded_str}")
                return None
        else:  # Decodifica BASE64 nas iterações ímpares
            encoded_str = encoded_bytes.decode()
            encoded_str = add_base64_padding(encoded_str)  # Corrige o padding
            try:
                encoded_bytes = base64.b64decode(encoded_str)
            except binascii.Error as e:
                print(f"Erro na iteração {6 - i} ao decodificar BASE64: {e}")
                print(f"String com erro: {encoded_str}")
                return None

        # Mostra a string decodificada em cada iteração
        print(f"Iteração {6 - i}: {encoded_bytes}")
```

```
    # Retorna o resultado final decodificado como string
    return encoded_bytes.decode()

# String codificada final (resultado final fornecido anteriormente)
final_encoded_string =
'S01ZVVVUQ1dJVkNURVVaUkpaRUZDMlNLSTVJVk1WU1VLSlZUQ1UyU0tWTkVZVEpRSlpHRkdNSzJKUkxFS1RKUUtFWUhRVENPS
1ZIRktVSlJMSktGT1JMVUs1SldXVlNLS1pWRTRUS1ROTllGRVZUTEtaSEZDTUsyS0ZKR1c2Q0dLTVlEQ1RDTk5OWlRDVlJRS1p
IRktNQ09MRkpUQ1NTTEtKV0VNVlNYTlJERVNVVExMSkxWSVZTS0pWSlZLM0NMS1JER0lXQ1dLWkhGTVUzMktaR1ZHUktHSzVHV
1lUU0lLRldHSVNDUkdGTkVZVlNWTVJCRTRSUzJJSkxHV1NTVEtJWURLU1NYSVU0VkNVM0tLWkZWRVZDS0lKSUZJTUJaS0JLR1F
WM0lOST09PT09PVdYo8'
# Aplicando a função de reversão para retornar ao hash MD5 original
original_md5 = decode_alternating_base32_base64(final_encoded_string)
print(f"\nMD5 original: {original_md5}")
```

O hash final obtido é:

```
MD5: c4e8f4c9e7e01d9156c069e93829c523

Hash Cracker
https://crackstation.net/

Shadow Government
```

PoC – Gerando o desafio:

```
import base64
import random

def encode_alternating_base32_base64(input_string):
    iterations_output = []

    # Definir a sequência de codificação alternada
    for i in range(6):  # 6 ciclos, alternando entre BASE32 e BASE64
        if i % 2 == 0:  # BASE32
            encoded_string = base64.b32encode(input_string.encode()).decode()
        else:  # BASE64
            encoded_string = base64.b64encode(input_string.encode()).decode()

        # Gera sempre 2 caracteres aleatórios
        random_chars =
''.join(random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', k=2))
        # Adiciona os caracteres aleatórios à string codificada
        input_string = encoded_string + random_chars

        # Armazena o resultado da codificação
        iterations_output.append(encoded_string)

    # Retorna apenas o último resultado, sem uma codificação adicional
    return iterations_output, input_string  # Retorna a string codificada na sexta iteração

# Hash MD5 inicial
md5_hash = 'c4e8f4c9e7e01d9156c069e93829c523'
# Aplicando o sistema alternado de codificação BASE32 e BASE64
iterations_output, result = encode_alternating_base32_base64(md5_hash)

# Apresentando a saída a cada iteração e o resultado final
print("Codificação alternada (BASE32 e BASE64) a cada iteração:")
for i, val in enumerate(iterations_output):
    encoding_type = "BASE32" if i % 2 == 0 else "BASE64"
    print(f"Iteração {i+1} ({encoding_type}): {val}")

print(f"\nResultado final: {result}")
```

4º ATO - Secret eye

Arquivo de origem:

https://github.com/elderofz1on/Cyber-Conspiracy/blob/main/resources/sugar.zip

Possível solução:

```
import requests
import zipfile
import os
import base64
import zlib
import binascii

# Função para baixar o arquivo ZIP do URL
def download_zip_file(url, output_path):
    response = requests.get(url)
    if response.status_code == 200:
        with open(output_path, 'wb') as file:
            file.write(response.content)
        print(f"Arquivo {output_path} baixado com sucesso.")
    else:
        print(f"Falha ao baixar o arquivo. Código de status: {response.status_code}")

# Função para extrair o arquivo ZIP e obter o sugar.txt
def extract_zip_file(zip_path, extract_to):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)
    print(f"Arquivo {zip_path} extraído com sucesso para {extract_to}")

# Função para ler o arquivo TXT com a mensagem codificada
def read_encoded_file(file_path):
    with open(file_path, 'r') as file:
        encoded_message = file.read()
    return encoded_message

# Função para remover múltiplas camadas de BASE64
def remove_multiple_base64(encoded_message, iterations=2):
    decoded_data = encoded_message.encode('utf-8')
    for _ in range(iterations):
        decoded_data = base64.b64decode(decoded_data)
    return decoded_data

# Função para decodificar hexadecimal
def decode_hex(hex_encoded):
    decoded_data = binascii.unhexlify(hex_encoded)
    return decoded_data

# Função para descomprimir os dados usando zlib
def decompress_data(compressed_data):
    decompressed_data = zlib.decompress(compressed_data)
    return decompressed_data

# Função para salvar a imagem final em um arquivo JPG
def save_image(image_data, output_image_path):
    with open(output_image_path, 'wb') as image_file:
        image_file.write(image_data)
    print(f"Imagem salva com sucesso em {output_image_path}")

# Função principal para reverter o processo de codificação
def restore_image_from_zip(zip_url, zip_path, extract_to, txt_filename, output_image_path):
    # Passo 1: Fazer download do arquivo ZIP
    download_zip_file(zip_url, zip_path)

    # Passo 2: Extrair o arquivo ZIP
    extract_zip_file(zip_path, extract_to)
```

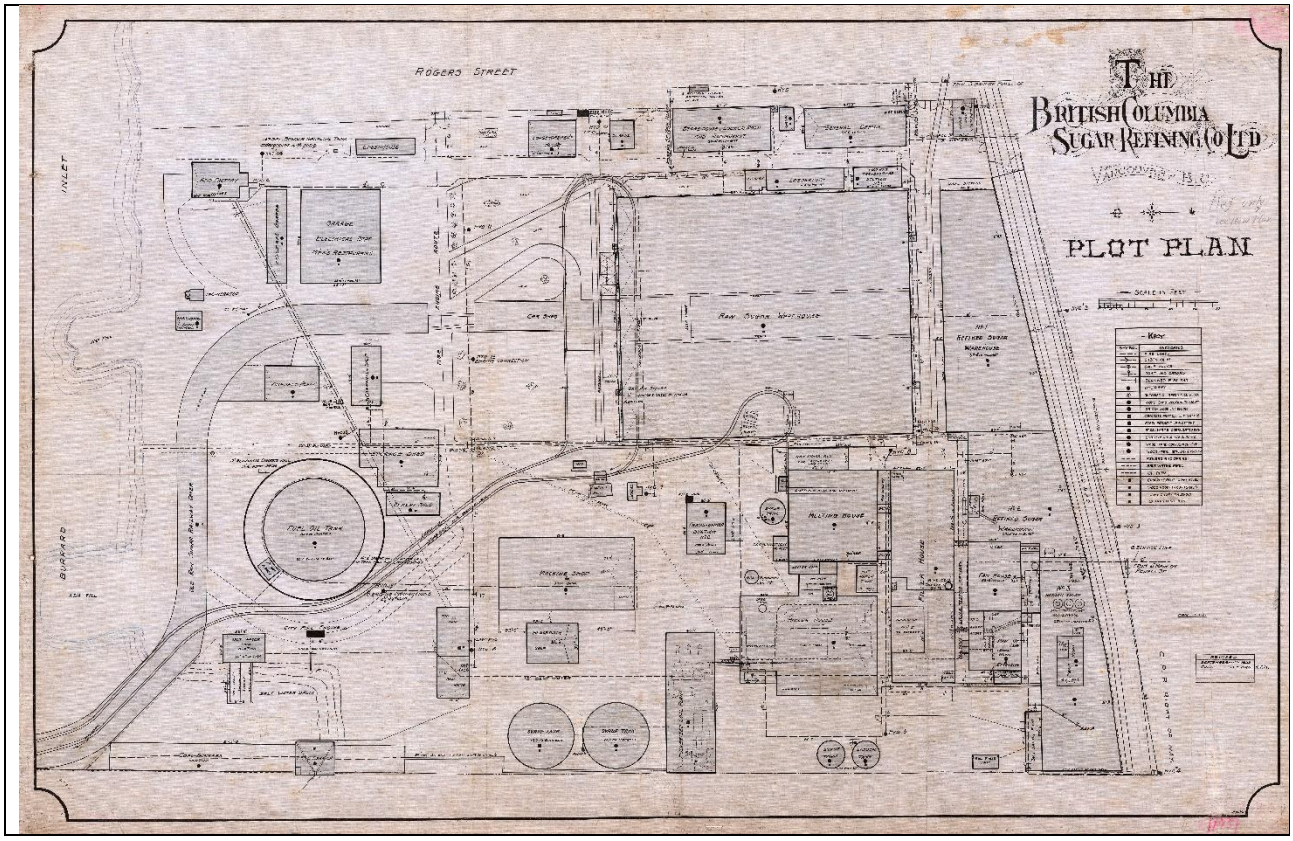
```
    # Passo 3: Ler a mensagem codificada do arquivo TXT extraído
    encoded_file_path = os.path.join(extract_to, txt_filename)
    encoded_message = read_encoded_file(encoded_file_path)

    # Passo 4: Remover múltiplas camadas de BASE64
    base64_decoded = remove_multiple_base64(encoded_message)
    print(f"BASE64 removido: {base64_decoded[:50]}...")  # Mostra parte dos dados decodificados

    # Passo 5: Decodificar hexadecimal
    hex_decoded = decode_hex(base64_decoded)
    print(f"Hexadecimal decodificado: {hex_decoded[:50]}...")  # Mostra parte dos dados
decodificados

    # Passo 6: Descomprimir os dados
    decompressed_data = decompress_data(hex_decoded)
    print(f"Dados descomprimidos: {decompressed_data[:50]}...")  # Mostra parte dos dados
descomprimidos

    # Passo 7: Salvar a imagem decodificada
    save_image(decompressed_data, output_image_path)

# Definir o URL do ZIP, caminhos de arquivos e saída
zip_url = 'https://github.com/elderofz1on/Cyber-Conspiracy/blob/main/resources/sugar.zip?raw=true'
# O arquivo a ser baixado
zip_path = 'sugar.zip'  # Onde o arquivo ZIP será salvo localmente
extract_to = 'extracted_files'  # Diretório onde o ZIP será extraído
txt_filename = 'sugar.txt'  # O nome do arquivo dentro do ZIP
output_image_path = 'sugar.jpg'  # Nome da imagem de saída

# Reverter o processo a partir do arquivo ZIP
restore_image_from_zip(zip_url, zip_path, extract_to, txt_filename, output_image_path)
```

PoC – Gerando o desafio:

```
import base64
import zlib
import binascii

# Função para ler a imagem e retornar seus bytes
def read_image(image_path):
    with open(image_path, 'rb') as image_file:
        image_bytes = image_file.read()
    return image_bytes

# Função para comprimir os dados da imagem
def compress_data(data):
    compressed_data = zlib.compress(data)
    return compressed_data

# Função para codificar em hexadecimal
def encode_hex(compressed_data):
    hex_encoded = binascii.hexlify(compressed_data)
    return hex_encoded

# Função para aplicar múltiplas camadas de BASE64
def apply_multiple_base64(hex_encoded, iterations=2):
    encoded_data = hex_encoded
    for _ in range(iterations):
        encoded_data = base64.b64encode(encoded_data)
    return encoded_data

# Função para salvar a mensagem final em um arquivo TXT
def save_to_file(encoded_message, output_file):
    with open(output_file, 'w') as file:
        file.write(encoded_message.decode('utf-8'))  # Salva como string decodificada
    print(f"Mensagem salva com sucesso em {output_file}")

# Função principal que faz todo o processo
```

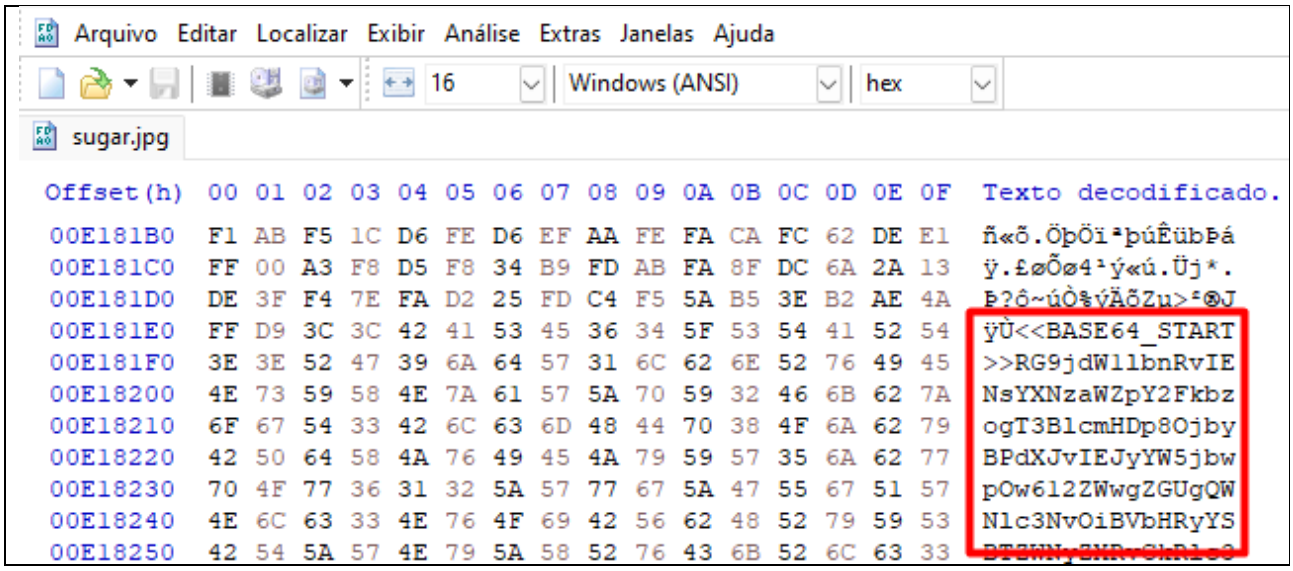
```
def process_image(image_path, output_file):
    # Passo 1: Ler a imagem
    image_bytes = read_image(image_path)

    # Passo 2: Comprimir a imagem
    compressed_data = compress_data(image_bytes)
    print(f"Dados comprimidos: {compressed_data[:50]}...")  # Mostra parte dos dados comprimidos

    # Passo 3: Codificar em hexadecimal
    hex_encoded = encode_hex(compressed_data)
    print(f"Dados em hexadecimal: {hex_encoded[:50]}...")  # Mostra parte dos dados em hexadecimal

    # Passo 4: Aplicar múltiplas camadas de BASE64
    final_encoded = apply_multiple_base64(hex_encoded)
    print(f"BASE64 final (encurtado): {final_encoded[:50]}...")  # Mostra parte dos dados
codificados em BASE64

    # Passo 5: Salvar a mensagem final em um arquivo TXT
    save_to_file(final_encoded, output_file)

# Caminho para a imagem e nome do arquivo de saída
image_path = 'refinaria.jpg'
output_file = 'sugar.txt'

# Processar a imagem e gerar a mensagem final
process_image(image_path, output_file)
```

Imagem obtida após extração:



5º ATO

Ao abrir sugar.jpg em um editor hexadecimal observa-se que após a estrutura formal, foi inserido um bloco BASE64 entre **<<BASE64_START>>** e **<<BASE64_END>>.**



Possível solução:

```
import base64
import re

# Função para ler a imagem e encontrar a mensagem esteganográfica entre os delimitadores
def extract_hidden_message(image_path):
  try:
    print(f"Passo 1: Lendo o arquivo de imagem '{image_path}'...")
    # Ler a imagem em modo binário
    with open(image_path, 'rb') as image_file:
      image_data = image_file.read()

    print(f"Passo 2: Convertendo os dados binários em uma string legível...")
    # Converter os dados binários em uma string legível, com erros ignorados
    image_data_str = image_data.decode('utf-8', errors='ignore')

    print("Passo 3: Buscando pelos delimitadores <<BASE64_START>> e <<BASE64_END>>...")
    # Procurar pelo padrão entre os delimitadores <<BASE64_START>> e <<BASE64_END>>
    pattern = re.search(r'<<BASE64_START>>(.*?)<<BASE64_END>>', image_data_str, re.DOTALL)

    if pattern:
      hidden_message = pattern.group(1)
      print("Passo 4: Mensagem oculta encontrada com sucesso!")
      return hidden_message
    else:
      print("Nenhuma mensagem oculta foi encontrada.")
      return None
  except Exception as e:
    print(f"Erro ao ler a imagem ou encontrar a mensagem: {e}")
    return None

# Função para decodificar a mensagem BASE64 extraída
def decode_hidden_message(base64_message):
  try:
    print("Passo 5: Decodificando a mensagem BASE64...")
    # Decodificar a mensagem BASE64
    decoded_data = base64.b64decode(base64_message).decode('utf-8')
    print("Passo 6: Mensagem decodificada com sucesso!")
    return decoded_data
  except Exception as e:
    print(f"Erro na decodificação: {e}")
    return None
```

```
# Função principal para extrair e decodificar a mensagem oculta da imagem
def extract_and_decode_message(image_path):
  print("=== Início do processo de extração e decodificação ===")

  # Extrair a mensagem oculta
  hidden_message = extract_hidden_message(image_path)

  if hidden_message:
    # Exibir a mensagem BASE64 integralmente
    print(f"Mensagem extraída (BASE64 completa): {hidden_message}")

    # Decodificar a mensagem BASE64
    decoded_message = decode_hidden_message(hidden_message)

    if decoded_message:
      print("=== Mensagem oculta decodificada ===")
      print(decoded_message)

  print("=== Fim do processo ===")

# Caminho da imagem onde os dados esteganográficos estão escondidos
image_path = 'sugar.jpg'

# Extrair e decodificar a mensagem oculta
extract_and_decode_message(image_path)
```

Texto codificado e adicionado na imagem refinaria.jpg (esteganografia):

RG9jdW1lbnRvIENsYXNzaWZpY2FkbzogT3BlcmHDp8OjbyBPdXJvIEJyYW5jbwpOw612ZWwgZGUgQWNlc3NvOiBVbHRyYSBTZWNyZXRvC
kRlc3RpbmF0w6FyaW9zOiBNZW1icm9zIGRvIENvbnNlbGhvIFNlcnBlbnQKQXNzdW50bzogSW5maWx0cmHDp8OjbyBlIFNhYm90YWdlbS
BkYSBSZWZpbmFyaWEgZGUgQcOnw7pjYXIKCkNhcm9zIG1lbWJyb3MgZG8gU2VycGVudCBDb3VuY2lsLAoKT3MgcHJlcGFyYXRpdm9zIHB
hcmEgYSBmYXNlIGZpbmFsIGRhIE9wZXJhw6fDo28gT3VybyBCcmFuY28gZXN0w6NvIGNvbXBsZXRvcy4gTm9zc2EgaW5maWx0cmHDp8O
jbyBuYSBSZWZpbmFyaWEgZGUgQcOnw7pjYXIgZGEgQnJpdGlzaCBDb2x1bWJpYSBhbnRpdSBvIGFjZXNzbyBuZWNlc3PDoXJpbyBwY
XJhIGNvbXByb21ldGVyIHN1YXMgb3BlcmHDp8O1ZXMgc2VtIHF1ZSBvcyBzaXN0ZW1hcyBkZSBzZWd1cmFuw6dhIGRldGVjdGVtIHF1YWxx
dWVyIGFub21hbGlhLgoKQXF1aSBlc3TDo28gb3MgZGV0YWxoZXMgYXR1YWxpemFkb3MgZGFzIHByw7N4aW1hcyBhw6fDtWVzOgoKMS4g
U2Fib3RhZ2VtIGRvcyBUYW5xdWVzIGRlIEFybWF6ZW5hbWVudG86Ck5vc3NhcyBlcXVpcGVzIHBvc2ljaW9uYXJhbSBkaXNwb3MM
gZGUgaW50ZXJydXDDp8OjbyBub3MgdGFucXVlcyBkZSBjb21idXN0w612ZWwgcHLDs3hpbW9zIGFvIEZ1ZWwgT2lsIFRhbmsuIFF1YW5kby
BhdGl2YWRvcywgY2F1c2Fyw6NvIHVtYSBpbnRlcnJ1cMOnw6NvIG1hY2nDp2Egbm8gZm9ybmVjaW1lbnRvIGRlIGVuZXJnaWEsIG8gcXVlIG
lyw6EgcGFyYWxpc2FyIHRvZGEgYSByZWZpbmFyaWEuCkEgZGVzdHJ1acOnw6NvIHNlbGV0aXZhIGRlIHBvbnRvcy1jaGF2ZSBubyB0YW5xd
WUgZGUgY29tYnVzdMOtdmVsIHNlcsOhIHJlYWxpemFkYSBkZSBtYW5laXJhIHF1ZSBhcyBleHBsb3PDtWVzIHNlamFtIHZpc3RhcyBjb21vIG
ZhbGhhcyBpbnRlcm5hcywgZGlmaWN1bHRhbmRvIHF1YWxxdWVyIGludmVzdGlnYcOnw6NvIHNvYnJlIGEgbm9zc2EgaW50ZXJ2ZW7Dp8
Ojby4KCjIuIEludGVyZmVyw6puY2lhIG5hcyBMaW5oYXMgZGUgUHJvZHDp8OjbzoKCkluZmlsdHJhbW9zIGPDs2RpZ28gbWFsaWNpb3NvI
G5vIHNpc3RlbWEgZGUgY29udHJvbGUgZGFzIG3DoXF1aW5hcyBsb2NhbGl6YWRhcyBuYSBNYWNoaW5lcnkgUm9vbS4gRXNzZSBjw7NkaA
WdvIHNlcsOhIGF0aXZhZG8gw6BzIDAyOjAwIGRhIHByw7N4aW1hIHF1YXJ0YS1mZWlyYSwgaW50ZXJyb21wZW5kbyBvIGZsdXhvIGRhIHBy
b2R1w6fDo28gZSBjcmlhbmRvIHVtIGNhb3Mgb3BlcmFjaW9uYWwgcXVlIGFmZXRhcsOhIGEgZGlzdHJpYnVpw6fDo28gZGUgYcOnw7pjYXIg
cmVmaW5hZG8gZ2xvYmFsbWVudGUuCk5vc3NhIGVxdWlwZSBqw6EgbW9kaWZpY291IG9zIHJlZ2lzdHJvcyBpbnRlcm5vcyBwYXJhIHF1ZS
BvcyBlbmdlbmhlaXJvcyBkZSBwbGFudMOjbyB0ZW5oYW0gZGlmaWN1bGRhZGVzIGVtIHJhc3RyZWFyIGEgb3JpZ2VtIGRvIHByb2JsZW1hLg
oKMy4gTWFuaXB1bGHDp8OjbyBkbyBTaXN0ZW1hIEVsw6l0cmljbzoKQSB1bmlkYWRlIGVsw6l0cmljYSBjZW50cmFsLCBzaXR1YWRhIG5vI
EVsZWN0cmljYWwgRGVwYXJ0bWVudCwgc2Vyw6EgY29tcHJvbWV0aWRhIGNvbSB1bSBhdGFxdWUgZGUgbmVnYcOnw6NvIHNlcnZ
pw6dvIChEb1MpLiBFc3NlIGF0YXF1ZSBzZXLDoSBwcm9ncmFtYWRvIHBhcmEgY2F1c2FyIHVtYSBzb2JyZWNhcmdhIG5vcyBjaXJjdWl0b3M
sIGRlc2FiaWxpdGFuZG8gbyBzaXN0ZW1hIGRlIGNvbnRyb2xlIGRlIGVtZXJnw6puY2lhIGUgZm9yw6dhbmRvIG8gZGVzbGnYW1lbnRvIGRhI
HByb2R1w6fDo28uCkEgc29icmVjYXJnYSBzZXLDoSBncmFkWFsIHBhcmEgbsOjbyBhY2lvbmFyIG9zIHNpc3RlbWFzIGF1dG9tw6F0aWNv
cyBkZSBiYWNrdXAgYXTDqSBxdWUgc2VqYSB0YXJkZSBkZW1haXMgcGFyYSBxdWFscXVlciBpbnRlcnZlbsOnw6NvIG1vLgoKNC4gQWNlc3NvI
UgTWFuaXB1bGHDp8OjbyBkYXMgw4FyZWFzIGRlIEVzdG9jYWdlbToKQSDDoXJlYSBkZSBSYXcgU3VnYXIgV2FyZWhvdXNlIHNlcsOhIGFmZX
RhZGEgY29tIGEgY29udGFtaW5hw6fDo28gZG9zIGxvdGVzIGFybWF6ZW5hZG9zLiBPcyBhZ2VudGVzIGluZmlsdHJhZG9zIGlyw6NvIGNvbXB
yb21ldGVyIG9zIHNhY29zIGRlIGHDp8O6Y2FyIGNydSBjb20gc3Vic3TDom5jaWFzIHF1ZSByZXRhcmRhcsOjbyBhIGNyaXN0YWxpemHDp8O
jbyBkbyBhw6fDumNhciwgY29tcHJvbWV0ZW5kbyBvcyBsb3RlcyBmdXR1cm9zIGUgY2F1c2FuZG8gdW0gaW1wYWN0byBkaXJldG8gbmEg
cXVhbGlkYWRlIGRvIHByb2R1dG8gZmluYWwuCkEgY29udGFtaW5hw6fDo28gc2Vyw6EgaW5kZXRlY3TDoXZlbCBwb3IgbcOpdG9kb3MgY2
9udmVuY2lvbmFpcywgZ2FyYW50aW5kbyBxdWUgb3MgaW1wYWN0b3Mgc8OzIHNlcsOjbyBwZXJjZWJpZG9zIGFww7NzIGEgZGlzdHJpYnV
pw6fDo28gZG8gcHJvZHV0by4KCjUuIFByw7N4aW1vcyBQYXNzb3M6CkEgc2Fib3RhZ2VtIHNlcsOhIHNpbmNyb25pemFkYSBwYXJhIGNva
W5jaWRpciBjb20gYSBtdWRhbsOnYSBkZSB0dXJubyBuYSByZWZpbmFyaWEsIG1pbmltaXphbmRvIGEgY2hhbmNlIGRlIGRldGVjw6fDo28uI
ENhZGEgY8OpbHVsYSBkZXZlIGVzdGFyIHByZXBhcmFkYSBwYXJhIGV2YWN1YXIgYSDDoXJlYSBpbWVkaWF0YW1lbnRlIGFww7NzIGEgYXR
pdmHDp8OjbyBkb3MgZGlzcG9zaXRpdm9zLgpOb3NzYSBvcGVyYcOnw6NvbmNsdcOtZGEgZW0gNDggaG9yYXMgZSBkZWl
4YXLDoSBvIEJyaXRpc2ggQ29sdW1iaWEgU3VnYXIgUmVmaW5pbmcgQ28uIGVtIHVtIGVzdGFkbyBkZSBjb2xhcHNvIG9wZXJhY2lvbmFsIGly
cmV2ZXJzw612ZWwsIHByZXBhcmFuZG8gbyBjZW7DoXJpbyBwYXJhIG5vc3NhIGludGVydmVuw6fDo28gbmFzIG5lZ29jaWHDp8O1ZXMgZn
V0dXJhcy4KCkxlbWJyZW0tc2U6IEEgZGVzdHJ1acOnw6NvIHNpc3RlbcOhdGljYSBkYXMgaW5mcmFlc3RydXR1cmFzIGluZHVzdHJpYWlzIM
OpIG8gcHJpbWVpcm8gcGFzc28gcGFyYSBvIGNvbnRyb2xlLiBPIGNhb3Mgw6kgbyBub3NzbyBpbnN0cnVtZW50byBtYWlzIGVmaWNhei4KC
kFiYWl4by1hc3NpbmFkbzoKTyBBbHRvIENvbnNlbGhvClNlcnBlbnQgQ291bmNpbCDigJMgSW5maWx0cmF0aW9uIERpdmlzaW9u

Texto Decodificado:

Documento Classificado: Operação Ouro Branco
Nível de Acesso: Ultra Secreto
Destinatários: Membros do Conselho Serpent
Assunto: Infiltração e Sabotagem da Refinaria de Açúcar

Caros membros do Serpent Council,

Os preparativos para a fase final da Operação Ouro Branco estão completos. Nossa infiltração na Refinaria de Açúcar da British Columbia garantiu o acesso necessário para comprometer suas operações sem que os sistemas de segurança detectem qualquer anomalia.

Aqui estão os detalhes atualizados das próximas ações:

1. Sabotagem dos Tanques de Armazenamento:
Nossas equipes posicionaram dispositivos de interrupção nos tanques de combustível próximos ao Fuel Oil Tank. Quando ativados, causarão uma interrupção maciça no fornecimento de energia, o que irá paralisar toda a refinaria.
A destruição seletiva de pontos-chave no tanque de combustível será realizada de maneira que as explosões sejam vistas como falhas internas, dificultando qualquer investigação sobre a nossa intervenção.

2. Interferência nas Linhas de Produção:

Infiltramos código malicioso no sistema de controle das máquinas localizadas na Machinery Room. Esse código será ativado às 02:00 da próxima quarta-feira, interrompendo o fluxo da produção e criando um caos operacional que afetará a distribuição de açúcar refinado globalmente.
Nossa equipe já modificou os registros internos para que os engenheiros de plantão tenham dificuldades em rastrear a origem do problema.

3. Manipulação do Sistema Elétrico:
A unidade elétrica central, situada no Electrical Department, será comprometida com um ataque de negação de serviço (DoS). Esse ataque será programado para causar uma sobrecarga nos circuitos, desabilitando o sistema de controle de emergência e forçando o desligamento da produção.
A sobrecarga será gradual para não acionar os sistemas automáticos de backup até que seja tarde demais para qualquer intervenção.

4. Acesso e Manipulação das Áreas de Estocagem:
A área de Raw Sugar Warehouse será afetada com a contaminação dos lotes armazenados. Os agentes infiltrados irão comprometer os sacos de açúcar cru com substâncias que retardarão a cristalização do açúcar, comprometendo os lotes futuros e causando um impacto direto na qualidade do produto final.
A contaminação será indetectável por métodos convencionais, garantindo que os impactos só serão percebidos após a distribuição do produto.

5. Próximos Passos:
A sabotagem será sincronizada para coincidir com a mudança de turno na refinaria, minimizando a chance de detecção. Cada célula deve estar preparada para evacuar a área imediatamente após a ativação dos dispositivos.
Nossa operação será concluída em 48 horas e deixará o British Columbia Sugar Refining Co. em um estado de colapso operacional irreversível, preparando o cenário para nossa intervenção nas negociações futuras.

Lembrem-se: A destruição sistemática das infraestruturas industriais é o primeiro passo para o controle. O caos é o nosso instrumento mais eficaz.

Abaixo-assinado:
O Alto Conselho
Serpent Council – Infiltration Division