Desarrollo Seguro de Aplicaciones - Práctica 1

Autores - Grupo ‘Error 404’

- Juan Cruz Cassera Botta, 17072/7
- Brian Llamocca, 02037/9
- Franco Camacho, 16194/1

Notas

- Fecha de entrega: 17/04/2025 a las 17:59.
- Para cada ejercicio, se deberá comprobar que fueron resueltos correctamente ingresando la respuesta en el CTF.
- La bandera (flag) a encontrar tiene el formato **flag{[a-z_]*}**.
- Término relacionado: OWASP top 10 (2021): A02 (Cryptographic Failures).

PGP

Ejercicios

1) La finalidad de este ejercicio es trabajar los conceptos de **Pretty Good Privacy (PGP)** vistos en clase, para ello cada alumno/a deberá formar grupo de hasta tres personas y comunicarlo a la cátedra de la siguiente manera:
    a) El correo debe contener apellido, nombre, nro. de alumno y usuario de github, usuario y grupo del CTF de todos sus miembros.
    b) El correo debe estar **firmado con PGP.**
    c) Adjuntar las claves PGP de los/as integrantes del grupo.
    d) Enviar el correo **cifrado con PGP** tanto como para sus compañeros como para:
        i) einar@info.unlp.edu.ar
        ii) mcarbone@linti.unlp.edu.ar
        iii) sandrazilla@gmail.com
        iv) nmacia@info.unlp.edu.ar

        Para este ejercicio haremos lo siguiente:

        1. Cada miembro de nuestro grupo generará sus claves **pública y privada** PGP.
        2. Uno de nosotros enviará un email **firmado y cifrado** con PGP a los otros 2 y a los 4 profesores de la cátedra, con la siguiente información (formato → apellido y nombre | número de alumno | usuario de github | usuario de CTF | grupo de CTF):

        - Juan Cruz Cassera Botta | 17072/7 | https://github.com/Juan-Cruz-GH | Juan Cruz Cassera | Error

404
- Brian Llamocca | 02037/9 | https://github.com/notBraii | Braii | Error 404
- Franco Camacho | 16194/1 | https://github.com/fran1999 | Fran | Error 404

```
gpg --encrypt --sign --armor \
  -r c.javierfranco99@gmail.com\
  -r llamoccas.brian@gmail.com\
  -r juancasserab@gmail.com \
correo.txt
  -r einar@info.unlp.edu.ar \
  -r mcarbone@linti.unlp.edu.ar \
  -r sandrazilla@gmail.com \
  -r nmacia@info.unlp.edu.ar \
```

Recibimos la flag de parte de Sandra:

flag{bien_ahi_enviaron_el_email}

Encoding

Herramientas sugeridas

- CyberChef.
- dcode.

Ejercicios

2) Develar el mensaje que se intentó ocultar utilizando un sistema de codificación:
    a) 102 108 97 103 123 101 109 112 101 122 97 110 100 111 95 97 95 101 110 99 111 100 101 97 114 125

    Los números pertenecen al estándar **ASCII**: 102 equivale a “f”, 108 equivale a “l”, 97 a “a”, y así sucesivamente. Por ende la flag completa es:

    flag{empezando_a_encodear}

    b) ZmxhZ3szbmMwZGVhcl9uMF8zc19lbmNyMXB0NHJ9

    El string presentado está escrito en **Base64**. Si lo traducimos de Base64 a texto plano, obtenemos la flag:

    flag{3nc0dear_n0_3s_encr1pt4r}

    c) 6-12-1-7{2-9-5-14-22-5-14-9-4-15-19-1-4-19-1}

    Estos **números** podemos inferir que se **mapean** a **letras** del abecedario, donde 1 → “a”, 2 → “b”, … , 26 → ”z” (a1 z26). Se asume que el mapeo no es case sensitive, por ende la flag completa es:

flag{bienvenidosadsa}

d) 66 6c 61 67 7b 68 33 78 63 6f 64 33 34 6e 64 30 5f 74 30 64 30 7d

Se ve claramente que los caracteres están en **hexadecimal**. La flag es:

flag{h3xcod34nd0_t0d0}

e) ..-. .-.. .- --. { .... --- .-.. .. ... }

Este último mensaje está codificado en código **Morse**. Su flag es:

FLAG{HOLIS}

Hashing

Tips de la cátedra

- Averiguar qué tipo de hash se está utilizando. Analizar la cantidad de caracteres.
- Herramientas sugeridas:
  - CrackStation.
  - dcode.

Ejercicios

3) Utilizando alguna página que permita la búsqueda inversa de hashes, busca el texto plano de los siguientes:
   a) bef58f652fddb1c20ecbfdb7cf31d932

      Este hash posee **32 caracteres**. Además, son caracteres hexadecimales, los cuales son de 4 bits cada uno. 32 * 4 = 128 bits. Un algoritmo de hashing que siempre produce un hash de 128 bits es el **md5**, por ende se puede asumir que este hash fue producido por ese algoritmo.

      Usando CrackStation obtenemos la siguiente flag a partir de este hash:

      weakpass

   b) c8074675a6310657d3ddf7f35a61bf393af41141

      Este hash posee **40 caracteres**. Igual que antes, son caracteres hexadecimales → 40 * 4 = 160 bits. Un algoritmo de hashing que siempre produce un hash de 160 bits es el **SHA-1**, por ende seguramente este hash fue producido así.

      Usando md5decrypt.net o md5hashing.net obtenemos la siguiente

flag a partir de este hash:

muyseguro

c) 111fca2d52def4c33f4d8f1be7e74d14b65d365e5ddb9161
0c3c0dbecc192073b0b0df28213e3828cc0321f6286baf944
49a4f8803203be3293595f4d67ff7e2

Este hash posee **128 caracteres**. Igual que antes, son caracteres hexadecimales → 128 * 4 = 512 bits. Un algoritmo de hashing que siempre produce un hash de 512 bits es el **SHA-512**.

Usando CrackStation obtenemos la siguiente flag al crackear el hash:

superpassword

En resumen, se obtiene la siguiente tabla:

Hash (incisos)	Cant. caracteres	Función	Texto
bef58f652fddb1c20ecbfdb7cf31d932	32	MD5	weakpass
c8074675a6310657d3ddf7f35a61bf393af411 41	40	SHA-1	muyseguro
111fca2d52def4c33f4d8f1be7e74d14b65d36 5e5ddb91610c3c0dbecc192073b0b0df28213 e3828cc0321f6286baf94449a4f8803203be32 93595f4d67ff7e2	128	SHA-512	superpassword

Tabla 1. Hashes con sus respectivos textos.

4) Hashing - MD5 hacker: Encontrá la clave correcta.

Para encontrar la clave correcta primero inspeccionamos la página, nos fijamos los archivos que provee la misma (md5.js y demo.js) y vemos que

el archivo demo.js chequea si la función md5 con parámetro = clave que el usuario ingresa da como resultado el hash **434d0c349e0e35ee9ed740e83a23fee5.** Por ende, hacemos una búsqueda inversa de md5 con ese hash usando la página md5decrypt mencionada anteriormente. Haciendo esto, hallamos que la contraseña válida es:

chinitalenda

Criptografía

Tips de la cátedra

- Qué algoritmo de criptografía se utiliza.
- Investigar sobre las funciones CryptoJS / Encrypted / Decrypted.
- Responder la pregunta que representa la flag.

Ejercicios

5) Criptografía Básica - AES: Para encontrar la flag en este reto vas a tener que desencriptar la variable “flagEncrypted” con la password “myPassword”

   Nuevamente, inspeccionamos la página. Vemos que hay un archivo js.js, el cual encripta el string “Esta no es la flag” usando la clave “myPassword” y luego desencripta ese resultado usando esa misma clave.

   La función que encripta y desencripta en este código proviene de CryptoJS, la cual es una biblioteca de cifrado de JavaScript que ofrece funciones criptográficas. Con Crypto.js, se puede cifrar y descifrar mensajes usando diferentes algoritmos de cifrado como AES (Advanced Encryption Standard, el que se usa en este caso), DES, Triple DES, RC4, Rabbit, entre otros.

   Para hallar la flag, modificamos el código para que en vez de desencriptar la variable **encrypted** desencripte el string **flagEncrypted.** Al hacer esto, obtenemos la flag:

   flag{Why_encrypt_in_clientside?}

   Pregunta: ¿Por qué encriptar del lado del cliente? El mayor motivo por el cual alguien podría encriptar del lado del cliente es en el cifrado de extremo a extremo (E2EE), caso en el cual el servidor no conoce la clave de desencriptado para mayor privacidad. Es el caso de las aplicaciones de mensajería.

Introducción a HTTP/HTML/JS

Tips de la cátedra

- Inspeccionar código fuente.
- Herramientas sugeridas:
  - Burp Suite Community Edition.
  - curl, postman, insomnia.
  - zaproxy.

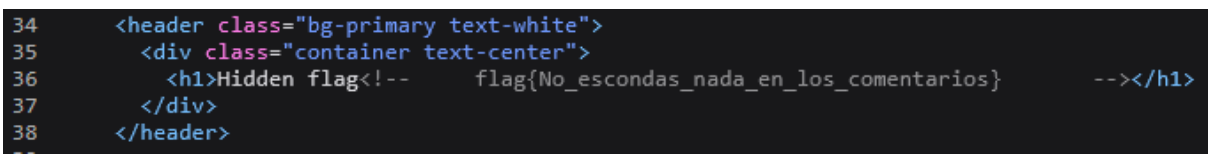
Ejercicios

6) Estos ejercicios sirven para aprender conceptos básicos de HTML, HTTP y JS. Todos se pueden resolver en el cliente (tu navegador). Para resolverlos vas a necesitar:
   a) Ver el código fuente de la página (HTML y JS).
   b) Ver las peticiones HTTP de respuesta.
   c) Debuggear JS para manipular el valor de una variable.
   d) Utilizar las herramientas de programador del navegador.

A. Mirá los source: Encontrá la bandera:

Para el siguiente ejercicio inspeccionamos la página, revisamos el archivo HTML ***comentario*** dentro del código fuente. Notamos que en la línea 36 (dentro del header “Hidden flag”) se encuentra un comentario con la siguiente flag:

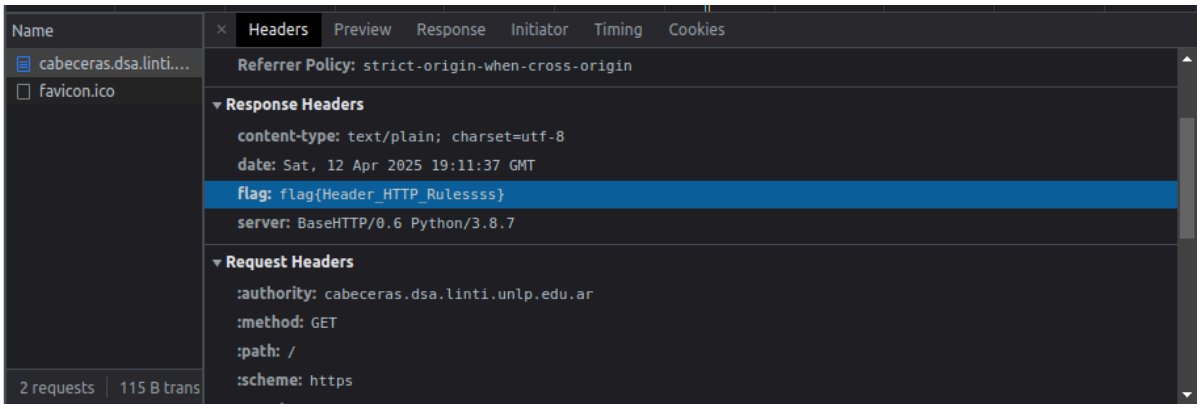
flag{No_escondas_nada_en_los_comentarios}



B. Cabeceras: Sabes ver los requerimientos HTTP de respuesta?: https://cabeceras.dsa.linti.unlp.edu.ar/

Viendo los headers de respuesta del requerimiento HTTP GET que nuestro browser hace a la página mencionada, vemos que uno los headers que la página nos envía es una flag:

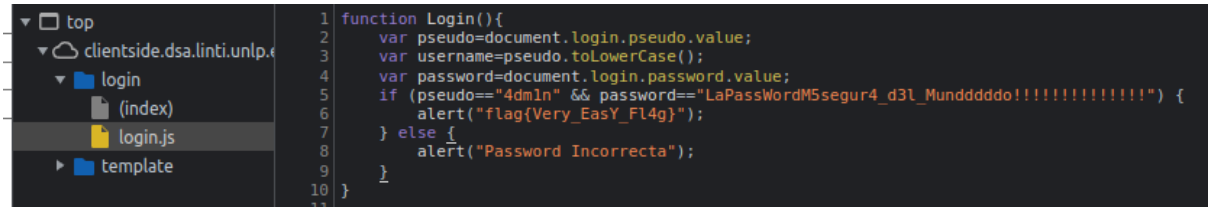
flag{Header_HTTP_Rulessss}



C. Login?: Encontrá las credenciales correctas para obtener la flag: https://clientside.dsa.linti.unlp.edu.ar/login

Inspeccionando la página, dentro la sección de Sources, dentro de la carpeta login, en el archivo login.js se encuentra las credenciales para loguearse y obtener la flag:

flag{Very_EasY_Fl4g}



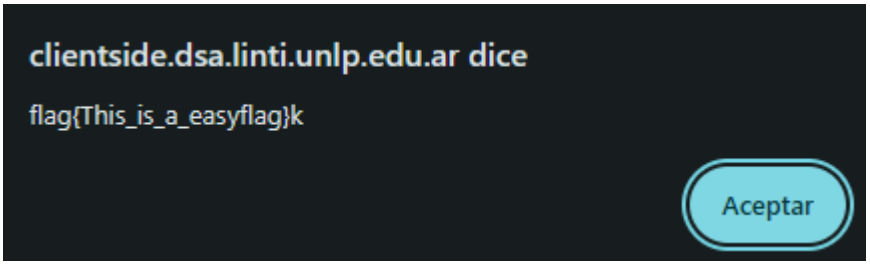
D. Debugging JS: Utilizá tu capacidad de debugger JS. Para encontrar la flag utiliza breakpoints y cambiá de valor la variable “getFlag”: https://clientside.dsa.linti.unlp.edu.ar/breakpoint/

   En este caso nuevamente inspeccionamos vemos que el archivo js.js tiene una variable booleana que indica si se nos mostrará la flag o no. Como está en **false** inicialmente, no se nos muestra nada al clickear el botón Obtenerflag.

   Entonces usamos el debugger del browser para modificar localmente esa variable booleana y settearla en true, y al hacerlo obtenemos la flag:

   flag{This_is_a_easyflag}k

```
21  $( ".fo" ).submit(function( event ) {
22    var getFlag = true;
23    var doNothing = "33";
24    if (getFlag){
25      y = hex_to_ascii("070e02031e32090b103b0c153e033c01041518040f05021b0a");
26      z = "abcdef";
27      x = xor_str(y,z);
28      alert(x);
29    }
```

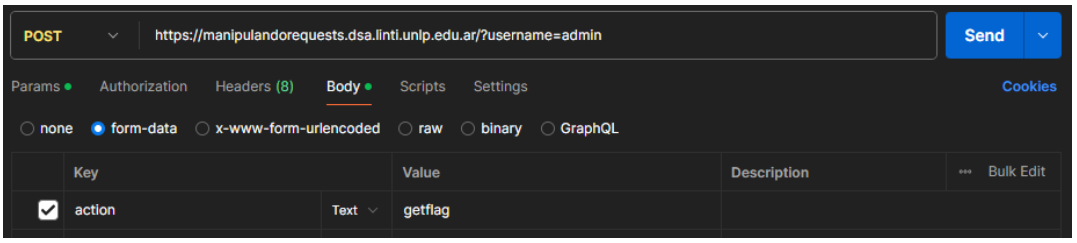


E. Manipulando requests nivel 1: ¿Podés mandar variables por POST sin tener un formulario? https://manipulandorequests.dsa.linti.unlp.edu.ar/

   En este último ejercicio, se fue siguiendo las indicaciones dadas por la página web:

Se realizó el pasaje de parámetros como se ve en la siguiente url: manipulando request con pasaje de parámetros. De esa manera, la página nos devuelve una nueva instrucción indicando que se debe realizar un POST con la siguiente clave-valor: ***action=getflag***.

En este último paso, se tuvo en cuenta la herramienta postman para realizar peticiones a la API de dicha página. Se realizó un método POST con la clave-valor en el **body** como se ve a continuación:



Hecho eso, en la página se muestra la siguiente flag:

flag{Y4_s3_M4ndar_v4r1ables!}