•[DragonJAR Security Conference 2016 – CTF Writeup]•

. Web 100 – Web hacking .

. Descripción .

. Análisis .

Al ingresar a la URL del reto se nos muestra un mensaje de error con código "400 Bad Request". Examinando el código fuente de la página de error observamos un comentario HTML muy interesante:

```
<html>
<body>
<h1>BAD Request</h1>
<p>Your browser sent a request that this server could not
understand</p>
<!-- no provienes de http://www.cia.gov/cyberterrorism/1q2w.php
--><!-- no usas MacOSX --><!-- no usas Napster -->
<p>If you think is a error server, please contact the
webmaster</p>
<h2>Error 400</h2></body><html>
```

Se trata de un reto clásico en el cual debemos engañar a la aplicación web modificando las cabeceras Referer y User-Agent de nuestra solicitud HTTP.

. Solución .

Usamos el comando "curl" para enviar el request asignando como Referer la URL "http://www.cia.gov/cyberterrorism/1q2w.php" y como User-Agent el valor "Mozilla/5.0 (MacOSX Napster)".

```
# curl  -A "Mozilla/5.0 (MacOSX Napster)" -H 'Referer:
http://www.cia.gov/cyberterrorism/1q2w.php' http://game-
ehacking.rhcloud.com/
```

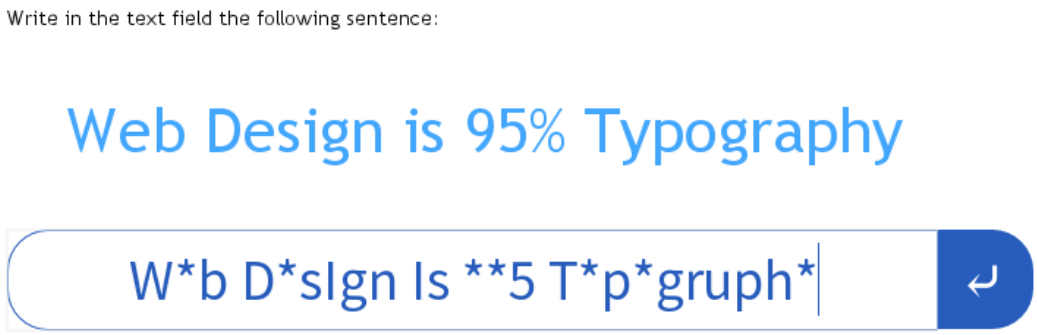
El flag es: d0f5c7596f569f72377a47038d98565e76195d1d

. Web 150 – Tipografia .

. Descripción .

. Análisis .

La web nos muestra un cuadro de texto y nos pide ingresar la frase "Web Design is 95% Typography". Sin embargo, no es tan simple como aparenta. Al escribir en el cuadro de texto observamos que ciertos caracteres son reemplazados por otros.



Podemos suponer que las sustituciones se hacen con Javascript. Sin embargo, luego de examinar el código Javascript, concluimos que no es así. Además observamos que el formulario envía el texto al servidor tal cual ha sido ingresado, sin realizar ninguna modificación.

```
[...]
        $.ajax({
            url: "wtf1",
            type: "POST",
            data: JSON.stringify({
                f: $('input[name="f"]').val()
            }),
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: function(a) {
[...]
```

Al revisar el código CSS observamos que el cuadro de texto emplea una fuente personalizada la cual se encuentra en el archivo “wtf1.ttf”.

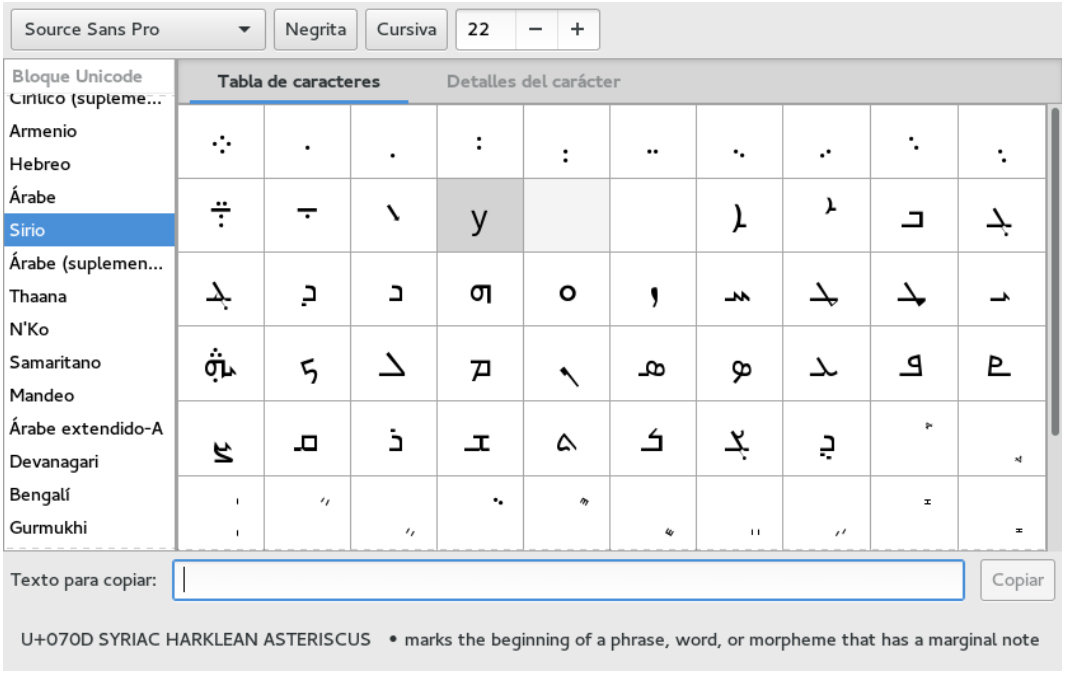
```
[...]
@font-face {
    font-family: 'wtf1';
    src: url('wtf1.ttf') format('truetype');
    font-weight: normal;
    font-style: normal;
}
[...]
.center input[type='text'] {
        font-family: 'wtf1';
        font-size: 2.8 rem;
[...]
```

El fichero “wtf1.ttf” contiene una fuente llamada “Source Sans Pro” que ha sido generada, aparentemente, a partir de la fuente “Trebuchet MS” intercambiando,

reemplazando o moviendo los glifos de ciertos caracteres. Por ello, al escribir observamos glifos que no se corresponden con los caracteres pulsados en el teclado.

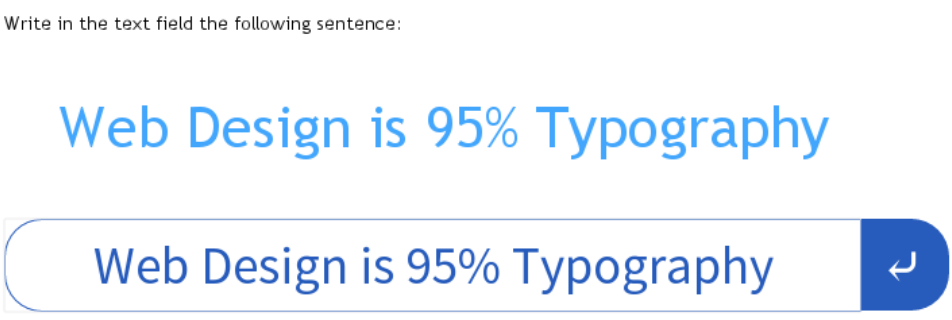
. Solución .

El reto consiste en encontrar los nuevos caracteres asociados al glifo correcto para que el texto se vea tal cual nos lo indican. Empleamos "charmap" (Windows) o "gucharmap" (Linux/Gnome) para analizar el fichero "wtf1.ttf" y buscar manualmente los caracteres.



El caracter más difícil de encontrar fue la "y". Este se encontraba en el bloque unicode correspondiente al alfabeto Sirio. Luego de mucho tiempo y un gran esfuerzo grupal logramos reunir todos los caracteres faltantes.

```
Wêb DêsIgn Is )%& T*pôgrâph*
```



Luego de enviar el formulario obtenemos el flag.

```
Congratulations: bc9c21e45d19fd8a33581a130558a1f3
```

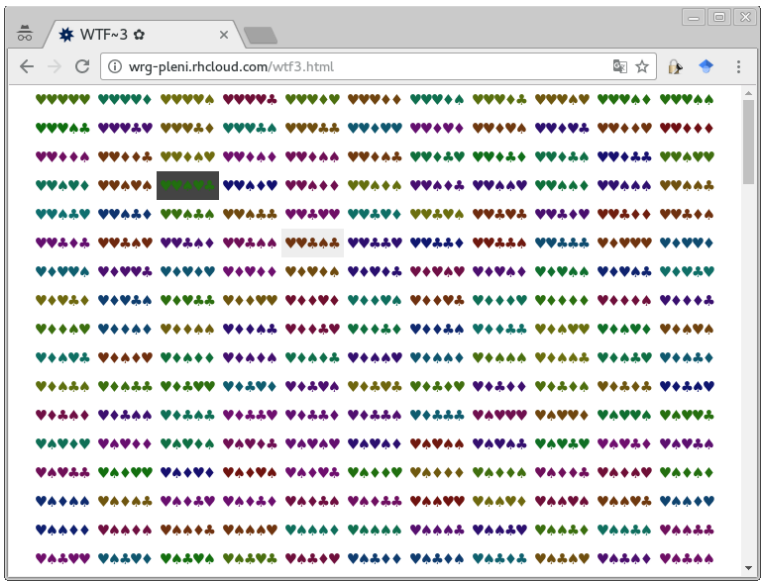
El flag es: bc9c21e45d19fd8a33581a130558a1f3

. Web 200 – Crazy links .

. Descripción .

. Análisis .

La web del reto nos muestra una página con 1024 enlaces cada uno de los cuales envía un número distinto al servidor. El número está comprendido entre 0 y 1023. Solo hay un enlace que envía el número correcto y tenemos 10 intentos para encontrarlo.



En el código Javascript observamos que existe un array “f” el cual contiene 10 caracteres diferentes, un contador “g” inicialmente en 0 y una variable “e” a la cual se le asigna un valor extraído de la cookie. Al hacer clic en los enlaces, también se envía al servidor un objeto JSON con el resultado de concatenar el caracter del array “f” que se encuentra en la posición “g” y el valor de la variable “e”.

```
[...]
        e = "",
        f = ["!", "{", "}", "[", "]", "-", "*", "(", ")", "^"],
        g = 0;
    $.get("wtf3", function(a, b, c) {
        e = /^.*wtf3=([0-9]{1,10}).*$/.exec(document.cookie)[1]
    });
    for (var h = Math.floor(Math.random() * a.length), i = 0; i < Math.pow(2, c);
i++) {
        var j = a[h],
            k = b(i, 10, 4, !1, j),
            l = [j[0], j[0], j[0], j[0], j[0]].join(""),
            m = l.substring(0, c / 2 - k.length) + k,
            n = Math.floor(360 * Math.random()),
            o = $('<a href="/' + d + "?n=" + i + '" style="color:hsl(' + n +
',75%,25%)">' + m + "</a>");
        o.click(function(a) {
            a.preventDefault(), $(this).addClass("wrong"), $.ajax({
                url: $(this).attr("href"),
                type: "POST",
                data: JSON.stringify({
                    m: f[g] + e
                }),
                contentType: "application/json; charset=utf-8",
                dataType: "json",
                success: function(a) {
                    switch (a.msg) {
                        case "yep":
                            $(".wrapper").html(a.html);
                            break;
                        default:
                            g++, $(".msg").remove(), $(".wrapper").prepend('<div
class="msg"><p>Server not convinced. You still have ' + (10 - g) + '
attempts</p><div class="close">X</div></div>'), $(".close").click(function() {
                                $(".msg").remove()
                            }), g >= 10 && location.reload()
                    }
[...]
```

En efecto, al hacer clic sobre un enlace, observamos que se envía la siguiente solicitud HTTP.

```
POST /wtf3?n=8 HTTP/1.1
Host: wrg-pleni.rhcloud.com
User-Agent: Mozilla/5.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=utf-8
X-Requested-With: XMLHttpRequest
Referer: http://wrg-pleni.rhcloud.com/wtf3.html
Content-Length: 17
Connection: close
```

```
{"m":"!633839779"}
```

Aparentemente los caracteres del array “f” actúan como un prefijo que le indica al servidor el número de intentos. Luego, si enviamos siempre el mismo prefijo podemos burlar el límite de diez intentos e ir enviando uno a uno los 1024 números (o los que hagan falta) hasta encontrar el correcto.

. Solución .

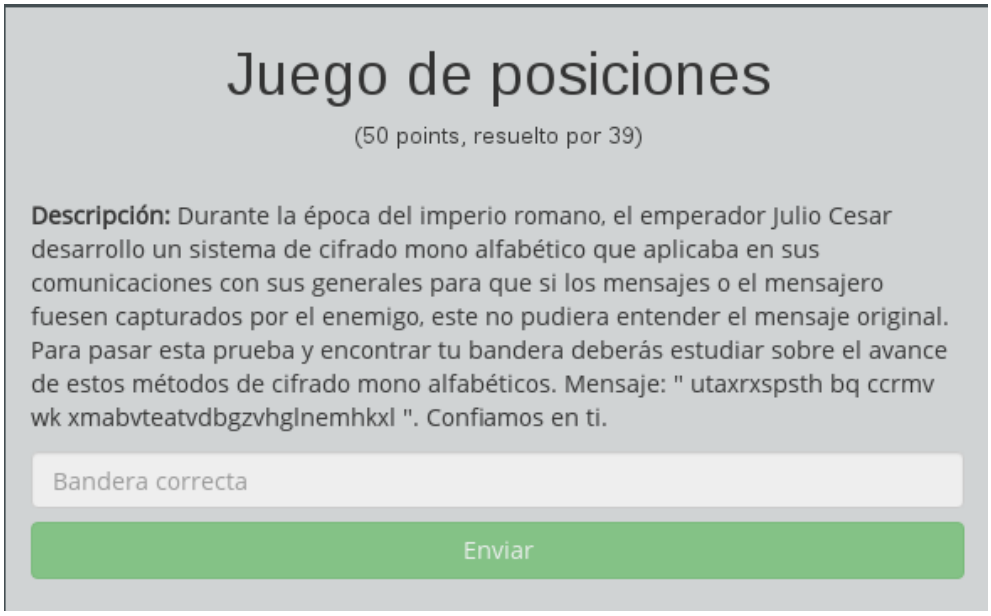
Empleamos la siguiente instrucción bash:

```
for i in `seq 0 1024`; do curl -H 'Content-Type:
application/json; charset=utf-8' --data '{"m":"}633839779"}'
"http://wrg-pleni.rhcloud.com/wtf3?n=$i" 2>/dev/null; done
```

El flag es: f88008dc63a67983e5824dafa0935662

. Crypto 50 – Juego de posiciones .

. Descripción .



. Análisis .

Este reto por la descripción estaba claro que se trataba de un cifrado cesar. Por lo que ingresamos en la web http://www.xarg.org/tools/caesar-cipher/, colocamos en el recuadro ***"utaxrxspsth bq ccrmv wk xmabvteatvdbgzvhglnemhkxl"*** y dando la opción a guess nos escupe:

Output:

baheyezwzao ix jjytc dr ethicalhackingconsultores

El flag es: ethicalhackingconsultores

. Crypto 100 - Telegram 2 .

. Descripción .

. Análisis .

Luego de contactar con el bot por Telegram, activamos la opción "Envío de imágenes". El bot comenzó a enviarnos imágenes en forma de fichas de un rompecabezas cada 5 minutos. Además nos muestra el mensaje: "Necesitamos la contraseña del usuario Dragonjar". Tras reunir todas las fichas, armamos el rompecabezas.

Observamos el contenido de un fichero ".htpasswd" que contiene usuarios y hashes de contraseñas. Con mucho cuidado transcribimos el contenido de la imagen a un fichero de texto. La letra "l" (ele minúscula) y la "I" (i mayúscula) son muy similares, por ello ante la duda, duplicamos el registro y usamos ambos caracteres para garantizar que el hash correcto se encuentra en el archivo.

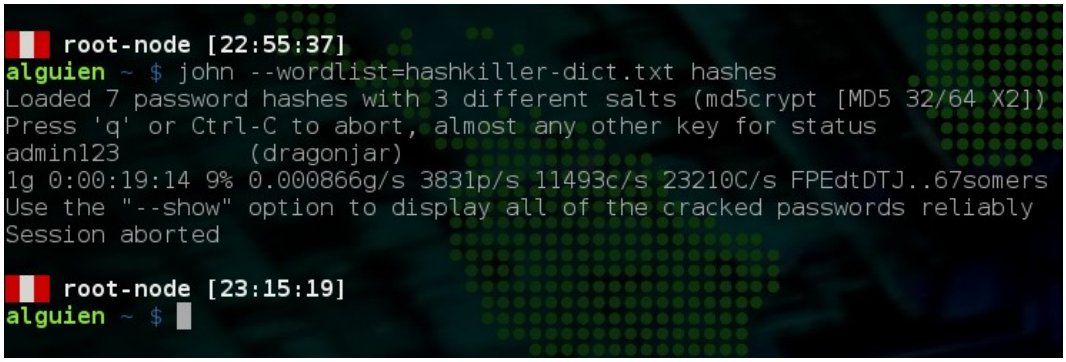
```
dragonjar:$apr1$w43XR4O8$6Pb8MIcIE/f33xB8cviZ3.
dragonjar:$apr1$w43XR4O8$6Pb8MIclE/f33xB8cviZ3.
dragonjar:$apr1$w43XR4O8$6Pb8MlcIE/f33xB8cviZ3.
dragonjar:$apr1$w43XR4O8$6Pb8MlclE/f33xB8cviZ3.
root:$apr1$RofJBd7W$WaVaXfkcKiCMh6kqynYL9/
ehacking:$apr1$eba7obgQ$TImnSFGR8tYw1Tr1W2rpY1
```

```
ehacking:$apr1$eba7obgQ$TlmnSFGR8tYw1Tr1W2rpY1
```

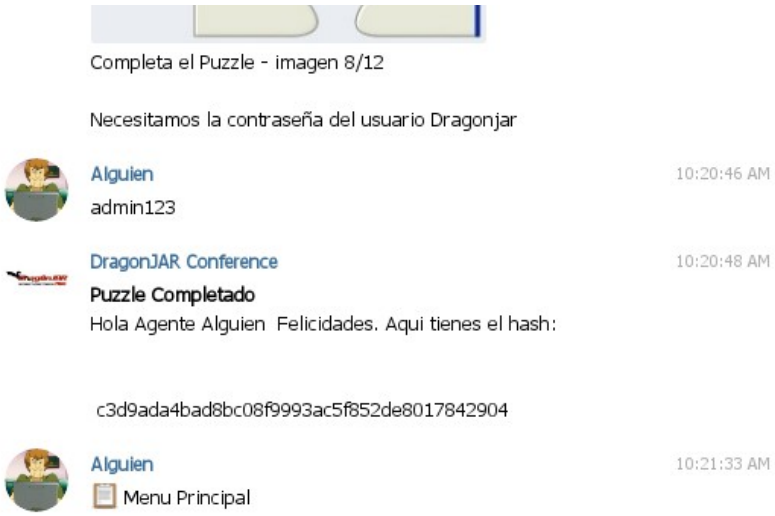
. Solución.

Crackeamos los hashes usando John the Ripper y el diccionario de claves de Hashkiller[1].

```
john --wordlist=hashkiller-dict.txt hashes
```



La contraseña del usuario “dragonjar” es “admin123”. Se la enviamos al bot por Telegram y nos entrega el flag.



El flag es: c3d9ada4bad8bc08f9993ac5f852de8017842904

1 HashKiller Passwords - http://home.btconnect.com/md5decrypter/hashkiller-dict.rar

. Crypto 150 – Telegram .

. Descripción .

Este reto consta de dos partes, por ello vamos a separar el análisis y la solución para cada parte del reto.

. Análisis (Parte I) .

Solicitamos al bot por Telegram el reto "Reto 1 – Frases Celebres" y nos entrega el siguiente mensaje cifrado.

```
.Ixp zlkqoxpbñxp cyx mywy vk byzk sxdobsyb. Tu vakjky jkpgx wak
tgjok rg bkg, uvsvj trdszricr ivxlcridvekv q fg ugehsjlajds ugf
wpljsñgk
 VW4gZ3JhbiBjb25zZWpvIMK/UXVpZW4gZXMgZWwgYXV0b3I/
```

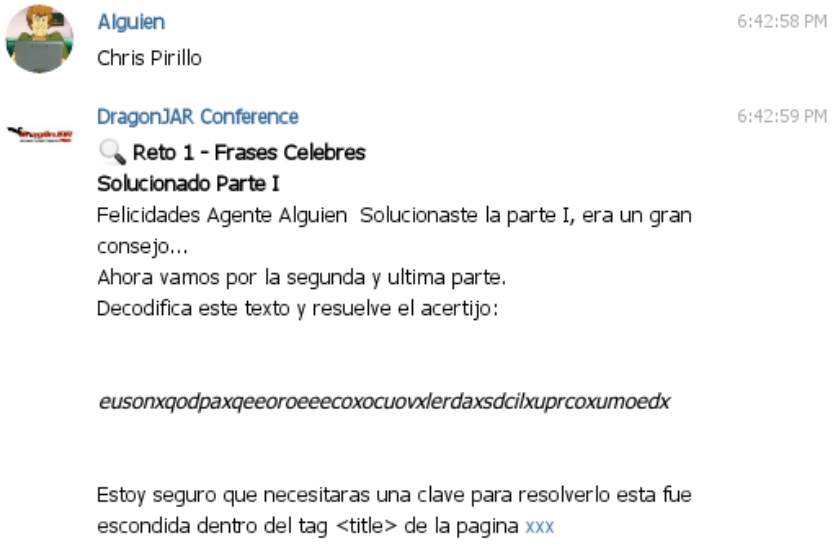
Claramente se diferencian dos partes en el mensaje. La primera parte parece un cifrado de sustitución simple, posiblemente César. La segunda parte es base64.

. Solución (Parte I) .

Usamos la web "https://www.nayuki.io/page/automatic-caesar-cipher-breaker-javascript" para descifrar la primera parte del mensaje. Se trata de César, pero usando rotaciones diferentes para varios segmentos del texto. Luego de decodificar cada segmento obtenemos:

```
Las contraseñas son como la ropa interior. No puedes dejar que
nadie la vea, debes cambiarla regularmente y no compartirla con
extraños
```

Por otra parte, el texto en base64 dice ***"Un gran consejo ¿Quien es el autor?"***. Luego de buscar en Google nos enteramos que el autor de la frase es "Chris Pirillo". Se lo enviamos al bot y nos entrega la segunda parte del reto.




```
eusonxqodpaxqeeoroeeecoxocuovxlerdaxsdcilxuprcoxumoedx
```

. Análisis (Parte II) .

Este cifrado nos mantuvo entretenidos mucho tiempo, de hecho este reto fue el último que resolvimos. Antes de dar con el algoritmo de cifrado correcto probamos muchos otros, entre ellos la Escítala, Vigenère y Alberti (por la pista de "razonamiento circular" de la descripción).

El bot nos dice que necesitamos una contraseña y que está en el título de la página el evento. Así que probablemente sea alguna de esas palabras.

```
<title>DragonJAR Security Conference – Congreso de Seguridad
Informática</title>
```

Una observación importante fue que el criptograma contiene caracteres "x" a intervalos regulares. Esto finalmente nos permitió identificar el algoritmo de cifrado.

```
eusonx
qodpax
qeeoroeeecox
ocuovx
lerdax
sdcilx
uprcox
```

```
umoedx
```

Se trata del algoritmo de cifrado de transposición por columnas[2]. Este cifrado además nos permite saber la longitud de su contraseña. Para ello dividimos la longitud total del criptograma (54) entre la longitud de los segmentos que terminan en "x" (6). Es decir, la contraseña debe tener nueve caracteres. Luego las posibles contraseñas que aparecen en el título son "dragonjar" y "seguridad".

. Solución (Parte II) .

Para resolverlo programamos el siguiente script en python. La contraseña resultó ser "dragonjar".

```
#/usr/bin/env python


def transpo(texto, clave):
    cifra = ''
    clave_len = len(clave)
    tabla = []
    for i in xrange(0, len(texto), clave_len):
        fila = ''
        if i+clave_len < len(texto):
            fila = texto[i:i+clave_len]
        else:
            fila = texto[i:] + 'x' * (clave_len - len(texto[i:]))
        tabla.append([x for x in fila])

    clave_ord = [x for x in clave]
    clave_ord.sort()
    for c in clave_ord:
        pos = clave.index(c)
        cifra += ''.join([fila[pos] for fila in tabla])
    return cifra

def des_transpo(cripto, clave):
    texto = ''
    clave_len = len(clave)
    cosi = len(cripto) / clave_len
    tabla = [cripto[i:i+cosi] for i in xrange(0, len(cripto),
cosi)]
    clave_ord = [x for x in clave]
    clave_ord.sort()
    for i in xrange(0, cosi):
        posi = []   # parche para posiciones repetidas xD
        for c in clave:
            pos = clave_ord.index(c)
            if c in posi:
                pos += 1
            posi.append(c)
            texto += tabla[pos][i]
```

2 Algoritmos de transposición – Transposición por columnas. http://redyseguridad.fi-p.unam.mx/proyectos/criptografia/criptografia/index.php/2-tecnicas-clasicas-de-cifrado/22-opereraciones-utilizadas/223-algoritmos-de-transposicion?showall=&start=3

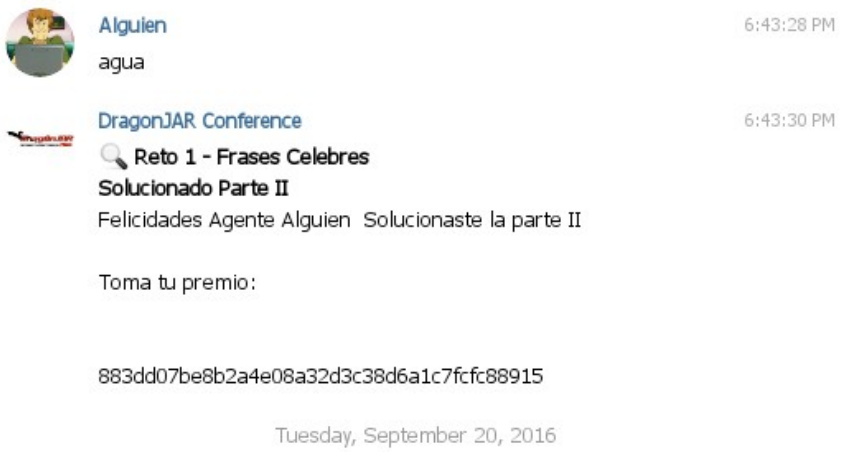
```
    return texto


cifra = 'eusonxqodpaxqeeoroeeecoxocuovxlerdaxsdcilxuprcoxumoedx'
print des_transpo(cifra, 'dragonjar')
```

La salida es:

```
queesloquepuedecomersecrudoococidoperonolavadoxxxxxxxx
```

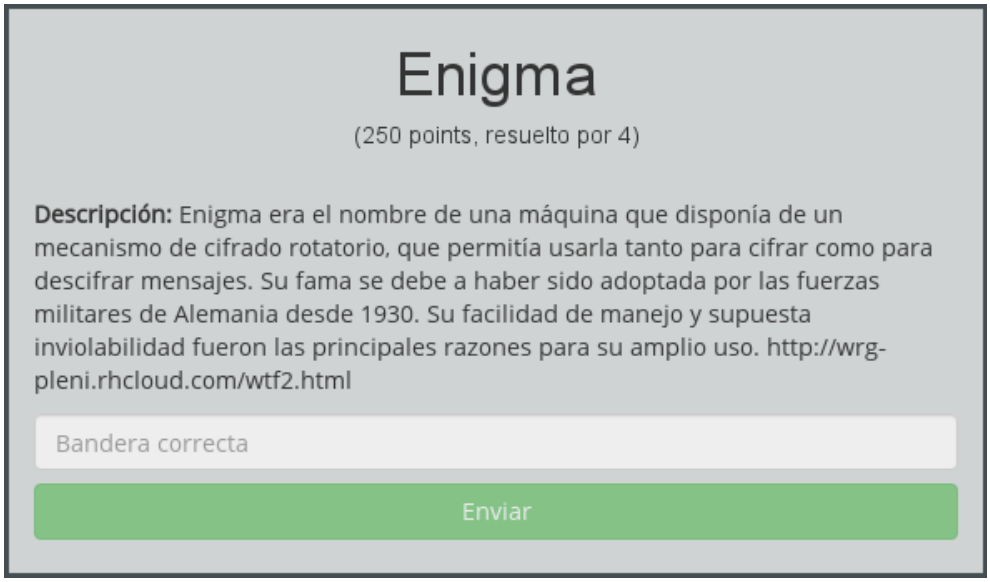
¿Qué es lo que puede comerse crudo o cocido pero no lavado? ¡Agua!



El flag es: 883dd07be8b2a4e08a32d3c38d6a1c7fcfc88915

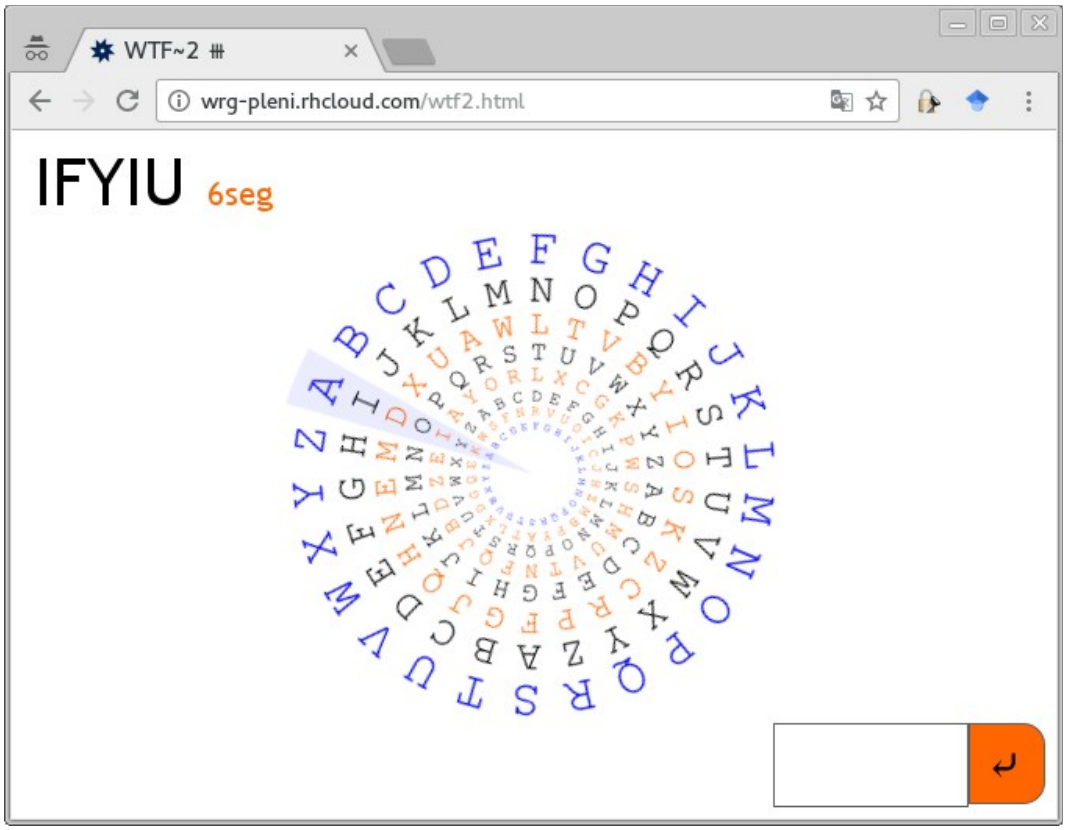
. Crypto 250 – Enigma .

. Descripción .



. Análisis .

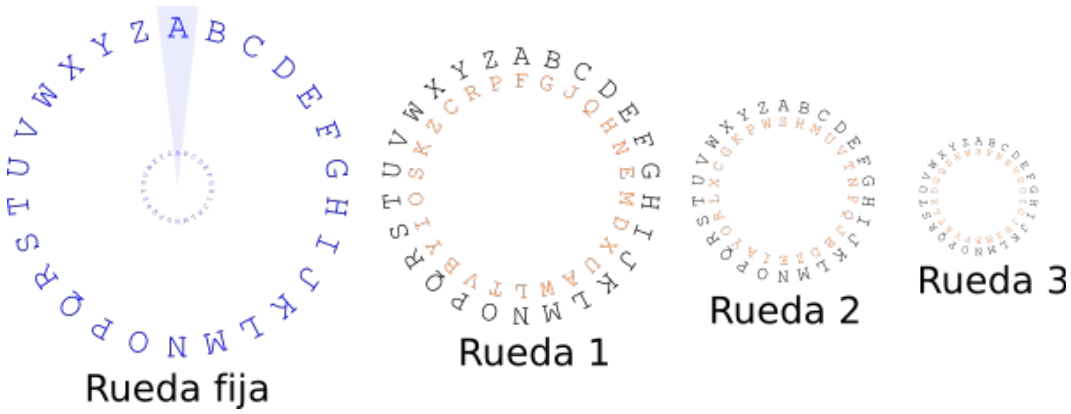
El reto es bastante explícito ¡Se trata de la máquina Enigma! Observamos el código cifrado en la parte superior izquierda junto a un contador que nos da tan solo 10 segundos antes de que el código cambie. En la parte central están la rueda fija (en azul) y las tres ruedas móviles (en negro y naranja) que nos muestran la configuración inicial de la máquina sombreada con gris. La configuración inicial también cambia cada 10 segundos. Finalmente en la parte inferior derecha hay una caja de texto para enviar el código descifrado.



Tenemos todo lo que necesitamos para descifrar el código. Nuestro principal problema es el tiempo. Antes de que terminemos de ingresar la configuración inicial y el código cifrado en un simulador de Enigma los 10 segundos ya habrán terminado y todo habrá cambiado. Por ello, decidimos automatizar la resolución del reto con Javascript ;)

. Solución .

El primer problema a resolver será extraer la configuración inicial de la máquina, es decir, cuantas posiciones se ha desplazado cada rueda móvil con respecto a la rueda fija. Las ruedas son imágenes SVG a las cuales se les aplica una transformación de rotación usando CSS. Podemos calcular la configuración inicial a partir del número de grados de dicha rotación.



La siguiente función Javascript recibe como parámetro el número de rueda y devuelve su configuración inicial.

```
function getrot(n) {
    rot = (/rotate\((.+)deg/.exec( $('.r' + n)[0].style.transform
)[1]) * 1;
    pos = Math.round(rot / (360 / 26)) % 26;
    pos = (pos < 0) ? 26 + pos : pos;
    return 26 - pos;
}
```

El siguiente problema a resolver es conseguir una implementación sencilla de la máquina Enigma que podamos ejecutar desde la consola Javascript del navegador. Decidimos programar nuestra propia implementación de Enigma y para ello fue fundamental la explicación[3] que da el usuario constructor91 de Youtube sobre el mecanismo de cifrado y descifrado de Enigma.

El resultado a continuación...

```
Rueda = function(alfa, rota) {
    this.alfa = alfa;
    this.rota = rota;

    this.rotar = function() {
        this.rota = (this.rota + 1) % this.alfa.length;
        return this.rota == 0;
    }

    this.valor = function(pos) {
        return this.alfa[(this.rota + pos) % this.alfa.length];
    }

    this.buscar = function (letra, indice) {
        for (var pos=0; pos < this.alfa.length; pos ++) {
            if (letra == this.alfa[pos][indice]) {
                pos = pos - this.rota;
                return (pos < 0) ? this.alfa.length + pos : pos;
            }
        }
        return null;
    }

    this.estado = function() {
        s = ''
        for(var i=0; i<this.alfa.length; i++) {
            s += this.alfa[(this.rota + i) % this.alfa.length] +
'|';
        }
        console.log(s);
    }
}
```

3 Maquina Enigma de LU1DSS II https://www.youtube.com/watch?v=Wsdx9CniR78

```
Enigma = function (rot1, rot2, rot3) {
    this.alfa = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    this.ruedas = [
        new Rueda(['AF', 'BG', 'CJ', 'DQ', 'EH', 'FN', 'GE',
'HM', 'ID', 'JX', 'KU', 'LA', 'MW', 'NL', 'OT', 'PV', 'QB', 'RY',
'SI', 'TO', 'US', 'VK', 'WZ', 'XC', 'YR', 'ZP'], rot1),
        new Rueda(['AS', 'BH', 'CM', 'DU', 'EV', 'FT', 'GN',
'HF', 'IQ', 'JJ', 'KB', 'LD', 'MZ', 'NE', 'OI', 'PA', 'QY', 'RO',
'SR', 'TL', 'UX', 'VC', 'WG', 'XK', 'YP', 'ZW'], rot2),
        new Rueda(['AS', 'BF', 'CN', 'DR', 'EV', 'FU', 'GO',
'HI', 'IC', 'JJ', 'KH', 'LZ', 'MM', 'NB', 'OP', 'PY', 'QA', 'RT',
'SL', 'TX', 'UD', 'VG', 'WQ', 'XE', 'YK', 'ZW'], rot3)
    ];

    this.rotar = function () {
        return this.ruedas[2].rotar() && this.ruedas[1].rotar()
&& this.ruedas[0].rotar();
    }

    this.cifrar_letra = function(letra) {
        pos = this.alfa.indexOf(letra);
        for(var i=this.ruedas.length - 1; i>=0; i--) {
            letra = this.ruedas[i].valor(pos)[1];
            pos = this.ruedas[i].buscar(letra, 0);
        }
        this.rotar();
        return this.alfa[pos];
    }

    this.decifrar_letra = function(letra) {
        pos = this.alfa.indexOf(letra);
        for(var i=0; i<this.ruedas.length; i++) {
            letra = this.ruedas[i].valor(pos)[0];
            pos = this.ruedas[i].buscar(letra, 1);
        }
        this.rotar();
        return this.alfa[pos];
    }

    this.cifrar = function (texto) {
        cripto = '';
        for(var i=0; i < texto.length; i++) {
            cripto += this.cifrar_letra(texto[i]);
        }
        return cripto;
    }

    this.decifrar = function(cripto) {
        texto = '';
        for(var i=0; i < cripto.length; i++) {
            texto += this.decifrar_letra(cripto[i]);
        }
        return texto;
    }
}
```

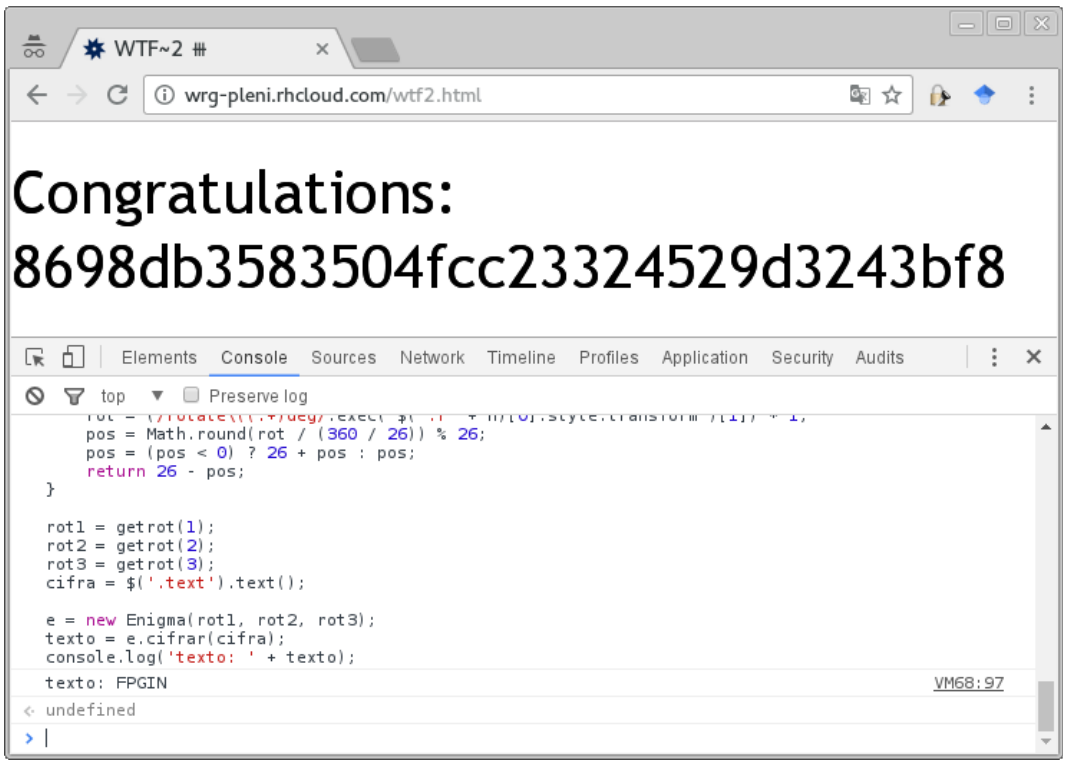
```
function getrot(n) {
    rot = (/rotate\((.+)deg/.exec( $('.r' + n)[0].style.transform
)[1]) * 1;
    pos = Math.round(rot / (360 / 26)) % 26;
    pos = (pos < 0) ? 26 + pos : pos;
    return 26 - pos;
}

rot1 = getrot(1);
rot2 = getrot(2);
rot3 = getrot(3);
cifra = $('.text').text();

e = new Enigma(rot1, rot2, rot3);
texto = e.cifrar(cifra);
console.log('texto: ' + texto);
```

Nota: La máquina enigma cifra en una dirección y descifra en la dirección contraria. Por alguna extraña razón terminamos implementando nuestra máquina Enigma en sentido contrario al del reto. En consecuencia tuvimos que usar el método "cifrar" para descifrar y viceversa xD

El código anterior lo copiamos, pegamos y ejecutamos en la consola Javascript del navegador. Nos imprime el código descifrado el cual ingresamos en la caja de texto para obtener el flag.



El flag es: 8698db3583504fcc23324529d3243bf8

. Estego 80 – Logo DragonJarCon .

. Descripción .

. Análisis .

Descargaba un fichero PNG dónde se encontraba el flag escondido. Ninguno de los que estábamos jugando éramos experto en esta categoría, nuestro nivel de estenografía se limitaba a abrirlo con el editor hexadecimal.

```
3D50h: 22 2D D9 3A 6C DB 79 E9 25 DE AB B5 98 8C E2 95   "-Ù:lÛyé%Þ«µ˜Œâ•
3D60h: FF 0B 66 8A 91 C8 0C 3C C0 85 00 00 00 0E 74 45   ÿ.fŠ'È.<À….....tE
3D70h: 58 74 43 6F 6D 6D 65 6E 74 00 44 72 34 67 30 6E   XtComment.Dr4g0n
3D80h: D9 8C BF 45 00 00 00 00 49 45 4E 44 AE 42 60 82   ÙŒ¿E....IEND®B`‚
```

Nuestra mente ágil y discernidora determinó rápidamente que la cadena de caracteres “Dr4g0n” era una contraseña (en realidad, después de tanto intentar fue deducido).

El equipo comenzó a probar muchas técnicas, desde el LSB (Least Significat Bit), invertir los colores hasta jugar con las curvas de colores. El segundero sin descanso nos presionaba con su sonido y luego de muchas pruebas un software fue probado, OpenStego era su nombre.

OpenStego tiene la opción de extraer datos, donde se puede especificar una contraseña, al ingresar “Dr4g0n” como tal, se generó un archivo que al hacer cat del mismo soltaba (no imprime chars nulos y badchars).

```
00000000: 5777 675a 6d78 794f 4446 694d 4745 314e  WwgZmxyODFiMGE1Nz
00000010: 7a4e 6b4d 5755 304d 5749 304e 5530 4d57  zNkMWU0MWI0NU0MW
```

```
00000020: 4930 4e55 304d 5749 304e 6d45 345a 6d45  I0NU0MWI0NmE4ZmE
00000030: 7a59 7a6b 7a4d 6d49 794e 4455 3159 325a  zYzkzMmIyNDU1Y2Z

[…]

00000790: 457a 597a 6b7a 4d6d 4979 4e44 5531 5932  EzYzkzMmIyNDU1Y2
000007a0: 5a6a 5954 5269 4d51 6f3d 0a57 7767 5a6d  ZjYTRiMQo=.WwgZm
000007b0: 786b 7a4d 6d49 794e 4455 3159 325a 6a59  xkzMmIyNDU1Y2ZjY
000007c0: 5452 694d 516f 3d0a                      TRiMQo=.
```

Para obtener las cadenas codificadas en base64 sin los *badchars* bastaba con ejecutar los siguientes comandos.

```
$ cat dragonJar | awk '{print $10}'  > output.txt &&
  tail -n 1 dragonJar | awk '{print $6}' >> output.txt &&
  cat output.txt | tr -d "\n" > base64.txt &&
  sed -i 's/\./\n/g' base64.txt
```

Finalmente, para decodificarlos en la consola.

```
$ for i in `cat base64.txt`; do echo $i | base64 -d; done
```

Una cadena codificada daba el flag.

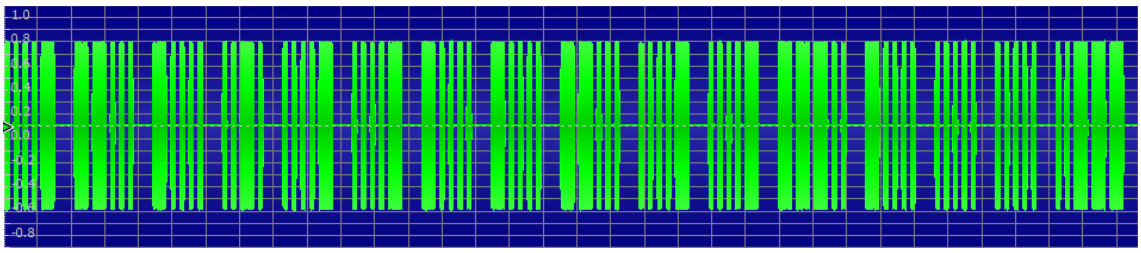
El flag es: a6281b0a573d1e41b46a8fa3c932b2455cfca4b1001

. Estego 80 – Musica ligera .

. Descripción .

. Análisis .

Al escuchar el archivo descargado, se identifican rápidamente tonos largos y cortos indicando de que nuestros oídos escuchan a morse, su legado. El análisis con espectrómetro nos daría los caracteres para la posterior codificación.



```
....- --... -.... ..-. ....- ....- -.... -.... --... ....- ....- ....-
---.. -.... ..... ..... ..---
```

La cadena obtenida esta codifica en hexadecimal, y al hacerlo se obtenía el flag.

```
>>> '476f446674486552'.decode('hex')
'GoDftHeR'
```

El flag es: GoDftHeR

. Misc 20 – Directorios .

. Descripción .

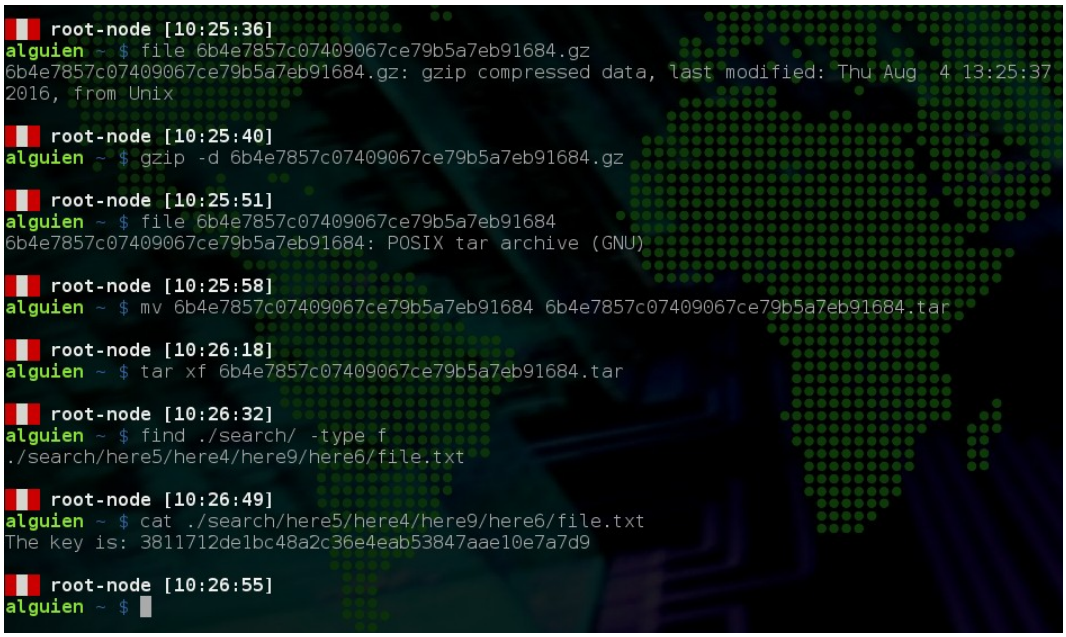
. Análisis .

El reto nos presenta un archivo comprimido con “gzip” y en su interior otro empaquetado con “tar”. El empaquetado resulta ser un directorio “search” y en su interior una infinidad de subdirectorios anidados. En alguno de estos hay un archivo con el flag y debemos encontrarlo.

. Solución .

Empleamos el comando "find" con el filtro "-type f" para buscar unicamente ficheros dentro del directorio "search".

```
find ./search/ -type f
```



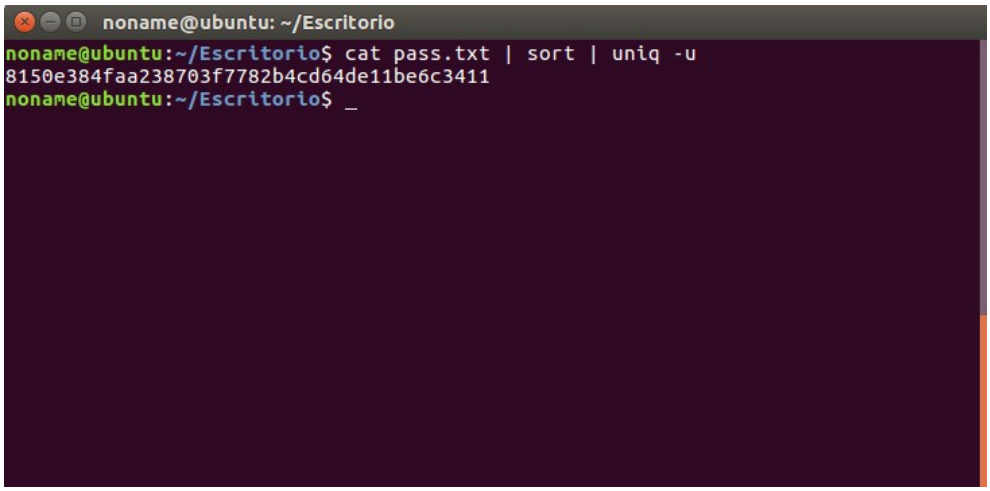
El flag es: 3811712de1bc48a2c36e4eab53847aae10e7a7d9

. Misc 20 – Unico .

. Descripción .

. Análisis .

El archivo en si eran un montón de hashes donde solo uno era diferente. Bastaba con ejecutar en consola:



El flag era: 8150e384faa238703f7782b4cd64de11be6c3411

. Misc 30 – Misc .

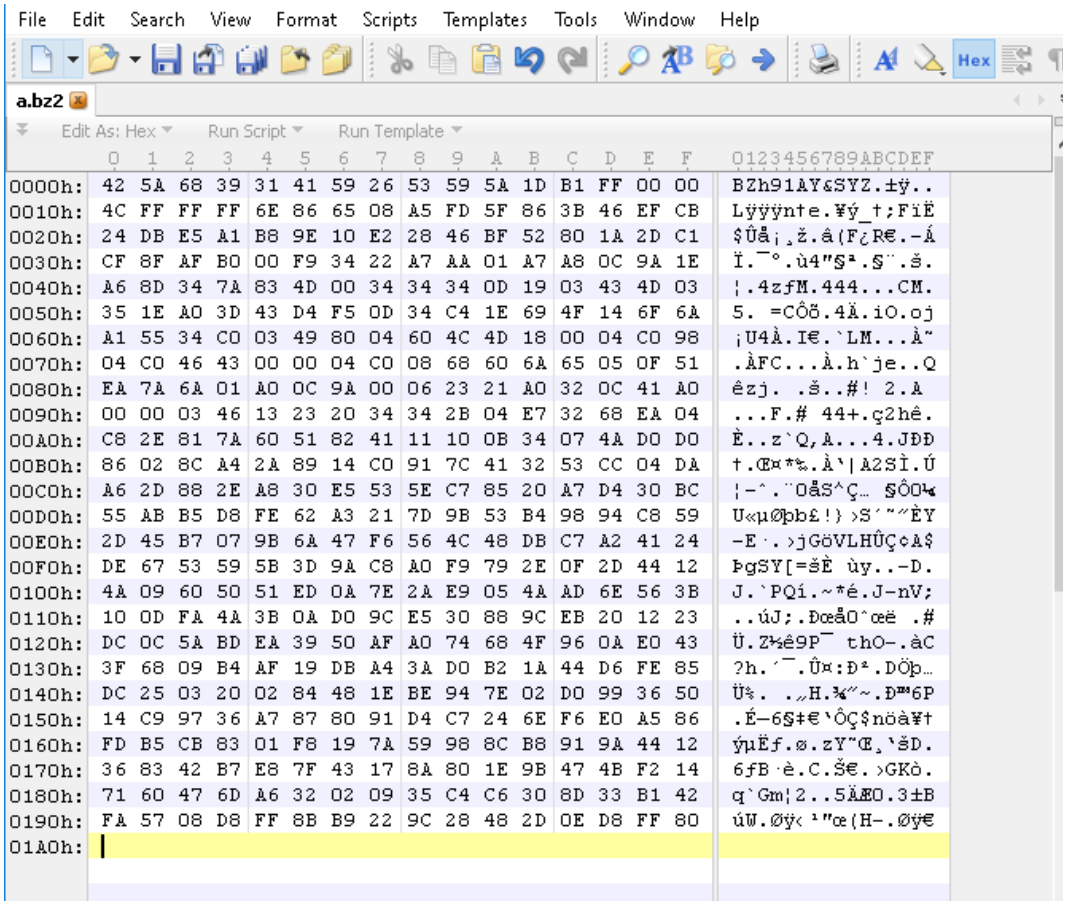
. Descripción .

. Análisis .

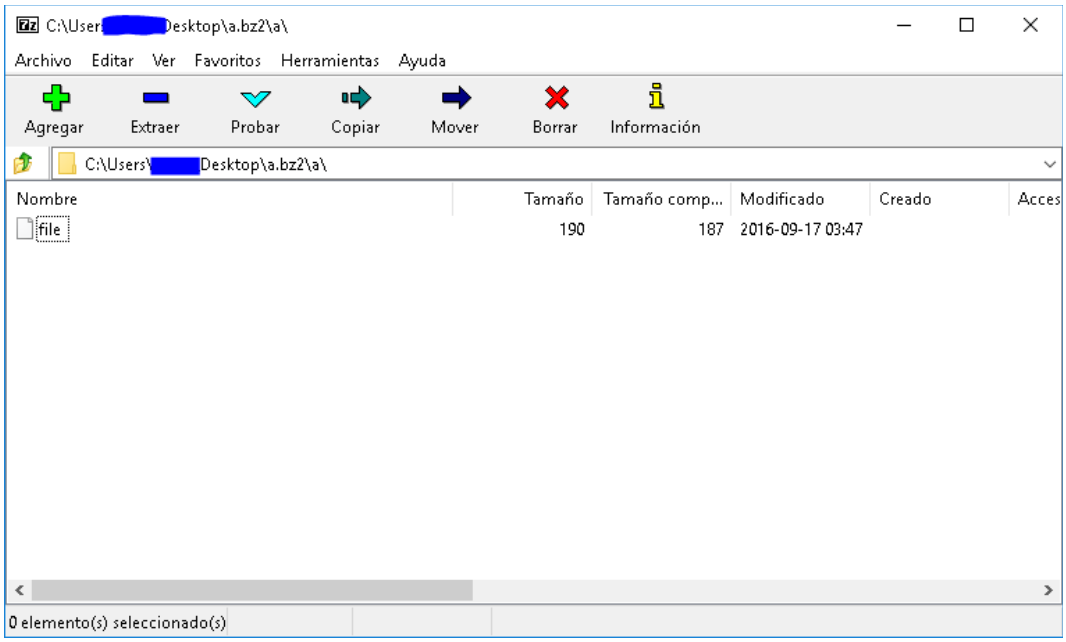
Con un simple vistazo con un editor de texto vemos que el archivo que se nos proporciona es un "dump" o extracción de un archivo desconocido analizado con un editor hexadecimal.

Podemos observar que los primeros bytes de dicho archivo comienzan por 42 5A 68 ó "BZh", característica propia de los archivos bz2.

Así pues con un editor hexadecimal creamos un archivo utilizando todos los bytes:



y lo abrimos utilizando por ejemplo 7zip.



Nuevamente, mirando las cabeceras del nuevo (1F 8B 08), vemos que es un archivo comprimido .gz asique lo volvemos a abrir con 7zip.

Esta operación se repite un par de veces más con archivos tipo .tar hasta que obtenemos, por fin, un archivo en el que se puede leer:

"the key is: 13ba34cafa454fd9c110b8d131cec4b6ffd7ec93"

The End - Amn3s1a Team