Writeup - UAM - Matrix - Episode 1

Julian J. M. 16/03/2019

Nos facilitan un binario con la que obtenemos un código y una web donde introducir ese código para obtener la flag, o simplemente avanzar en el reto.

Analizando el código vemos que todo se realiza en su función main. Va haciendo una serie de comparaciones (4 en total), y si falla te echa fuera. Al llegar al final (sin hacer trampas), te proporciona el código. Digo sin hacer trampas porque si atajamos la clave que nos facilitará no será válida.

Hay dos formas de enfocar el reto:

1) Analizar el código, y resolver los 4 sistemas de ecuaciones resultantes.
2) Usar angr, indicarle a dónde queremos llegar, y esperar que haga su magia.

En este caso las 2 son factibles, por la longitud de la clave (32 bits). En caso de números más grandes, solo nos queda la primera opción, si queremos sacar el reto este lustro. En este writup nos centraremos en la primera.

Partimos de un array de bytes “24372347142643281943534639”. Justo después de pedirnos la key, llama a un par de funciones que utilizan la función ptrace. El resultado de estas dos llamadas modifican las dos primeras posiciones de array anterior.

Usando un poco de magia (IDA Decompiler), vemos lo que hace la función:

```
1 char __cdecl comprobacion_ptrace()
2 {
3   char result; // al
4
5   if ( ptrace(0, 0LL, 1LL, 0LL) >= 0 )
6     result = '3';
7   else
8     result = '1';
9   return result;
10 }
```

Básicamente llama a ptrace(PTRACE_TRACEME, ….). Esta función pone el proceso actual en modo debug. Si ya está siendo debugueado, devuelve un número negativo. Cuando estamos analizando con IDA, ya está en modo debug, y por eso siempre devuelve ‘1’. Ahora bien, en condiciones normales, la primera llamada devuelve un número positivo, así que devolvería ‘3’, pero la siguiente sería ‘1’.

```
data[1] = comprobacion_ptrace();
data[0] = comprobacion_ptrace();
```

Es decir, el array de bytes, en condiciones normales, pasaría a ser:

13372347142643281943534639

Continuamos con las otras dos funciones. La primera de ellas devuelve un número, convirtiendo parte de una cadena de caracteres a entero. Recibe 3 parámetros, una secuencia de números en ASCII, inicio y longitud. Vemos que va acumulando `(data[i]-48) * 10^posicion`, siendo i la variable del bucle. posición es 0 para las unidades, 1 las decenas, etc.

```
1  int __fastcall ATOI_substring(char *data, int inicio, int longitud)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    acum = 0;
6    v7 = inicio + longitud;
7    for ( i = inicio; i < v7; ++i )
8    {
9      val = (data[i] - '0');
10     acum = (pow(10.0, (v7 - i - 1)) * val + acum);
11   }
12   return acum;
13 }
```

Por ejemplo ATOI_substring(“12345678”, 4, 3) devolvería el entero 567. Empezando en la posición 4, usa los siguientes 3 caracteres.

La otra función calcula el módulo de un número con otro. Sería equivalente (si ambos números fuesen positivos) a:

result = a % b

La función hace un par de comprobaciones para devolver siempre un módulo positivo, aunque cualquiera de los operandos puedan ser negativos:

```
1  int __fastcall modulo_p1_p2(signed int p1, int p2)
2  {
3    int result; // [rsp+1Ch] [rbp-4h]
4
5    if ( p2 < 0 )
6      return modulo_p1_p2(p1, -p2);
7    result = p1 % p2;
8    if ( p1 % p2 < 0 )
9      result += p2;
10   return result;
11 }
```

Una vez tenemos las piezas del puzle, vemos que las comprobaciones que realiza el programa son las siguientes. Buscamos una clave X, tal que:

```
42486    = X % 133723
6293904  = X % 47142643
234      = X % 2819
31311341 = X % 43534639
```

Este tipo de ecuaciones se pueden resolver por fuerza bruta… iteramos X para que cumpla una de las ecuaciones, y vamos comprobando si el resto también las cumple. Para números pequeños, aquí hablamos de 32bits, se puede llegar a hacer.

Para números grandes, se hace uso del Teorema del Resto Chino (filipino).
https://es.wikipedia.org/wiki/Teorema_chino_del_resto

Haciendo uso de la siguiente página: https://www.dcode.fr/chinese-remainder obtenemos que nuestra clave es 902004121.

```
$ ./getcode
Insert the correct key to get unlock code:
902004121
Correct key!
Here is your unlock code: 943589633
```

Al introducir ese código en la web, nos facilitan un enlace de Google Drive con una captura de red (pcap) y un fichero zip con contraseña.

Vemos mucho tráfico (really), pero nos llama la atención una llamada telefónica SIP. Usando wireshark extraemos el audio y escuchamos a Morfeo dándole la chapa al pobre Neo con algo de unas pastillas y un viaje. Haciendo un poco de guessing, y suponiendo que la clave está relacionada con la llamada creamos un diccionario parecido a esto:

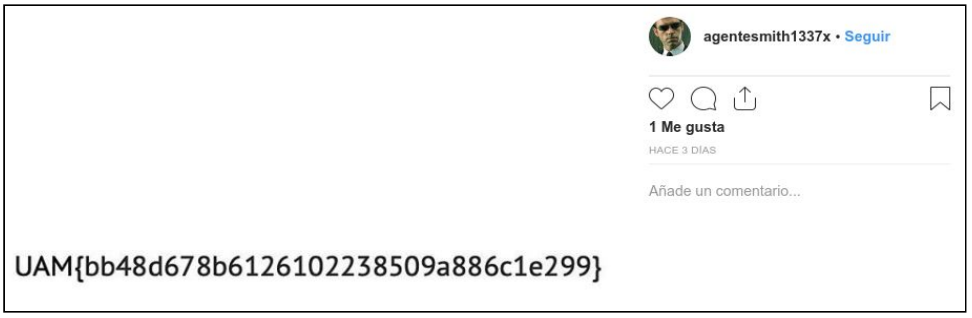
```
pastilla
roja
azul
pastilla roja
pastilla azul
madriguera
conejo
```

Y dejamos que John the Ripp3r haga el trabajo sucio de ir probando combinaciones de mayúsculas minúsculas, añadir números, etc.

```
$ zip2john Usuario.zip > Usuario.hashes
$ john Usuario.hashes -w=dict -rules=dive
```

En pocos segundos tenemos que la clave del zip es “`pastillaroja`”, todo junto.

Tras descomprimir el zip, vemos que el contenido de flaguser.txt es “@agentesmith1337x”. Haciendo un poco de OSINT llegamos a la red social Instagram, donde vemos la flag:



Julian J. M.
Telegram: @julianjm
Email: julianjm@gmail.com