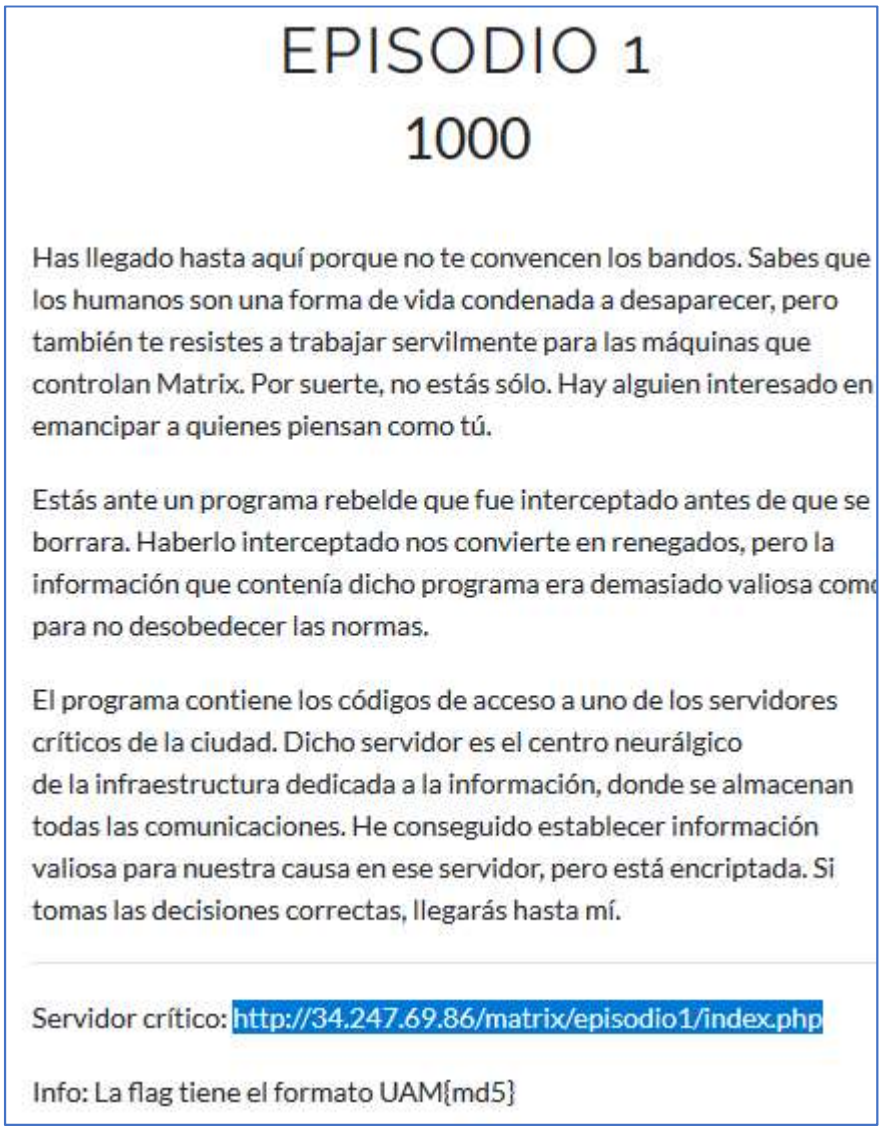
20190315 – Reto UAM. Matrix – Episodio 1



EPISODIO 1

1000

Has llegado hasta aquí porque no te convencen los bandos. Sabes que los humanos son una forma de vida condenada a desaparecer, pero también te resistes a trabajar servilmente para las máquinas que controlan Matrix. Por suerte, no estás sólo. Hay alguien interesado en emancipar a quienes piensan como tú.

Estás ante un programa rebelde que fue interceptado antes de que se borrara. Haberlo interceptado nos convierte en renegados, pero la información que contenía dicho programa era demasiado valiosa como para no desobedecer las normas.

El programa contiene los códigos de acceso a uno de los servidores críticos de la ciudad. Dicho servidor es el centro neurálgico de la infraestructura dedicada a la información, donde se almacenan todas las comunicaciones. He conseguido establecer información valiosa para nuestra causa en ese servidor, pero está encriptada. Si tomas las decisiones correctas, llegarás hasta mí.

Servidor crítico: http://34.247.69.86/matrix/episodio1/index.php

Info: La flag tiene el formato UAM{md5}

Descargamos el fichero getCode y Kali nos dice que es un binario de Linux 64 bits:

```
nacho@kali:~/UAM/201903-Matrix-Episodio1$ file getcode
getcode: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib
ildID[sha1]=fa966968762bff2e58360b56ac30754d91ebaa25, stripped
```

Lo abrimos con IDA y lo vamos analizando. Vemos como al principio va cargando unas constantes y metiéndolas en memoria:

```
00007FFD30075FC0  32 34 33 37 32 33 34 37  31 34 32 36 34 33 32 38  2437234714264328
00007FFD30075FD0  31 39 34 33 35 33 34 36  33 39 00 F9 76 55 00 00  1943534639..vU..
```

Después ejecuta dos llamadas a una función, que a su vez invoca a “ptrace”. “ptrace” es una función usada para hacer attach a un proceso existente, y en algunos programas se utiliza como técnica antidebugging, para alterar el comportamiento del programa en caso de estar ejecutándose bajo GDB o similar.

La idea se basa en que “ptrace” solo se puede ejecutar una vez por un proceso, debido a que no podemos hacer “attach” a más de un proceso. Y si el proceso está bajo GDB ya lo usó éste para debugear el proceso y cargarlo, por tanto la función dará un error:

The general idea is that debuggers, such as gdb, utilize the `ptrace()` function to attach to a process at runtime. Because only one process is allowed to do this at a time, having a call to `ptrace()` in your code can be used as an anti-debugging technique.

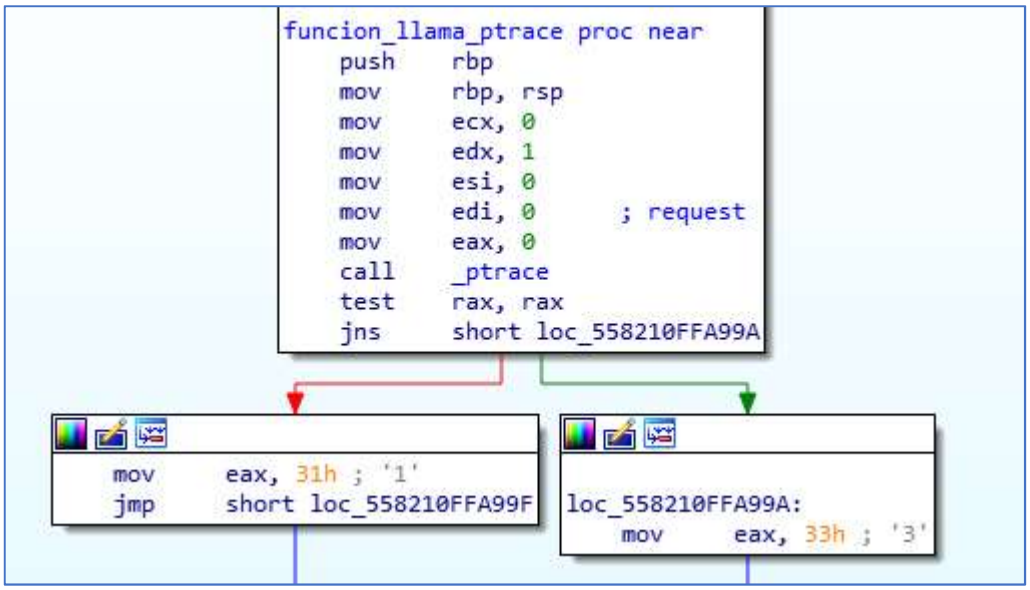
```
#include <stdio.h>
#include <sys/ptrace.h>

int main()
{
    if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
    {
        printf("don't trace me !!\n");
        return 1;
    }
    // normal execution
    return 0;
}
```

`ptrace()` can be detected by the fact that an executable can only call `ptrace()` once. if `ptrace()` was already called by the `strace` executable, we can detect it in runtime:

Al ejecutar el programa bajo un strace, o GDB, la llamada daría error y devolvería el valor “-1”:

```
$ strace -i ./e7bc5d2c0cf4480348f5504196561297 1 2
[00007fb6d7559ae7] execve("./e7bc5d2c0cf4480348f5504196561297", ["./e7bc5d2c0cf4480348f550419656
"2"], [/* 18 vars */]) = 0
[00000000004a9297] uname({sys="Linux", node="debian", ...}) = 0
[00000000004aa78a] brk(0)               = 0x148d000
[00000000004aa78a] brk(0x148e1c0)       = 0x148e1c0
[000000000045e3f5] arch_prctl(ARCH_SET_FS, 0x148d880) = 0
[00000000004aa78a] brk(0x14af1c0)       = 0x14af1c0
[00000000004aa78a] brk(0x14b0000)       = 0x14b0000
[000000000047431b] ptrace(PTRACE_TRACEME, 0, 0x1, 0) = -1 EPERM (Operation not permitted)
```

En el programa nuestro, vemos como hace la llamada, y en función de si devuelve 0 (OK) o -1 (KO) devolverá un “1” o un “3”.



Y este valor devuelto, irá directamente a la zona de memoria donde teníamos antes las constantes:

```
call    funcion_llama_ptrace
mov     byte ptr [rbp+var_valor_constante_tras_ptrace+1], al
mov     eax, 0
call    funcion_llama_ptrace
mov     byte ptr [rbp+var_valor_constante_tras_ptrace], al
```

Por tanto, los dos primeros valores que vimos pueden cambiar a “1” o “3” según como estemos ejecutando el programa.

Ahora analizamos el comportamiento del programa. Vemos que primero nos pide un valor por teclado, y después ejecuta la misma funcionalidad en cuatro pasos consecutivos, que se resumen en:

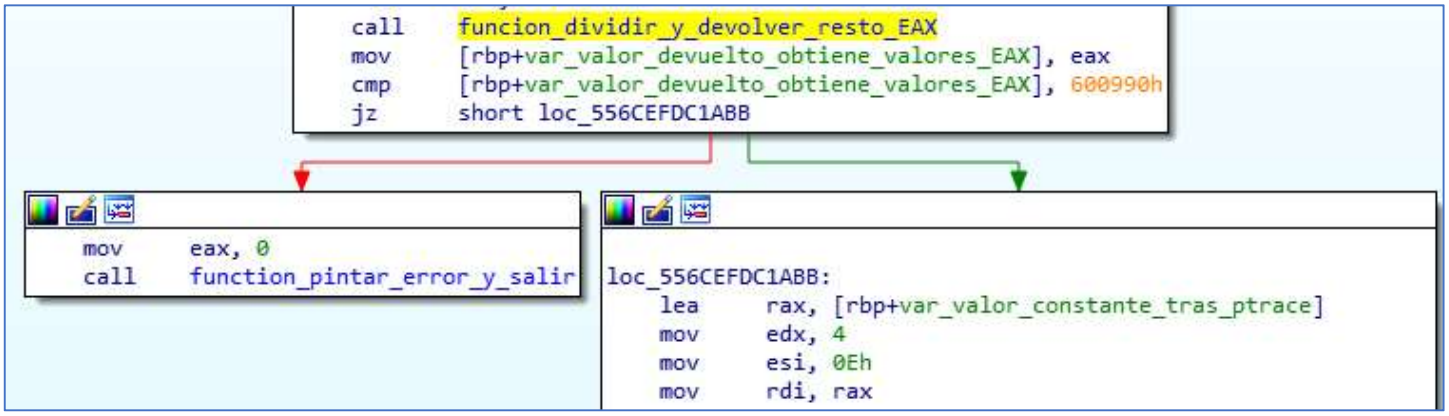
- Obtiene unas posiciones concretas de la variable de constantes inicial (en este caso de ejemplo, 8 posiciones a partir del índice 6):

```
lea     rax, [rbp+var_valor_constante_tras_ptrace]
mov     edx, 8
mov     esi, 6
mov     rdi, rax
call    funcion_obtiene_valores_constante_EAX
```

- Después divide el valor que metimos por teclado, por el valor obtenido en la función anterior (trozo de la variable de constantes). Y se queda con el valor del resto (EDX) que devuelve la función:

```
mov     eax, [rbp+param_valor_introducido_teclado]
cdq
idiv    [rbp+param_valor_porcion_constantes]
mov     [rbp+var_resto_division], edx ; EDX Resto division. EAX Cociente division
```

- Por último, devuelve el resto y lo compara con una constante. Si acierta, sigue. Si falla, sale del programa:



Este comportamiento lo ejecuta cuatro veces, dividiendo cada vez el valor que metimos por teclado, por unas posiciones concretas de la variable de constante, y comparando en cada caso el resto con una constante distinta. Los valores en cada caso serán:

```
Divisiones:
  - (6 pos) Cociente 1BC3B         Resto A5F6
  - (8 pos) Cociente 2CF56F3       Resto 600990
  - (4 pos) Cociente B03           Resto EA
  - (8 pos) Cociente 298492F       Resto 1DDC5EDh
```

Por tanto, tenemos que encontrar un número para meter por teclado, que sus divisiones por cada cociente, nos den el resto adecuado, respectivamente. Igual hay una forma “matemática” de encontrar este valor, pero a mí no se me ocurre otra manera que hacerme un programilla Java que pruebe combinaciones y me lo encuentre.

Para evitar millones de combinaciones, parto de la premisa de que el numero elegido debe ser, al menos, tan grande con el resto más grande (1DDC5DEh), y además deberá cumplir que sea compatible con ese mismo número, más el propio cociente 298492F * n. Por tanto, este será el valor que vayamos buscando en el bucle, y lo que haremos será validar si además es compatible con las otras tres divisiones por cociente1, cociente2 y cociente3.

Y también debemos traducir todos los valores Hexadecimal que vimos antes, a sus correspondientes Decimales, ya que el programa obtendrá el valor por teclado en decimal.

Por tanto, programamos lo siguiente:

```java
public class CalculaUAM201903MatrixEpisodio1 {

    long coc1 = 133723, coc2 = 47142643, coc3 = 2819, coc4 = 43534639;
    long resto1 = 42486, resto2 = 6293904, resto3 = 234, resto4 = 31311341;
    long resultado = -1;

    public CalculaUAM201903MatrixEpisodio1() {}

    public void calcula()    {

        for (long a = resto4; a < resto4*100000L; a+=coc4)    {
            if (a % coc1 != resto1)   continue;
            if (a % coc2 != resto2)   continue;
            if (a % coc3 != resto3)   continue;
            resultado = a;
            break;
        }


    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        CalculaUAM201903MatrixEpisodio1 calc = new CalculaUAM201903MatrixEpisodio1();
        calc.calcula();

        System.err.println("Resultado: " + calc.resultado);
    }
```

Es importante el valor del cociente1 (coc1), que son los 6 primeros dígitos de la cadena de constantes inicial. Modifico el valor sustituyendo los dos primeros caracteres por “13”, que son resultado de aplicar la primera llamada al “ptrace” que va a funcionar bien y por tanto modifica la posición 2 por un “3”, y la segunda llamada “ptrace” que NO va a funcionar OK y por tanto va a cambiar la posición 1 por un “1”:

```
call    funcion_llama_ptrace
mov     byte ptr [rbp+var_valor_constante_tras_ptrace+1], al
mov     eax, 0
call    funcion_llama_ptrace
mov     byte ptr [rbp+var_valor_constante_tras_ptrace], al
```

Por tanto, los valores de cocientes serán:

```
long coc1 = 133723, coc2 = 47142643, coc3 = 2819, coc4 = 43534639;
long resto1 = 42486, resto2 = 6293904, resto3 = 234, resto4 = 31311341;
long resultado = -1;
```

Lo ejecutamos, y enseguida nos saca un resultado compatible.

```
Console  Tasks
<terminated> CalculaUAM201903MatrixEpisodio1 [Java Application] C:\Program Files\Java\jdk1.8.0_181\bin\javaw.exe (16 mar. 2019 12:31:16
Resultado: 902004121
```

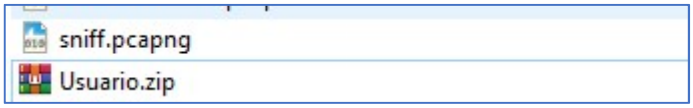
Así que, metemos este valor al ejecutar el programa y nos da el Code:

```
nacho@kali:~/UAM/201903-Matrix-Episodio1$ getcode
Insert the correct key to get unlock code:
902004121
Correct key!
Here is your unlock code: 943589633
```

Ahora ya el code lo metemos en la página que nos ponía el reto, y nos saca un enlace para descargar:

Descarga de comunicaciones:
https://drive.google.com/open?id=1CmHuHxlPXJz5uqz6KyhYblAcwiu14jfz

Lo descargamos, y nos saca dos ficheros, un pcapng de tramas de red, y un fichero ZIP protegido por contraseña, que dentro parece que lleva el fichero con la flag:

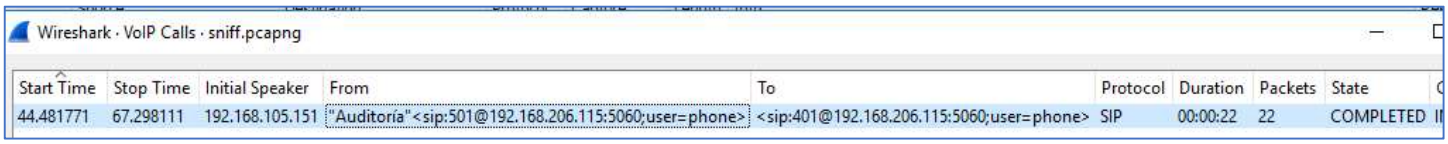


Abrimos el fichero pcapng, y lo vamos analizando. Lleva muchas tramas de trafico SSL, y SSH, que al estar encriptado no vamos a poder encontrar nada.

Dentro de los protocolos no cifrados, encontramos RTP y SIP, que sirven para la comunicación por VoIP. Mediante SIP establecemos la comunicación entre un terminal y el otro, y con RTP enviamos la información de Voz.

5580	44.514015417	192.168.105.151	192.168.206.115	SIP/SDP	5580	1249 Request: INVITE sip:401@192.168.206.115:5060;user=phone |
5581	44.517590862	192.168.105.151	192.168.206.115	SIP	5581	548 Request: ACK sip:401@192.168.206.115:5060;user=phone |
5582	44.517769084	192.168.105.151	192.168.206.115	SIP/SDP	5582	1249 Request: INVITE sip:401@192.168.206.115:5060;user=phone |
5583	44.519536734	192.168.206.115	192.168.105.151	SIP	5583	619 Status: 100 Trying |
5584	44.525606924	192.168.206.115	192.168.105.151	SIP	5584	619 Status: 100 Trying |
5598	44.783504762	192.168.206.115	192.168.105.151	SIP	5598	635 Status: 180 Ringing |
5599	44.789590861	192.168.206.115	192.168.105.151	SIP	5599	635 Status: 180 Ringing |
5600	44.804209547	192.168.206.115	192.168.105.151	SIP	5600	635 Status: 180 Ringing |
5601	44.805583362	192.168.206.115	192.168.105.151	SIP	5601	635 Status: 180 Ringing |
5704	45.456175359	192.168.206.115	192.168.105.151	SIP/SDP	5704	960 Status: 200 OK |
5705	45.461597352	192.168.206.115	192.168.105.151	SIP/SDP	5705	960 Status: 200 OK |
5706	45.469734168	192.168.105.151	192.168.206.115	SIP	5706	728 Request: ACK sip:401@192.168.206.115:5060;user=phone |
5707	45.477592874	192.168.105.151	192.168.206.115	SIP	5707	728 Request: ACK sip:401@192.168.206.115:5060;user=phone |
5716	45.596784218	192.168.206.115	192.168.105.151	RTP	5716	214 PT=ITU-T G.711 PCMU, SSRC=0x1E9D509A, Seq=45881, Time=24609
5717	45.597706862	192.168.206.115	192.168.105.151	RTP	5717	214 PT=ITU-T G.711 PCMU, SSRC=0x1E9D509A, Seq=45881, Time=24609
5718	45.615852362	192.168.206.115	192.168.105.151	RTP	5718	214 PT=ITU-T G.711 PCMU, SSRC=0x1E9D509A, Seq=45882, Time=24609
5722	45.621564903	192.168.206.115	192.168.105.151	RTP	5722	214 PT=ITU-T G.711 PCMU, SSRC=0x1E9D509A, Seq=45882, Time=24609
5723	45.639970680	192.168.206.115	192.168.105.151	RTP	5723	214 PT=ITU-T G.711 PCMU, SSRC=0x1E9D509A, Seq=45883, Time=24609
5725	45.645646512	192.168.206.115	192.168.105.151	RTP	5725	214 PT=ITU-T G.711 PCMU, SSRC=0x1E9D509A, Seq=45883, Time=24609

Con WireShark podemos acceder directamente a la información de voz, y escucharla:



Podemos acceder al audio y escucharlo con la opción “Play Stream”. Escuchamos un fragmento de la película Matrix, donde Morfeo ofrece dos opciones relacionadas con la pastilla Azul o Roja.

Probamos distintas contraseñas del ZIP relacionadas con el audio, y con la password “pastillaroja” el ZIP se descomprime. Dentro del fichero de texto está esto:

Parece un usuario de redes sociales, vamos buscando en Twitter, Instagram, y en esta última el usuario existe:

Buscamos en sus publicaciones, solo tiene una, y sale la FLAG!:

```
UAM{bb48d678b6126102238509a886c1e299}
```

José Ignacio de Miguel González:

User UAM: nachinho3

Telegram: @jignaciodemiguel