EPISODIO 1
927

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}

TOP 3:

1. oreos
2. DarkEagle
3. julianjm

getcode

Descargamos el binario getcode y lo analizamos brevemente.
Vemos que es un binario que al ejecutarlo nos pide una key:

root@kali:~/unaalmes/matrix/EP1# ./getcode
Insert the correct key to get unlock code:
1234
Incorrect key!

Visto esto, es un candidato excelente para probar angr: https://github.com/angr
Lo único que necesitamos para hacer funcionar angr es la dirección de memoria a la cual queremos llegar, esto lo conseguiremos con, por ejemplo, r2:

Haciendo un pdf en la función main vemos:

0x00000b5c 488d3df50000. lea rdi, qword str.Correct_key___Here_is_your_unlock_code:__d ; 0xc58 ; "Correct key!\nHere is your unlock code: %d\n"

Como decíamos anteriormente, aquí nos interesa la dirección de memoria 0x00000b5c (donde nos muestra el unlock code que queremos). Angr nos dice que el ejecutable está compilado con PIE y que debemos sumarle **0x400000**, así que la dirección queda: 0x00400b5c

Código utilizado python3 usando angr:

############################ BOF ########################

```
import angr
import sys

def main(argv):
  # Create an Angr project.
  # If you want to be able to point to the binary from the command line, you can
  # use argv[1] as the parameter. Then, you can run the script from the command
  # line as follows:
  # python ./scaffold00.py [binary]
  # (!)
  path_to_binary = "/root/unaalmes/matrix/EP1/getcode"  # :string
  project = angr.Project(path_to_binary)

  # Tell Angr where to start executing (should it start from the main()
  # function or somewhere else?) For now, use the entry_state function
  # to instruct Angr to start from the main() function.
  initial_state = project.factory.entry_state()

  # Create a simulation manager initialized with the starting state. It provides
```

```
  # a number of useful tools to search and execute the binary.
  simulation = project.factory.simgr(initial_state)

  # Explore the binary to attempt to find the address that prints "Good Job."
  # You will have to find the address you want to find and insert it here.
  # This function will keep executing until it either finds a solution or it
  # has explored every possible path through the executable.
  # (!)
  print_good_address = 0x00400b5c  # :integer (probably in hexadecimal)
  simulation.explore(find=print_good_address)

  # Check that we have found a solution. The simulation.explore() method will
  # set simulation.found to a list of the states that it could find that reach
  # the instruction we asked it to search for. Remember, in Python, if a list
  # is empty, it will be evaluated as false, otherwise true.
  if simulation.found:
    # The explore method stops after it finds a single state that arrives at the
    # target address.
    solution_state = simulation.found[0]

    # Print the string that Angr wrote to stdin to follow solution_state. This
    # is our solution.
    print (solution_state.posix.dumps(sys.stdin.fileno()))
  else:
    # If Angr could not find a path that reaches print_good_address, throw an
    # error. Perhaps you mistyped the print_good_address?
    raise Exception('Could not find the solution')

if __name__ == '__main__':
    main(sys.argv)
```

############################## EOF ##########################

root@kali:~/unaalmes/matrix/EP1# time python3 test_angr.py
WARNING | 2019-03-18 14:39:30,403 | cle.loader | The main binary is a position-independent executable. It is being loaded with a base address of 0x400000.
WARNING | 2019-03-18 14:39:31,229 | angr.state_plugins.symbolic_memory | Register rbx has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:32,185 | angr.state_plugins.symbolic_memory | Register rbx has an unspecified value; Generating an unconstrained value of 8 bytes.

WARNING | 2019-03-18 14:39:32,516 | angr.state_plugins.symbolic_memory | Register r15 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:32,518 | angr.state_plugins.symbolic_memory | Register r14 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:32,520 | angr.state_plugins.symbolic_memory | Register r13 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:32,522 | angr.state_plugins.symbolic_memory | Register r12 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:32,526 | angr.state_plugins.symbolic_memory | Register rbx has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:32,577 | angr.state_plugins.symbolic_memory | Register cc_ndep has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,382 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,407 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,442 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,469 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,511 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,545 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,572 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,611 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
WARNING | 2019-03-18 14:39:34,647 | angr.state_plugins.symbolic_memory | Register 232 has an unspecified value; Generating an unconstrained value of 8 bytes.
b'0902004121'

real 1m59,042s
user 1m57,880s
sys 0m0,322s

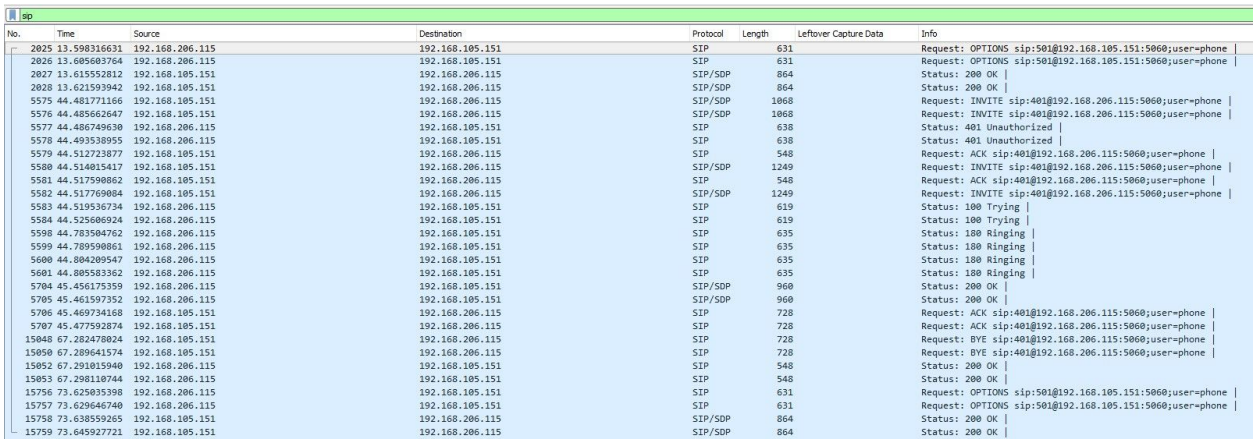
En menos de 2 minutos tenemos el resultado 0902004121, así que, vamos a probarlo:

root@kali:~/unaalmes/matrix/EP1# ./getcode
Insert the correct key to get unlock code:
0902004121
Correct key!
Here is your unlock code: 943589633

Introducimos el unlock code en la web que nos han dado:

Nos descargamos el fichero de google drive **https://drive.google.com/open?id=1CmHuHxlPXJz5uqz6KyhYblAcwiu14jfz** el cual es una captura de tráfico de red y un fichero zip con contraseña.

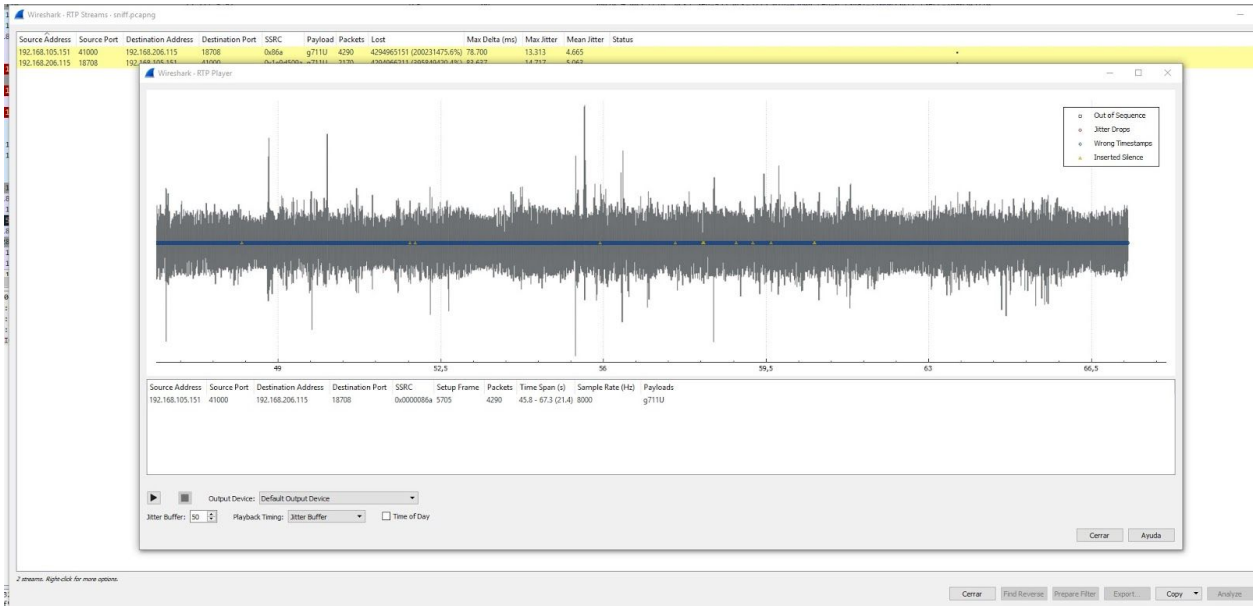
Analizando el tráfico nos llama la atención que hay una llamada sip



sip

No.	Time	Source	Destination	Protocol	Length	Leftover Capture Data	Info
2025	13.598316631	192.168.206.115	192.168.105.151	SIP	631		Request: OPTIONS sip:501@192.168.105.151:5060;user=phone |
2026	13.605603764	192.168.206.115	192.168.105.151	SIP	631		Request: OPTIONS sip:501@192.168.105.151:5060;user=phone |
2027	13.615552812	192.168.105.151	192.168.206.115	SIP/SDP	864		Status: 200 OK |
2028	13.621593942	192.168.105.151	192.168.206.115	SIP/SDP	864		Status: 200 OK |
5575	44.481771166	192.168.105.151	192.168.206.115	SIP/SDP	1068		Request: INVITE sip:401@192.168.206.115:5060;user=phone |
5576	44.485662647	192.168.105.151	192.168.206.115	SIP/SDP	1068		Request: INVITE sip:401@192.168.206.115:5060;user=phone |
5577	44.486749630	192.168.206.115	192.168.105.151	SIP	638		Status: 401 Unauthorized |
5578	44.493538955	192.168.206.115	192.168.105.151	SIP	638		Status: 401 Unauthorized |
5579	44.512723877	192.168.105.151	192.168.206.115	SIP	548		Request: ACK sip:401@192.168.206.115:5060;user=phone |
5580	44.514015417	192.168.105.151	192.168.206.115	SIP/SDP	1249		Request: INVITE sip:401@192.168.206.115:5060;user=phone |
5581	44.517590862	192.168.105.151	192.168.206.115	SIP	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	1249		Request: INVITE sip:401@192.168.206.115:5060;user=phone |
5583	44.519536734	192.168.206.115	192.168.105.151	SIP	619		Status: 100 Trying |
5584	44.525606924	192.168.206.115	192.168.105.151	SIP	619		Status: 100 Trying |
5598	44.783504762	192.168.206.115	192.168.105.151	SIP	635		Status: 180 Ringing |
5599	44.789590861	192.168.206.115	192.168.105.151	SIP	635		Status: 180 Ringing |
5600	44.804209547	192.168.206.115	192.168.105.151	SIP	635		Status: 180 Ringing |
5601	44.805583362	192.168.206.115	192.168.105.151	SIP	635		Status: 180 Ringing |
5704	45.456175359	192.168.206.115	192.168.105.151	SIP/SDP	960		Status: 200 OK |
5705	45.461597352	192.168.206.115	192.168.105.151	SIP/SDP	960		Status: 200 OK |
5706	45.469734168	192.168.105.151	192.168.206.115	SIP	728		Request: ACK sip:401@192.168.206.115:5060;user=phone |
5707	45.477592874	192.168.105.151	192.168.206.115	SIP	728		Request: ACK sip:401@192.168.206.115:5060;user=phone |
15048	67.282478024	192.168.105.151	192.168.206.115	SIP	728		Request: BYE sip:401@192.168.206.115:5060;user=phone |
15050	67.289641574	192.168.105.151	192.168.206.115	SIP	728		Request: BYE sip:401@192.168.206.115:5060;user=phone |
15052	67.291015940	192.168.206.115	192.168.105.151	SIP	548		Status: 200 OK |
15053	67.298110744	192.168.206.115	192.168.105.151	SIP	548		Status: 200 OK |
15756	73.625035398	192.168.206.115	192.168.105.151	SIP	631		Request: OPTIONS sip:501@192.168.105.151:5060;user=phone |
15757	73.629646740	192.168.206.115	192.168.105.151	SIP	631		Request: OPTIONS sip:501@192.168.105.151:5060;user=phone |
15758	73.638559265	192.168.105.151	192.168.206.115	SIP/SDP	864		Status: 200 OK |
15759	73.645927721	192.168.105.151	192.168.206.115	SIP/SDP	864		Status: 200 OK |

Vamos a escuchar la llamada, para hacerlo, en Wireshark podemos ir a: Telephony→ RTP → RTP Streams

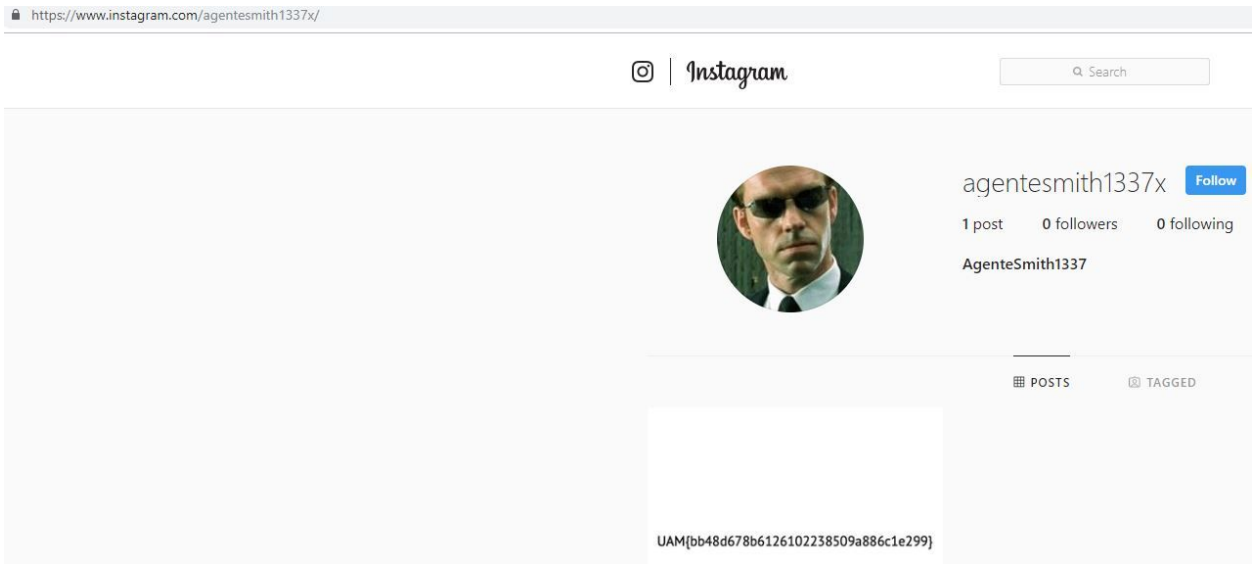


El audio que se escucha en esta conversación podemos encontrarlo también en youtube: https://www.youtube.com/watch?v=wAQCW63_VNQ donde Morpheos le da a escoger a Neo entre la pastilla azul y la roja. La contraseña era: **pastillaroja**

root@kali:~/unaalmes/matrix/EP1# unzip Usuario.zip
Archive: Usuario.zip
[Usuario.zip] flaguser.txt password:
 extracting: flaguser.txt
root@kali:~/unaalmes/matrix/EP1# cat flaguser.txt
@agentesmith1337x

Después de complicarme la vida con que podía ser el contenido de flaguser.txt .. tiramos de OSINT con la web: https://namechk.com introduciendo el nombre agentesmith1337x

Como vemos según la leyenda en Instagram ese nombre se está utilizando, así que vamos a ver que hay:



Y aquí tenemos la preciada flag:

Flag: UAM{bb48d678b6126102238509a886c1e299}

Found : **Era_inevitable_señor_Anderson**
(hash = bb48d678b6126102238509a886c1e299)

DarkEagle