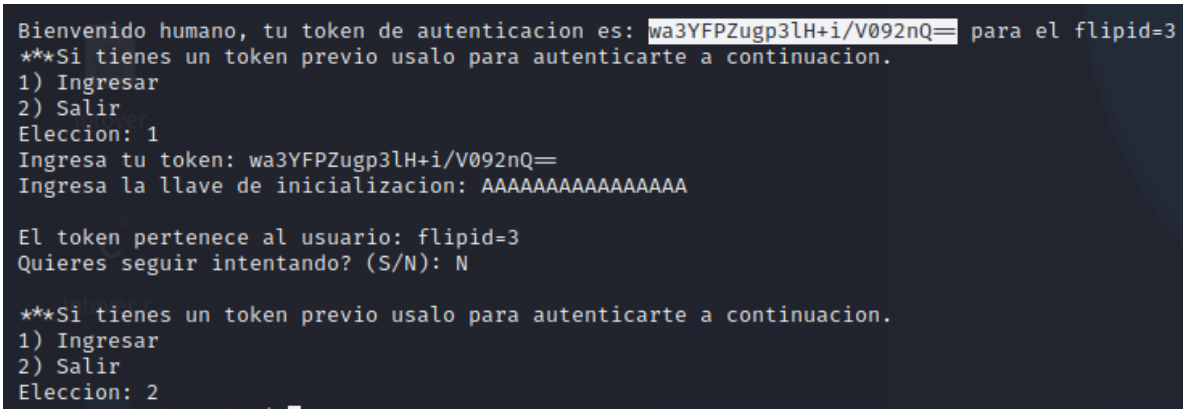
Reto: Padding CBC*.

Descripción:

Por alguna razón a todo nuevo usuario se le entrega, una llave "AAAAAAAAAAAAAAAA" a la que se le dice quizás incorrectamente "llave de inicialización", se cuenta con capacidad para 6 IDs de usuarios invitados (0-5), siendo el usuario con id 6 el administrador del sistema y con esto el que te dará la flag si te haces pasar por el ;-)

Ejecutando de forma normal el código nos muestra lo siguiente:

Probando el token proporcionado con el vector de inicialización dado como A*16 , el servidor arroja el flipid=3:



Lo que se busca es obtener el flipid=6 que da acceso a la cuenta de administrador y por lo tanto la bandera.

Si se decodifica el token que aparentemente esté en base64 nos devuelve lo siguiente:

```
Á.Ø.ön..å.è¿WOv.
```

Que probablemente será el cifrado CBC, así que se analiza el código fuente.

Se crea una llave de 16 caracteres y el vector de inicialización de A*16. También se crea el mensaje con flipid=[0-4].

```
key = gen_random_key(16)
iv = "\x41"*16
self.request.send(bytes(flipper,"utf-8"))
mensaje = "flipid=" + str(random.choice(range(100)) % 5)
```

Este mensaje es cifrado y lo muestra como token:

```
tu token de autenticacion es: %s para el %s " % (str(delfin_encrypt(iv,key,mensaje),"utf-8"),mensaje),"utf-8"))
```

Durante la petición del cliente, se puede ver que solicita la llave de inicialización con longitud mínima de 16.

Llama a la función delfin_decrypt con la llave (IV) que la recorta hasta 16 caracteres, la llave generada aleatoriamente y el token.

Si regresa flipid=6 lanza el mensaje Felicidades: flag.

```
if opcion == 1:
    while True:
        self.request.send(b"Ingresa tu token: ")
        token = self.request.recv(1024).strip()
        self.request.send(b"Ingresa la llave de inicializacion: ")
        llave = self.request.recv(1024).strip()
        if len(llave) < 16:
            self.request.send(b"\r\nLa llave de inicializacion no cumple con el tamano requerido\r\n")
        id = delfin_decrypt(llave[:16],key,token)
        if "flipid=6" in id:
            self.request.send(b"\r\nFelicidades: %b" % bytes(flag,"utf-8"))
            self.request.send(b"\r\n")
            break
        else:
            try:
                self.request.send(b"\r\nEl token pertenece al usuario: %s\r\n" % bytes(id,"utf-8"))
            except:
                self.request.send(b"\r\nHay un problema con tu token\\r\n")
```

Analizando la función delfin_encrypt, se encontró que recibe tres parámetros, el IV, la llave y el mensaje.

```
def delfin_encrypt(iv,key,msg):
    cipher = AES.new(bytes(key,"utf-8"),AES.MODE_CBC,bytes(iv,"utf-8"))
    los_bytes = cipher.encrypt(pad(bytes(msg,"utf-8"),AES.block_size,style="pkcs7"))
    return base64.b64encode(los_bytes)
```

Cifra el mensaje con AES CBC con el IV y agrega el Padding restante con el estándar PKCS7, lo que significa que dependiendo de la cantidad que falta bytes que faltan para rellenar el bloque de 16 bytes le agrega Padding correspondiente a dichos bytes, por ejemplo:

AAAABBBB88888888

AAAABBBBCCCC4444

Entonces el mensaje quedaría de la siguiente manera:

flipid=688888888

Para posteriormente cifrarlo y codificarlo en base64.

La segunda función es delfin_decrypt que recibe el IV, la llave y el mensaje:

```
def delfin_decrypt(iv,key,msg):
    try:
        ciphertext = base64.b64decode(msg)
        cipher = AES.new(bytes(key,"utf-8"),AES.MODE_CBC,iv)
        el_string = unpad(cipher.decrypt(ciphertext),AES.block_size,style="pkcs7")
        return str(el_string,"ascii")
    except:
        return "Padding Incorrecto"
```

Decodifica el mensaje de base64 y lo descifra con el mismo Padding.

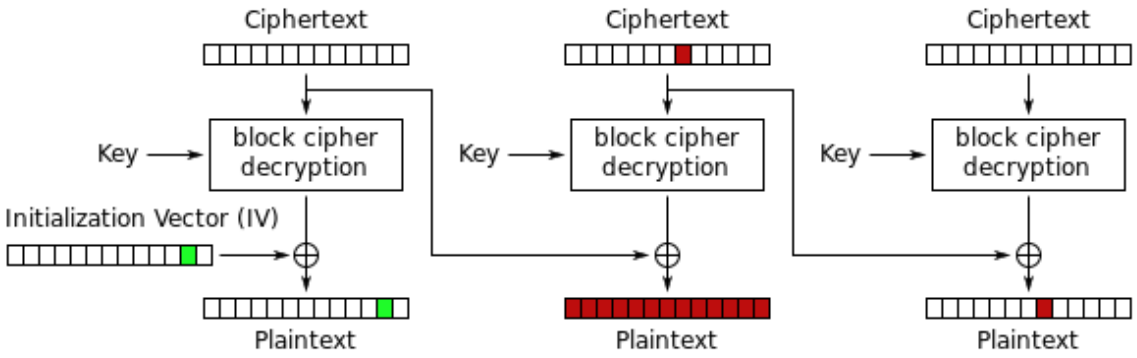
Si ocurrió algún error durante le descifrado regresa Padding Incorrecto. Este error es muy genérico, por lo que puede ocurrir cuando se ingresa un token incorrecto, un IV erróneo o una llave diferente.

Regresando al ataque, lo único donde se puede hacer es realizar un ataque de fuerza bruta offline sobre el token, ya que se conoce el IV. Sería cuestión de bastante tiempo, por lo que se descarta esta opción.

La segunda opción es modificar el IV, a esto se le llama bit flipping attack, el cual consiste en cambiar los valores del IV para generar una respuesta diferente en el proceso de descifrado, conociendo el IV se pueden hacer pruebas para entender lo que genera un pequeño cambio:

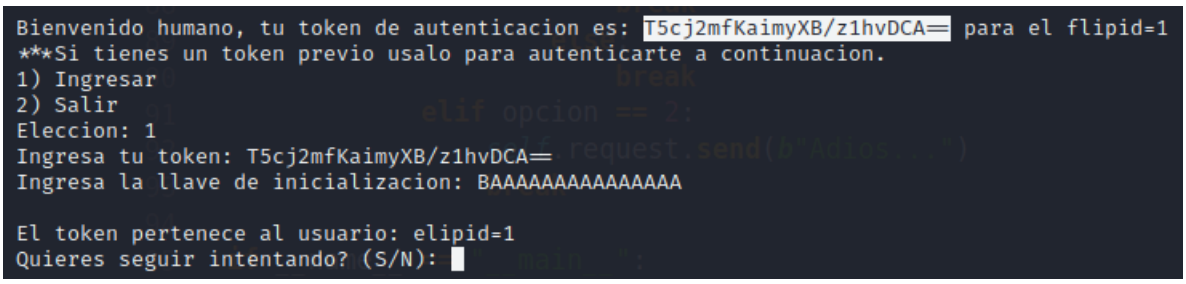
IV = B + A*15

Token = El mismo otorgado.

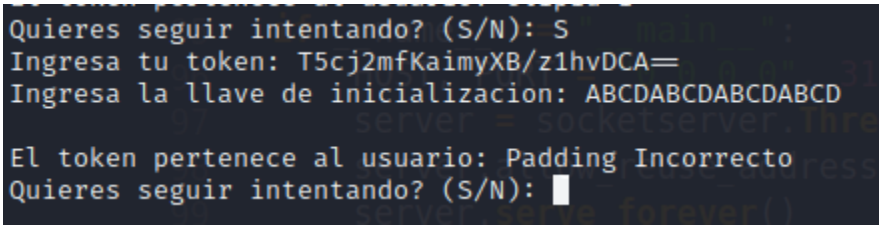


Cipher Block Chaining (CBC) mode decryption

Resultado: elipid=1



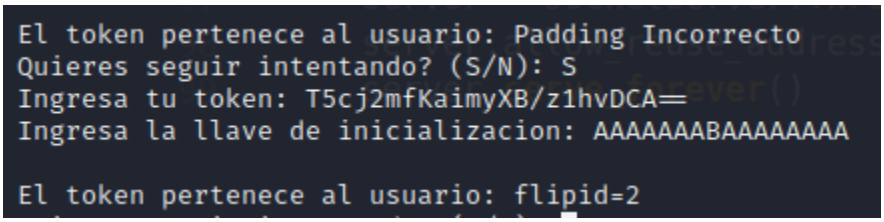
Si se cambia por completo genera el error esperado “Padding incorrecto”.



Por lo que un IV aleatorio es un error.

Lo que se busca es obtener el flipid=6, por lo que se debe cambiar un solo carácter, específicamente el 8°.

IV = A*7 + B + A*8



Se obtiene el id 2, probablemente sea secuencial el 8° carácter, así que suma 5 (esto dependerá del id generado) posiciones después de la A y se obtendrá el flipid=6.

IV = A*7 + F + A*8