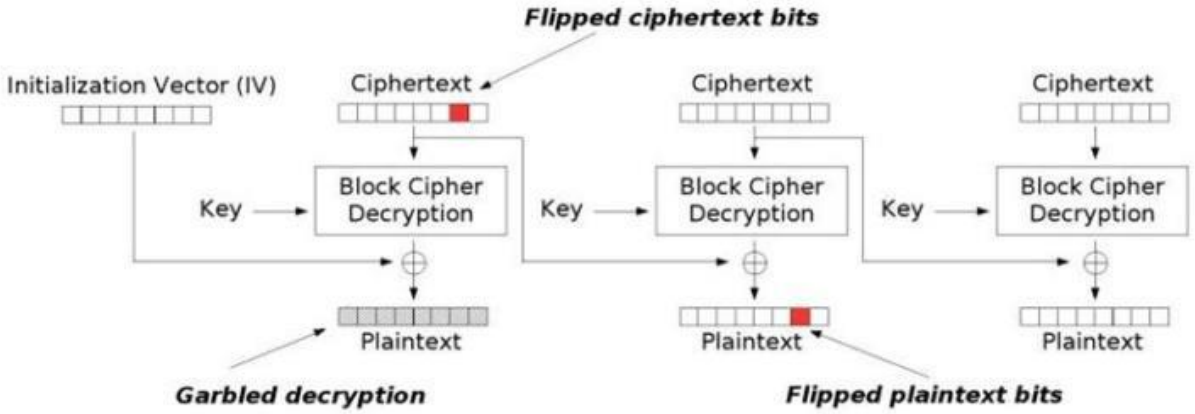
CRYPTO 300

Esté reto abusa de la implementación del modo de operación CBC para el algoritmo de cifrado AES, en la cual debido a la naturaleza del modo de operación CBC, podemos emplear el primer bloque para afectar la salida del segundo en el proceso de descifrado.

El siguiente diagrama, explica a la perfección la vulnerabilidad implementada.



Modification attack on CBC

Para este reto se dan los archivos siguientes:

- safe.py
- database.bin
- encrypt-decrypt_db.py

Donde safe.py es una “caja fuerte” digital en la cual se almacena la llave de cifrado para el archivo database.bin, que contiene la base de datos cifrada por el archivo encrypt-decrypt_db.py

Encrypt-decrypt.py emplea el modo de operación EAX, para AES, esto con la finalidad de que el archivo no sea vulnerable a ataques de fuerza bruta o algún otro tipo de ataques por criptoanálisis al ser criptografía autenticada, es decir, si alguno de los elementos en el proceso de descifrado es ligeramente diferente, la firma del archivo se corrompe y no se puede recuperar el archivo original. Esto se implemento con la finalidad de forzar a los concursantes a resolver el reto a través de safe.py

Safe.py tiene la siguiente salida, al ser un servicio de red, es necesario conectarse al puerto especificado.

```
f@war-ensemble:~/Documents/hackdef/300$ nc 127.0.0.1 3102

               CAJA FUERTE DE GRADO MILITAR

                  #REPUBLICA DE HACKDEF#

       Ejemplo de uso con su token en formato:
        8COQ1V2S_zzhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

  Solo el administrador con el ID 01 tiene acceso a la informacion
   aqui contenida, toda la interaccion esta siendo monitoreada.

Ingresa tu llave:
```

La finalidad del reto es obtener el id de usuario, electionadministrator=01, que es la cadena que internamente se está intentando descifrar. Si usamos el ejemplo obtenemos lo siguiente:

```
Ingresa tu llave: 8COQ1V2S_zzhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

ID incorrecto: 99, intente nuevamente...
```

Lo cual nos da indicio de que la validación se hace solo sobre el valor numérico del ID. Revisando el código fuente de la aplicación podemos ver que se trata de una implementación de AES-CBC con padding pkcs7, ningún problema real, hasta la validación del id de usuario:

```
if administrador_id[22:24] == b"01":
    self.request.send(b"\r\nLlave de descifrado para la DB: ")
    self.request.send(bytes(flag,"utf-8"))
    break
```

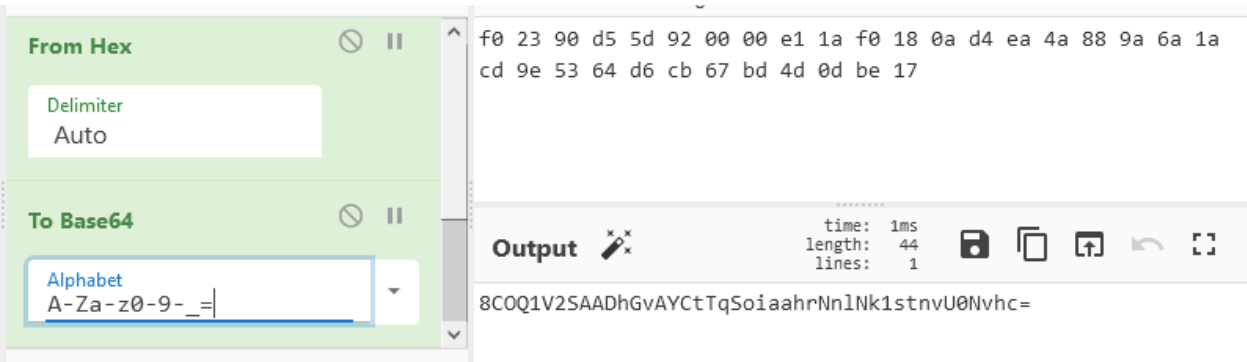
Donde nos podemos dar cuenta de que solo validamos los bytes que nos importan, pudiendo desechar el resto. El reto nos da un token de ejemplo, el cual podemos utilizar ya que fue generado con la misma llave que se espera se haya generado el token del usuario 01

Si tomamos en cuenta que AES CBC funciona con bloques de 16 bytes, podemos suponer que

```
electionadminist
rator=01
```

AL ser el primer bloque usado como el IV del segundo bloque, podemos modificar los valores posicionales del ID que se busca, es decir el 01. Los bytes contenidos en o y n deben ser modificados para que descifren como 01 y se pueda validar la autenticidad del usuario o hacerse pasar por el.

La función decrypt safe nos muestra que el token que se recibe a través del token se encuentra en formato urlsafe base64.



En la imagen anterior modifique los bytes 7 y 8 por valores en 0, es decir, apagando todos los bytes y se vuelve a generar el token en urlsafe base64, al enviarlo de vuelta al servidor se obtiene lo siguiente

```
ID incorrecto: 99, intente nuevamente...
Ingresa tu llave: 8COQ1V2SAADhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

ID incorrecto: , intente nuevamente...
Ingresa tu llave: 8COQ1V2SBlDhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

ID incorrecto: U, intente nuevamente...
Ingresa tu llave: 8COQ1V2SRlDhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

ID incorrecto: U, intente nuevamente...
Ingresa tu llave: 8COQ1V2S9nDhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

ID incorrecto: 0u, intente nuevamente...
```

Haciendo fuerza bruta o bit flipping en los valores de 0 a ff en los bytes posicionales podremos llegar a un punto en el que pueda descifrar al ID que buscamos, sin embargo, el primer bloque quedará inutilizable, afortunadamente no se esta trabajando con el.

```
ID incorrecto: U, intente nuevamente...
Ingresa tu llave: 8COQ1V2S9nDhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

ID incorrecto: 0u, intente nuevamente...
Ingresa tu llave: 8COQ1V2S9izhGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

ID incorrecto: 0), intente nuevamente...
Ingresa tu llave: 8COQ1V2S9jThGvAYCtTqSoiaahrNnlNk1stnvU0Nvhc=

Llave de descifrado para la DB: _!h4ckD3fs3kr3T@
```

Una vez teniendo la llave secreta, podemos descifrar la base de datos, con el archivo

```
f@war-ensemble:~/Documents/hackdef/300$ python3 encrypt-decrypt_db.py
[-] Uso: <encrypt/decrypt> <llave> <ruta_archivo de entrada> <ruta archivo de salida>
```

```
f@war-ensemble:~/Documents/hackdef/300$ python3 encrypt-decrypt_db.py decrypt '_!h4ckD3fs3kr3T@' ./database_export.bin ./database_export.csv
f@war-ensemble:~/Documents/hackdef/300$ head database_export.csv
id_votante,nombre_completo,codigo_de_estado,anio_registro,anio_nacimiento
1337,hackdef{b1t_fl1pp1ng-strik3sback!!},0xh4ckd3f,2017,2017
71148200,Amalleli Galnares Mendibuey,6,1965,1947
60348400,Carmela Obeso Maide,14,2019,2001
25063000,Leida Benagassi Tejeyra,13,1976,1958
88793400,Kerim Polanco Gorostiza,7,1965,1947
109195600,Timmy Campos Rojica,10,1988,1970
125922200,Yoel Morais Buergo,19,1992,1974
```

hackdef{b1t_fl1pp1ng-strik3sback!!}

Pistas:

- Usa el token de ejemplo para tu interacción con el servidor. Si, es base64, pero hay algo diferente. ¿Puedes identificarlo?
- En el modo de operación CBC, el primer bloque se usa como IV del segundo. Te trae memorias de la ronda de calificación?