El siguiente contenido está publicado con fines académicos y de concienciación. No nos hacemos responsables de las actividades que se lleven a cabo con la información mostrada a continuación.

El siguiente post forma parte de una serie de posts dedicados a mostrar las soluciones a los retos presentados en el **Evento de CiberSeguridad de Secuma 2018** celebrado en **Málaga el 15 de Noviembre de 2018.**

Cabe destacar que vamos a proceder a publicar el código fuente de cada uno de los retos (de los retos de Web) para que todo el mundo pueda bajárselo, modificarlo y conseguir así customizar lo a su gusto para lograr nuevas variantes procedentes del mismo reto que puedan inspirar a la creación de nuevos retos y/o soluciones. Así que estar atentos al **Gitlab**.

Tan sólo pedimos que siempre que se modifique el código se mencione al autor del mismo así como que se ponga a disposición del público el resultante. Dicho esto vamos a proceder a explicar, de forma detallada los pasos que seguimos así como consejos que puedan ayudar al lector y/o aclarar conceptos. Empecemos!

FASE DE CONTACTO.

El segundo reto de crypto puede que a alguno le suene, pues no es nada novedoso, es exactamente igual, por no decir calcado de un **reto random** que subió un chaval a twitter y cuyo writeup me animó a incluirlo en el CTF.

Antes de nada agradecer al creador, así como a la persona que subió el writeup. Comencemos!

Nos encontramos en la descripción del reto una **imagen** con una pequeña descripción y un **fichero** de texto adjunto.

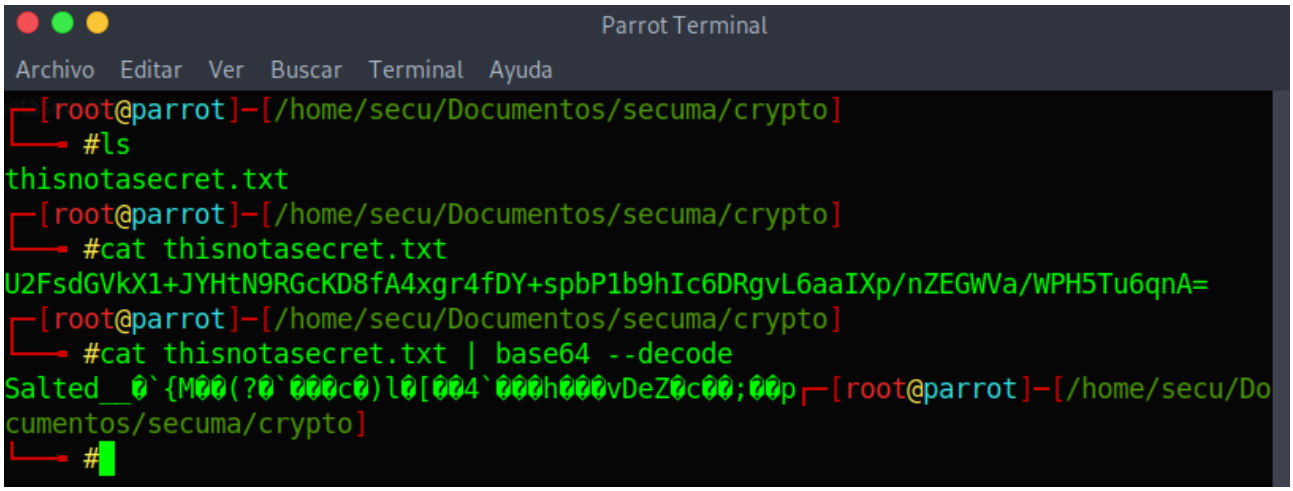
La descripción del reto dice algo así: "**pienso que ella tiene que ser la clave**", y hace referencia a la chica de la foto, que si hemos visto Mr. robot o si hacemos una pequeña búsqueda con google imágenes, sabremos que se llama **darlene**.

Fucksoci3ty
300

I think she has to be the key to all this...

MD5: 1968be52e3160b5bfc745d8886df27de
SHA1: 7fc537348aa7a0d4102bd25f11fc6f6c98a43825

Pasemos al fichero de texto, que contiene un bonito **base64**. Que codifica una cadena cifrada con algún tipo de algoritmo criptográfico.



Ese "**Salted**" nos da la pista de que es un texto cifrado con **OpenSSL**. Pero claro, no tenemos forma de conocer a priori con cuál de los cifrados que ofrece OpenSSL. La solución es guarra, pero simple y eficaz:

```
Cipher commands (see the `enc' command for more details)
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       aria-128-cbc      aria-128-cfb
aria-128-cfb1     aria-128-cfb8     aria-128-ctr      aria-128-ecb
aria-128-ofb      aria-192-cbc      aria-192-cfb      aria-192-cfb1
aria-192-cfb8     aria-192-ctr      aria-192-ecb      aria-192-ofb
aria-256-cbc      aria-256-cfb      aria-256-cfb1     aria-256-cfb8
aria-256-ctr      aria-256-ecb      aria-256-ofb      base64
bf                bf-cbc            bf-cfb            bf-ecb
bf-ofb            camellia-128-cbc  camellia-128-ecb  camellia-192-cbc
camellia-192-ecb  camellia-256-cbc  camellia-256-ecb  cast
cast-cbc          cast5-cbc         cast5-cfb         cast5-ecb
cast5-ofb         des               des-cbc           des-cfb
des-ecb           des-ede           des-ede-cbc       des-ede-cfb
des-ede-ofb       des-ede3          des-ede3-cbc      des-ede3-cfb
des-ede3-ofb      des-ofb           des3              desx
rc2               rc2-40-cbc        rc2-64-cbc        rc2-cbc
rc2-cfb           rc2-ecb           rc2-ofb           rc4
rc4-40            seed              seed-cbc          seed-cfb
seed-ecb          seed-ofb          sm4-cbc           sm4-cfb
sm4-ctr           sm4-ecb           sm4-ofb
```

Si hay todos estos **cifrados en OpenSSL**, la solución será hacer un script que nos pruebe cada uno de ellos.

Pero estos cifrados requieren de una clave, ¿cuál utilizamos?

Pues aquí es cuando volvemos a la descripción del reto y aplicamos literalmente lo que nos dice: "**Ella tiene que ser la clave**".

Pues vamos a probar con la clave darlene. Y con cada uno de los cifrados que tenemos separados en un archivo de texto.

```
┌─[secu@parrot]─[~/Documentos/secuma/crypto]
└──╼ $cat crypts.txt
aes-128-cbc aes-128-ecb aes-192-cbc aes-192-ecb aes-256-cbc aes-256-ecb aria-128
-cbc aria-128-cfb aria-128-cfb1 aria-128-cfb8 aria-128-ctr aria-128-ecb aria-128
-ofb aria-192-cbc aria-192-cfb aria-192-cfb1 aria-192-cfb8 aria-192-ctr aria-192
-ecb aria-192-ofb aria-256-cbc aria-256-cfb aria-256-cfb1 aria-256-cfb8 aria-256
-ctr aria-256-ecb aria-256-ofb base64 bf bf-cbc bf-cfb bf-ecb bf-ofb camellia-12
8-cbc camellia-128-ecb camellia-192-cbc camellia-192-ecb camellia-256-cbc camell
ia-256-ecb cast cast-cbc cast5-cbc cast5-cfb cast5-ecb cast5-ofb des des-cbc des
-cfb des-ecb des-ede des-ede-cbc des-ede-cfb des-ede-ofb des-ede3 des-ede3-cbc d
es-ede3-cfb des-ede3-ofb des-ofb des3 desx rc2 rc2-40-cbc rc2-64-cbc rc2-cbc rc2
-cfb rc2-ecb rc2-ofb rc4 rc4-40 seed seed-cbc seed-cfb seed-ecb seed-ofb sm4-cbc
 sm4-cfb sm4-ctr sm4-ecb sm4-ofb
┌─[secu@parrot]─[~/Documentos/secuma/crypto]
└──╼ $
```

Podemos utilizar un **script** como este (es muy muy simple) que irá probando cada uno de los cifrados que hemos separado en un txt, y que iremos leyendo, y con la clave **darlene**, volcaremos el resultado a un archivo precedido por el nombre del cifrado utilizado.

```
┌─[secu@parrot]─[~/Documentos/secuma/crypto]
└──╼ $ls
bruteDec.sh  crypts.txt  thisnotasecret.txt
┌─[secu@parrot]─[~/Documentos/secuma/crypto]
└──╼ $cat bruteDec.sh
#!/bin/bash

for i in $(cat crypts.txt);
do
  openssl enc -d -$i -in thisnotasecret.txt -out $i.txt -a -k darlene;
done

┌─[secu@parrot]─[~/Documentos/secuma/crypto]
└──╼ $
```

Posteriormente buscaremos en todos esos archivos la palabra "**secuma18**" que es el comienzo de la flag.

```
┌─[secu@parrot]─[~/Documentos/secuma/crypto]
└──╼ $strings -f * | grep "secuma18"
des3.txt: secuma18{N0b0dY_MaKe_R00tk1ts_L1K3_m3}
des-ede3-cbc.txt: secuma18{N0b0dY_MaKe_R00tk1ts_L1K3_m3}
┌─[secu@parrot]─[~/Documentos/secuma/crypto]
└──╼ $
```

Aplicamos todo esto y conseguimos la **FLAG**!

by **@Secury**