CÁTEDRA DE CIBERSEGURIDAD CIBERUGR, INCIBE-UGR

Nombre	OpenDoor
Categoría	MISC (REVERSING + FORENSE)
Dificultad	DIFÍCIL
Puntos	500

DESCRIPCIÓN DEL RETO

¿Sabes que el otro día me descargué de internet un firmware para mi router? Ahora está remolón y cada vez que enciende suena una musiquita de Dragon Ball. ¿Te gustaría echarle un ojo?

WRITEUP

1. Descomprimimos el archivo comprimido con gunzip

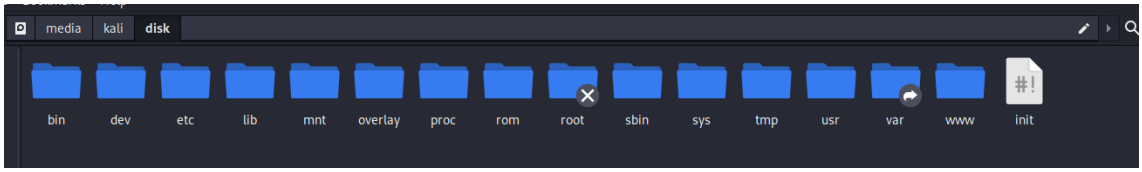
gunzip firmware.gz

```
┌──(kali㉿kali)-[/tmp/open]
└─$ gunzip firmware.gz

┌──(kali㉿kali)-[/tmp/open]
└─$ ls
firmware
```

2. Extraemos el firmware usando binwalk

binwalk -e firmware

```
┌──$ binwalk -e firmware

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
40960         0xA000          uImage header, header size: 64 bytes, header CRC: 0x52CD2ED6, created: 2024-03-24 20:20:46
, image size: 551540 bytes, Data Address: 0x4A000000, Entry Point: 0x4A000000, data CRC: 0xC07025B4, OS: Firmware, CPU:
ARM, image type: Firmware Image, compression type: none, image name: "U-Boot 2023.04-OpenWrt-r25659-00"
43584         0xAA40          CRC32 polynomial table, little endian
60832         0xEDA0          LZ4 compressed data
329204        0x505F4         SHA256 hash constants, little endian
392190        0x5F8FE         Android bootimg, kernel size: 1920091392 bytes, kernel addr: 0x203A726F, ramdisk size: 163
5151433 bytes, ramdisk addr: 0x2064696C, product name: ""
569252        0x8AFA4         Flattened device tree, size: 23312 bytes, version: 17
1107968       0x10E800        uImage header, header size: 64 bytes, header CRC: 0x5922CEA8, created: 2024-03-24 20:20:46
, image size: 411 bytes, Data Address: 0x0, Entry Point: 0x0, data CRC: 0x3D4C3F4D, OS: Linux, CPU: ARM, image type: Scr
```



3. Después de revisar minuciosamente el sistema de archivos, encontrar el script de inicio “backdoor” en la carpeta “/etc/init.d”, que ejecutará un binario en “/usr/bin” llamado “backdoor”. Este fichero es un ELF de 32 bit compilado para la arquitectura ARM.

file backdoor

```
└─$ file backdoor
backdoor: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /li
b/ld-musl-armhf.so.1, no section header
```

4. Abrir el binario en cualquier herramienta de ingeniería inversa. En este caso se usará “Binary Ninja”. Al abrirlo con este programa, nos dirigimos a la función “main”.

```
        if (r0_4 s< 0)
            s = "listen"
        else
            printf("Server listening on port %d\n", 0x539)
            int32_t fd_1
            pid_t status_1
            while (true)
                fd_1 = accept(fd)
                void* const s_1
                if (fd_1 s< 0)
                    s_1 = "accept"
                else
                    status_1 = fork()
                    if (status_1 == 0)
                        break
                    if (status_1 s<= 0)
                        s_1 = "fork"
                    else
                        close(fd_1)
                        continue
                perror(s_1)
            close(fd)
            sub_10994(fd_1)
            status = status_1
    else
        s = "bind"
if (fd s< 0 || (fd s>= 0 && r0_2 s< 0) || (fd s>= 0 && r0_2 s>= 0 && r0_4 s< 0))
    perror(s)
```

5. Tras analizarlo, podemos ver cómo abre un servidor TCP y, cuando un cliente se conecta y envía un mensaje, ejecuta el método sub_10994. Dentro de este método se pide una contraseña y, en caso de que sea correcta, se da luz verde a ejecutar comandos de manera remota.

```
int32_t sub_10994(int32_t arg1)

uint32_t* const var_14 = __stack_chk_guard
send(arg1)
void p1
*(&p1 + recv(arg1) + 0x468 - 0x469) = 0
sub_1093c(&p1, &data_10bed)
if (strcmp(&p1, &data_10bfe) != 0)
    puts("Incorrect password")
    close(arg1)
    exit(1)
    noreturn
puts("Password accepted. You can now e…")
void* const s
while (true)
    void var_414
    memset(&var_414, 0, 0x400)
    send(arg1)
    ssize_t r0_10 = recv(arg1)
```

Para ello, se llama a sub_1093c con argumentos arg1 (input del usuario) y la constante data_10bed.

---

f1 e4 2d 5e c9 3a b8 2f 91 d7 4c 7b a0 8f e8 06

Dentro de sub_1093c, se hace un XOR con los parámetros de la función y se guarda el resultado en la dirección de memoria de arg1

```
int32_t sub_1093c(int32_t arg1, char* arg2)

int32_t r0_1 = strlen(arg2)
int32_t i = strlen(arg1)
char* r2 = arg1 - 1
int32_t r3 = 0
while (i s> 1 - arg1 + r2)
    r2 = &r2[1]
    if (r3 == r0_1)
        r3 = 0
    char r1_1 = arg2[r3]
    r3 = r3 + 1
    *r2 = r1_1 ^ *r2
return i
```

Por último, ya en la función anterior, se compara el resultado con la constante data_10bfe.

---

a4 a3 7f 01 8c 6e eb 66 d8 83 13 38 f4 c9 da 32 8a a9 19 32 a5 4d f9
5d a2 88 7d 15 ff da 9a 59 83 d4 58 0a fa 48 c5

Podemos llegar a la conclusión de que la contraseña se podrá obtener haciendo un XOR entre data_10bed y data_10bfe.

6. Abrir Cyberchef en el navegador y descifrar con XOR la contraseña con el texto cifrado, lo cual nos dará la flag.