Trabajo Práctico: ARMv8 en QEMU

Introducción

El trabajo práctico se basa en el uso de la herramienta QEMU para emular un microprocesador ARMv8 y el uso del debugger GDB para depurar un programa que deberán construir, y un programa dado para analizar.

Para visualizar fácilmente el resultado de los códigos implementados, la **primer actividad** de este trabajo se basa en la modificación de imágenes generadas en código ASCII (ASCII art) para resolver un laberinto de forma similar al parcial.

La **segunda actividad** propuesta se basa en analizar un programa criptográfico que descifra un mensaje secreto mediante un password de **32 bytes**. Dicho programa es inseguro y comprendiendo su funcionamiento se debería poder quebrar.

Condiciones

- Realizar el laboratorio individualmente.
- Entregar el laboratorio resuelto por mail hasta el **lunes 15 de Junio** inclusive. Se enviará confirmación de la recepción. Los laboratorios entregados después de esa fecha se consideran desaprobados.
- En caso de tener algún problema para resolver el laboratorio (falta de hardware, conectividad, etc.) comunicarse por email a: **`odc.famaf@gmail.com`** antes del **miércoles 27 de Mayo**.
- Defender en una exposición oral el trabajo presentado. Se organizarán reuniones en Meet con cada uno y deberán responder preguntas acerca del desarrollo del mismo. La defensa puede realizarse en cualquier momento, una vez finalizado el desarrollo del laboratorio. La fecha límite para la defensa está prevista para el día **miércoles 17 de Junio**.
- La aprobación de este trabajo es requisito **obligatorio** para obtener la PROMOCIÓN de la materia.

Formato de entrega

Deben entregar una carpeta comprimida (tarball o zip) con el nombre: `Laboratorio_Apellido_Nombre`. La carpeta debe contener los directorios “Laberinto” y “Crypto”, respetando la estructura del archivo entregado en el aula virtual. Los archivos deben seguir el estilo de código de los programas de ejemplo y contener comentarios que ayuden a comprender la manera en que solucionaron el problema. La segunda actividad además del archivo *main.s* modificado deberá incluir un pdf con las respuestas solicitadas.

La carpeta comprimida debe enviarse por email a: **`odc.famaf@gmail.com`** con el asunto: `Laboratorio_Apellido_Nombre.`

Calificación

El Trabajo Práctico se aprueba o desaprueba (A o N). Para aprobar, los códigos deben realizar la tarea pedida, que será corroborada cargando, ensamblando y ejecutando el programa. Finalmente se deberá defender el trabajo oralmente.

Aunque no se califica: estilo de código, simpleza, elegancia, comentarios adecuados y velocidad; si el código está muy por fuera de los parámetros aceptables, se podrá desaprobar el trabajo aunque sea funcionalmente correcto.

Actividad 1

Desarrollar un solucionador de laberintos en código ASCII de dimensión Nx16 similar al resuelto en el segundo parcial de la materia. Para ello modificar el archivo '*main.s*' que les fue provisto (a través del aula virtual) en la carpeta Laberinto.
Se deberá construir teniendo en cuenta los símbolos mínimos:

X	Personaje principal	0x58
#	Objetivo	0x23
|	Pared vertical	0x7c
+	Esquinas	0x2b
-	Pared horizontal	0x2d

[Opcionalmente]: Se puede expandir con símbolos adicionales con reglas propias.
Por ejemplo: Un símbolo **@** me redirige a otro símbolo **@**.

Ejemplos que deberán resolver:

```
+-+---+--------+
| |     |     -|
| | --+ | +--  |
| |   | | |    |
| +-- | | |  --+
|  X  |   |   #|
+-----+---+----+
```

```
+-+---+--------+
|X|     |     -|
| | --+ | +--  |
| |   | | |    |
| +-- | | |  --+
| |   |   |   #|
| ----+ |    --|
|     |  | -----|
|  |  |  |     |
+-----+---+----+
```

```
+-+---+--------+
| |            -|
| | --+ | +--  |
| |   | | |    |
| +-- | | +----+
| |X  | +-+    |
| ----+ |    +-|
|       | ---+ |
|  +----+      |
|           |#  |
+-----+---+----+
```

El resolvedor a entregar debería poder resolver **cualquier** laberinto que se le pase en el segmento de datos como:

```
.data:
    maze: .dword 0x2d2b2d2d2d2b2d2b ,  0x2b2d2d2d2d2d2d2d , ….
```

Como referencia se puede tener en cuenta el siguiente algoritmo:

Método de la mano (izquierda[1])

En cada paso:

- Siempre que pueda girar a la izquierda, hacerlo (y dar un paso).
- Si no girar a la izquierda pero puedo ir hacia adelante, avanzar (dar un paso).
- Si no puedo girar a la izquierda, ni avanzar, pero puedo girar a la derecha, hacerlo (y dar un paso).
- Sino, Dar la vuelta y avanzar

Notar que las indicaciones izquierda y derecha son relativas a la orientación del personaje.

Actividad 2

Criptografía Básica

Un cipher[2] (simétrico) consiste en un par de “funciones eficientes” E (encrypt) y D (decrypt).

$E : K \times M \rightarrow C$	$D : K \times C \rightarrow M$

Donde *M, K, C* son el espacio de mensajes posibles, espacio de claves posibles y espacio de textos cifrados posibles respectivamente.

Que además debe cumplir que:

$$D(k, E(k,m)) = m \quad \forall\ m \in M,\ k \in K$$

Por simplicidad, en esta actividad vamos a asumir que:

$$M = K = C = \{0,1\}^{32 \times 8}$$

La técnica de encriptación (o cipher) conocida como “One Time Pad[3]” posee propiedades importantes en términos de seguridad como “Perfect Secrecy”. Esto lo clasifica como un cipher “seguro”. A pesar de ello este cipher no es usado en la práctica por razones que no abordaremos ya que escapan a la materia.

En un cipher OTP, las funciones E y D están definidas de la siguiente manera:

$E(k, m) = k \oplus m$	$D(k, c) = k \oplus c$

[1] https://github.com/joshepcs/TinyMazeSolver (Implementacion en Python de referencia)
[2] https://crypto.stanford.edu/~dabo/courses/OnlineCrypto/slides/02-stream-v2-annotated.pdf
[3] https://www.coursera.org/lecture/crypto/information-theoretic-security-and-the-one-time-pad-cbnX1

De este modo encriptar un mensaje *m = "HOLA...",* dada una clave *k="ASDF..."* será de la forma:

m	H	O	L	A	...
k	A	S	D	F	...
m (Hexa)	0x48	0x4f	0x4c	0x41	...
k (Hexa)	0x41	0x53	0x44	0x46	...
m (Bin)	01001000	01001111	01001100	01000001	...
k (Bin)	01000001	01010011	01000100	01000110	...
c = m ⊕ k	00001001	00011100	00001000	00000111	...
c (Hexa)	0x09	0x1c	0x08	0x07	...

Una vez obtenido el mensaje encriptado **c** (o ciphertext). Cualquiera que sepa la clave podrá desencriptar el mismo aplicando la función D.

k	01000001	01010011	01000100	01000110
c	00001001	00011100	00001000	00000111
m = c ⊕ k	01001000	01001111	01001100	01000001

Teniendo en cuenta esta definición, se puede pensar en construir una implementación de estos ciphers de la forma:

```
def encrypt(k,m):
    ...
    for i from 0 to 31:
        c[i] = chr(ord(m[i]) ^ ord(k[i]))     # /*c[i] = m[i] ⊕ k[i] */
    ...
```

(La función `decrypt` puede ser implementada de forma similar)

Pensando en LEGv8 se puede pensar que un loop que implemente OTP puede contener instrucciones de la forma:

```
...
ldurb w0, [x6]    // x6 apunta a m[i]
ldurb w1, [x7]    // x7 apunta a k[i]
eor x2, x0, x1    // x2 guardará   x0 ⊕ x1
sturb ...
```

Con estas ideas en mente, los alumnos deberán analizar el código que les fue entregado.

Enunciado

Realizar un **análisis** del programa entregado en el directorio “Crypto”.

El archivo principal a analizar es *main.s* y contiene un algoritmo **decrypt** que implementa la función OTP, pero que no es exactamente OTP.
Es decir, que si conocen la password **k** y tienen un texto cifrado **c**.
$D(k, c) \neq k \oplus c$

Compilar el código base dado y correr el programa siguiendo los pasos indicados en el archivo “HowTo: Debug AArch64 GDB”. El programa tiene cargados 3 arreglos con datos en hexadecimal, correspondientes a un password (incorrecto) de 32 bytes, un texto cifrado de 32 bytes y una cadena auxiliar de 32 bytes usada en el proceso de descifrado:

Código Hexadecimal	Código ASCII
41 41 41 41 41 41 41 41 42 42 42 42 42 42 42 42 43 43 43 43 43 43 43 43 44 44 44 44 44 44 44 44	AAAAAAAABBBBBBBB CCCCCCCCDDDDDDDD
39 c1 da a1 de 37 cc 5c c8 0f fd 16 4c 0b e3 35 30 34 3e f5 09 11 1a 0d 07 02 2b 25 4c 62 72 6c	9····7·\····L··5 04>·····+%Lbrl
4d 53 43 7c 51 58 47 75 4d 78 7e 7b 85 7f 51 74 54 80 79 7f 55 84 7a 85 7d 82 7e 89 81 8f 8e 60	MSC|QXGuMx~{··Qt T·y·U·z·}·~····`

Importante:
Los estudiantes deberán analizar el programa dado para entender el código de descifrado.
SOLO se podrá modificar en todo el programa main.s, el arreglo ‘password’ del segmento .data y agregar comentarios que ayuden a entender la funcionalidad.
De esta manera tendrán que[4]:

- Determinar cuál de los arreglos some1 o some2 es el mensaje cifrado o ciphertext.
- Determinar cuál de los arreglos some1 o some2 es una cadena auxiliar y como se usa.
- Obtener la password para decodificar el mensaje secreto.
- Obtener el mensaje secreto o plaintext.
- Decidir si el programa está completo o incompleto en algún sentido justificando.

Estos items deberan ser desarrollados, respondidos y agregados a la carpeta “Crypto” en un archivo (pdf).

[4] Hint: Analizar por partes. Si les sirve, pasar a un lenguaje de alto nivel.