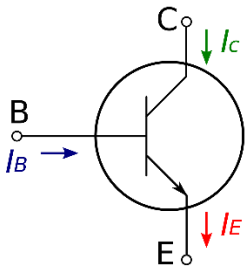
Apuntes del curso.

Semana 0. Electrónica:

- La historia de la electricidad: https://www.youtube.com/watch?v=NUUeGianTKM
- El semiconductor:
  - Actúa como un conductor o un aislante dependiendo de alguna condición.
  - Extrínsecos (Dopaje) vs Intrínsecos (Un solo átomo).
  - Semiconductores extrínsecos: Tipo p (Exceso de protones), Tipo n (Exceso de electrones).
  - https://www.youtube.com/watch?v=cy50YR7kr8c
- El transistor:



  Permite el flujo de electrones entre colector (C) y emisor (E) siempre que haya una tensión determinada en la base (B) (tensión umbral).

  ¡Es un interruptor sin partes móviles!

- Lógica digital:
  - Agrupando transistores y resistencias podemos implementar todas las puertas lógicas que queramos:
  - http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/trangate.html#:~:text=The%20use%20of%20transistors%20for,for%20the%20TTL%20logic%20family.
  - Podemos modelar expresiones lógicas complejas con puertas simples:
  - https://courses.edx.org/courses/course-v1:UTAustinX+UT.PQ.14.01x+1T2017/f6757b627d464e35985a16919bbdd4db/
  - La salida de una función lógica para cualquier entrada se define mediante una tabla de verdad.
- Memoria:
  - Con puertas lógicas también podemos hacer memoria:
  - https://www.youtube.com/watch?v=j-nUVi215yE
- Otros circuitos:
  - Sumadores, mutiplexores, comparadores, codificadores.
- Computación, la máquina de Turing:
  - https://www.youtube.com/watch?v=iaXLDz_UeYY
- Práctica: Diseña tu esolang Turing completo: https://esolangs.org/wiki/Main_Page

Semana 1. Hardware:

- Simplez: Una arquitectura didáctica: (Son todos lo mismo, elige el que mejor te vaya, yo recomiendo el último)
    - http://www1.frm.utn.edu.ar/tecnicad2/tec_dig2/tools/manual_simplez.pdf
    - https://grvc.us.es/FC_grado/docs/material/Tema4.pdf
    - https://github.com/TretornESP/bec/blob/main/simplez.pdf
- Tamaño de palabra: Unidad arbitraria mínima de acceso a datos: Normalmente tamaño de un registro y de bus. Es decir: Cuando cargo datos desde la memoria a un registro para hacer una operación, cuantos bits cargo? Pues ese número suele ser una palabra.
- Arquitectura Von Neumann (pronunciado Noyman): Tanto datos como instrucciones están en la misma localización, depende del contexto diferenciarlos.
- Memoria:
    - Un conjunto de unidades de memoria agrupadas en bloques (palabra, byte). Se interactúa con ella emitiendo una dirección, una orden (leer/escribir) y recibiendo/enviando el dato.
    - Internamente se suele visualizar como una matriz de unidades a las que se hace un and con la dirección emitida.
    - https://www.researchgate.net/figure/Esquema-de-una-memoria-RAM-2D-Organizacion-3D-Tridimensional-Cada-celda-binaria-se_fig28_282335835
- Procesador:
    - UC: unidad de control
        - Máquina de estados encargada de ejecutar el pipeline del procesador.
        - Pipeline: Secuencia que rige el comportamiento de la máquina:
            - IF: Instruction fetch, obtener la instrucción a ejecutar desde la memoria.
            - ID: Instruction decode, decodificar la instrucción en sus partes elementales:
                - Opcode: Codigo de operación, identifica la operación a ejecutar (p ej, ADD suma, SUB resta)
                - Registro: Con que registro operar
                - Modo de direccionamiento: Que modo de direccionamiento aplicar
                - Campo de datos: Datos con lo que operar.
            - EX: Execution, ejecuta la instrucción.
            - MEM: Acceso a memoria, si hace falta acceder a memoria se hace aquí
            - WB: WriteBack, guardar los resultados de la operación.
        - Registro de estado: Contiene el estado de la máquina, algunos elementos:
            - Z: Zero, la ultima op resultó en un 0
            - N: Negativo, “”
            - I: Interrupt enabled: Se admiten interrupciones
            - H: Halt, la máquina está en suspensión.

            - C: Carry, la última operación excedió el tamaño de la unidad, pero ese valor extra puede almacenarse en una unidad aritmética de orden jerárquico superior para su proceso.
            - V: Overflow, igual que el carry pero ya no podemos pasarle esa información a ninguna otra unidad.
        - Banco de registros:
            - Pequeña memoria en la que almacenamos los datos con los que operamos.
        - ALU: Unidad arimético lógica:
            - Permite realizar operaciones aritméticas o lógicas en función de unos parámetros de entrada. La operación se codifica con dos inputs, modo (lógico/aritmético), id de operación.
            - https://www.researchgate.net/figure/Complete-Functional-Table-of-the-Designed-ALU_tbl1_249316170
        - Buses:
            - Permiten comunicar unidades funcionales, son un medio compartido, pueden ser abiertos (salen de la cpu y comunican unidades como la gpu, tarjeta de sonido, etc (simplez)) o cerrados (solo comunican elementos internos de la cpu y la comunicación con el exterior se realiza mediante bridges (cpus modernas))
            - En simplez tenemos: Bus de datos, direcciones y control.

- Modos de direccionamiento: (CD -> Campo de datos, DE -> Dirección efectiva, X -> registro índice, MEM(0x5) -> contenido de la memoria en la posición 5)
    - Directo: DE = CD
    - Indirecto: DE = MEM(CD)
    - Indexado: DE = CD + X
    - Indirecto indexado: DE = MEM(CD + X) ó DE = MEM(CD) + X (al gusto del arquitecto).
    - Inmediato (no direccionamos nada): Dato = CD
    - Hay muchos más…
- Gestión de la periferia:
    - Pollying: Bucle infinito (preguntamos al dispositivo si tiene algo para nosotros, si lo tiene lo leemos, si no lo tiene volvemos a preguntarle)
    - Interrupciones: El dispositivo nos notifica que tiene algo para nosotros, paramos nuestra ejecución normal y “atendemos” al dispositivo. Luego volvemos a nuestra ejecución como si nada hubiera ocurrido.
    - ADM: Acceso directo a memoria: El propio dispositivo escribe o lee de nuestra memoria principal, para eso inhibe al procesador durante n ciclos (robo de ciclo) o la escritura en los buses (robo de bus).
- Programación reentrante: El concepto más importante también es el más difícil de resumir. La idea es que cuando se termine la rutina de servicio a la interrupción, el programa principal vuelva a ejecutarse como si nada hubiera pasado, para eso es necesario guardar el estado de la máquina antes de pausar el programa (registros, stack,etc). Esto lo puede hacer el programa (caller save) o la isr (callee save) (usamos este). Este proceso se llama “cambio de contexto”.

Semana 2: Sistemas operativos:

El kernel actúa como un intermediario entre el usuario y el hardware, administrando los recursos de la máquina y abstrayendo de las especializaciones debidas a las diferentes arquitecturas de los dispositivos.

Principales funciones del kernel:

- Manejo de acceso a memoria: Maneja las tablas de páginas del sistema, la protección de las mismas. Permite asignar (allocate) y revocar (deallocate) memoria.
- Manejo de procesos: Planifica los procesos en la cpu, administra sus estados, interactúa con ellos para mandarles señales de control y define su estructura en memoria.
- Entrada/Salida e IPC. Se encarga de permitir la comunicación del sistema con los periféricos mediante el concepto de device driver.
    - Los device drivers son código del kernel que se puede cargar o eliminar en caliente (Sin recompilar). Permiten ampliar la funcionalidad del sistema para permitir soportar nuevo hardware.
- Concurrencia:
    - https://www.youtube.com/watch?v=nlHIuG3RQ0g
    - https://www.youtube.com/watch?v=nVESQQg-Oiw
- Memoria virtual:
    - El concepto más importante de este bloque es la memoria virtual: Por este proceso se dota a cada proceso de su espacio de direcciones único (y virtual), que luego se mapea a direcciones reales (físicas) mediante la memory management unit (MMU) empleando las tablas de páginas.
    - Esto nos permite cosas como:
        - Compartir páginas físicas (ahorra memoria).
        - Simplificar la programación (código independiente de la posición).
        - Mayor seguridad: ASLR
        - Enviar páginas no usadas al disco duro (swapping).
    - Existen dos mecanismos de memoria virtual (segmentación, obsoleto) y paginación.
    - La traducción de direcciones se acelera mediante una memoria caché llamada TLB (Translation lookaside buffer).
    - https://www.youtube.com/watch?v=pJ6qrCB8pDw
- Administrar el arranque del disco:
    - https://www.youtube.com/watch?v=atNMrBVwRjk (vídeo casero (:)
- Práctica final: Hazte tu propio kernel!
    - https://www.youtube.com/watch?v=mpPbKEeWIHU UEFI
    - https://www.youtube.com/watch?v=7LTB4aLI7r0 BIOS

Semana 3: Programas de usuario:

- Formato ELF: (Executable and Linkable Format).
    - https://github.com/TretornESP/bec/blob/main/s3/devconf2012.pdf
    - Permite crear varios tipos de programas:
        - Librerías

        - Ejecutables
        - Core dumps
        - Ejecutables dinámicos (el más usado)
    - Define donde está almacenado el código en disco y donde colocarlo en memoria cuando se cargue mediante secciones y segmentos.
    - Define donde iniciará la ejecución de nuestro proceso.
    - Define los permisos de cada sección.
    - Almacena los símbolos.
    - Almacena información de debug (DWARF)
- Carga dinámica:
    - En vez de incluir las librerías en nuestro programa, podemos dejarlas referenciadas y solicitar su carga conforme las necesitemos.
    - Esto se consigue con las tablas plt (procedural linkage table) y got (global offset table).
    - Cuando llamamos a printf, estaremos entrando a la dirección de printf en la tabla plt, esta nos redirigirá al cargador dinámico si no está presente, que escribirá la dirección correcta en la entrada de la got y saltará ahí para completar la petición. A partir de ahora cuando llamemos a printf, saltaremos directamente a esta entrada y a la función real.
- Protecciones de los ejecutables:
    - Canaries: Cada vez que ejecutamos un return susceptible de inyección, comprobamos si un valor preestablecido (fs+28) ha sido modificado. Este valor se encuentra entre nuestro buffer y las entradas susceptibles de inyección, con lo que para tomar el control del programa debemos obligatoriamente modificarlo. En caso afirmativo, el programa se aborta con el mensaje: stack smashing detected.
    - W^X: Básicamente evita que podamos ejecutar páginas en las que podemos escribir.
    - ASLR: Address space layout randomization: Aleatoriza la situación de las librerías de nuestro programa. Evita ataques como ret2libc.
    - PIE: Position independent code: Como aslr pero para el propio binario. Se dividen básicamente porque un programa elf de tipo EXEC no permite aslr.
- Práctica: Carga dinámica:
    - https://www.youtube.com/watch?v=r4auCn-_axU

Semana 4: Hacking:

Esto se entiende muchísisisimo mejor con vídeos.

- Stack overflow:

En el siguiente pseudocódigo:

```
void vulnerable(char * input) {
    char buffer1[64];

    int i = 0;

    while (input[i] i= 0) {
```

```
            buffer[i] = input[i];
            i++;
        }
        return;
}
```

Asumiendo que controlamos input, podemos sobrepasar el tamaño de buffer sin mayor dificultad. Ahora observamos que la instrucción buffer[i] = input[i] es en realidad:

Memoria ram en (dirección de buffer + valor de i) = memoria ram en (dirección de input + valor de i)

o por ejemplo, si buffer vale 0x7ffe450000 e input vale 0x7ffe650000, para i = 400:

memoria(0x7ffe450000+400) = memoria(0x7ffe650000 + 400)

Si os fijáis la primera dirección NO está dentro de buffer (que solo tiene 64 elementos). Esto es un buffer overflow.

Sabiendo que después del buffer está la dirección de retorno, podemos hacer que cuando se ejecute el return, se salte a la dirección que nosotros prefiramos. Si inyectamos código ensamblado (Shellcode) y saltamos a el, podemos hacer maravillas.

- https://www.youtube.com/watch?v=T03idxny9jE
- https://www.youtube.com/watch?v=T03idxny9jE&list=PLhixgUqwRTjxglIswKp9mpkfPNfHkzyeN&index=13
- https://www.youtube.com/watch?v=xSQxaie_h1o&list=PLUl4u3cNGP62K2DjQLRxDNRi0z2IRWnNh&index=3

- Return oriented programming:
    - Podemos evitar inyectar código en secciones sin permisos de ejecución, saltando a fragmentos del código que ya existan y hagan lo que queramos. Para no perder el control, asumiendo que no hay ninguna función que hace exactamente lo que queremos. Podemos inyectar múltiples direcciones de retorno y buscar código útil que termine en un ret (gadgets). Por ejemplo:
        - Si queremos ejecutar una Shell:
            - Escribimos en el stack el dato /bin/sh.
            - Buscamos gadgets que hagan:
                - pop rax; ret
                - pop rsi; ret
                - pop rdi; ret
                - pop rdx; ret
                - syscall
            - En nuestro exploit introduciremos:

/bin/sh

Basura para rellenar el buffer

dirección del 1er gadget (pop rax; ret)

nº de la syscall para execve (59)

dirección del 2do gadget (pop rdi, ret)

dir de memoria en la que empieza la string /bin/sh

dirección del 3er gadget (pop rsi; ret)

0x0 (puntero nulo se ignora)

Dirección del 4to gadget (pop rdx; ret)

0x0

Dirección del 5to gadget (syscall)

¡Con esto conseguimos una Shell sin necesidad de inyectar ni una sola instrucción!

ROP es la técnica, si sacamos gadgets de libc podemos decir que estamos ante Ret2Libc.

- Vulnerabilidades de formato de string:
    - Si en algún caso se ve en el código *printf(input)* y nosotros controlamos ese input, podremos visualizar direcciones de memoria del proceso, con lo que podremos saltarnos la mayoría de protecciones (ASLR, PIE y SC).
- Sigreturn oriented programming:
    - Si no tenemos gadgets para todo pero :
        - Podemos escribir 300 bytes en el buffer antes del overflow.
        - Tenemos un gadget para el RAX.
    - Podemos usar la llamada 15 (rt_sigreturn) y definir un frame para modificar los registros que queramos. (Ver documentación)
- Práctica: Hackea un servidor!