Análisis de vulnerabilidades

@truerand0m

Temario día 1

DIA 1

- 1. Conceptos básicos
- 2. Lenguaje ensamblador

@truerand0m

Evaluación

- Examen Práctico ??%
- Examen Teórico ??%
- Tareas ??%
- Prácticas ??

@truerand0m

Introduccion

@truerand0m

Objetivos

- **Exposición de técnicas para realizar:**
- **análisis de vulnerabilidades**
- **ingeniería inversa**
- **desarrollo de exploits**

@truerand0m

Porque VA y RE

- **No puedes protegerte de algo que no conoces.**
- **Entender cómo funcionan los programas.**
- **Habilidad no muy común**
- **Mercado en crecimiento.**

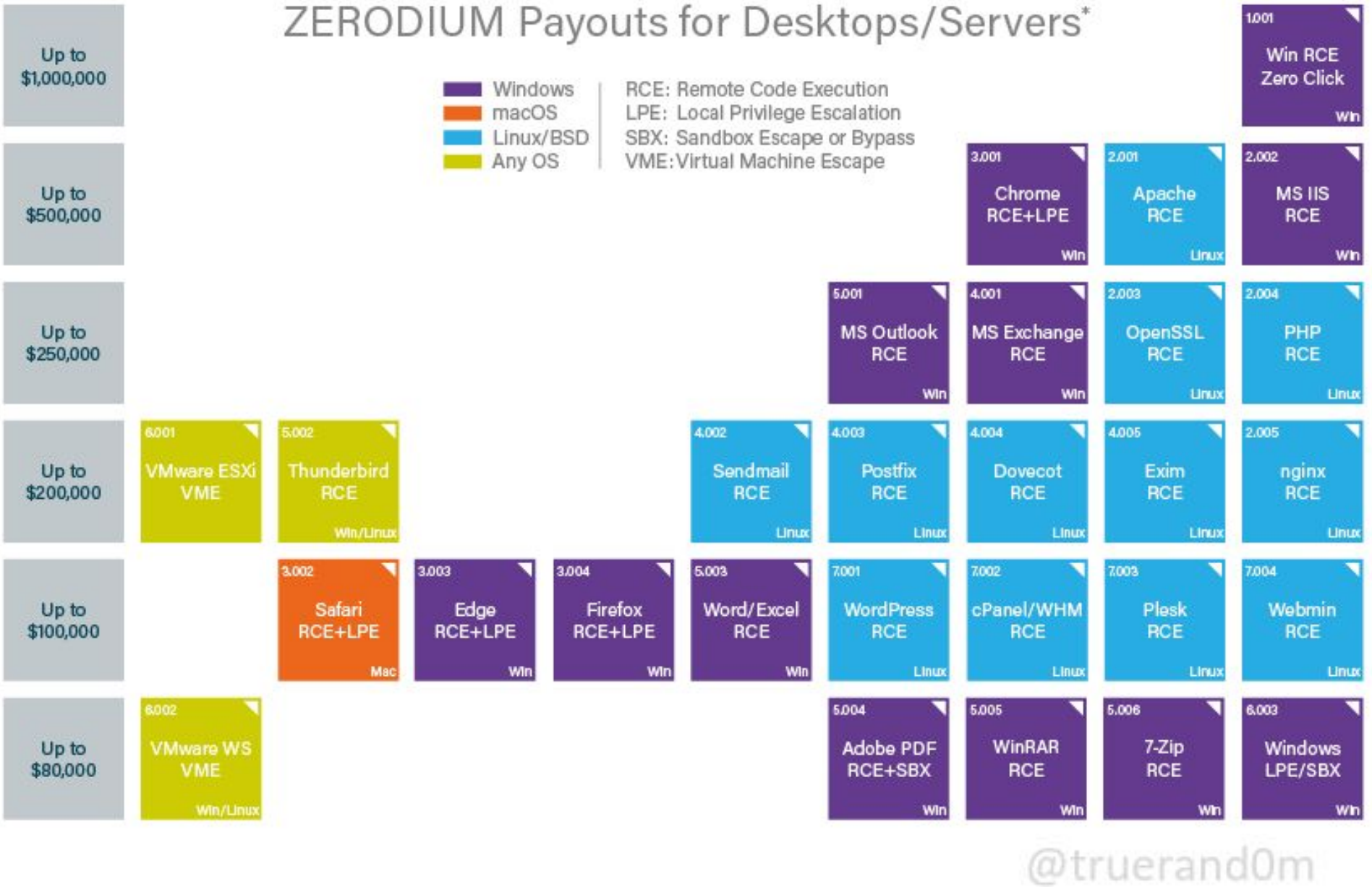
Therac-25

Dates: 1985-1987

Cause: "race condition"

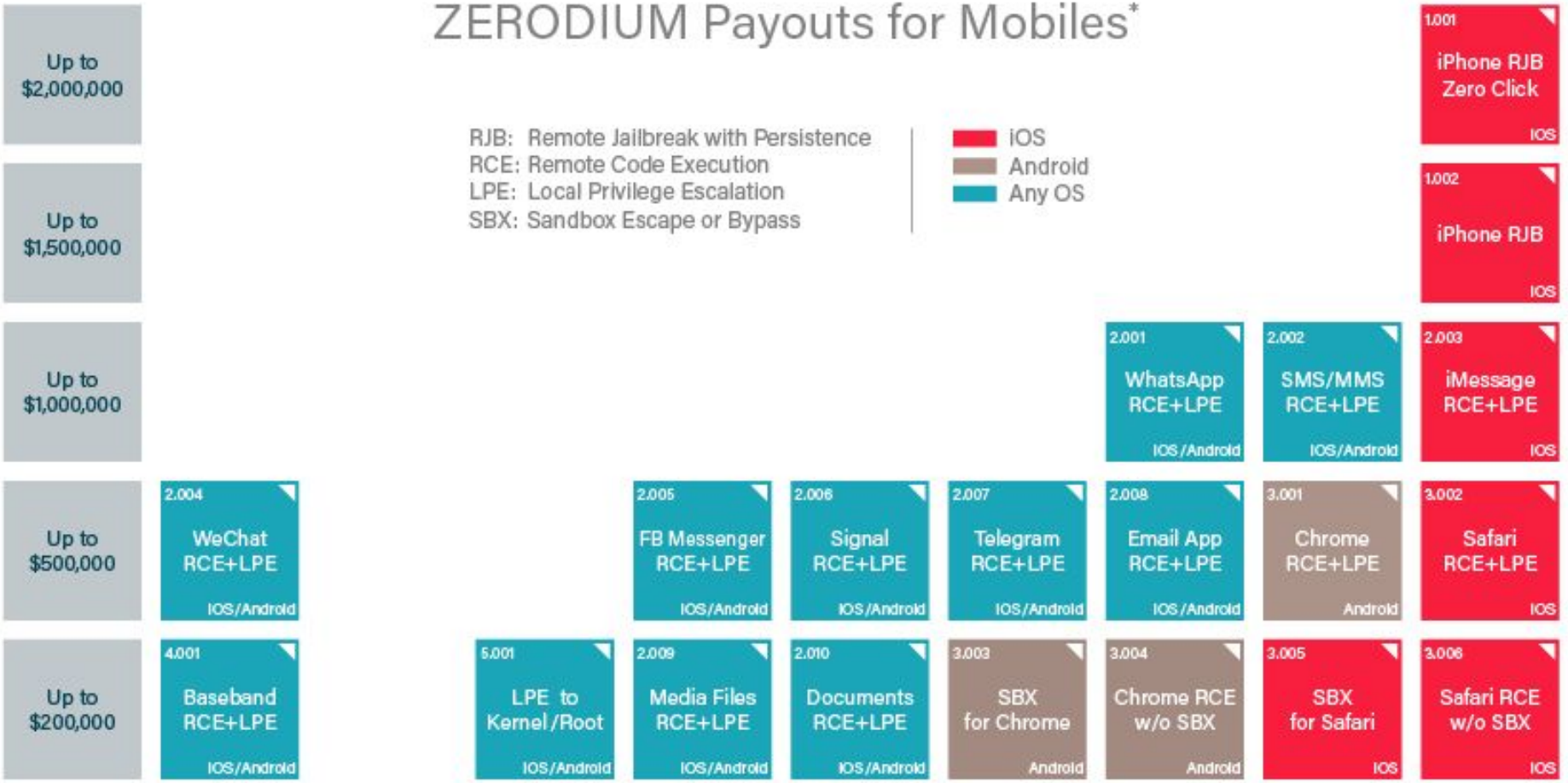
Consequences: at least 5 patients died, others were seriously injured

@truerand0m



@truerand0m

https://www.zerodium.com/program.html



@truerand0m

https://www.zerodium.com/program.html


ZERO DAY
INITIATIVE


beyond
SECURITY


SSD
SecuriTeam
Secure
Disclosure

Explotación Binaria

- Los fallos pueden convertirse en:
- Tomar el control del servidor
- Obtener información sensible
- Instalar malware
- Dañar activos de información

Explotación Binaria

Pokémon: Yellow Version "arbitrary code execution" by MrWint

https://youtu.be/zZCqoHHtovQ?t=108

ruerand0m

Historia

1972: El plan de estudios de la USAF describe los overflows

1985: Primer virus “Brain”. Infecta sector de arranque.

1988: Se propaga morris debido a un overflow

1996: Se publica ‘Smash the stack’ por aleph-1

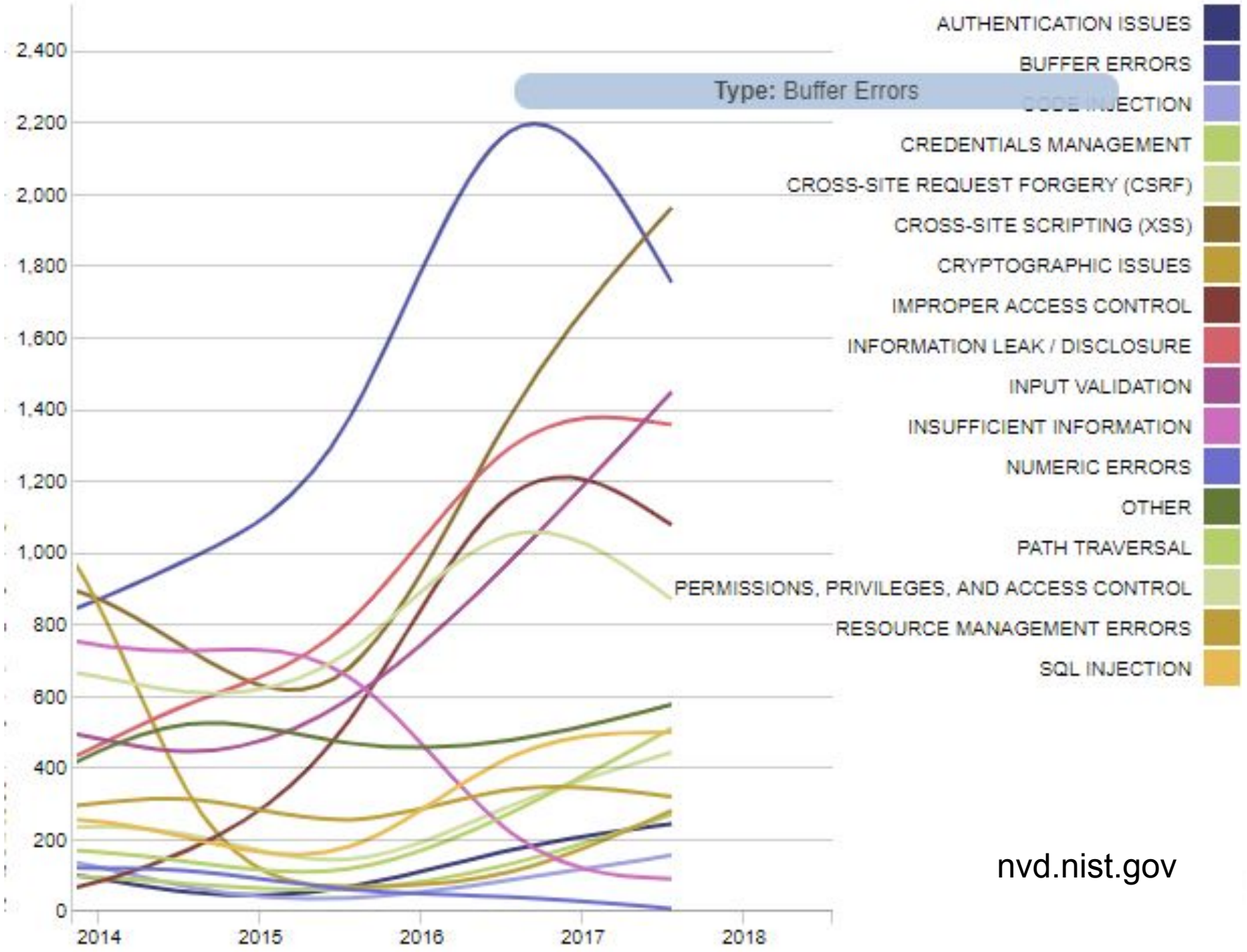
2001: Code red infecta millones de servidores

2004: XP SP2 es liberado

2007: JailBreak al iphone

2008: Stuxnet

@truerand0m


Type: Buffer Errors
2,400
2,200
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
200
0
2014
2015
2016
2017
2018
AUTHENTICATION ISSUES
BUFFER ERRORS
CODE INJECTION
CREDENTIALS MANAGEMENT
CROSS-SITE REQUEST FORGERY (CSRF)
CROSS-SITE SCRIPTING (XSS)
CRYPTOGRAPHIC ISSUES
IMPROPER ACCESS CONTROL
INFORMATION LEAK / DISCLOSURE
INPUT VALIDATION
INSUFFICIENT INFORMATION
NUMERIC ERRORS
OTHER
PATH TRAVERSAL
PERMISSIONS, PRIVILEGES, AND ACCESS CONTROL
RESOURCE MANAGEMENT ERRORS
SQL INJECTION

nvd.nist.gov

1. Conceptos básicos

@truerand0m

Conceptos básicos

- **Incidente de seguridad**: Es cualquier evento no esperado que afecte la continuidad del negocio y atente contra la confidencialidad, integridad o disponibilidad de la información
- **Riesgo**: Posibilidad de un incidente, que puede resultar en daños o pérdidas de activos de una organización
- **Amenaza**: Causa potencial de un incidente, que puede resultar en daños o pérdida de activos de una organización

@truerand0m

Conceptos básicos

- **Vulnerabilidad**: fallo o hueco de seguridad
  - Desbordamiento de buffer
  - Escalamiento de privilegios

- **Exploit**: es un código o técnica que amenaza con tomar ventaja de una vulnerabilidad.
  - Para ejecutar comandos
  - Ganar acceso a un sistema
  - Escalar provilegios

@truerand0m

Conceptos básicos

- **Payload**: Son las acciones posteriores a explotar una vulnerabilidad. Generalmente son tareas automatizadas para un determinado objetivo. (Ejemplos: crear usuarios, modificar archivos, obtener una terminal remota)

- **Compilador**: Herramienta que traduce de un lenguaje a otro. El resultado de este código máquina es llamado archivo objeto.

@truerand0m

Conceptos básicos

- **Lenguaje de alto nivel**: Lenguaje diseñado para facilitar la comprensión por parte de los humanos. Es convertido a código máquina por un compilador.
- **Lenguaje de bajo nivel**: Es la versión legible del conjunto de instrucciones (mnemónicos) de una arquitectura de computadoras, conocido como lenguaje ensamblador.

@truerand0m

Conceptos básicos

- **Código máquina**: Consiste en opcodes (dígitos en hexadecimal) que le indican al procesador lo que debe realizar. Se crea cuando un programa en lenguaje de alto nivel es compilado.

@truerand0m

Creación de un ejecutable

- En primera instancia el código fuente es traducido por un compilador a código objeto.
- Posteriormente el enlazador proporciona las bibliotecas requeridas, se puede ligar de dos maneras: estáticamente o dinámicamente.
  - Ligado estático: las bibliotecas requeridas se colocan como parte del ejecutable.
  - Ligado dinámico: las bibliotecas requeridas se satisfacen al momento de la ejecución.

@truerand0m

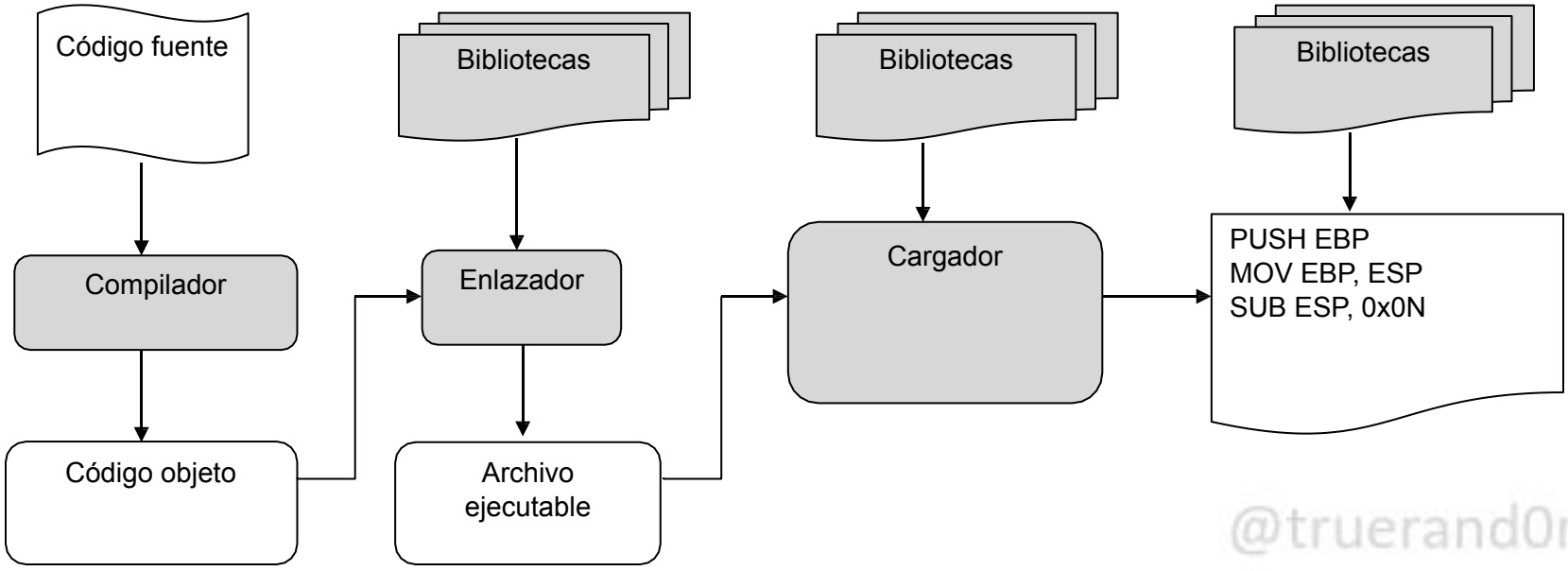
Creación de un ejecutable

- El resultado del enlazador es un archivo ejecutable.
- Finalmente, cuando el archivo ejecutable es utilizado, éste es colocado en memoria por el cargador, el cual se encarga de reservar memoria y de llamar al enlazador dinámico para resolver las dependencias restantes.
- Como consecuencia se carga el *opcode* necesario para la ejecución del programa.

@truerand0m

Creación de un ejecutable

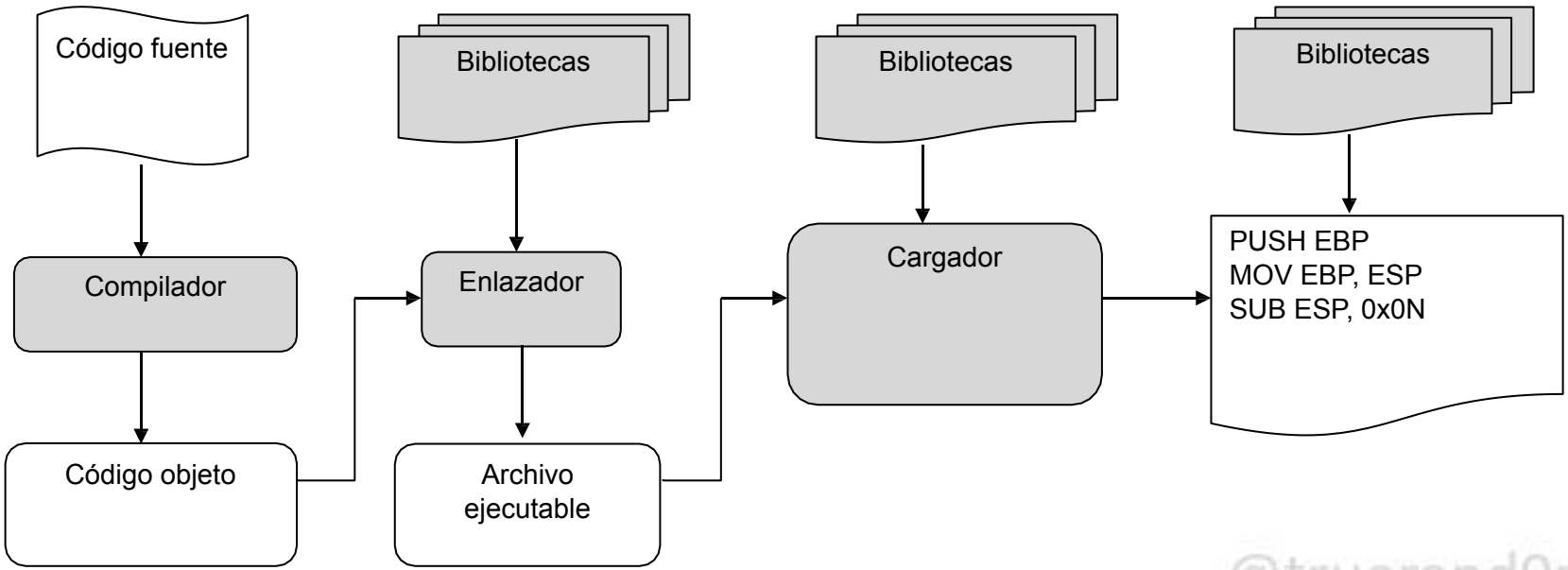
- El proceso para convertir código fuente a un ejecutable, tiene que pasar por distintos componentes como se muestra en la siguiente imagen:



@truerand0m

Creación de un ejecutable

- nasm -f elf32 -o $fuente.obj ;
- ld -m elf_i386 -o $fuente $fuente.obj
  file hello_world.obj
  objdump -m Intel -h hello_world.obj



@truerand0m

2. Lenguaje ensamblador

@truerand0m

Lenguaje ensamblador

- Es el lenguaje de bajo nivel por excelencia.
- Las instrucciones en lenguaje ensamblador son conocidas como mnemónicos.
- Un ensamblador traduce las instrucciones a código máquina.

@truerand0m

Sintaxis básica

- Una instrucción en ensamblador está compuesta por los siguientes elementos:

Sintaxis básica			
[Etiqueta:]	Mnemónico	[Operando(s)] (Destino,Origen)	[; Comentario]

@truerand0m

Sintaxis básica

- **Mnemónico**. Mnemónico de la instrucción.
- **Operandos**. Contiene los datos requeridos por la instrucción, pueden ser valores constantes, provenir de direcciones en memoria o de registros. El número de operandos pueden ser 0, 1 o 2: (

  RET;

  INC EAX;

  MOV EAX, 1

  respectivamente).

@truerand0m

Sintaxis básica

- **Etiqueta**. Sirve como punto de control para el flujo del programa. Debe comenzar con un carácter alfabético o un punto, el resto de la etiqueta puede contener letras, números o signos especiales [-$.@%].

- **Comentario**. Información adicional para propósitos de documentación. Cualquier texto posterior al signo ';' será excluido de la compilación.

@truerand0m

Acceso a datos

- Para hacer referencia a los datos se pueden utilizar:
  - Valores inmediatos. Un dato es almacenado en uno de los registros de propósito general. (MOV EAX, 0x01)
  - Registros. El dato contenido en un registro es almacenado en otro registro. (MOV EAX, EBX).

@truerand0m

```
section .text
global _start                    ; must be declared for linker (ld)
_start:                          ; entry point
    mov eax,0x1234               ; movs valores inmediatos
    mov ebx,0x5678
    mov edi,eax                  ; movs entre registros
    mov esi,ebx
    mov edi,mem1                 ; movs direcciones
    mov esi,mem2                 ; check x/x &mem1, content x/s &mem1

    mov [name],dword 'Done'      ; x/s 0x804...
    mov eax,1                    ; system call number (sys_exit)
    int 0x80                     ;call kernel

section .data
    name: db 'Test', 0xa
    mem1: db "hello",0xa
    mem2: db "world",0xa
```

@truerand0m

Acceso a datos

- Localidades de memoria
  - **Direccionamiento directo**: busca y accede a los datos en la dirección de memoria especificada, el registro es el destino. (MOV EAX, **[**0x08040200**]**).
  - **Direccionamiento indirecto**: también llamado "direccionamiento por referencia" calcula la dirección del destino, llamada "dirección efectiva" (en el registro reside la dirección del destino). (MOV EAX, [EBX+8])
  - **Implícitos**. El operando está definido dentro de la misma instrucción, dichos operandos pueden ser registros o la pila. (PUSH 0x23
    PUSH CLC).

@truerand0m

Registros

- Se tienen 8 registros de propósito general, cada registro tiene un fin específico, dependiendo del tipo de instrucción que se haya ejecutado.

- Estos son: **EAX, EBX, ECX, EDX, EBP, ESP, ESI y EDI**, donde
  la letra “E” inicial proviene de “extended”.

@truerand0m

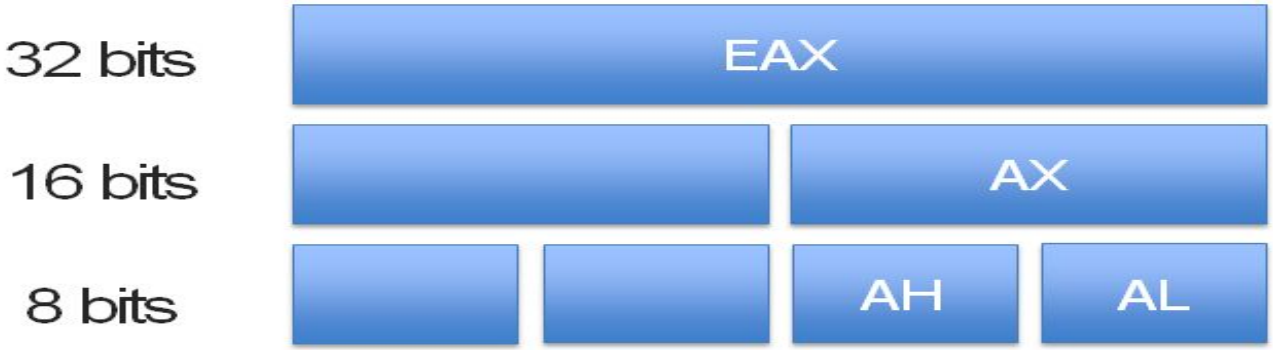
Registros

- EAX: Registro acumulador, utilizado en operaciones aritméticas (suma, resta, multiplicación y división), almacenar valores de retorno de funciones y para lectura/escritura en periféricos de I/O.
- EBX: Registro base o apuntador, utilizado para indicar el desplazamiento de direcciones en segmentos de memoria y almacenar datos.
- ECX: Registro contador, utilizado en operaciones iterativas (ciclos/loops).
- EDX: Registro de datos, utilizado para realizar operaciones aritméticas junto con EAX.

@truerand0m

Registros

- Se puede acceder a los primeros 16 bits de EAX,EBX,ECX y EDX, omitiendo la ‘E’, especificandolos de la siguiente forma:AX, BX, CX y DX.
- Se puede acceder a los 8 bits más significativos de AX,BX,CX y DX, usando AH, BH, CH y DH.
- Se puede acceder a los 8 bits menos significativos de AX,BX,CX y DX, usando AL, BL, CL y DL.



@truerand0m

Registros

- ESP (*Stack Pointer*): Apuntador de pila, contiene la dirección de memoria del último valor almacenado en la pila que está usando ~~el~~ un programa en ese momento.
- EBP (*Base Pointer*): Apuntador base, indica la dirección inicial de la pila y es usado para referenciar argumentos y variables locales.

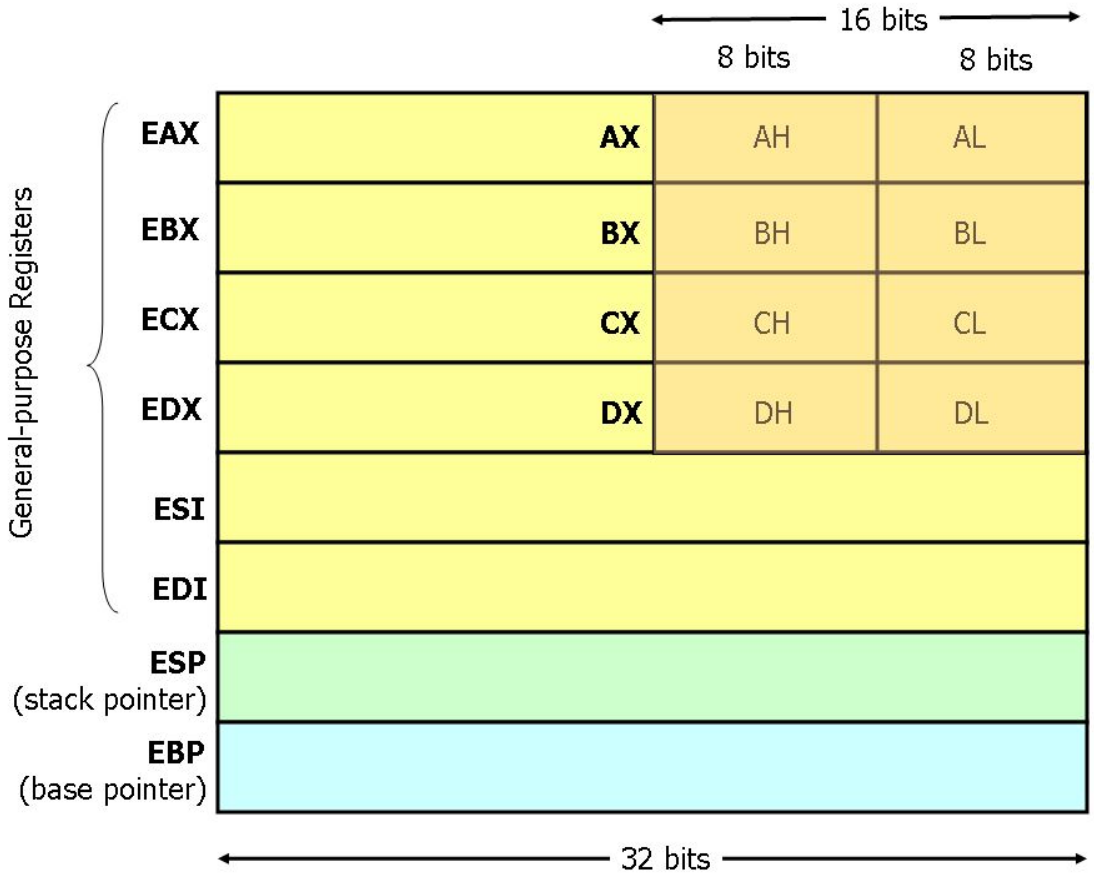
@truerand0m

Registros

- ESI (*Source Index*): Índice origen, indica dónde se encuentra el búfer de datos de entrada.
- EDI (*Destination Index*): Índice destino, contiene la dirección donde se copiará el búfer de datos indicado por ESI.

@truerand0m

Registros



http://www.cs.virginia.edu/~evans/cs216/guides/x86.html

@truerand0m

Registros x64

- Los registros de propósito general en procesadores de **arquitectura x64** (64 bits) son 16: RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8, R9, R10, R11, R12, R13, R14 Y R15.

- Registro apuntador a instrucciones:

  EIP (*Instruction Pointer*): Registro que contiene la dirección de memoria de la siguiente instrucción a ejecutar.
  También conocido como contador.

@truerand0m

Registros

- Registro de banderas o de estado (EFLAGS).

Las banderas indican el estado actual de la computadora dependiendo del procesamiento de instrucciones, ya que las comparaciones o cálculos aritméticos cambian su estado. El registro está formado por 32 bits, donde cada bandera es un bit.

@truerand0m

Registros

- Algunas banderas importantes son:
  - *Sign* (SF - 7): Se establece en 1 si el resultado tiene signo negativo, si es positivo tomará el valor 0.
  - *Zero* (ZF - 6): Se establece en 1 si el resultado de una instrucción fue 0.
  - *Trap* (TF - 8): Ejecución paso a paso.

@truerand0m

Registros

- *Overflow* (OF - 11): Se establece si el resultado de una instrucción aritmética con signo genera un número el cual es demasiado largo para el destino.
- *Interruption* (IF - 9): Indica si una interrupción de los dispositivos externos se procesa o no, está reservada para el sistema operativo en modo protegido.
- *Carry* (CF - 0): Se establece si el resultado de una operación aritmética acarrea (adición) o toma prestado (resta) un bit más allá del MSB

@truerand0m

Tipos de instrucciones

- En lenguaje ensamblador, existe un número considerable de instrucciones las cuales pueden clasificarse en las siguientes categorías.

  - Manipulación de datos
  - Transferencia de datos
  - Instrucciones condicionales e instrucciones para ramificación

@truerand0m

Tipos de instrucciones

- Manipulación de datos
  - Instrucciones aritméticas: ADD, SUB, MUL, IMUL, DIV, IDIV, INC, DEC.
  - Operaciones booleanas: NOT, AND, OR, XOR.
  - Manipulación de bits: SHR, SHL, ROR, ROL.

- Transferencia de datos
  - Incluye instrucciones como MOV, XCHG, PUSH y POP.

@truerand0m

Tipos de instrucciones

- Instrucciones condicionales e instrucciones para ramificación
  - Saltos (JMP, JZ, JNZ, JE, JNE, LOOP, etc.)*
  - Llamadas y servicios (CALL, RET, IRET e INT)*
  - Comparaciones (NEG, CMP, TEST, etc.)
  - **Nota:** se modifica el valor de EIP

@truerand0m

- **Aritmética**

El formato de la instrucción de la adición es

“ADD destino,valor” y el de la sustracción es

“SUB destino, valor”.

La sustracción modifica dos banderas importantes:

- **ZF** *(zero flag)*; se establece en 1 si el resultado de la operación es 0.

- **CF** *(carry flag)*; se establece si se efectúa un préstamo en la sustracción y replica el valor saliente en corrimientos y rotaciones.

@truerand0m

lenguaje ensamblador

- **Aritmética**

  - ADD EAX, EBX ; Asigna el valor de EBX a EAX y almacena el resultado en EAX
  - SUB EAX, 0x10 ; Resta 10 en hexadecimal a EAX
    - EAX-=16
    -

Las instrucciones **INC** y **DEC** incrementan o decrementan un registro en uno.

  - INC EDX ; Incrementa EDX en 1
  - DEC ECX ; Decrementa ECX en 1

@truerand0m

lenguaje ensamblador

```
section .text
global _start
_start:
    mov eax,0x1                     ; eax=1
    add eax,0x2                     ; eax+=2
    ; bl+= (byte en la dir de memoria a la que apunta mem1
    add bl, byte [mem1]
    mov dl,0x5                      ; lowest 8 bits edx  = 5
    ; resta a lo que haya en la dirección de memoria mem2 dl
    sub byte [mem2], dl
section .data
    mem1: db 0x2
    mem2: db 0x8
```

@truerand0m

lenguaje ensamblador

- **Aritmética**

El formato de la multiplicación sin signo es **“MUL valor”** y **siempre multiplica al registro EAX**, por lo que este último debe estar configurado apropiadamente antes de que ocurra la operación.

El producto es almacenado como resultado de 64 bits a través los registros EDX y EAX. EDX almacena los 32 bits más significativos de la operación y EAX almacena los 32 bits menos significativos.

@truerand0m

lenguaje ensamblador

- **Aritmética**

Multiplicador (valor)	Multiplicando	Producto
8 bits	AL	+ AH:AL -
16 bits	AX	+ DX:AX -
32 bits	EAX	+ EDX:EAX -

@truerand0m

lenguaje ensamblador

- **Aritmética**

```
MOV EAX,0x44332211 ; Asigna el valor 0x44332211 a EAX
MUL 0x50 ; Multiplica EAX (0x44332211) por 0x50
         ; y almacena el resultado (154FFAA550) en EDX:EAX
```

@truerand0m

lenguaje ensamblador

- **Aritmética**

El formato de la división sin signo es **“DIV valor”** y hace lo mismo que la instrucción **MUL** pero de manera opuesta, divide EDX y EAX entre un valor, por lo que estos últimos deben estar configurados apropiadamente antes de que ocurra la operación.

@truerand0m

lenguaje ensamblador

- **Aritmética**

El cociente se almacena en EAX (AX o AL) y el residuo se almacena en EDX (DX o DL).

```
MOV EDX,0x0
MOV EAX,0x150
DIV 0x75      ; Divide los registros EDX:EAX (0x150) entre
              ; 0x75 y almacena el resultado en el registro EAX
              ; y el residuo en EDX
```

@truerand0m

lenguaje ensamblador

- **Aritmética**

Divisor (valor)	Dividendo	Cociente	Residuo
8 bits	AX	AL	AH
16 bits	DX:AX	AX	DX
32 bits	EDX:EAX	EAX	EDX

@truerand0m

lenguaje ensamblador

- **Boolean**

Instrucción NOT: Lleva a cabo la negación bit a bit, es decir, invierte todos los bits y el resultado se guarda en el mismo operando.

- MOV EAX, 0xF049 ; EAX = 0xF049
- NOT EAX ; EAX = 0x0FB6

@truerand0m

lenguaje ensamblador

A B	A·B
0 0	0
0 1	0
1 0	0
1 1	1

- **Boolean**

Instrucción AND: Realiza la conjunción bit a bit y el resultado se guarda en el operando destino (AND destino, origen).

- MOV EAX, 0xF049 ; EAX = 0xF049
- MOV EBX, 0x05CA ; EBX = 0x05CA
- AND EAX, EBX ; EAX = 0x0048 y EBX = 0x05CA

@truerand0m

lenguaje ensamblador

A B	A+B
0 0	0
0 1	1
1 0	1
1 1	1

- **Boolean**

Instrucción OR: Realiza la disyunción bit a bit y el resultado se guarda en el operando destino (OR destino, origen).

- MOV EAX, 0xF049 ; EAX = 0xF049
- MOV EBX, 0x05CA ; EBX = 0x05CA
- OR EAX, EBX ; EAX = 0xF5CB y EBX = 0x05CA

@truerand0m

lenguaje ensamblador

A B	A^B
0 0	0
0 1	1
1 0	1
1 1	0

- **Boolean**

Instrucción XOR: Realiza la disyunción exclusiva bit a bit y el resultado se guarda en el operando destino (XOR destino, origen).

En algunos lenguajes de programación se usa el acento circunflejo para efectuar la operación XOR.

@truerand0m

```
global _start
section .text
_start:
        mov eax,0x11111111          ; and
        mov edx,0x10101010
        and eax,edx
        mov eax,0x11010101          ; or
        mov edx,0x10101010
        or eax,edx
        mov eax,0x11010011          ; xor
        mov edx,0x11011101
        xor eax,edx
        mov al,0b11001100          ; not
        not al
```

@truerand0m