Ciberseguridad

23 de agosto de 2025

José Luis

Índice general

1. Temario tentativo

Temario en orden cronológico: Wed 16 Jul 25

(1) Redes y protocolos

(2) Sistemas operativos

(3) Programación baja, vulnerabilidades y reversing

(4) Pentesting

(5) Hardening, blue team y forensics

(6) Criptografía

1.1. Algunos objetivos a mediano plazo

- Completar libros.
- Resolver máquinas o labs como `Hack The Box`, `PicoCFT`, `Root-Me`.
- Experimentar con laboratorios pequeños en casa: una red virtual con PfSense, Kali Linux, Window vulnerable, etc.
- Construir un portafolio de aprendizajes.
- Sacar las certificaciones eLearnSecurity Junior Penetration Tester v2 (`eJPTv2`), eLearnSecurity Certified Professional Penetration Tester v2 (`eCPPTv2`), eLearnSecurity Web Application Penetration Testing (`eWPT`), CompTIA Security +, Offensive Security Certified Professional (`OSCP`), `CISM` o `CISSP`.

2. Networking

2.1. Algunas definiciones básicas Thu 17 Jul 25

La **Internet** es una red de computadoras que conecta a cientos de millones de dispositivos informáticos en todo el mundo. Estos dispositivos incluyen PC's, Linux workstations, servers, laptops, smartphones, tablets, TV's, consolas de videojuegos, Webcams, automóviles, dispositivos de detección ambiental o sistemas domésticos eléctricos y de seguridad. Cualquier dispositivo que se conecte a Internet se llamará **host** o **end system**, es decir, anfitrión o sistema final.

Wed 23 Jul 25

Los sistemas finales se conectan entre sí con una red de **communication links** o enlaces de comunicación, y **packet switches** o conmutadores de paquetes. Diferentes enlaces pueden transmitir datos a diferentes tasas, siendo la **transmission rate** o tasa transmisión de un

enlace medida en `bits/second`. Cuando un sistema final envía datos a otro sistema final, el sistema final que envía segmenta los datos y agrega **header bytes** o bytes de encabezado a cada segmento. Los paquetes de información resultantes, conocidos como **packets** o paquetes en la jerga de redes, se envían a través de la red hacia el sistema final de destino, donde son reensamblados en los datos originales.

Un conmutador de paquetes toma paquetes que llegan de alguno de sus enlaces de comunicación entrantes y reenvía el paquete a uno de sus enlaces de comunicación salientes. Existen varios tipos de conmutadores de paquetes, pero los más destacados hoy en día son los **routers** o enrutadores, y los **link-layer switches** o conmutadores de capa de enlace. Ambos conmutadores reenvían los paquetes hacia su destino final. Los conmutadores de capa de enlace generalmente se usan en **access networks** o redes de acceso, mientras que los enrutadores se suelen usar en el **network core** o el núcleo de red. La sucesión de enlaces de comunicación y conmutadores de paquetes atravesados por un paquete del sistema final que envía al sistema final que lo recibe, se llama **route** o **path**, o ruta o camino del paquete.

Los sistemas finales acceden a Internet a través de un **Internet Service Provider (ISP)**, incluyendo ISPs residenciales como compañías locales de cable o teléfono, ISPs corporativos, ISPs universitarios y ISPs que proveen acceso a WiFi en aeropuertos, hoteles, cafeterías y otros lugares públicos. Cada ISP es en sí misma una red de enlaces de comunicación y conmutadores de paquetes.

Los ISPs proporcionan una variedad de tipos de acceso a los sistemas finales, incluyendo *residential broadband access* o acceso banda ancha residencial como un módem de cable o Digital Subscriber Line (DSL), *high-speed local area network access* o acceso a red de área local de alta velocidad, *wireless access* o acceso inalámbrico, o *56kbps dial-up modem access* o acceso a módem telefónico de 56kbps. Los ISPs también proveen acceso a Internet a proveedores de contenido, conectando los sitios Web directamente a Internet. Los ISPs también están conectados entre sí para dar acceso a más sistemas finales. Estos **lower-tier ISPs** o ISPs de nivel inferior están conectados a través de **upper-tier ISPs** o ISPs de nivel superior nacionales e internacionales como Level 3 Communications, AT&T, Sprint y NTT. Cada red ISP se gestiona de forma independiente, usa el protocolo IP y se adhiere a ciertas convenciones de nombres y direcciones.

Los sistemas finales, conmutadores de paquetes y otras piezas de Internet ejecutan protocolos que controlan el envío y recibo de información dentro de Internet. El **Transmission Control Protocol (TCP)** y el **Internet Protocol (IP)** son dos de los más importantes. El protocolo IP especifica el formato de los paquetes que se envían y reciben entre los enrutadores y los sistemas finales. Los protocolos principales de Internet se conocen colectivamente como **TCP/IP**. Para estableces estándares de protocolos se crearon los **Internet standards** y son desarrollados por la Internet Engineering Task Force (IETF). Los documentos estándar de la IETF se llaman **request for comments (RFCs)** y entre estos están las definiciones de los protocolos TCP, IP, HTTP (para la Web) y SMTP (para e-mail). Para especificaciones de estándares de componentes de red y en particular enlaces de red, existen otros cuerpos de información. Por ejemplo, el IEEE 802 LAN/MAN Standards Committee especifica los estándares de Ethernet y wireless WiFi.

Tue 29 Jul 25

La Internet también se puede describir como una *infraestructura que provee servicios a aplicaciones*. Entre estas aplicaciones se encuentran el *electronic mail* o correo electrónico, *Web surfing* o navegación web, *social networks* o redes sociales, *instant messaging* o mensajería instantánea, *Voice over-IP* (voIP), *video streaming* o transmisión de video, *distributed games* o juegos distribuidos, *peer-to-peer (P2P) file sharing* o intercambio de archivos peer-to-peer, *remote login* o acceso remoto, etc. Estas aplicaciones se llaman **distributed applications** o aplicaciones distribuidas porque involucran varios sistemas finales que intercambian datos entre sí.

Las aplicaciones de Internet se ejecutan en sistemas finales y no en los conmutadores de paquetes. Los sistemas finales conectados a Internet proveen una **Application Programming Interface** (**API**) o Interfaz de Programación de Aplicaciones, que especifica cómo un programa ejecutándose en un sistema final solicita a la infraestructura de Internet el envío de datos a un programa destino específico ejecutándose en otro sistema final. Finalmente, un **protocol** o protocolo, define el formato y el orden de los mensajes intercambiados entre dos o más entidades de comunicación, así como las acciones tomadas el la transmisión y/o recepción de un mensaje u otro evento. Dominar el campo de las redes informáticas es equivalente a entender el qué, por qué y cómo de los protocolos de red.

2.2. Redes borde y de acceso

El nombre "end systems" o sistemas finales se debe a que estos dispositivos se encuentran en la **networkd edge** o borde de la red. También son referidos como *hosts* porque alojan y ejecutan programas de aplicaciones como navegadores Web, servidores Web o clientes de e-mail. Los hosts se pueden dividir en las categorías de **clientes** y **servers**, o clientes y servidores. Informalmente, los clientes suelen ser PC'de escritorio, smartphones, etc., y los servidores suelen ser máquinas más poderosas que alojan y distribuyen páginas Web, transmiten video, relay e-mail o retransmiten correo electrónico, etc. Hoy en día, muchos de los servidores de los que recibimos resultados de búsqueda, páginas Web y videos residen en grandes **data centers** o centros de datos.

La **access network** o red de acceso es la red que conecta físicamente al sistema final con el primer enrutador, también conocido como el **edge router** a un camino desde este sistema final a otro sistema final lejano.

3. Operating Systems

Thu 17 Jul 25

Existen tres conceptos clave para entender como funciona un sistema operativo, cómo decide qué programa ejecutar después en la CPU, cómo maneja la sobrecarga de memoria en un sistema de memoria virtual, cómo funcionan los monitores de máquinas virtuales, cómo manejar la información en discos o incluso cómo construir un sistema distribuido que funcione cuando algunas partes hayan fallado. Estos conceptos son los de **virtualización**, **concurrencia** y **persistencia**.

3.1. Sistemas informáticos

El código máquina `x86-64` es el último en un camino de evolución seguido por Intel y su competencia que empezó con el procesador `8086` en 1978. Esta clase de procesadores se conoce coloquialmente como `x86`. Consideraremos como las máquinas ejecutan código `C` en Linux. Otros sistemas operativos que tienen herencia del sistema operativo Unix son Solaris, FreeBSD y MacOS X. Estos sistemas han mantenido un buen nivel de compatibilidad gracias a Posix y al Single UNIX Specification. Se puede usar Linux en un entorno de máquina virtual como VirtualBox o VMWare, que permiten ejecutar programas escritos para un sistema operativo, el *guest OS* en otro, el *host OS*.

Tue 29 Jul 25

Un *computer system* o sistema informático, consiste en sitemas de hardware y software que trabajan juntos para ejecutar programas de aplicación. Implementaciones específicas de los sistemas cambian con el tiempo, pero los conceptos subyacentes no. Todos los sitemas informáticos poseen sistemas de hardware y software similares que llevan a cabo funciones similares.

La vida un programa empieza con un *source program* o *source file*, o programa fuente o archivo fuente que el programador crea con un editor de texto y guarda en un archivo de texto. El programa fuente es una sucesión de *bits*, cada uno con valor 0 o 1, organizados en *chunks* o fragmentos de ocho bits llamados *bytes*. Cada byte representa un caracter de texto en el programa. La mayoría de sistemas informáticos representan los caracteres de texto usando el estándar *ASCII* que representa cada caracter con un entero único del tamaño de un byte, aunque otros métodos de codificación se usan para representar texto en lenguages distintos al inglés. Los archivos que consisten únicamente en caracteres ASCII se conocen como *text files* o archivos de texto. Cualquier otro archivo se conoce como un *binary file* o archivo binario.

Los programas fuente están diseñados para ser leídos y escritos por humanos, sin embargo, para ejecutarlos en el sistema, las declaraciones individuales del lenguage en uso deben traducirse con otros programas en una sucesión de *low-level machine language instructions* o instrucciones en lenguage máquina de bajo nivel. Estas instrucciones se empaquetan en un formato llamado *executable object program* o programa objeto ejecutable y almacenado como un *binary disk file* o archivo de disco binario. A los programas objeto también se les llama *executable object files* o archivos objeto ejecutables. En un sistema Unix, la traducción del archivo fuente al archivo objeto la realiza un *compiler driver* o unidad de compilación.

Por ejemplo, el `GCC Compiler` transforma un archivo fuente `file.c` en un programa objeto ejecutable `file`. La traducción se realiza en las cuatro fases siguientes y los programas que la realizan son conocidos colectivamente como el *compilation system* o el sistema de compilación:

- *Preprocessing phase* o fase de preprocesamiento. El *preprocessor* o preprocesador (`cpp`) modifica el programa `C` original de acuerdo con las directivas que empiezan con el caracter `#`. Por ejemplo, el comando `#include <stdio.h>` le dice al preprocesador que lea los contenidos del *system header file* o archivo de encabezado del sistema `stdio.h` y los inserte directamente en el programa texto. El resultado es otro programa en `C`, típicamente con el sufijo `.i`.

- *Compilation phase* o fase de compilación. El *compiler* o compilador (`cc1`) traduce el archivo de texto `file.i` en el archivo de texto `file.s`, el cual contiene un *assembly-language program* o programa en lenguage ensamblador. El lenguage ensamblador es útil porque provee un lenguage de salida común para diferentes compiladores de distintos programas de alto nivel.
- *Assembly phase* o fase de ensamblado. Después, el *assembler* o ensamblador (`as`) traduce el archivo `file.s` en instrucciones en lenguage máquina y las empaqueta en un formato llamado *relocatable file object* u archivo objeto reubicable, y almacena el resultado en el archivo objeto `file.o`.
- *Linking phase* o fase de enlace. Si nuestro programa `file` llamara, por ejemplo, a la función `printf`, la cuál es parte de la `standard C library` proporcionada por cualquier compilador de `C`, esta tendría que ser fusionada con nuestro programa `file.o`, y se encuentra en otro archivo precompilado llamado `printf.o`. El *linker* o enlazador se encarga de esta fusión y el resultado es un *executable object file* o archivo objeto ejecutable, o simplemente llamado ejecutable, en este caso llamado `file` listo para ser cargado en la memoria y ejecutado por el sistema.

Para ejecutar este archivo ejecutable en un sistema Unix, se escribe su nombre en un programa de aplicación conocido como *shell*:

```
1 linux> ./hello
2 hello, world
3 linux>
```

La shell es un *command-line interpreter* o intérprete de línea de comandos que imprime un *prompt* o mensaje de aviso, espera a que se escriba un comando y luego ejecuta el comando. Si la primera palabra del comando no corresponde a un *built-in shell command* o comando de shell integrado, entonces la shell asume que es el nombre de un archivo ejecutable que debe cargar y ejecutar.

3.2. Organización del hardware de un sistema

- *Buses*. A través del sistema existen conductos eléctricos llamados *buses*, que llevan información de un lado a otro entre componentes. Los buses generalmente están diseñados para transferir *chunks* o fragmentos de bytes de tamaño fijo, conocidos como *words* o palabras. El número de bytes en una palabra es un parámetro fundamental del sistema. Muchas máquinas hoy en día tienen tamaños de palabra de 4 bytes (32 bits) u 8 byes (64 bits).
- *I/O Devices*. Los *I/O Devices* o dispositivos de entrada y salida, son la conexión entre el sistema y el mundo externo. Por ejemplo, el ratón y el teclado, el monitor o un disco duro externo.

- *Main Memory*. Es un almacenamiento temporal que contiene un programa y los datos que este manipula cuando el procesador está ejecutando el programa. Físicamente, la memoria principal consite en una colección de chips de *dynamic random access memory* (DRAM). Lógicamente, la memoria se organiza como un arreglo lineal de bytes, cada uno con su dirección única (*array index*), empezando en cero.

- *Processor*. La *central process unit* (CPU), o unidad central de procesamiento, o simplemente el procesador, es la máquina que ejecuta las instrucciones almacenadas en la memoria principal. En su núcleo está un *word-size storage device* o (*register*), un dispositivo de almacenamiento del tamaño de una palabra, llamado *program counter* (PC). En cualquier momento, el PC apunta (contiene la dirección) de alguna instrucción en lenguage máquina en la memoria principal. Desde que el sistema recibe energía eléctrica hasta que se le deja de suministrar, el procesador ejecuta repetidamente la instrucción hacia la que apunta el contador de programa y actualiza este para apuntar a la siguiente instrucción. El procesador aparenta funcionar de acuerdo a un modelo simple de instrucciones, definido por su *instruction set architecture*, o arquitectura del conjunto de instrucciones.

  Existen pocas de estas operaciones simples y giran alrededor de la memoria principal, del *register file* o archivo de registro, y de la *arithmetic/logic unit* o unidad aritmética/lógica (ALU). El archivo de registro es un pequeño dispositivo que consiste en una colección de registros del tamaño de una palabra, cada uno con un nombre único. La ALU calcula nuevos datos y valores de dirección. Se puede distinguir la arquitectura del conjunto de instrucciones de un procesador, que describe el efecto de cada instrucción, y su *microarquitectura*, que describe cómo el procesador está realmente implementado.

4. Reversing

4.1. Reverse Engineering

Thu 17 Jul 25

La **ingeniería inversa** es un conjunto de técnicas y herramientas para entender lo que hace un software en realidad. Formalmente, es "el proceso de analizar un sistema sujeto para identificar los componentes del sistema y su interrelación, así como crear una representación del sistema de una forma diferente o a un nivel mayor de abstracción." [3]. La **ingeniería inversa binaria** tiene como objetivo extraer información valiosa de programas cuyo código fuente no está disponible. En algunos casos, la persona o grupo de personas que conocen el sistema, no quieren que la información extraída sea compartida. En otros casos, dicha información suele estar perdida o ha sido destruida.

A la ingeniería inversa de software se le suele llamar coloquialmente **reversing**. La práctica del **reversing** puede ser utilizada por desarrolladores de cógido malicioso para detectar vulnerabilidades en sistemas operativos y demás software. Tales vulnerabilidades se pueden usar para penetrar las capas defensa del sistema y permitir la infección. Generalmente, los culpables utilizan dichas vulnerabilidades para permitir que programas maliciosos ganen acceso a

información sensible o incluso tomar control total del sistema. Por otro lado, los desarrolladores de antivirus para software analizan programas maliciosos y usan técnicas de ingeniería inversa para rastrear todas las etapas del programa y evaluar el daño que podría causar, la tasa esperada de infección, cómo se podría remover de sistemas infectados y si la infección se puede evitar por completo.

4.2. Low-level software

Low-level software, **system software**, o en español, **software de bajo nivel**, se refiere a la infraestructura del mundo del software. Caben dentro de este concepto los compiladores, linkers, debuggers, sistemas operativos y lenguages de programación de bajo nivel como el lenguage ensamblador. Los aspectos low-level de un programa son a menudo la única cosa con la que se tiene que trabajar haciendo ingeniería inversa. La persona que quiera aplicar ingeniería inversa a software debe ser consciente de literalmente cualquier cosa que se interponga entre el código fuente del programa y la CPU.

5. Pentesting

Bibliografía

[1] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, Inc., 2014. `http://www.ostep.org`.

[2] Randal E. Bryant and David R. O'Hallaron. *Computer Systems: A Programmers Perspective*. Pearson Education, 3 edition, 2016.

[3] Eldad Eilam. *Reversing: Secrets of Reverse Engineering*. Wiley Publishing, Inc., 2005.

[4] James F. Kurose and Keith W. Ross. *Computer Networking*. Pearson Education, 6 edition, 2013.