Apuntes 4: Viernes 27 de octubre

Diego Granados Retana 2022158363

Bases de datos II

27 de octubre, 2023

Apuntes de la clase:

Una aplicación normalmente se organiza en capas. Hace mucho tiempo se usaba el Model View Controller, el cual especifica tres capas que tienen las aplicaciones. Modelo: Parte de base de datos. View: Ui Controlador: Business Logic

Ahora las aplicaciones han evolucionado en muchas capas, y aparecieron los web services y restful apis, etc.

En una red on premise una ubicación geográfica que puede ser una oficina o un servidor en las instalaciones del Tec, se va a organizar con un switch. Un switch es un dispositivo que es puesto por un internet service provider, los cuales dan un router que traen modem etc. Trae mucho puertos donde conectamos cables RJ45. Este switch lo tenemos que convertir a bilanes

Una bilán es una porción lógica de la red.

Eso me provoca una red separada. Las computadoras que se sientan en cada red van a tener diferente forma de comunicación. Si nos metemos a un switch que tiene una conexión al Internet Service Provider, nosotros en la red tenemos la figura del router. El router normalmente va a funcionar como un Firewall router y tiene una pata en cada uno de las redes. El firewall va a tener el acceso a internet y el router le da salida a cada una de las máquinas en las redes. Por un lado tenemos la red donde hay un servidor público, que se conoce como un public network. Aquí hay un demilitarized zone, es el punto más vulnerable de nuestra red. Es el dispositivo que se encuentra expuesto. En el momento que alguien irrumpe en el servidor, la red va a estar comprometida. En otro lado podemos tener una red privada con acceso a internet de salida y en otro lado podemos tener una red privada sin acceso a Internet. Si una red está comprometida, el otro server está en peligro de ser alcanzado por el comprometido. En una red pública nunca vamos a poner una base de datos. Si nos hackean la red, nos hackean la base y queda comprometida. Podrían extraer toda la información y publicarla o encriptar toda la información.

La buena práctica es que las bases de datos se van a manejar en una red sin Internet. Esto implica que nadie puede entrar y salir de esa red. En la realidad, no se usa este modelo. En la práctica, se ponen en una red privada con acceso limitado a Internet.

En un servidor privado con acceso a Internet, la DB se puede conectar al servidor de updates, donde puede haber un backdoor afuera, entonces el backdoor puede llegar hasta la base de datos y hacerla explotar, pero no es común que esto suceda. Si está privada sin internet, para hacer que un update fluya del servidor de updates, pasa por un servidor de la red privada que se encarga de descargar los updates. El update pasa por un proceso de análisis de antivirus, checksums, exploits, etc., y luego el update se mueve al servidor privado sin Internet cuando el update está listo o probado. Ya luego permitimos que la base de datos acceda al update. Esto me garantiza que la probabilidad que alguien se meta hasta la base de datos sea muy baja.

Normalmente, nosotros tenemos que buscar cómo organizar una aplicación. Si tenemos una aplicación con tres capas, un UI, un API y una base de datos, la organización debería . El API tiene reglas de confianza para entrar a la base de datos. Estas reglas las controla el Firewall del Router. Hay cosas que tenemos que tener presentes. El Source IP es el IP desde el cual me estoy conectando a la base de datos. Fuera de eso, vamos a tener un Source Port. En la base de datos, tenemos un destination IP y destination Port. El que se encarga de manejar las reglas es el Firewall.

Cuando el API pone una petición en la red, va al router y el router va a redireccionar la solicitud hacia la base de datos. Si la regla de SRC port o DST IP no existe, o el SRC IP no es adecuado, el router va a bloquear el acceso. Esto se asegura si mantenemos el mínimo de permisos a los accesos a la base de datos.

También tenemos Firewalls a nivel de base de datos. Esto quiere decir que el router, al igual que la base de datos va a hacer bloqueo de tráfico para evitar algún tipo de problema. Se hace en la dB y en el router, es que si se da un error a nivel humano o no, y deja pasar algún request de una máquina que fue hackeada, a nivel de red o de firewall de la base de datos, no va a permitir el acceso a la base de datos.

Lo otro que podemos tener de seguridad es que la base de datos tenga usuario y password. Este usaurio y password se va a poner en la aplicación que va a estar accediendo a la base de datos. A parte de permitir el acceso con IP fuente y destino, se puede asegurar con usuario y password. Normalmente, las bases de datos hacen que uno tenga usuario, password e IP. Esto implica que el usuario y password solo se pueden utilizar de una máquina específica. Si alguien irrumpe en una base de datos, y logra robarse las credenciales y los firewalls permiten el acceso, todavía la base de datos puede bloquear el acceso porque solo se va a permitir acceso a la DB a través de un IP específico.

Otra cosa que vamos a meter a nivel de una aplicación on premise es que en las conexiones internas vamos a meter TLS o HTTPS. Esto me da la ventaja de que si alguien irrumpe a otra aplicación que vive en la misma red donde se encuentra la base de datos, lo que sucede es que si me ponen un sniffer, el sniffer va a empezar a capturar todo lo que pase a través de la red. Como la información está protegida por TLS o HTTPS, no va a venir la información en texto plano y el sniffer no puede agarrar nada útil.

Podemos implementar certificados o un certificate authority. Valida que los certificado son válidos, aseguramos comunicaciones con llave pública y privada, permitimos autenticación con un certificado. El nivel de encriptación con un certificado es mayor a que si se tiene usuario y contraseña.

Una base de datos internamente va a tener un Firewall, usuario, passwords, certificados, encryption keys (encriptar toda la información cuando se escribe al disco).

Todas estas cosas de seguridad tienen implicaciones de rendimiento. Un encryption key quiere decir que cada vez que ocupo sacar o meter información de la base de datos, necesito encriptar o desencriptar la información. Aparte, da el problema adicional es un Key Management Service.

La llave de encriptación tenemos que manejar en algún lado. Vamos a hacer las letras de un texto se mappen a la llave. Establecemos un conteo de letras tomando en cuenta todos los caracteres. Cuando agarro esto, tengo que reacomodar las letras de forma que quedan en el mismo orden de la llave. Las llaves de enripción se usan para reacomodar los textos. Cada vez que hay que recuperar datos, tengo que desencriptar, y cada vez que guardo datos hay que encriptar. Con una llave simple, hay que hacer una manipulación de datos bastante grande.

La administración de las llaves puede darse en varios niveles. Una vez que se dé a disco de la máquina. La otra es con un sistema externo. Esto se refiere a un servicio en Internet especializado. La última opción es en un

device físico. Un disco es tal vez es el más riesgoso porque si alguien ataca la base de datos y de alguna forma se brincaron todos los seguros, si la llave de encriptación se encuentra en disco simplemente la sacan. El atacante puede sacar la información como la llave. Cuando la uso en un servicio externo, implica que la base de datos tiene que pedir la llave. El servicio se asegura que esa llave nunca llegue al disco. Esa llave tiene un time to live, que me dice cada cuanto tiempo tengo que solicitar la llave.

Cuando me den la llave, la empiezo a usar y la mantengo en memoria durante 30 minutos. Pasados los 30 minutos, la base de datos expira la llave. Esto fuerza que la llave se descarte, se limpie la memoria, se hace un formateo profundo (tirar 0s en toda la memoria para que un proceso no pueda venir y volver a leer la memoria para recuperar la llave). La base de datos luego vuelve a pedir la llave. Una información sensible como la llave puede que no esté siendo utilizada el 100% del tiempo. Si yo doy la llave y la Db la deja en memoria el 100% del tiempo que la pide, no hay un sistema de detección de intrusos que pueda estar auditando el uso de la llave de encriptación. Cuando expiro las llaves, en los audit logs y en la detección de intrusos, queda en qué momento alguien está pidiendo la llave, y si no es normal, va a prender todas las banderas. Esto nos ayuda a aprender y entender el patrón de uso de las llaves para detectar intrusos.

No es completamente seguro, ningún método es infalible, pero nos da más capas de seguridad. La ventaja que nos da esto es que para que nos hackeen, tienen que hackear el servicio externo y la base de datos.

El manejo físico es un dispositivo físico que para lo único que sirve es encriptar. Lo único que tiene es una interfaz donde se la manda información o datos y el sistema devuelve datos encriptados. Las llaves de encriptación están en ese dispositivo. No tiene una interfaz de administración. No corre Windows ni Linux. No tiene acceso de la red más que recibir y enviar la información encriptada. No hay probabilidad que alguien pueda sacarme las llaves de encriptación. Todos estos servicios tienen sistemas de auditoría. El más seguro de todos es un mecanismo físico. Normalmente estos dispositivos son muy caros. Esta encriptación implica que a todas las transacciones, les voy a meter una penalización en rendimiento para poder proteger los datos.

La parte de certificados es autenticación donde hay un certificate authority que emite certificados para una aplicación cliente, como un API. El certificado se instala en la máquina. Está siendo generado a partir de un certificado raíz, que es del cual se genera todos los demás certificados. Implica que es algo bastante importante que puede comprometer al red o esquema de seguridad. Lo que va a pasar es que cuando la aplicación se conecta a la base de datos, dice quién es al enviar el certificado. La base de datos se comuncia con el CA para que este le diga si el certificado es real.

Cuando un browser recibe el certificado dado por una página web, lo agarra, se lo pregunta al CA y le pregunta si es válido. Los certificados itenen un número serial, entonces si alguien crea un certificado que no tiene el mismo número, hay problemas. Además, se utilizan ciertos strings encriptados que se envían al CA, el cual los desencripta y verifica que el certificado fue generado a partir del root certificate.

Aquí, voy a utilizar al certificate authority como entidad autorizadora.

Cada request lo tengo que verificar contra un sistema externo. Si utilizo usuario y password, tiene una complejidad similar, nada más que se maneja dentro de la base de datos, pero también podemos usar un manejador de passwords, como un Lightweight Directory Access Protocol. A nivel de Firewall, cada paquete de información que entra y sale de la máquina tiene que ser analizado por el Firewall y se tiene que verificar que la información que esté saliendo venga y vaya al lugar correcto.

Toda esta parte es la authentication, verificar que el usuario es quien dice ser. Falta un esquema de autorización, que dice si puedo o no puedo acceder a las cosas. La función de una base de datos es guardar

información y recuperarla. Para asegurar la seguridad a nivel de la base de datos, le estamos metiendo toneladas de overhead para asegurarnos que la información está completamente segura. Esto es a nivel puramente de base de datos.

Otras cosas que tenemos que estar verificando: Security updates. Latest versions Backups Sistemas operativos autorizados Puertos de administración SSH, como 3389, estén cerrados. Si por error dejamos el puerto 22 abierto en todos los firewalls y alguien se mete, si alguien dejó abierto el puerto en el router, esto implica que alguien puede meterse al puerto 22 de la base de datos. Si alguien logra quebrar el protocolo de seguridad, está completamente expuesto al ataque. Alguien puede entrar con algún tipo de malicia hasta el servidor. Normalmente, en bases de datos, no están abiertos los puertos de administración a menos que sean estrictamente necesarios. Si están abiertos, se hace para un set de usuarios muy reducidos, credenciales efímeras (para entrar y tienen una duración limitadas). Los puertos de administración deberían estar cerrados. Es común que en empresas formales, como de marketing, biomédica, petróleo, etc, estén muy preocupadas por la seguridad. Hay que llenar formularios, ser aprobados por los managers, utilizar un VPN, descargar todos los parches, etc, solo para entrar al servidor como administrador.

Fuera de eso, tenemos toda la parte de observabilidad, auditoriá y todo eso se va a centralizar en un sistema completo de observabilidad. Con todo esto de Prometheus, usamos uno de los tres pilares de la observabilidad: recolección de métricas en tiempo real con base de datos tiem series. Existen dos pilares más. Los logs son errores, warnings o información que se pueden presentar. Ayudan a ver el historial de actividad. Si tenemos un sistema de métricas y alguien intenta robar algo, tal vez veamos un aumento de uso de CPU y de memoria. Esto no tiene mucho problema dentro de una base de datos, pero puede ocurrir que sospechosamente se pasen de los parámetros. El slow query logs guarda los tiempos de consultas, la cantidad de consultas. Uno puede ver que hay queries que están durando mucho. En el log de auditoría podemos detectar que alguien se metió a la base de datos en una hora normal. Cuando el sistema de observabilidad detecta estas cosas raras, las métricas juntas pueden significar actividad maliciosa, aún cuando por separado sigan dentro de los parámetros establecidos.

Podemos empezar a recoger traces. Un trace es básicamente obtener las banderas o métricas propias del software o lenguaje de programación. Son las banderas que genera mientras está en ejecución. Java por ejemplo, tiene Runtime Metrics. Se va a dar un sistema de logging especializado de los lenguajes de programación que me va a estar generando ciertas estadísticas de dónde el programa va a generar más cuellos de botella. Con los traces más logs y métricas puedo tener una visión mayor del comportamiento de la base de dats. Cuando se dan errores, se usa inteligencia artificial para encontrar patrones y analizar los casos de error.

Los servidores en un servidor en producción, todo es cableado. No se usan redes wireless. El problema con wireless es que tiene un 50% de pérdida de paquetes. En una aplicación de alto rendimiento, si se pierde la mitad de la información y tiene que reenviarlos, se desperdicia muchos errores y se gasta mucho ancho de banda. En redes y BDs manejamos unidades de transferencia de datos. En una red que se manejan páginas de información en una red de BDs, son de 64 KBytes, el cual se mappea al tamaño de página que usamos en el sistema operativo. Cuando se hace paginación con el disco, se mueven 64ks, la red también va a manejar páginas de 64kbs, porque si vamos a disco y hay que mandar una página y mandarla por la red para sincronizarla, si la red no soporta los 64ks y soporta 10, hay que hacer 10 paquetes, ensamblarlos del otro lado y verificar que llegaron en orden. En un ambiente wireless, al tener una pérdida tal alta de paquetes, cuando movemos un paquete completo por una red, la página de 64KBytes es atómica. Si enviamos 64KBs en un paquete de red, pero a la mitad de los 64Ks, se jodió un bit, las entidades que tienen que ver en la comunicación, la receptora tiene que descargar todos los 64Ks que se mandaron. En una red wireless, es muy

riesgoso usar 64Ks en la red es muy riesgoso porque se pierde mucha información. Hay que repetir mucha información, incluso con el caso en que se pierda un bit. En wireless, la transmisión de datos se transforma en segmentos. Un segmento tiene el tamaño de 1535 bytes, que es el mínimo tamaño que puede soportar una comunicación Ethernet. Esto implica que si uso páginas de 64Ks para hacer transmisiones rápidas sobre una red, cuando toco la parte wireless, tengo que agarrar esos 64Ks y despedazarlos en un montón de segmentos para poder enviarlos. Esto no quiere decir que en ambientes de desarrollo no usemos una BD corriendo en una red wireless, sino que se usa en ambientes de desarrollo.

Vamos a tener un router con firewall a nivel de red. Todo paquete que entra y sale de la base de datos va a pasar un segundo filtro a nivel de red. Verifico ip fuente, destino y puerto fuente, destino. Toda comunicación interna sobre una red tiene que ir encriptada. Toda la comunicación va a ir encriptada en tránsito.

Después venimos a la parte de aplicación o lógica de negocio. maneja la comunicación en HTTPs y TLS. Todo lo que sale va a estar filtrado a través del router y la base de datos. Vamos a encontrar algo que se llama SSL Termination. Esto es que la comunicación entre el API y el router va a estar encriptado. Una vez que llega al router se desencripta y se vuelve a encriptar. No todo esto se da al mismo tiempo. A parte del router Firewall, le podemos poner un proxy o un Webapplication Firewall, que desencripta la petición, analizan la petición, se aseguran que no es un ataque y la vuelven a encriptar y la vuelven a sacar. Toda comunicación entre nodos va encriptada. Otra forma es que el proxy no exista, que es la más común. Solamente se va a encriptar entre el api y la base de datos y el router solo hace que pase la petición directamente.

El API necesita un usuario y password, certificados, correr la aplicación con un usuario de sistema operativo sin privilegios. Esto hace que no pueda correr como root. Esto previene que explote todo el sistema. Aquí se complica el manejo de usuarios, passwords o certificados, porque caemos en el mismo problema de antes. Aquí se podría usar un servicio como un Key Management Service, pero esto no es muy común utilizarlo. Es común ver servicios de encriptación física con hardware como tarjetas que con un password especial queman passwords, certificados y se garantiza que la información de esos dispositivos no lleguen al disco duro. La única forma de que me hackeen es que me roben el yubikey.

Algo importante que hay que tener presente es que también hay Firewall en el API. En un esquema de sguridad bien paranoíco hay que tener Firewalls en aplicaciones, redes, antivirus en redes y dispositivos.

La parte pública es un UI, el cual maneja la comunicación a través de la red con https y TLS. Hay el mismo problema con el manejo de keys. Aquí se manejan API keys que son mecanismos de autenticación propios de un API. No usamos usuarios y passwords para autenticar con un API. Normalmente, autenticamos y manejamos algún tipo de sesión. Hay que manejar el acceso por usuarios de esos keys, por lo que hay que verificar los usuarios y su autorización. Una forma simple de hacer esto es usar un Single Sign On o un LDAP.

A nivel de UI, API y DB, un usuario es exactamente el mismo. Cuando un usuario inicia sesión en el Single Sign On, este le devuelve un token (una cadena de texto alfanumérico con algún formato y tamaño) que se usa como firma de una petición que va hacia un UI o API. Con esto, todas las partes se benefician de la seguridad del single sign on, como two factor authentication. Cuando esto pasa, por debajo, se pasa un token. El cliente lo agarra y lo manda al UI, luego ese token se valida preguntándole al SSO, y si es válido, deja pasar al UI y este manda el token al API. El API luego le pregunta al SSO si el token es autenticado y si tiene la autorización de leer la información que está pidiendo, como para que un usuario no lea la información de otro. Cuando la petición llega a la DB, también le pregunta al SSO. Así, se centraliza todo en un dispositivo o sistema. Estos sistemas no son inviolables, pero su seguridad es bastante robusta. Además, hay sistemas de verificación extra. Esto nos da el poder de que nunca voy a manejar API Keys a nivel de UI, sino el UI es una pasarela que

recibe un token generado por un Single Sign On, y la validación de los permisos con los API Keys se hace a través del SSO.

Aseguramos toda la base de datos con un sistema de autenticación externo, como SSO y LDAP. LDAP se utilizaba antes. En realidad, todo sistema de SSO utiliza un provider, el cual es un LDAP o algo como Gmail, Facebook, Instagram, Twitter, etc.

Todas estas capas se usan para servicios On Premise, pero por cada capa, se mete overhead que se va a sumarizar en el tiempo de respuesta que va a tener la base de datos.

Cada máquina en una red va a tener un número de identificación. Un IP dentro de una red no es único de una computadora. El identificador de una computadora es con un IP y un puerto.