

30-4-2022



INSTITUTO TECNOLÓGICO DE
TLALNEPANTLA

3.2 Controles en Aplicaciones en desarrollo

MATERIA:

TECNOLOGIAS DE
SEGURIDAD EN SOFTWARE

PROFESORA:
MPEDT HILDA DÍAZ RINCÓN

INTEGRANTES:
SAAVEDRA ROJAS
BRANDON
HUITRON UC TABATA
ELIZABETH
GARCIA MADUJANO SAID

GRUPO:

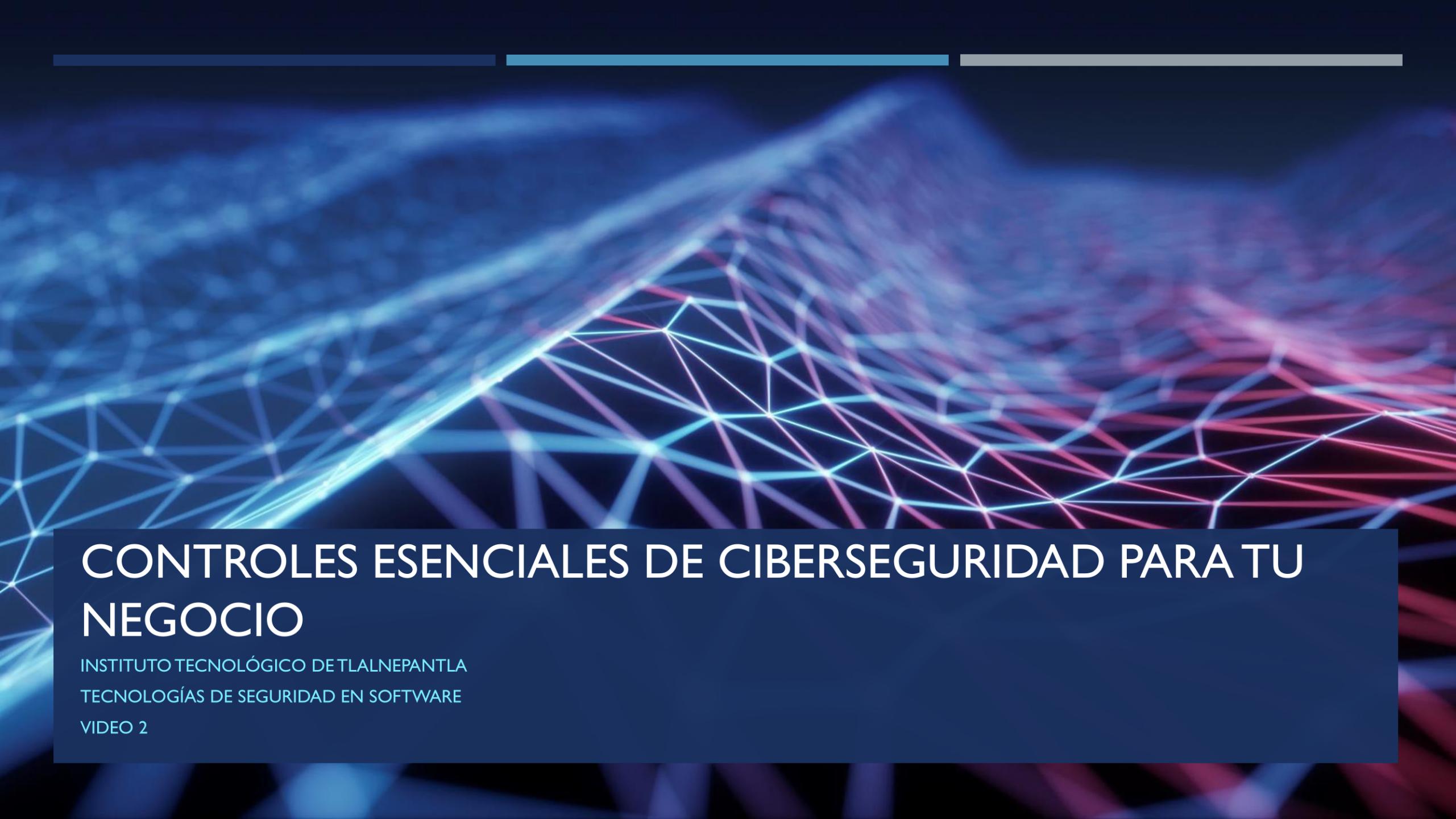
T92

Actividad:Video I

revisar y realizar un video animado que incluya un caso de acuerdo a lo que menciona el autor del video

Enlace del video:

[Ver video](#)

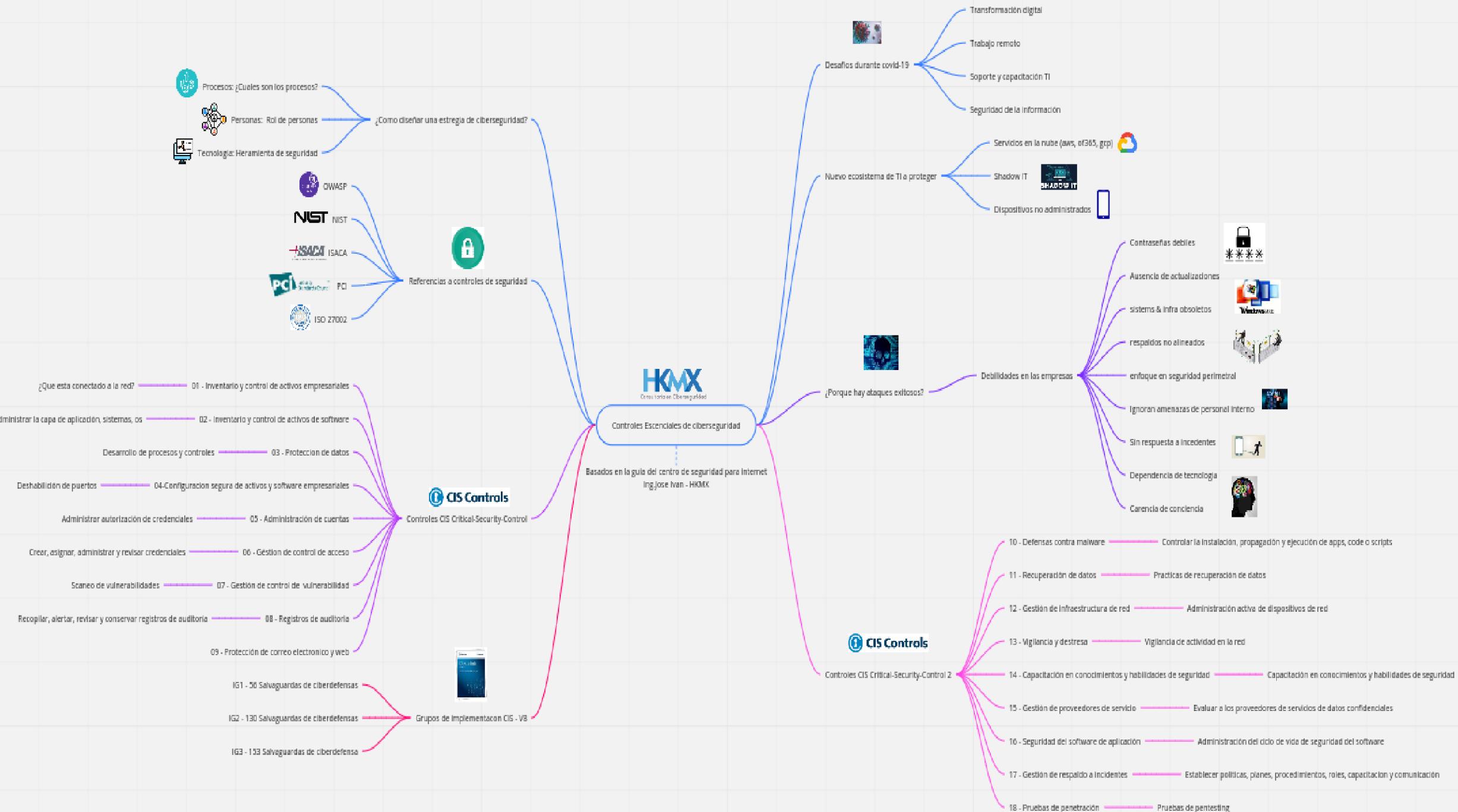


CONTROLES ESENCIALES DE CIBERSEGURIDAD PARA TU NEGOCIO

INSTITUTO TECNOLÓGICO DE TLALNEPANTLA

TECNOLOGÍAS DE SEGURIDAD EN SOFTWARE

VIDEO 2



COMO SE RELACIONA CON EL CASO DE ESTUDIO

La empresa Bike si bien utiliza sistemas propuestos por nosotros, aun no integra algún tipo de control en las aplicaciones que se encuentran en producción.

Se debe proteger y tratar todos los sistemas dependiendo de la información que contenga o maneje.

Existen diferentes referencias para implementar controles de seguridad, por lo que en este video se implementa CIS (Critical-Security-Control).

Para mitigar ataques o reducir su impacto teniendo como referencia a la empresa bike se debe realizar los 18 puntos de control que nos ofrece CIS:

- I. **Saber que esta conectado en la red de la empresa BIKE:** esto nos permitirá conocer los dispositivos que se encuentran dentro de la red
- II. **Inventarios y controles de activos de software:** Administrar todos los sistemas de los diferentes departamentos de la empresa bike

COMO SE RELACIONA CON EL CASO DE ESTUDIO

- III. **Implementar protección de datos:** Desarrollar procesos y políticas sobre el uso y tratamiento de los datos dentro de la empresa Bike
- IV. **Configuración segura de activos:** Mantener los activos con configuraciones seguras, deshabilitando o no configurando funciones que no se usan como lo son los puertos de entrada de los servidores
- V. **Administrar las cuentas:** Llevar un control de las cuentas del personal y personal directivo de la empresa Bike y saber quien tiene acceso a que.
- VI. **Gestión de control de acceso:** Gestionar los accesos y autorizaciones de los usuarios, así como realizar auditorias donde las credenciales de los empleados de la empresa Bike para reducir posibles ataques como lo son los de ingeniería social

COMO SE RELACIONA CON EL CASO DE ESTUDIO

- VII. Gestión de controles de seguridad: Realizar escaneos dentro de la red de la empresa periódicamente
- VIII. Registro de auditoria: Durante cada auditoria, se debe resguardar, revisar, alertar sobre los registros de auditoria
- IX. Protección de correo electrónico: Capacitar a los empleados sobre las buenas prácticas de uso del correo electrónico para no caer en trampas de fishing o Ramsomeware
- X. Defensa contra malware: Cuando se instale algo en la organización, controlar su instalación y su propagación utilizando sistemas de antivirus para la detección de código malicioso
- XI. Recuperación de datos: Dentro de la empresa, a las personas de TI enseñarles la cultura de hacer respaldos periódicos a la información para reducir los impactos ante un ataque como Ramsomeware

COMO SE RELACIONA CON EL CASO DE ESTUDIO

XII. Gestión de infraestructura de red: Administrar activamente los dispositivos que entran y salen de la red de empresa Byke, esto permitirá estar alerta ante un infiltración o ataque

XIII. Vigilancia: Vigilar la red de la empresa enfocado en la actividad de los usuarios, de esta forma podríamos detectar rápidamente si alguien esta extrayendo grandes volúmenes de información

XIV. Capacitación de conocimientos y habilidades de seguridad: Capacitar a los empleados sobre temas de seguridad como la creación adecuada de contraseñas, actualizaciones constantes, respaldos alineados, amenazas no solo son dentro de la empresa

XV. Gestión de proveedores de servicio: Evaluar a todos los proveedores con los que labora la empresa y determinar si alguno trata con información sensible que pueda dañar a la empresa, si es el caso implementar controles de seguridad con el proveedor

XVI. Gestión de respaldo a incidentes: Establecer políticas, planes, procedimientos, roles, capacitación y comunicación dentro de la empresa, esto permitirá saber como actuar ante un ataque siguiendo los procedimientos

COMO SE RELACIONA CON EL CASO DE ESTUDIO

Como vemos los 18 puntos que nos presenta CIS- Critical-Security-Control pueden ser adaptados a cualquier tipo de negocio o empresa, porque son marcos de referencia para la implementación de controles de seguridad,

Existen diversos marcos como OWASP Top 10, NIST, ISACA, etc.

Hoy en día ya no es recomendable no contar con controles de seguridad, una empresa siempre va a manejar información que es sensible para los atacantes, es por eso que es nuestro deber proteger nuestros activos y salvaguardar la reputación de la empresa y sus clientes.

VIDEO #3.- Vulnerabilidades, amenazas y controles - Introducción a la seguridad informática

Una vez que conocemos los problemas y en nivel de riesgo del caso de estudio ya visto con anterioridad ahora pasaremos a ver cuál sería el mejor control de seguridad que podemos aplicar, un control de seguridad es una medida de seguridad o contramedidas para evitar, contrarrestar o minimizar seguridad riesgos relativos a la propiedad personal, o cualquier propiedad de la empresa.

Para business-to-business frente a las empresas cuyo servicio puede afectar los Estados financieros de la compañía, la perspectiva puede requerir los informes de auditoría exitosa de controles de políticas tales como un SSAE 16 Informe antes de concederles autorización como vendedor.

Una vez que se tiene claro lo que es un control de seguridad, debemos ver cual seria la mejor contra medida para los riesgos encontrados, para realizar esto se deben de tomar en cuenta los frameworks y estándares que existen.

Uno de los aspectos que garantiza una correcta implementación de cualquiera de los métodos de acceso es cumplir con las normas, regulaciones y/o estándares. De acuerdo a cada norma, regulación o estándares pueden aplicarse sanciones de todo tipo si es que se deja de lado su cumplimiento.

Por qué mencionamos esto, más que nada porque no existe un método que es estrictamente más conveniente que otro. Cada entorno de trabajo cuenta con necesidades en específico. Igualmente, es bueno tener presente que, de los tipos de control de acceso citados, el más moderno y reciente es el basado en atributos. Es, especialmente conveniente porque los permisos se conceden o limitan de acuerdo a la situación del momento del usuario.



VIDEO #4.- Programación defensiva.

CASO 1:

Reducir complejidad del código fuente.

Nunca hacer el código más complejo de lo necesario. La complejidad genera bugs, incluyendo problemas de seguridad. Esta meta puede tener conflicto con el objetivo de escribir programas que puedan recuperarse de cualquier error y manejar cualquier entrada de datos. Manejar todas las ocurrencias inesperadas en un programa requiere que el programador añada código extra, el cual pudiera también tener bugs.

CASO 2:

Revisiones del código fuente

Una revisión de código es donde alguien diferente al autor original realiza una auditoría de código. Una auditoría de código hecha por el mismo creador es insuficiente. La auditoría la debe hacer alguien que no sea el autor, como cuando se escribe un libro, debe ser revisado por alguien que no sea el autor.

Simplemente haciendo el código disponible para que otros lo lean (software libre) es insuficiente, pues no hay garantía que el código sea efectivamente revisado, no dejando que sea rigurosamente revisado.

CASO 3:

Pruebas de software

Las pruebas de software deberán ser para tanto que el software trabaje como debe, como cuando se supone que pase si se realice deliberadamente malas entradas.

Las herramientas de prueba pueden capturar entradas de teclado asociadas con operaciones normales. Luego las cadenas de texto de estas entradas capturadas pueden ser copiadas y editadas para ensayar todas las permutaciones y combinaciones, luego ampliarlas para tests posteriores después de cualquier modificación. Los defensores de la clave de registro afirman que los programadores que usan este método deberán asegurar que las personas a las cuales se les están capturando las entradas estén al tanto de esto, y con qué propósito, para evitar acusaciones de violación de privacidad.

CASO 4:

Reutilización inteligente del código fuente

Si es posible, reutilizar código. La idea es capturar beneficios de un bien escrito y bien probado código fuente, en vez de crear bugs innecesarios.

Sin embargo, reutilizar código no siempre es la mejor manera de progresar, particularmente cuando la lógica del negocio está involucrada. Reutilizar en este caso puede causar serios bugs en los procesos de negocio.

CASO 5:

Entrada segura / manejo de la salida

Canonicalizar

Los crackers tienden a buscar representaciones diferentes de los mismos datos.

Por ejemplo, si un programa verifica un nombre de archivo contra, un cracker puede intentar usar un nombre diferente pero que se refiere al mismo archivo.

Para evitar bugs debido a entradas no canónicas, se deben emplear las API de canonicalización.

COMENTARIOS GENERALES DEL VIDEO.

- Con esto nos podemos dar cuenta que la programación defensiva es una forma de diseño defensivo aplicada al diseño de software que busca garantizar el comportamiento de todo elemento de una aplicación ante cualquier situación de uso por incorrecta o imprevisible que ésta pueda parecer.
- Las técnicas de programación defensiva se utilizan especialmente en componentes críticos cuyo mal funcionamiento, ya sea por descuido o por ataque malicioso, podría acarrear consecuencias graves o daños catastróficos.
- La programación defensiva se basa en un conjunto de buenas prácticas basadas en la idea que no es posible saber cómo se van a utilizar nuestros programas. Con esta idea en mente el objetivo es tomar medidas para garantizar el funcionamiento correcto aún con información o datos inválidos.

Programación defensiva

Cuando realizamos desarrollos, muchas veces por falta de experiencia o conocimientos realizamos software de una manera poco segura, susceptible a errores.

La programación defensiva existe porque hay programadores que no hacen uso de buenas prácticas en sus softwares. Este concepto busca mitigar errores que pueden crashear una aplicación en entornos de producción.

Esta programación abarca los puntos:

- Manejo de excepciones:

Si bien, a veces cuando escribimos bloques de código podemos olvidarnos de manejar las excepciones de nuestras funciones, por ejemplo cuando escribimos bloques que realizan lógica aquí es donde es un punto crítico de nuestro programa

Debemos de controlar todos los valores de entrada, por ejemplo, si utilizamos lenguajes no tipados como JavaScript podemos inicializar variables que nos podría ayudar a salir del problema

Otra solución con este tipo de lenguajes permisivos es el uso de bloques try catch para manejar los errores que pueden manejar el error y salir de el.

- Datos no cifrados

Siempre que creamos un software y guardemos información sensible como datos de tarjetas o contraseñas entre otros, es de suma importancia poder cifrar los datos para que si alguna vez son secuestrados o revisados por personas ajenas no sepan acerca de lo que significa utilizando algoritmos de encriptación para salvaguardar información sensible

- Software legacy

Cuando se le tiene que dar mantenimiento a algún software, muchas veces nos encontramos con malas prácticas, versiones obsoletas de paquetes, etc. Por lo que nos toca pagar la deuda técnica.

La deuda técnica en pocas palabras la paga la persona que este dando el mantenimiento a dicho código, esto es muy costoso a la larga porque puede involucrar mas horas de desarrollo.

Podemos hacer uso de buenas prácticas de programación para mitigar o reducir la deuda.

- Mensajes no controlados

Cuando estamos depurando muchas veces solemos utilizar impresiones de consola para saber el valor de una variable, si bien existe el entorno de depuración y no es malo realizar impresiones de consola, si no que lo malo es dejar esa impresión en la versión de producción

Si dejaste algún log con información sensible, esto es muy susceptible que pueda ocurrir un ataque.

Mitigar

Siempre las pruebas unitarias y de integración permitirán saber los puntos críticos de nuestra aplicación y podemos realizar muchas pruebas a componentes separados o en conjunto por lo que nos cercioramos que nuestro código sirve de la forma esperada

Realizar documentación del software es de ley

Aplicar conceptos de bajo acoplamiento y alta cohesión en el código

Utilizar buenas prácticas de programación para reducir la deuda técnica

PROGRAMACIÓN DEFENCIVA.

- Protégete de cualquier entrada.

Si el programa, función, clase, etc. recibe una entrada o parámetro no importa si viene de una base de datos, de algo que el usuario ingresa en un formulario, si lo recibe de otro programa, API, etc. A esto se le llama entrada externa y debemos desconfiar de ella a toda costa, especialmente si programas en JS, porque ahí ni siquiera existe la validación de tipos de datos, tu función puede esperar un número y recibir un texto.

- Assertions (Aserciones, Afirmaciones)

Esta es una función que se usa durante el desarrollo para que el código se pruebe a si mismo mientras lo ejecutamos, cuando la aserción es verdadera es porque el programa está funcionando como debería y cuando es falsa significa que hemos encontrado un error. Estas funciones usualmente tienen dos parámetros uno es la condición, y el otro es un mensaje que se muestra en caso de que la condición sea falsa. Si tu lenguaje de programación no tiene una función Assert, puedes crearla tu mismo.

- Manejo de errores.

Se usan las aserciones para situaciones que no deberían de pasar “nunca”, pero cuando esperamos que suceda un error, usamos técnicas para manejo de errores.

- Excepciones.

Las excepciones son un medio por el cual se pueden manejar los errores en los que el sistema no tiene forma de recuperarse o de responder ante el error. Básicamente no sabe qué hacer, sin embargo, la pieza de código que lo llamó podría saber qué hacer y puede capturar la excepción y hacer que el programa logre recuperarse y continuar su ejecución.

Se puede hacer una función que regrese un código de error en caso de que se presente algo inesperado y esperar que el programa que lo llame, revise ese código

de error y haga algo al respecto. Si se genera una excepción puedes estar seguro de que harán algo, de lo contrario el programa termina con un error. Usa excepciones siempre que deseas que se tome una acción correctiva ante una situación inesperada, solo ante errores críticos y solamente si tú no puedes encargarte de ese error.

- Aserciones.

Entre funciones, se puede querer verificar que se está referenciando algo que no es válido y que el tamaño de los arreglos es válido antes de referenciar elementos, especialmente instanciando de forma temporal/local. Una buena heurística es no creer en las bibliotecas que se hayan o no se hayan escrito. Cada vez que se llamen estas, verificar lo que se quiere que estas devuelvan. A veces ayuda crear una pequeña biblioteca de funciones de “aserción” y “verificación” para hacerla junto a un logger, para así poder trazar la ruta y reducir la necesidad de extensos ciclos de depuración. Con la llegada de bibliotecas de loggeo y la Programación orientada al Aspecto, muchos de los tediosos aspectos de la programación defensiva son mitigados.

COMENTARIO GENERAL.

- ❖ Todo lo que he leído y escuchado acerca de la programación defensiva me ha llevado a la conclusión de que se trata de una metodología destinada a ayudarnos a anticipar los fallos, no sólo los del presente sino en el futuro de modo que, cuando se produzcan, nuestra aplicación esté preparada adecuadamente para responder a ellos.

VIDEO #5

TIPOS Y CLASES DE CONTROLES PARA SEGURIDAD DE LA INFORMACIÓN

