Universidad Nacional de Ingeniería (UNI)
Escuela Profesional de
Ciberseguridad
Sílabo 2024-II

1. CURSO

CY221. Seguridad de Software (Obligatorio)

2. INFORMACIÓN GENERAL

2.1 Curso	:	CY221. Seguridad de Software
2.2 Semestre	:	8^{vo} Semestre.
2.3 Créditos	:	3
2.4 horas	:	2 HT; 2 HP;
2.5 Duración del periodo	:	16 semanas
2.6 Condición	:	Obligatorio
2.7 Modalidad de aprendizaje	:	Presencial
2.8 Prerrequisitos	:	CS3I1. Seguridad en Computación. (7^{mo} Sem)

3. PROFESORES

Atención previa coordinación con el profesor

4. INTRODUCCIÓN AL CURSO

Este curso aborda los principios y prácticas para el desarrollo de software seguro, capacitando a los estudiantes para construir aplicaciones resistentes a vulnerabilidades. Se exploran técnicas de diseño, implementación y pruebas, considerando las responsabilidades éticas y legales.

5. OBJETIVOS

- Aplicar principios y prácticas para diseñar e implementar software seguro.
- Identificar y mitigar vulnerabilidades comunes en el desarrollo de software.
- Comprender el impacto ético y legal del desarrollo de software seguro.

6. RESULTADOS DEL ESTUDIANTE

2) Diseñar, implementar y evaluar una solución basada en la computación para satisfacer un conjunto dado de requisitos de computación en el contexto de la disciplina del programa. (Assessment)

4) Reconocer las responsabilidades profesionales y tomar decisiones informadas en la práctica de la computación basadas en principios legales y éticos. (Usage)

6) Aplicar principios y prácticas de seguridad para mantener las operaciones en presencia de riesgos y amenazas. (Assessment)

7. TEMAS

Unidad 1: Principios fundamentales (12 horas)	
Resultados esperados: 2,6	
Temas	**Objetivos de Aprendizaje (*Learning Outcomes*)**
• Mínimo privilegio – Esta unidad de conocimiento presenta los principios que subyacen tanto al diseño como a la implementación. Los primeros cinco son principios de restricción, los tres siguientes son principios de simplicidad y el resto son principios de metodología. • Valores predeterminados a prueba de fallos – El estado inicial debería ser denegar el acceso a menos que se requiera explícitamente. Entonces, a menos que al software se le dé acceso explícito a un objeto, se le debe negar el acceso a ese objeto y el estado de protección del sistema debe permanecer sin cambios. • Mediación Completa – El software debe validar cada acceso a los objetos para garantizar que el acceso esté permitido. • Separación – El software no debe otorgar acceso a un recurso ni realizar una acción relevante para la seguridad basándose en una única condición. • Minimizar la confianza – El software debe verificar todas las entradas y los resultados de todas las acciones relevantes para la seguridad. • Economía del mecanismo – Las funciones de seguridad del software deben ser lo más simples posible • Minimizar el mecanismo común – Reducir los recursos compartidos al máximo • El menor asombro – Las características de seguridad del software y los mecanismos de seguridad que implementa deben diseñarse de manera que su funcionamiento sea lo más lógico y simple posible. • Diseño abierto – La seguridad del software, y de lo que ese software proporciona, no debería depender del secreto de su diseño o implementación. • Capas – Organice el software en capas de modo que los módulos de una capa determinada interactúen solo con los módulos de las capas inmediatamente superiores y inferiores. Esto le permite probar el software una capa a la vez, utilizando	• Analice las implicaciones de confiar en el diseño abierto o el secreto del diseño para la seguridad [Usar] • Enumere los tres principios de seguridad [Usar] • Describa por qué cada principio es importante para la seguridad. [Usar] • Identificar el principio de diseño necesario [Usar]

Unidad 2: Diseño (8 horas)	
Resultados esperados: 2	
Temas	**Objetivos de Aprendizaje (*Learning Outcomes*)**
• Derivación de requisitos de seguridad. – Comenzando con el negocio, la misión u otros objetivos, determine qué requisitos de seguridad son necesarios para tener éxito. Estos también pueden derivarse o modificarse a medida que evoluciona el software. • Especificación de requisitos de seguridad. – Traducir los requisitos de seguridad a una forma que pueda usarse (especificación formal, especificaciones informales, especificaciones para pruebas). • Ciclo de vida de desarrollo de software/Ciclo de vida de desarrollo de seguridad – Incluya los siguientes ejemplos: modelo en cascada, desarrollo ágil y seguridad. • Lenguajes de programación y lenguajes de tipo seguro – Analice los problemas que introducen los lenguajes de programación, qué hace la seguridad de tipos y por qué es importante.	• Analice las implicaciones de confiar en el diseño abierto o el secreto del diseño para la seguridad. [Usar] • Enumere los tres principios de seguridad. [Usar] • Describa por qué cada principio es importante para la seguridad. Identificar el principio de diseño necesario [Usar]
Lecturas : [**McGraw2006**]	

Unidad 3: Implementación (10 horas)	
Resultados esperados: 2,6	
Temas	**Objetivos de Aprendizaje (*Learning Outcomes*)**
• Validar la entrada y comprobar su representación. – Verifique los límites de los buffers y los valores de los números enteros para asegurarse de que estén dentro del rango – Verifique las entradas para asegurarse de que sean las esperadas y que se procesen/interpreten correctamente. • Utilizando las API correctamente – Verifique los resultados del uso de la API para detectar problemas – Asegúrese de que los parámetros y entornos estén validados y controlados para que la API aplique la política de seguridad correctamente. • Uso de características de seguridad – Utilice aleatoriedad criptográfica – Restrinja adecuadamente los privilegios del proceso. • Comprobación de relaciones de tiempo y estado. – Compruebe que el archivo sobre el que se actúa es aquel para el que se comprueban los atributos relevantes – Verifique que los procesos se ejecuten. • Manejar excepciones y errores adecuadamente – Bloquear o poner en cola señales durante el procesamiento de señales, si es necesario – Determine qué información se debe brindar al usuario, equilibrando la usabilidad con cualquier necesidad de ocultar cierta información, y cómo y a quién reportar esa información. • Programación de robusta – Solo desasignar la memoria asignada, – Inicializar variables antes de usarlas – No confíe en un comportamiento indefinido. • Encapsulación de estructuras y módulos. – Procesos de aislamiento. • Teniendo en cuenta el medio ambiente – Ejemplo: no incluya información confidencial en el código fuente.	• Analice las implicaciones de confiar en el diseño abierto o el secreto del diseño para la seguridad. [Usar] • Enumere los tres principios de seguridad. [Usar] • Describa por qué cada principio es importante para la seguridad. Identificar el principio de diseño necesario [Usar]
Lecturas : [**Seacord2005**]	

Unidad 4: Análisis y pruebas (8 horas)	
Resultados esperados: 2,6	
Temas	**Objetivos de Aprendizaje (*Learning Outcomes*)**
• Análisis estático y dinámico. – Este tema describe los diferentes métodos para cada uno de ellos, incluye cómo funcionan juntos el análisis estático y dinámico, y los límites y beneficios de cada uno, además de cómo realizar estos tipos de análisis en sistemas de software de gran tamaño. • Pruebas unitarias – Este tema describe cómo probar componentes del software, como módulos. • Pruebas de integración – Este tema describe cómo probar los componentes de software a medida que se integran. • Pruebas de software – Este tema describe cómo probar el software en su conjunto y colocar las pruebas unitarias y de integración en un marco adecuado.	• Explique por qué los requisitos de seguridad son importantes [Usar] • Identificar vectores de ataque comunes [Usar] • Describir la importancia de escribir programas seguros y robustos [Usar] • Describir el concepto de privacidad, incluida la información de identificación personal [Usar]
Lecturas : [**Whittaker2012**]	

Unidad 5: Implementación y mantenimiento (10 horas)	
Resultados esperados: 2,6	
Temas	**Objetivos de Aprendizaje (*Learning Outcomes*)**
• Configurando – Este tema cubre cómo configurar el sistema de software para que funcione correctamente. • Parches y ciclo de vida de la vulnerabilidad – Este tema incluye la gestión de informes de vulnerabilidad, la reparación de las vulnerabilidades, la prueba del parche y la distribución del parche. • Comprobando el entorno – Este tema cubre cómo garantizar que el entorno coincida las suposiciones hechas en el software, y si no, cómo manejar el conflicto • DevOps – Este tema combina desarrollo y operación, y la automatización y monitoreo de ambos. • Desmantelamiento/Retiro – Este tema describe lo que sucede cuando se elimina el software y cómo eliminarlo sin causar problemas de seguridad.	• Explique por qué son necesarias la validación de entradas y la desinfección de datos [Usar] • Explica la diferencia entre números pseudoaleatorios y números aleatorios [Usar] • Diferenciar entre codificación segura y parcheo y explicar la ventaja de utilizar técnicas de codificación segura [Usar] • Describa un desbordamiento del búfer y por qué es un posible problema de seguridad [Usar]
Lecturas : [**Humble2010**]	

8. PLAN DE TRABAJO

8.1 Metodología

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

8.2 Sesiones Teóricas

Las sesiones de teoría se llevan a cabo en clases magistrales donde se realizarán actividades que propicien un aprendizaje activo, con dinámicas que permitan a los estudiantes interiorizar los conceptos.

8.3 Sesiones Prácticas

Las sesiones prácticas se llevan en clase donde se desarrollan una serie de ejercicios y/o conceptos prácticos mediante planteamiento de problemas, la resolución de problemas, ejercicios puntuales y/o en contextos aplicativos.

9. SISTEMA DE EVALUACIÓN

********* EVALUATION MISSING ********

10. BIBLIOGRAFÍA BÁSICA