Laboratorio Guiado

Del Código a Historias de Usuario con GitHub Copilot

Proyecto: `dotnet-architecture/eShopOnWeb`

https://github.com/NimblePros/eShopOnWeb.git

Objetivos del laboratorio

Al finalizar el laboratorio el participante podrá:

- Introducir seguridad desde el código usando Copilot como asistente.
- Identificar riesgos evidentes:
    - Falta de validaciones
    - Exposición de datos sensibles
    - Manejo inseguro de errores
- Formular preguntas efectivas de seguridad.
- Identificar amenazas básicas basadas en OWASP Top 10.
- Proponer controles mínimos de mitigación.

IMPORTANTE:
Este laboratorio **NO reemplaza**:

- Security Review formal
- Pentesting
- Threat modeling avanzado (STRIDE formal, DFD detallado, etc.)

🧭 Parte 1 – Contexto del sistema (15 min)

Paso 1 – Comprender qué estamos protegiendo

Abrir el repositorio y explorar:

- Web (UI)
- ApplicationCore
- Infrastructure
- Controllers / Endpoints
- Identity / autenticación

Prompt inicial con Copilot Chat:

```
Analiza este proyecto ASP.NET Core y explícame:

1. Qué tipo de aplicación es.
2. Qué datos sensibles podría manejar.
3. Qué superficies de ataque observas a primera vista.
4. Qué componentes son más críticos desde el punto de vista de seguridad.
```

Discusión grupal:

- ¿Maneja información personal?
- ¿Maneja pagos?
- ¿Tiene autenticación?
- ¿Existen roles?

Parte 2 – Identificación de Riesgos Evidentes

Trabajaremos sobre endpoints reales (por ejemplo: Basket, Orders, Catalog).

Caso 1 – Falta de Validaciones

Abrir un endpoint tipo:

```
[HttpPost]
```

```
public async Task<IActionResult>
CreateOrder(CreateOrderRequest request)
{
    var order = await _orderService.CreateOrderAsync(request);
    return Ok(order);
}
```

Prompt clave:

```
¿Qué riesgos de seguridad ves en este endpoint?
¿Faltan validaciones?
¿Qué tipo de ataques podría recibir?
```

Lo que esperamos que Copilot identifique:

- No validación de:
  - ModelState
  - Campos nulos
  - Longitudes
  - Valores fuera de rango

- Posible Mass Assignment
- Falta de autorización explícita
- Confianza excesiva en el cliente

Prompt de profundización:

```
Si un usuario manipula el request manualmente desde Postman,
¿qué escenarios de abuso podrías imaginar?
```

💡 Discusión:

- Inyección de IDs
- Manipulación de precios
- Referencias a recursos que no le pertenecen

Caso 2 – Exposición de Datos Sensibles

Buscar un endpoint que devuelva entidades completas.

Ejemplo típico:

```
return Ok(order);
```

Prompt:

```
¿Se exponen datos sensibles en este response?
¿Es buena práctica devolver la entidad completa del dominio?
¿Qué riesgos de privacidad existen?
```

Riesgos esperados:

- Exposición de:
  - IDs internos
  - Relaciones
  - Datos de usuario
- Fuga de estructura interna
- Acoplamiento del dominio al contrato público

Prompt de mejora:

```
¿Qué controles mínimos faltan aquí para reducir exposición de datos?
Propón una mejora concreta.
```

Esperado:

- Uso de DTOs
- Mapeo selectivo
- Minimización de datos (principio de least data)

🚨 Caso 3 – Manejo Inseguro de Errores

Buscar bloques tipo:

```
catch (Exception ex)
{
    return BadRequest(ex.Message);
}
```

Prompt:

```
¿Qué problemas de seguridad ves en este manejo de errores?
¿Qué información sensible podría filtrarse?
```

Esperado:

- Filtrado de stack traces
- Exposición de mensajes internos
- Filtrado de estructura de base de datos
- Información útil para atacante

Prompt de mejora:

```
¿Qué patrón de manejo de errores seguro recomendarías en
ASP.NET Core?
¿Qué debería mostrarse al cliente y qué debería registrarse
internamente?
```

Parte 3 – OWASP Básico Aplicado (30–40 min)

Ahora usamos Copilot como generador de amenazas.

Prompt General OWASP:

```
Basado en OWASP Top 10,
¿qué amenazas podrían aplicar a este proyecto?
Explícalas en contexto específico del código que estamos
viendo.
```

Esperamos:

- A01: Broken Access Control
- A02: Cryptographic Failures
- A03: Injection
- A04: Insecure Design
- A05: Security Misconfiguration

- A09: Logging & Monitoring Failures

Prompt específico por endpoint:

```
Si este endpoint fuera público en internet,
¿qué 5 amenazas realistas podrías listar?
```

Discusión:

- IDOR
- Enumeración de recursos
- Manipulación de parámetros
- Fuerza bruta
- Ataques de automatización

🧠 Parte 4 – Copilot como Checklist Inteligente

Ahora enseñamos el enfoque correcto.

No preguntar:
❌ "¿Es seguro este código?"

Sí preguntar:
✅ "¿Qué controles mínimos de seguridad faltan aquí?"

Prompt estructurado (Plantilla recomendada)

```
Actúa como un security reviewer senior.

Analiza este endpoint y responde:

1. Validaciones faltantes.
2. Problemas de autorización.
3. Riesgos de exposición de datos.
4. Posibles ataques.
5. Controles mínimos recomendados.

No hagas cambios grandes de arquitectura,
solo mejoras mínimas realistas.
```

Parte 5 – Mini Threat Modeling Ligero

No formal, pero práctico.

Paso 1 – Identificar Activos

Prompt:

```
Identifica los activos críticos de este sistema.
```

Ejemplos:

- Datos de usuario
- Órdenes
- Información de autenticación
- Configuraciones

Paso 2 – Identificar Actores

```
¿Quiénes podrían atacar este sistema?
```

- Usuario malicioso autenticado
- Usuario anónimo
- Bot automatizado
- Insider

Paso 3 – Identificar Amenazas

```
Enumera amenazas concretas contra el módulo de órdenes.
```

Paso 4 – Identificar Controles

```
¿Qué controles mínimos deberían existir para mitigar estas amenazas?
```

Parte 6 – Ejercicio Final Integrado

Cada grupo selecciona un endpoint distinto y responde:

1. ¿Qué hace?
2. ¿Qué datos procesa?
3. ¿Qué datos devuelve?
4. ¿Qué riesgos evidentes tiene?
5. ¿Qué amenazas OWASP aplican?
6. ¿Qué controles mínimos faltan?

Luego presentan:

- Riesgo identificado
- Impacto potencial
- Mejora propuesta

📌 Sección Crítica (Debe explicarse claramente)

🚫 Qué NO hace Copilot

Copilot:

- No ejecuta pruebas dinámicas
- No detecta vulnerabilidades runtime
- No reemplaza SAST/DAST
- No reemplaza pentesting
- No modela amenazas formales (STRIDE completo, DFD, etc.)

Debe quedar explícito en el laboratorio:

```
Copilot es un asistente cognitivo, no una herramienta de
seguridad certificada.
```

🏁 Cierre del Laboratorio

Reflexión Final

Preguntar a los alumnos:

- ¿Cambió su forma de leer código?
- ¿Qué tipo de preguntas funcionaron mejor?
- ¿Qué errores de prompting cometieron?
- ¿Dónde Copilot fue superficial?