Reporte de Vulnerabilidad

Documento Creador	Carlos Arizandy Rodriguez Preciado
Revisión	1
Ultima actualización	06-12-2023

Contenido

Recomendaciones

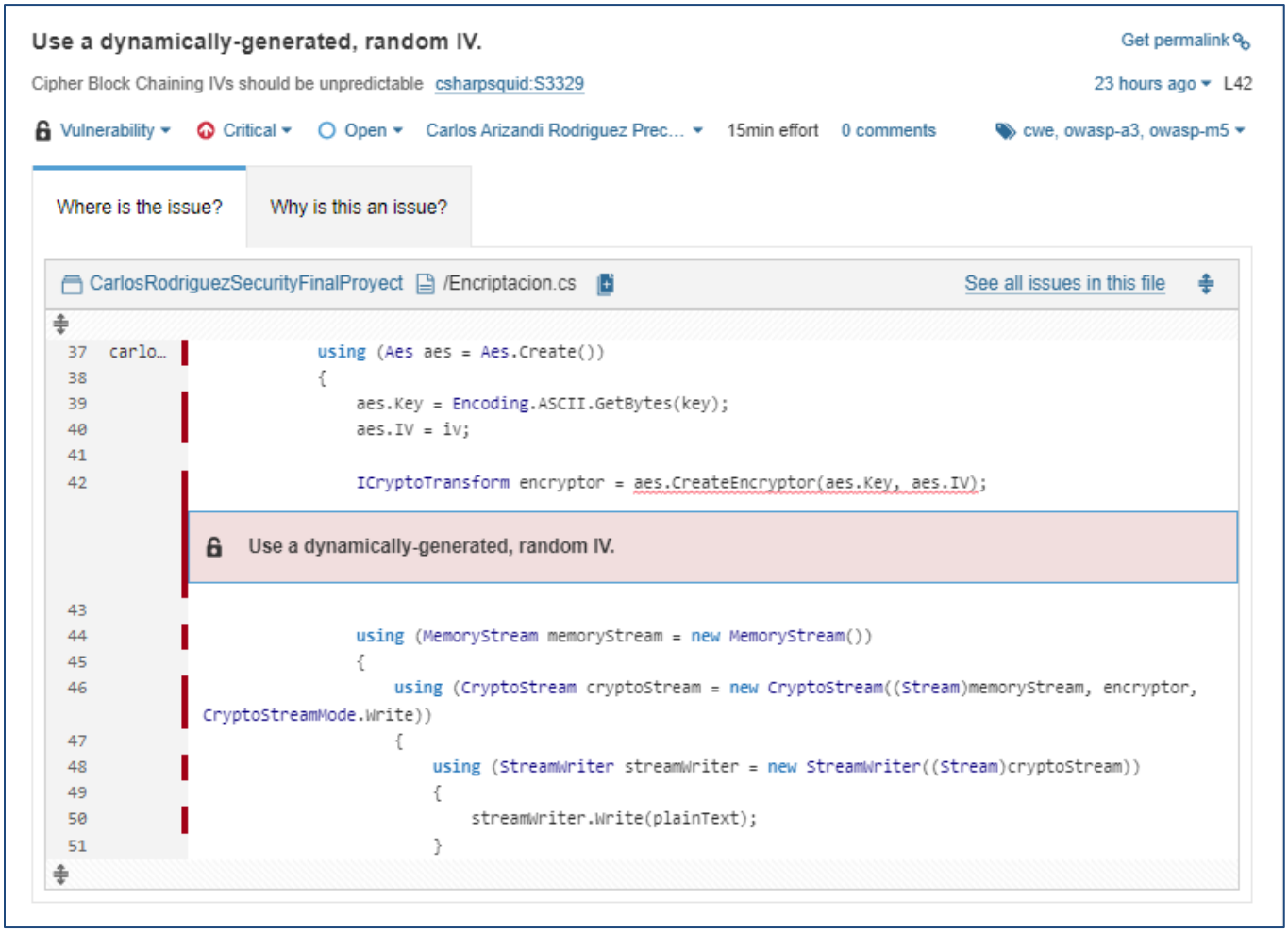
Vulnerabilidades de gravedad critica.

Use a dynamically-generated, random IV.

Esta vulnerabilidad consiste en el cifrado **Cipher Block** donde recomiendan utilizar un vector de inicialización (IV) para que el cifrado sea aleatorio.

No es necesario que el IV sea secreto, pero debe ser impredecible para evitar el **"Chosen-Plaintext Attack".**

Evidencia:



Recomendaciones

- Se recomienda generar vectores de inicialización, “NIST” recomienda utilizar un generador de números aleatorios seguro.
- Definición de claves aleatorias.

```
namespace SecurityProyect2
{
    2 references
    internal class Encriptacion{
        private readonly byte[] key = new byte[32];
        private readonly byte[] iv = new byte[16];

        1 reference
        private static string hashinSHA256(string dato)
        {
            // Create a SHA256 hash from string
            using (SHA256 sha256Hash = SHA256.Create())
            {
                // Computing Hash - returns here byte array
                byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(dato));

                // now convert byte array to a string
                StringBuilder stringbuilder = new StringBuilder();
```

Solución:

```
1 reference
private static byte[] Encrypt(string plaintext, byte[] key, byte[] iv)
{
    using (Aes aesAlg = Aes.Create())
    {
        aesAlg.Key = key;
        aesAlg.IV = iv;
        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
        byte[] encryptedBytes;
        using (var msEncrypt = new System.IO.MemoryStream())
        {
            using (var csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
            {
                byte[] plainBytes = Encoding.UTF8.GetBytes(plaintext);
                csEncrypt.Write(plainBytes, 0, plainBytes.Length);
            }
            encryptedBytes = msEncrypt.ToArray();
        }
        return encryptedBytes;
    }
}
```

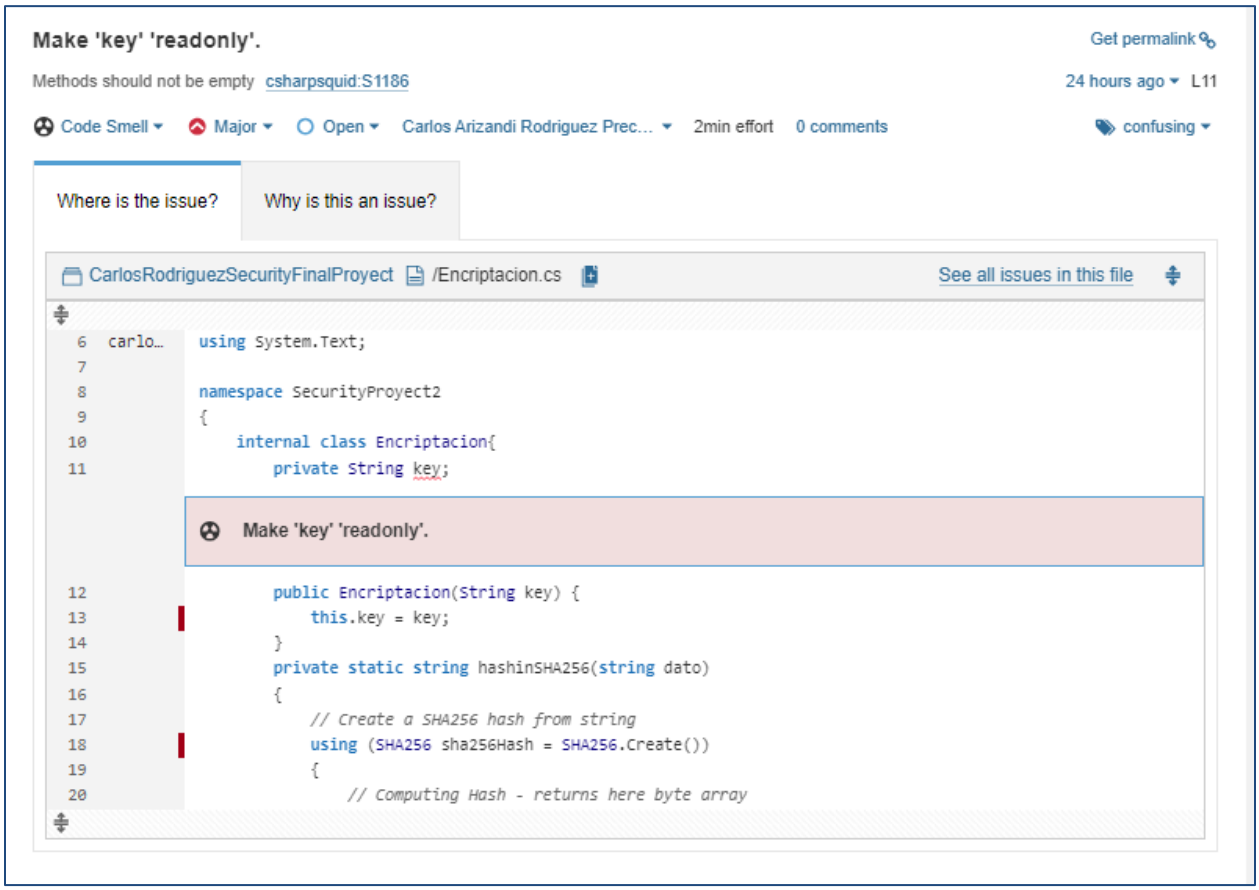
Vulnerabilidades de seguridad alta

En la revisión de las vulnerabilidades del proyecto, se detectó una vulnerabilidad de nivel alto, esta consiste en lo siguiente:

Make 'key' 'readonly'.

Esta vulnerabilidad indica que la llave debe estar definida como solo de lectura para que su valor no sea modificado.

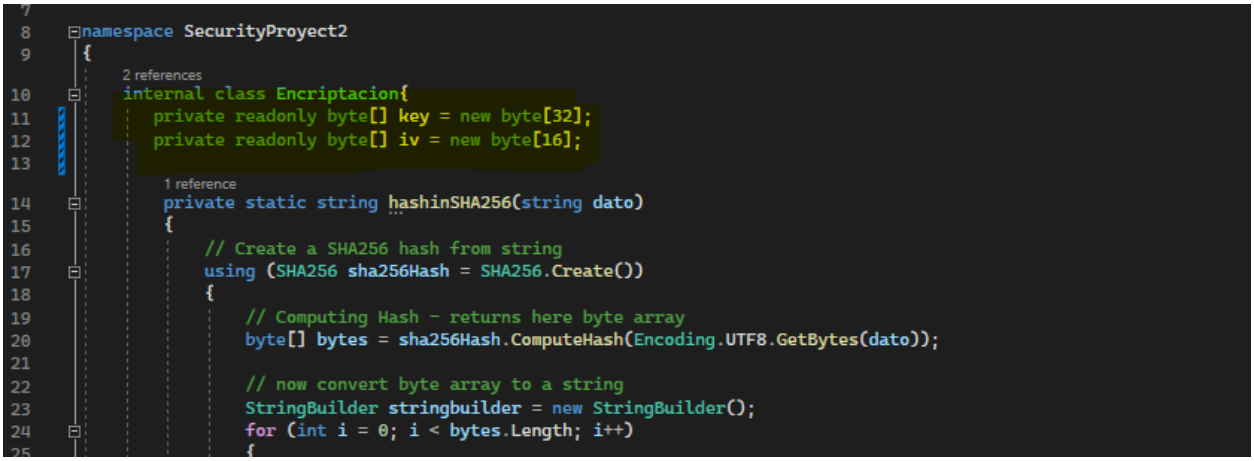
Evidencia:



Recomendaciones

- Se recomienda definir la variable como privada dentro de la clase con el atributo readonly, definir valor al iniciar la clase.

Solución



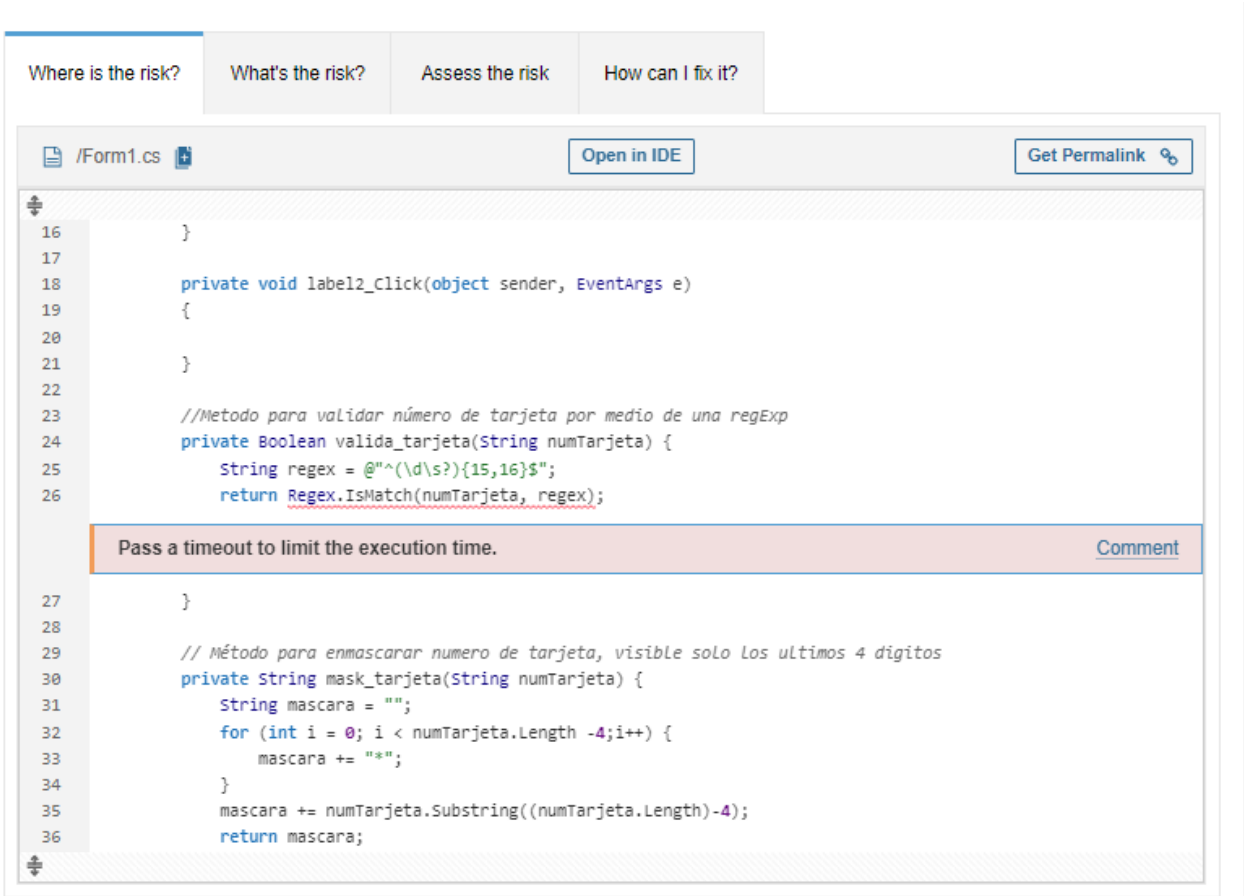
```
namespace SecurityProyect2
{
    2 references
    internal class Encriptacion{
        private readonly byte[] key = new byte[32];
        private readonly byte[] iv = new byte[16];

        1 reference
        private static string hashinSHA256(string dato)
        {
            // Create a SHA256 hash from string
            using (SHA256 sha256Hash = SHA256.Create())
            {
                // Computing Hash - returns here byte array
                byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(dato));

                // now convert byte array to a string
                StringBuilder stringbuilder = new StringBuilder();
                for (int i = 0; i < bytes.Length; i++)
                {
```

Vulnerabilidades de seguridad media

Denegación de servicio (DoS)

Se detectan 1 vulnerabilidades de este tipo, el cual nos están indicando una serie de expresiones reguladores, dichas expresiones regulares no están validando los datos de entrada de manera correcta, en algunos casos puede causar problemas de rendimiento y en el peor de los casos provocar una denegación de servicio de la aplicación.

Evidencia

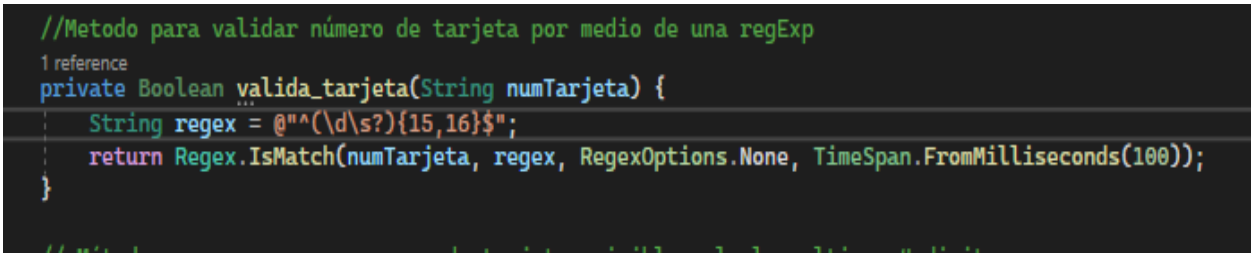


Recomendaciones

- Se recomienda utilizar o definir “matchTimeout”.
- Se recomienda revisar los patrones en busca de vulnerabilidades que puedan ocasionar una denegación de servicio

- Considerar utilizar un algoritmo sin retroceso especificando “RegexOptions.None”.

Solución

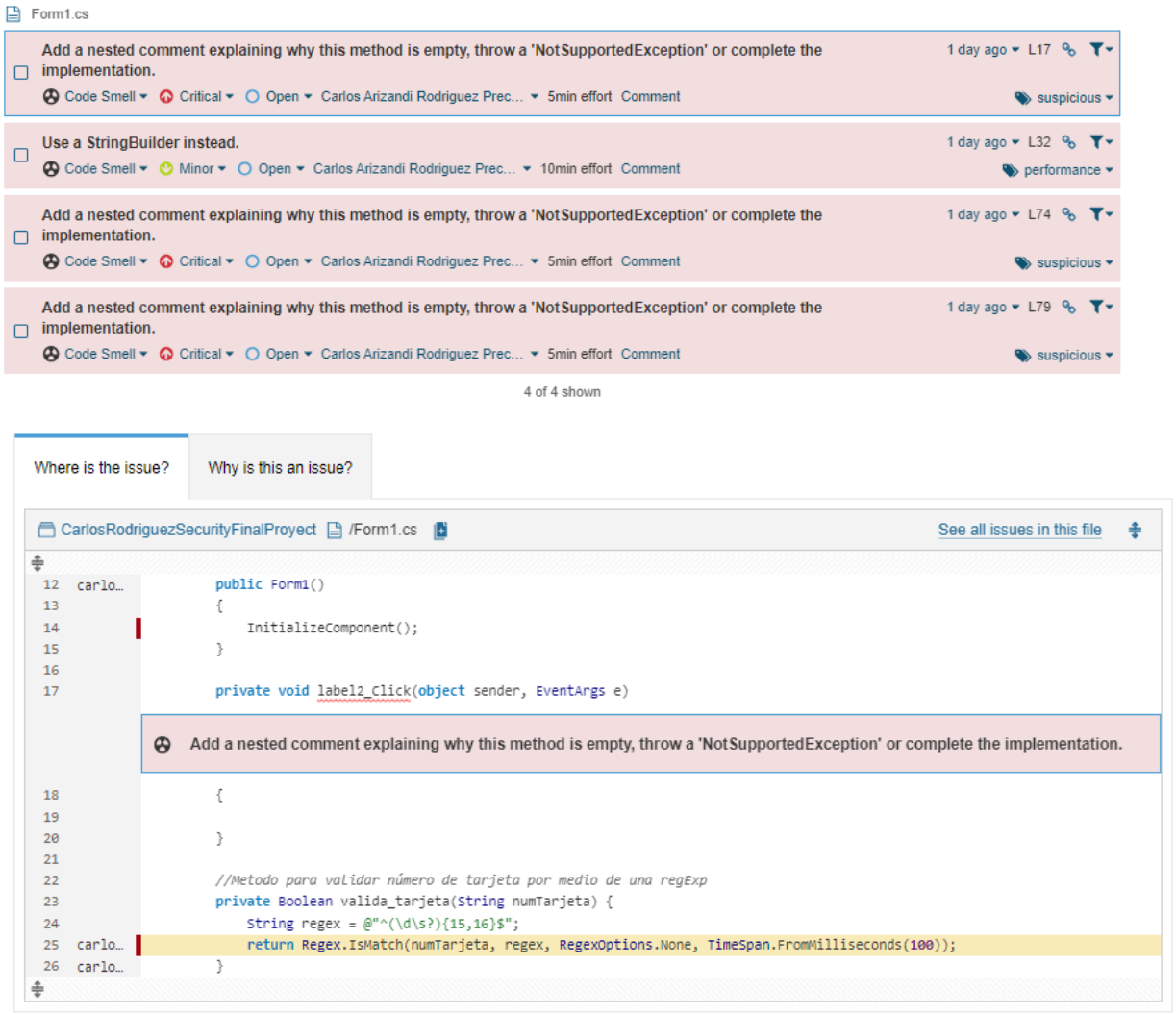


Vulnerabilidades de seguridad baja.

Hotspots Reviewed

En la revisión de las vulnerabilidades del proyecto, se encontraron 3 funciones definidas pero que no contienen código.

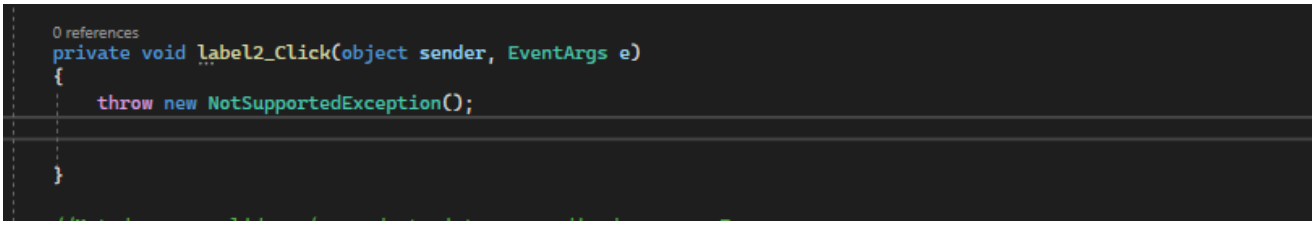
Evidencia:



Recomendaciones

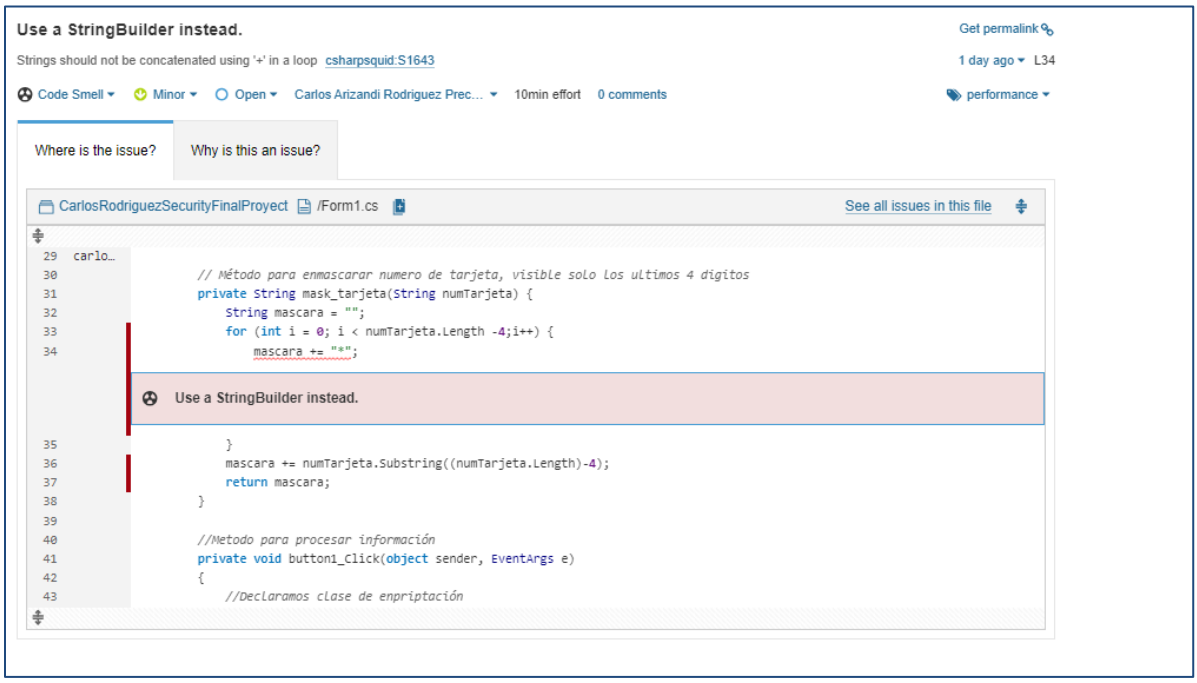
- Eliminar funciones que no cuentan con código
- Agregue método “throw new NotSupportedException()”

Solución



Code Smells

Evidencia:




```
29  carlo…
30          // Método para enmascarar numero de tarjeta, visible solo los ultimos 4 digitos
31          private String mask_tarjeta(String numTarjeta) {
32              String mascara = "";
33              for (int i = 0; i < numTarjeta.Length -4;i++) {
34                  mascara += "*";
35              }
36              mascara += numTarjeta.Substring((numTarjeta.Length)-4);
37              return mascara;
38          }
39
40          //Metodo para procesar información
41          private void button1_Click(object sender, EventArgs e)
42          {
43              //Declaramos clase de enpriptación
```

Recomendaciones

- "StringBuilderes" más eficiente que la concatenación de cadenas, especialmente cuando el operador se repite una y otra vez como en bucles.

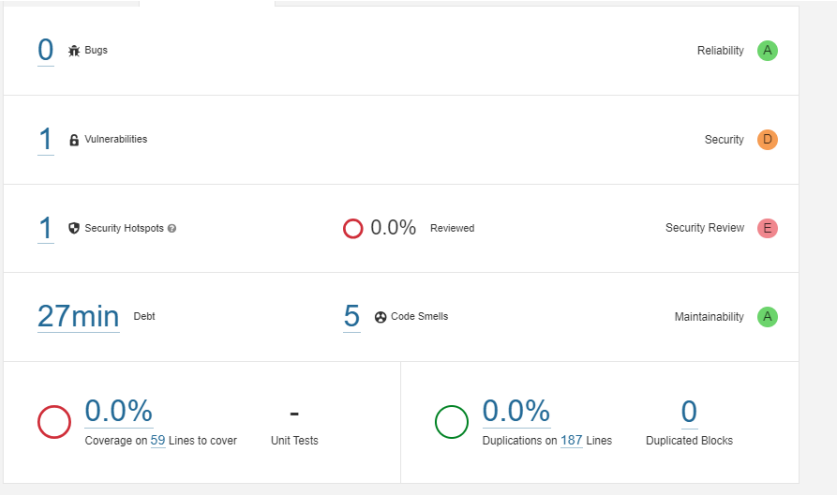
Solución

```
// Método para enmascarar numero de tarjeta, visible solo los ultimos 4 digitos
1 reference
private String mask_tarjeta(String numTarjeta) {
    StringBuilder bld = new StringBuilder();

    for (int i = 0; i < numTarjeta.Length -4;i++) {
        bld.Append("*");
    }

    bld.Append((numTarjeta.Length) - 4);

    return bld.ToString(); ;
}
```

Resumen

Con ayuda de SonarQube se detectaron las vulnerabilidades del proyecto, con lo cual se ha podido solucionar de manera efectiva estas problemáticas, dando como resultado un proyecto seguro y con las mejores prácticas de desarrollo implementadas.

Antes de aplicar la solución.



Después de aplicar soluciones.