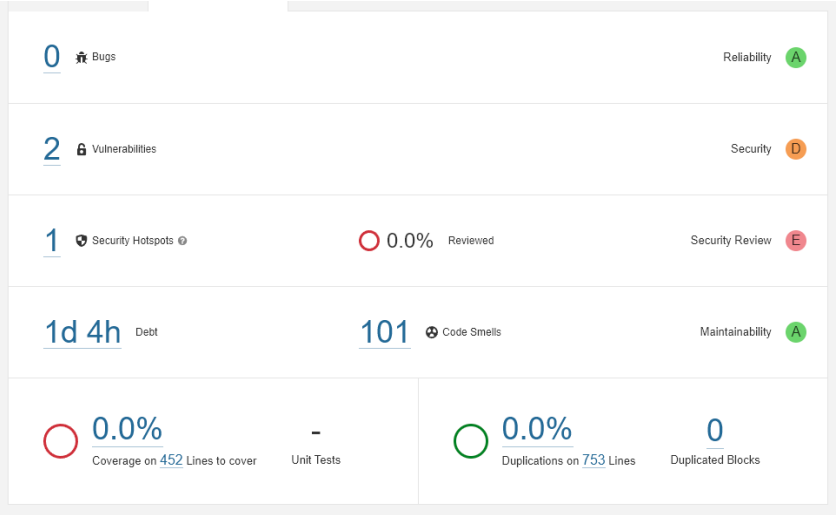
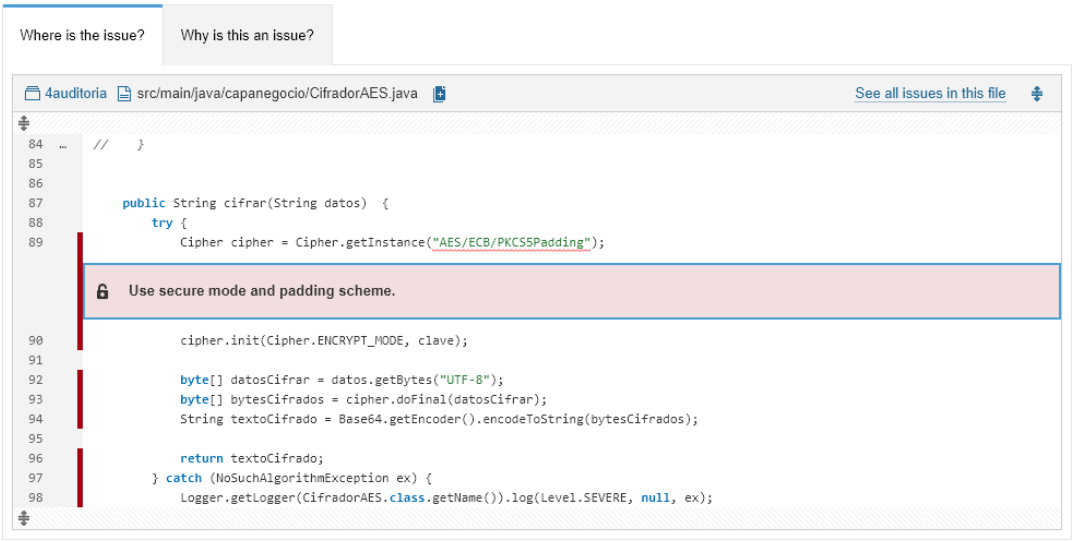
Informe herramienta SAST

Primer análisis de código:



SONAR detectó 2 vulnerabilidades.



Al entrar a ver cuáles eran se indicó que el problema estaba en esas líneas de código y se planteó la siguiente solución:

Encryption operations should use a secure mode and padding scheme so that confidentiality and integrity can be guaranteed.

- For block cipher encryption algorithms (like AES):
  - The ECB (Electronic Codebook) cipher mode doesn't provide serious message confidentiality: under a given key any given plaintext block always gets encrypted to the same ciphertext block. This mode should never be used.
  - The CBC (Cipher Block Chaining) mode by itself provides only data confidentiality. This cipher mode is also vulnerable to padding oracle attacks when used with padding. Using CBC along with Message Authentication Code can provide data integrity and should prevent such attacks. In practice the implementation has many pitfalls and it's recommended to avoid CBC with padding completely.
  - The GCM (Galois Counter Mode) mode which works internally with zero/no padding scheme, is recommended, as it is designed to provide both data authenticity (integrity) and confidentiality. Other similar modes are CCM, CWC, EAX, IAPM and OCB.
- For RSA encryption algorithm, the recommended padding scheme is OAEP.

Al encontrarse el problema en el cifrado y descifrado de la aplicación, se cambió a utilizar el algoritmo GCM.

Antes:

```java
public String cifrar(String datos)  {
    try {
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, clave);

        byte[] datosCifrar = datos.getBytes("UTF-8");
        byte[] bytesCifrados = cipher.doFinal(datosCifrar);
        String textoCifrado = Base64.getEncoder().encodeToString(bytesCifrados);

        return textoCifrado;
    } catch (NoSuchAlgorithmException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (NoSuchPaddingException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (UnsupportedEncodingException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (InvalidKeyException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IllegalBlockSizeException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (BadPaddingException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    }

    return null;
}
```

```java
public String descifrar(String datosCifrados) {
    try {
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, clave);

        byte[] bytesCifrados = Base64.getDecoder().decode(datosCifrados);
        byte[] datosDescifrados = cipher.doFinal(bytesCifrados);
        String datos = new String(datosDescifrados);

        return datos;
    } catch (NoSuchAlgorithmException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (NoSuchPaddingException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (InvalidKeyException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IllegalBlockSizeException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    } catch (BadPaddingException ex) {
        Logger.getLogger(CifradorAES.class.getName()).log(Level.SEVERE, null, ex);
    }

    return null;
}
```

Después:

```java
public String cifrar(String plaintext) throws Exception {

    byte[] iv = new byte[GCM_IV_LENGTH]; //NEVER REUSE THIS IV WITH SAME KEY
    secureRandom.nextBytes(iv);
    final Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
    GCMParameterSpec parameterSpec = new GCMParameterSpec(128, iv); //128 bit auth tag length
    cipher.init(Cipher.ENCRYPT_MODE, clave, parameterSpec);

    byte[] cipherText = cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF_8));

    ByteBuffer byteBuffer = ByteBuffer.allocate(iv.length + cipherText.length);
    byteBuffer.put(iv);
    byteBuffer.put(cipherText);

    String textoCifrado = Base64.getEncoder().encodeToString(byteBuffer.array());

    return textoCifrado;
}
```

```java
public String descifrar(String datosCifrados) throws Exception {
    final Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");

    byte[] encodedBytes = Base64.getDecoder().decode(datosCifrados);
//use first 12 bytes for iv
    AlgorithmParameterSpec gcmIv = new GCMParameterSpec(128, encodedBytes, 0, GCM_IV_LENGTH);
    cipher.init(Cipher.DECRYPT_MODE, clave, gcmIv);

    //use everything from 12 bytes on as ciphertext
    byte[] plainText = cipher.doFinal(encodedBytes, GCM_IV_LENGTH, encodedBytes.length - GCM_IV_LENGTH);

    return new String(plainText, StandardCharsets.UTF_8);
}
```

En el segundo análisis del código el resultado fue satisfactorio con 0 vulnerabilidades: