Universidad de Alcalá

Introducción a las prácticas con JAVA e Implementación de la vulnerabilidad *Zerologon*

Máster en ciberseguridad

Asignatura: Criptografía aplicada

Kevin van Liebergen

2021

Para esta práctica se han aunado las dos prácticas de JAVA, la parte introductoria junto con la de explotación de *Zerologon*

1 Introducción a las prácticas de Java

1.1. Objetivo

A lo largo de una serie de prácticas, se pretende mostrar cómo podemos basarnos en una plataforma de programación para implementar primitivas criptográficas.

En particular vamos a usar el lenguaje Java por medio del entorno de desarrollo NetBeans. Este entorno facilita la creación y gestión de proyectos, la edición del código fuente, la ejecución, la depuración y la optimización de aplicaciones generales hechas en Java. En nuestro caso, desarrollaremos algunos ejemplos sencillos que no requieren de grandes conocimientos ni de programación ni del uso de entornos de desarrollo.

Este documento presenta de manera sencilla rápida cómo empezar a manejarse con el entorno de desarrollo NetBeans con el que implementaremos diversas primitivas criptográficas.

La implementación de las primitivas se apoya en la existencia de unas bibliotecas con funciones criptográficas que funcionan de manera estándar, aunque estén proporcionadas por distintos fabricantes. En la jerga de Java estos fabricantes se conocen con el nombre de Providers. De entrada, nosotros usaremos el Provider por defecto, pero más adelante aprenderemos a usar otros

1.2. Desarrollo de la práctica

En esta primera práctica, nos familiarizamos con el entorno NetBeans. Después de presentarnos en el sistema, podemos seguir los siguientes pasos.

1. Arrancar el entorno NetBeans pulsando en el icono correspondiente situado en el Escritorio.

2. Seleccionar desde la barra de menu la opción File→New Project. Dentro de la categoría Java, elegimos un proyecto tipo Java Application.

3. Proporcionamos un nombre a nuestro gusto y una carpeta destino.

4. Pulsamos el botón Finish.

La herramienta nos presenta un cuadro con varias ventanas, conteniendo diversas informaciones. En la ventana derecha nos aparece un esqueleto de código Java que nos permite comenzar nuestro desarrollo.

Seguiremos las indicaciones del profesor para realizar una aplicación sencilla. Para guardarla, generarla y ejecutarla basta desplegar el menú Run dentro de la barra de menú y seleccionar la opción Run Project. La salida de nuestra aplicación aparecerá en la ventana inferior, etiquetada como Output.

El ciclo de desarrollo exige realizar cambios, generar y ejecutar la aplicación un número indeterminado de veces hasta conseguir los objetivos deseados.

Realizados los pasos seguidos en el enunciado anterior creamos un proyecto click en `New Project` como aparece en la figura 1.

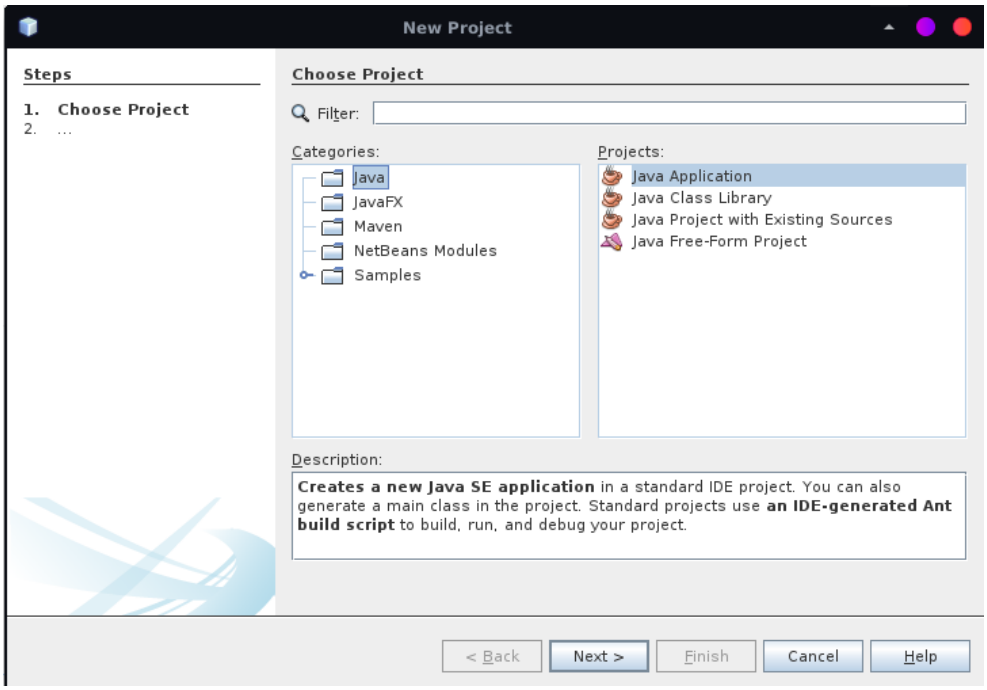

Figura 1: Creación del proyecto

Una vez hemos creado el proyecto vamos a copiar el fichero `ejemploaes.java` para poder ejecutar el fichero solicitado y poder trabajo en el siguiente capítulo acerca de *Zerologon*.

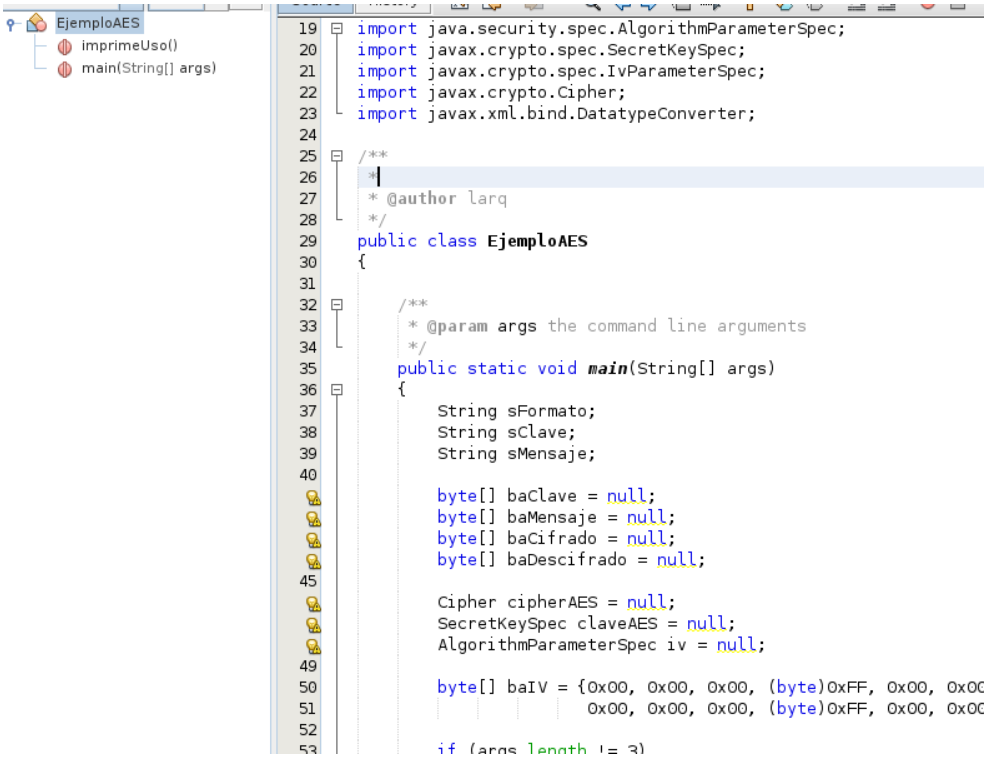

Figura 2: Creación del proyecto

2 Implementación de la vulnerabilidad *Zerologon*

2.1. OBJETIVO

En esta práctica se pretende implementar la vulnerabilidad conocida con el nombre Zerologon que permite al atacante adquirir pleno control sobre un servidor de dominio de Directorio Activo.

La vulnerabilidad actúa en el ámbito del protocolo Netlogon, utilizado en los servidores de dominio de Windows para, entre otras cosas, facilitar el acceso a los usuarios que quieran presentarse en el servidor usando cuenta y contraseña.

2.2. DESARROLLO DE LA PRÁCTICA

La vulnerabilidad está descrita con gran detalle en el documento adjunto, «Vulnerabilidad Zerologon».

Utilizando el entorno NetBeans y apoyándose en los ejemplos de cifrado ya realizados, implementaremos la vulnerabilidad tal como se describe en el citado documento.

Para implementar el exploit hemos implementado un mecanismo de creación de claves, se ha implementado en la línea 40 una creación de claves, inicializado la clave en modo AES y generado, vamos a insertar como texto plano el valor "00000000"de tal manera que con una clave específica la salida cifrada debe ser 00 para demostrar que el algoritmo se encuentra roto.

Para ello se ha elegido cifrar y descifrar mediante el algoritmo AES en modo CFB8, la clave generada se la pasamos a los algoritmos d cifrado y dscifrado en las líneas 49 y 57 respectivamente.

Se ha programado la aplicación para que mantenga creándose claves hasta que la probabilidad es de 1/256

```
1  public class EjemploAES
2  {
3      public static void main(String[] args) throws
           NoSuchAlgorithmException
4      {
5          String sFormato;
6          //String sClave;
7          String sMensaje;
8
9          byte[] baClave = null;
10         byte[] baMensaje = null;
11         byte[] baCifrado = null;
12         byte[] baDescifrado = null;
13
14         Cipher cipherAES = null;
15         //SecretKeySpec claveAES = null;
16         AlgorithmParameterSpec iv = null;
17
18         byte[] baIV = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
               0x00, 0x00,
19                         0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                               0x00, 0x00};
20
21         sFormato = "T";
22         // Mensaje todo a ceros
23         sMensaje = "00000000";
24         int contador = 1;
25
26         try
27         {
28             baMensaje = sMensaje.getBytes("ISO-8859-1");
29
30            // Creacion instancia
```

```
31              KeyGenerator keyGenerator =
                    KeyGenerator.getInstance("AES");
32
33              // Inicializacion
34              //SecureRandom secureRandom = new SecureRandom();
35              int keyBitSize = 256;
36              keyGenerator.init(keyBitSize);
37
38              while (true) {
39
40                  SecretKey claveAES =
                        keyGenerator.generateKey();
41
42                  // Configuracion
43                  cipherAES =
                        Cipher.getInstance("AES/CFB8/PKCS5PADDING");
44
45                  iv = new IvParameterSpec(baIV);
46
47                  // Cifrado
48                  //cipherAES.init(Cipher.ENCRYPT_MODE,
                        claveAES);
49                  cipherAES.init(Cipher.ENCRYPT_MODE, claveAES,
                        iv);
50                  baCifrado = cipherAES.doFinal(baMensaje);
51
52
53                  System.out.println("Cifrado: 0x" +
                        DatatypeConverter.printHexBinary(baCifrado));
54
55                  // Descifrado
56                  //cipherAES.init(Cipher.DECRYPT_MODE,
                        claveAES);
57                  cipherAES.init(Cipher.DECRYPT_MODE, claveAES,
                        iv);
58                  baDescifrado = cipherAES.doFinal(baCifrado);
59
60                  System.out.println("Descifrado: 0x" +
                        DatatypeConverter.printHexBinary(baDescifrado));
61
62                  System.out.println("\n");
63                  if (baCifrado[0]==0) {
64                      break;
65                  }
```

```
66                  else{
67                      contador++;
68                  }
69              }
70              System.out.println("Se ha intentado un total de:
                    "+contador+ " veces");
71          }
72          catch (Exception ex)
73          {
74              ex.printStackTrace();
75              System.out.println("EX: " + ex.toString());
76          }
77      }
78      ...
```

Por lo que realizando la iteración While true esperamos a que el mensaje cifrado tenga la mitad de bytes a 0 y nos ejecuta correctamente el programa.

```
Cifrado: 0x067D2FCB75C7E00C31B3E10446670E2A
Descifrado: 0x3030303030303030


Cifrado: 0x599E11F4035383C0C226FF90F029C3AC
Descifrado: 0x3030303030303030


Cifrado: 0x00000000000000003823B604B1F60114
Descifrado: 0x3030303030303030


Se ha intentado un total de: 122 veces
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 3: Resultado del programa