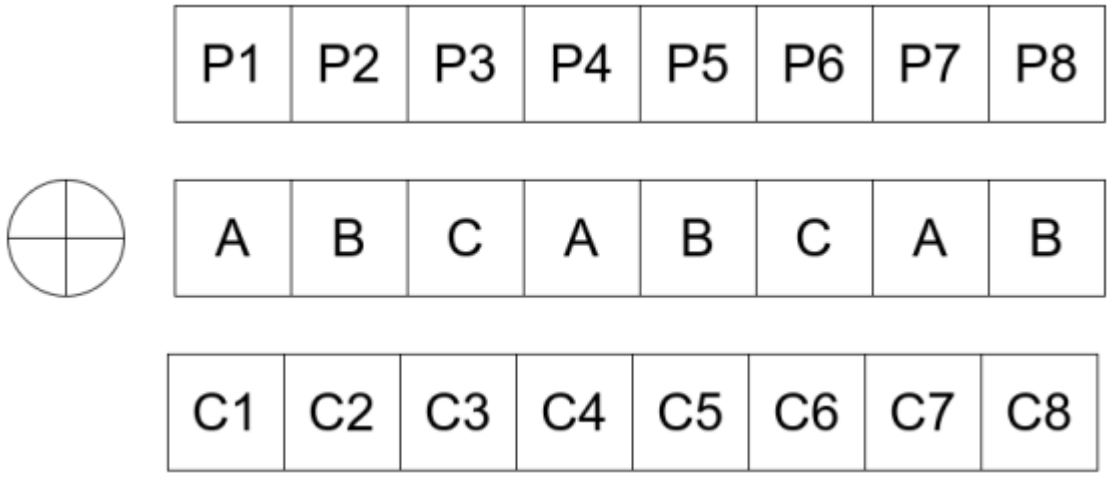
PRÁCTICA DE CRIPTOGRAFÍA

Yesurún González Román

1. Junto a este documento se adjunta un archivo ej1.txt que contiene ciphertext en base64. Sabemos que el método para encriptarlo ha sido el siguiente: La clave es de 10 caracteres. Frodo ha pensado que era poco práctico implementar one time pad, ya que requiere claves muy largas (tan largas como el mensaje). Por lo que ha decidido usar una clave corta, y repetirla indefinidamente. Es decir el primer byte de cipher es el primer byte del plaintext XOR el primer byte de la clave. Como la clave es de 10 caracteres, sabemos que el carácter 11 del ciphertext también se le ha hecho XOR con el mismo primer carácter de la clave. Debajo vemos un ejemplo donde la clave usada es “ABC”.



¿Por qué no ha sido una buena idea la que ha tenido Frodo? Explica cómo se puede encontrar la clave y desencriptar el texto a partir del archivo.

En este caso Frodo ha tomado una mala decisión. Al observar la cantidad de bytes del del ciphertext, se obtienen 1757. Como la clave contiene 10 caracteres, Frodo ha empleado la misma clave 175 veces con el objetivo de encriptar el texto. 175 es un número muy grande que nos ofrece muchas muestras en las que poder analizar qué caracteres debería tener la clave, a través de métodos entrópicos, ya que sabemos que Gandalf y Frodo se comunican en inglés y el tipo de frecuencia de uso que tiene cada letra de su abecedario.

Para encontrar la clave, hay que dividir el ciphertext en grupos de diez bytes. De forma que los diez primeros bytes integran un grupo, los diez siguientes otro, y así sucesivamente.

Posteriormente empleamos fuerza bruta de encriptación XOR en el primer byte de cada grupo con los primeros 128 posibles bytes de los cuales Frodo habrá utilizado uno (presuponemos que ha empleado una clave con caracteres ascii). Después el mismo proceso con el segundo byte de cada grupo, y así hasta realizarlo con el décimo byte.

De los 128 bytes empleados en la decodificación XOR retenemos el byte que haya mostrado mayor puntuación en términos entrópicos.

Como resultado, averiguamos que la clave empleada ha sido “keepcoding” y el texto original es la letra de la canción “Never gonna give you up”

Programa el código y encuentra la clave junto con el texto. Pista: no debería ser muy costoso, ya que se puede reutilizar mucho código de un ejercicio hecho en clase 😉

La parte de código que he programado se encuentra en el archivo descifrandoFrodo.py

2. En el archivo ej2.py encontramos dos funciones. Una de ellas obtiene una string cualquiera y genera una cookie encriptada con AES CTR, la función genera cuentas con rol user, y se encarga de verificar que la string no contiene ni ';' ni '='. La otra función desencripta la cookie y busca la string ';admin=true;'. Realizando llamadas a estas dos funciones, genera una cookie y modifica el ciphertext tal que al llamar la función de check te devuelva permisos de admin. (Obviamente se puede acceder a la clave de encriptación de la clase, pero el ejercicio consiste en modificar el sistema tal y como si el código se ejecutara en un servidor remoto al que no tenemos acceso, y solo tuvieramos acceso al código).

Pista: Piensa en qué efecto tiene sobre el plaintext en CTR cuando cambia un bit del ciphertext

Como tenemos conocimiento de cuál es el plaintext tan solo tenemos que emplear XOR en el plaintext con el ciphertext para obtener la clave. Ejecutamos XOR entre la clave y un nuevo plaintext que contenga ";admin=true;" para generar ciphertext malicioso y obtener acceso de administrador.

3. Explica cómo modificarías el código del ejercicio 2 para protegerte de ataques en los que se modifica el ciphertext para conseguir una cookie admin=true.

Guardaría un hash de la cookie generada relacionada al usuario. Cada vez que se inspeccione que si un usuario tiene acceso privilegiado, comprobaría que el hash de la nueva cookie y el hash de la cookie guardada por usuario coinciden idénticamente. Si no coincide, no se ejecuta la función.

4. Existe otro error en la implementación del Server que le puede hacer vulnerable a otro tipo de ataques, concretamente en los parámetros que usa para CTR. ¿Sabrías decir cuál es? ¿Cómo lo solucionarías?

La clave aleatoria se genera en el momento que se crea la clase. Esto nos permite generar infinidad de cookies con la misma clave en la misma clase. Esto es un error, ya que nos permite descubrir la clave con fuerza bruta sin necesidad de saber el plaintext. Se debería añadir una variable "contador" para que cada vez que se emplee la clave, el contador sume una posición y en conjunción con el nonce, se genere una nueva clave.

5. En el archivo ej5.py encontrarás una implementación de un sistema de autenticación de un servidor usando JWT. El objetivo del ejercicio es encontrar posibles vulnerabilidades en la implementación del servidor, y explicar brevemente cómo podrían ser atacadas.

- La función register crea un nuevo usuario en el sistema (en este caso guardado localmente en el objeto AuthServer en la lista users).
- La función login verifica que el password es correcto para el usuario, y si es así, devuelve un JWT donde sub = user y con expiración 6h después de la creación del token.
- La función verify verifica que la firma es correcta y devuelve el usuario que está autenticado por el token.

Para JWT usamos la librería PyJWT: https://pyjwt.readthedocs.io/en/latest/

Para funciones criptográficas, usamos la librería PyCryptoDome usada anteriormente en clase.

He identificado dos vulnerabilidades en AuthServer:

1. **El servidor comete un error al ejecutar la función de registro. Si dos usuarios se registran con el mismo nombre, se guardará solamente el hash de la contraseña de la última persona en registrarse. Este fallo permite a cualquier atacante obtener acceso a la cuenta de cualquier usuario, reemplazando además, su contraseña por una nueva que desconoce.**
2. **Cuando se ejecuta la función de verificación de firma de JWT, se aceptan JWT que no contengan ningún algoritmo. Este fallo permite a cualquier atacante falsificar su autenticación.**

6. Modifica el código para solucionar las vulnerabilidades encontradas.

El archivo que contiene el código modificado se llama AuthServer_seguro.py

He implementado un selector que no permite el registro con nombres de usuario ya existentes, y he eliminado la opción "None" de algoritmos en la función verify.

7. ¿Qué problema hay con la implementación de AuthServer si el cliente se conecta a él usando http? ¿Cómo lo solucionarías?

Cada vez que una persona se registra o inicia sesión, manda datos de los cuales se encuentran el nombre de usuario y la contraseña en claro (sin encriptar). Esto permite a un tercero que observe la comunicación, obtener información sensible que le otorgaría acceso ilegítimo a la cuenta.

Para evitar esto, se recomienda el uso de protocolo https, que transmite la información de forma encriptada. Para poder implementar el protocolo https es tan simple como generar un certificado SSL (incluso de forma gratuita).

8. Sabemos que un hacker ha entrado en el ordenador de Gandalf en el pasado. Gandalf entra en internet, a la página de Facebook. En la barra del navegador puede ver que tiene un certificado SSL válido. ¿Corre algún riesgo si sigue navegando?

Un certificado SSL nos asegura que el servidor con el que nos estamos comunicando (en este caso Facebook) al menos está relacionado con su verdadero dueño y no está controlado por un ciberdelincuente, por ejemplo.

Sin embargo, el hacker podría haberse infiltrado y ser capaz de observar cualquier comunicación que Gandalf mantuviese con cualquier servidor. Para que el navegador no notase la presencia del hacker como maligna o insegura, el hacker podría haberle emitido un certificado SSL propio y haber hecho que el navegador lo aceptase en el momento que consiguió entrar al ordenador en el pasado.

Luego sí, se corre el riesgo de que toda información sensible transmitida en la comunicación con Facebook, sea expuesta.

9. En el archivo sergi-pub.asc mi clave pública PGP:

- Comprueba que la fingerprint de mi clave es: 4010179CB16FF3B47F7D09C4751DD875E2FBBF59
- Genera un par de claves, y añade tu clave pública a los archivos entregados, el archivo con tu clave pública tiene que estar encriptado para que lo pueda ver sólo con mi clave privada.
- Usa tu clave privada para firmar el archivo de entrega, y adjunta la firma en un documento firma.sig (Recuerda que deberás hacer la firma con la versión final que entregues, ya que si realizas algún cambio, la firma no será válida).

Explica los pasos que has seguido.

Para importar la clave he ido a la terminal y he ejecutado el siguiente comando: gpg --import sergi-pub.asc.

Para comprobar que la fingerprint de la clave importada coincida con 4010179CB16FF3B47F7D09C4751DD875E2FBBF59 he ejecutado el comando gpg --list-keys. La fingerprint coincide.

Para generar clave privada y clave pública, he empleado el comando gpg2 --gen-key.

Para poder compartir la clave he realizado el comando gpg --armor --export yesurun > yesurun-pub y lo he exportado al archivo yesurun-pub.

Para que Sergi sea el único que pueda ver mi clave he empleado el comando gpg --output yesurun-pub.gpg --encrypt --armor --recipient Sergi yesurun-pub

Para firmar esta práctica he empleado el comando gpg --output firma.sig --detach-sign practica_criptografia.pdf