Ciberseguridad Full Stack Bootcamp III Edición

PRÁCTICA CRIPTOGRAFÍA: Alumno José David Ortiz Cruz

Comentarios para el profesor.

De conformidad con lo expuesto en el feedback del profesor sobre la primera entrega de la práctica de criptografía, el alumno que suscribe, ha llevado a cabo la revisión de los ejercicios considerados deficientes, omitiendo en todo caso repetir aquellos ejercicios considerados correctos por el profesor en la primera entrega.

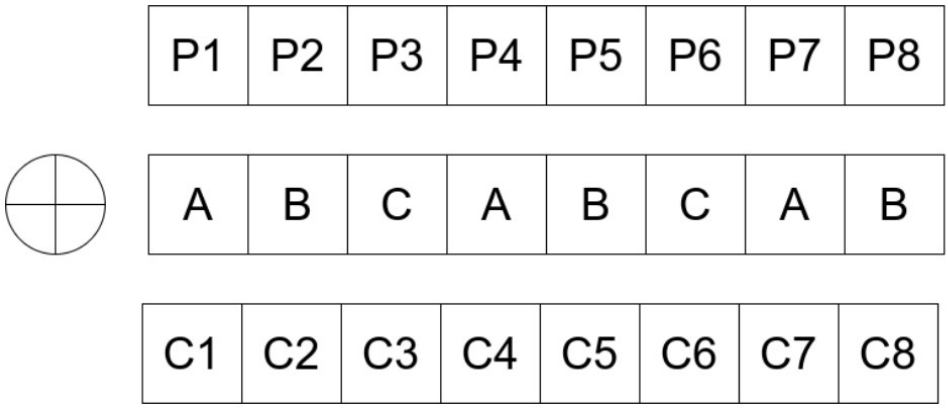
Las modificaciones llevadas a cabo, se consignan de **color azul** a lo largo de la presentación de la práctica, manteniendo en color negro el texto de repuesta de la primera entrega.

Dado que en el feedback del profesor se advirtió como mayor deficiencia la falta de detalle en las respuestas y que el código elaborado por el alumno que suscribe no funcionaba, para esta reentrega he considerado idóneo centrarme en desarrollar mis explicaciones y no entregar código, ya que tampoco dispongo del conocimiento de desarrollo necesario para elaborarlo con garantías de funcionamiento. Esperando con ello, que la ampliación de esta reentrega cumpla los mínimos suficientes para considerar APTA la práctica.

Ejercicios:

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.

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 ;)

En One Time Pad la key debería tener el mismo tamaño que el plaintext. Al utilizar una clave de menor tamaño, los bytes con los que se hace XOR se repiten y por tanto se puede analizar la frecuencia de aparición de las letras.

Observando que la clave tiene una longitud de 10 caracteres, la aplicación del análisis de frecuencia consistiría la división del ciphertext en grupos de 10 caracteres para posteriormente reagruparlos según suposición, generando así un paquete con las primeras letras de cada grupo, otro paquete con las segundas letras de cada grupo y así sucesivamente hasta tener diez paquetes.

Ejecutaremos XOR sobre cada uno de los referidos paquetes probando todos los posibles bytes, obteniendo cadenas de caracteres.

Compararemos los caracteres de estas cadenas con las letras del idioma en cuestión (en este caso inglés) que tengan mayor frecuencia de aparición y así montaremos frases que al recolocar cada carácter en su posición original puedan tener sentido.

El análisis manual al que se ha hecho referencia puede automatizarse mediante el uso un script que nos permitirá obtener la clave y por tanto pasar el cyphertext a plaintext.

Para realizar el análisis de frecuencia utilizaremos un diccionario en el que se ordenan las letras más frecuentes de la lengua inglesa, a estas letras se les tiene asignada una puntuación en función de si tienen una frecuencia de aparición más alta o más baja.

Agruparemos el ciphertext en bloques de 10 caracteres (ya que éste es el tamaño de la clave).

A continuación llamamos a la función "transpose" para crear mensajes compuestos de todas las primeras letras, todas las segundas letras, etc…

Seguidamente, con la función "brute_force" vamos probando combinaciones y asignando una puntuación a cada combinación, dejando almacenadas en memoria aquellas combinaciones que alcancen mayor puntuación mediante el uso de búcles for e if. Al final del búcle devolverá la mejor hallada.

Al repetir esta acción con todo el cipher text obtendremos la clave.

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

   Bit-flipping Attack. Cuándo modificamos un bit en el ciphertext, al desencriptarlo se observará la modificación en el plaintext. Esto debido a que la encriptación se ha realizado ejecutando XOR sobre el plaintext con el keystream.

   Tenemos que conseguir que la función check compruebe que el plaintext se incluya ';amin=true;'. Como sabemos que al alterar el ciphertext se verá modificado el plaintext, realizaremos cambios en la información de la cookie.

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.

   AES CTR es vulnerable a modificaciones en el ciphertext. Podemos añadir un authentication code.

   Podrían evitarse este tipo de ataque utilizando HMAC (Hash Message Authentication Code).

   Con HMAC obtenemos un primer hash de la clave y un segundo hash de la clave y el cypher text juntos.

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?

   Si reutilizamos el NONCE que se genera en el servidor para encriptar diferentes mensajes estaremos realizando XOR a multiples plaintext con el mismo keystream, similar al ejercicio uno dónde utilizábamos los mismos caracteres de la clave para encriptar diferentes caracteres del mensaje (One Time Pad), siendo por tanto vulnerable al análisis de frecuencia codificado en el adjunto del ejercicio ej1.py. Tendremos que generar un NONCE para cada mensaje y enviarlo con éste para no caer en el error del primer ejercicio.

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.

    La función hash utilizada es SHA1 la cuál es vulnerable por haberse detectado colisiones en la misma. Una forma más segura de almacenar la contraseña sería con una función SHA2 o SHA3, el problema de esto es que estos algoritmos suelen ser más pesados.

    Una posible solución sería modificar el código para que por cada password se almacenasen pares de hashes md5 y SHA1, de esta manera además de hacerse prácticamente imposible que se diesen colisiones, se limitaría la obtención del hash mediante un ataque de fuerza bruta.

    Otra vulnerabilidad en sí es que se almacene únicamente el hash de la password, posibilitando su obtención mediante la consulta de rainbowtables. Sería más conveniente modificar el código para que en lugar de obtener el hash únicamente de la password, se obtuviese de convinar un factor aleatorio (salt) con la misma.

    Una última posibilidad de mejora del código sería implementar el uso de funciones de derivación de claves como PBKDF2 que hashean los propios hashes sucesivamente lo que limita la eficacia de ataques de fuerza bruta al hacer que estos arrojasen resultados de forma infinitamente más lenta.

6. Modifica el código para solucionar las vulnerabilidades encontradas.

    Respondido en el ejercicio 5.

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

Las llamadas de registro y acceso (def register y def login) no serian encriptadas, siendo vulnerables a ser capturadas por un ataque de Man In The Middle utilizando un snnifer y leidas fácilmente al encontrarse en plaintext. También podría obtener el Json Web Toquen con la llamada de verificación (def verify). Para solucionarlo deberíamos utilizar el protocolo https validando su correcta configuración e implementación mediante la obtención de un certificado SSL/TLS.

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/TLS valida la conexión a través https, esto implica que los datos quedan protegidos de un atacante que estuviese observando la comunicación (Man In The Middle), pero no oculta el hecho de la comunicación. Es decir, que una persona que estuviese observando esta comunicación, vería que nos estamos comunicando con Facebook. Dicho esto, si el hacker entró en el ordenador de Gandalf en el pasado, podría tener acceso a las claves y descifrar los mensajes. Además mediante un ataque de Man In The Browser, instalando el atacante una extensión del navegador maliciosa en el equipo de Gandalf, éste podría estar accediendo a un sitio Web del atacante, haciéndole éste creer a Gandalf que accede a Facebook e introduciendo sus credenciales de acceso (usuario y contraseña).

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.

Utilizando una distribución de Kali Linux arrancada como maquiina virtual con la aplicación Virtual Box, accedemos a la terminal de comandos e introducimos los siguientes mandatos utilizando el protocolo de encripatación asimétrico GPG:

1. Para generar un clave, usamos el comando: > gpg2 --gen-key

2. Exportamos la clave generada utilizando el fingerprint mostrado en la consola con el comando: gpg --armor --export FA0737DDC33070C5C5BC78D39BE62E2E3889C350 > miclave.txt

3. Generamos otro fichero que contenga la clave privada utilizando el comando gpg --armor --export --secret-key FA0737DDC33070C5C5BC78D39BE62E2E3889C350 > miclavePrivada.txt

4. Encriptamos el fichero miclave.txt con la clave del fichero Sergi.asc utilizando el siguiente comando: > gpg --output miclave.txt --encrypt --recipient yokran Sergi.asc y lo renombraremos como miclaveEncriptada.txt

5. Crearemos una firma utilizando el comando > gpg --output mifirma.sig –clearsign miclave.txtgp

6. Podemos verificar la firma utilizando el comando gpg –verify mifirma.sig

7. Tomamos el directorio denominado PracticaCriptografia y lo convertimos en un fichero.zip para firmarlo utilizando el comando > gpg –armor –output mifirma.sig –detach-sign PracticaCriptogradia.zip

Adjunto a la presente practica, se encuentran los ficheros que contienen el código utilizado en los diferentes ejercicios cuya resolución lo requería.

Cáceres a 6 de mayo de 2.022

El alumno:

Fdo.: José David Ortiz Cruz