UNIVERSIDAD DE LOS ANDES

DEPARTAMENTO DE INGENIERIA DE SISTEMAS Y COMPUTACIÓN

LABORATORIO: Cifrado de Datos

ISIS 3311 – Ciberseguridad

Grupo 2

Edward Camilo Sánchez Novoa – 202113020

Daniel Rudas Bohorquez - 202112926

Manuel

Contenido

Laboratorio 2

1. Introducción

En el presente laboratorio se analiza la seguridad de las comunicaciones en un entorno distribuido compuesto por un servidor FTP y un servidor de base de datos MySQL, accesibles desde una máquina cliente Kali Linux. El objetivo principal es identificar vulnerabilidades relacionadas con la transmisión de información sensible y aplicar mecanismos de protección que mitiguen los riesgos asociados a la exposición de credenciales y datos en la red.

Inicialmente, se evalúan las comunicaciones utilizando los servicios en su configuración por defecto, con el fin de determinar si la información transmitida viaja en texto plano. Mediante el uso de herramientas de análisis de tráfico como Wireshark y tcpdump, se inspeccionan las conexiones establecidas hacia el servidor FTP y la base de datos, identificando posibles debilidades en términos de confidencialidad.

Posteriormente, se implementan mecanismos de protección orientados a asegurar tanto los datos en tránsito como los datos en reposo. En el caso del servicio FTP, se utiliza SFTP sobre SSH para garantizar el cifrado del canal de comunicación. Para la base de datos, se establecen conexiones mediante TLS, verificando el uso de cifrado moderno. Adicionalmente, se desarrolla un script en Python que implementa cifrado simétrico AES en modo GCM, permitiendo proteger archivos antes de su transmisión.

Finalmente, se realiza un análisis comparativo entre las comunicaciones inseguras y las protegidas, evidenciando las diferencias mediante capturas de red. Este proceso permite comprender el impacto real de la ausencia de cifrado en servicios críticos y la importancia de implementar mecanismos adecuados de seguridad en entornos productivos.

2. Análisis de comunicaciones inseguras

2.1 FTP EN TEXTO PLANO

Creación de Archivo con Datos Sensibles

Para simular un escenario real, se creó un archivo conteniendo información bancaria confidencial relacionada con datos bancarios del hospital:

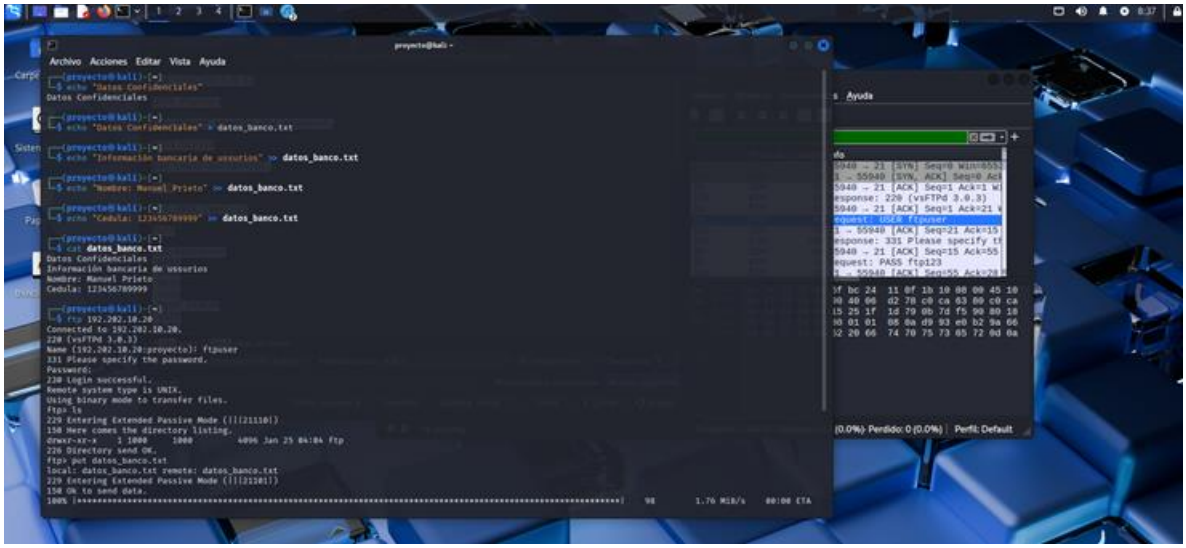


Figura 1: *Creación y transferencia del archivo por FTP*

En la imagen se observa la sesión FTP donde se transfirió el archivo datos_banco.txt conteniendo información sensible.

Intercepción de Credenciales FTP

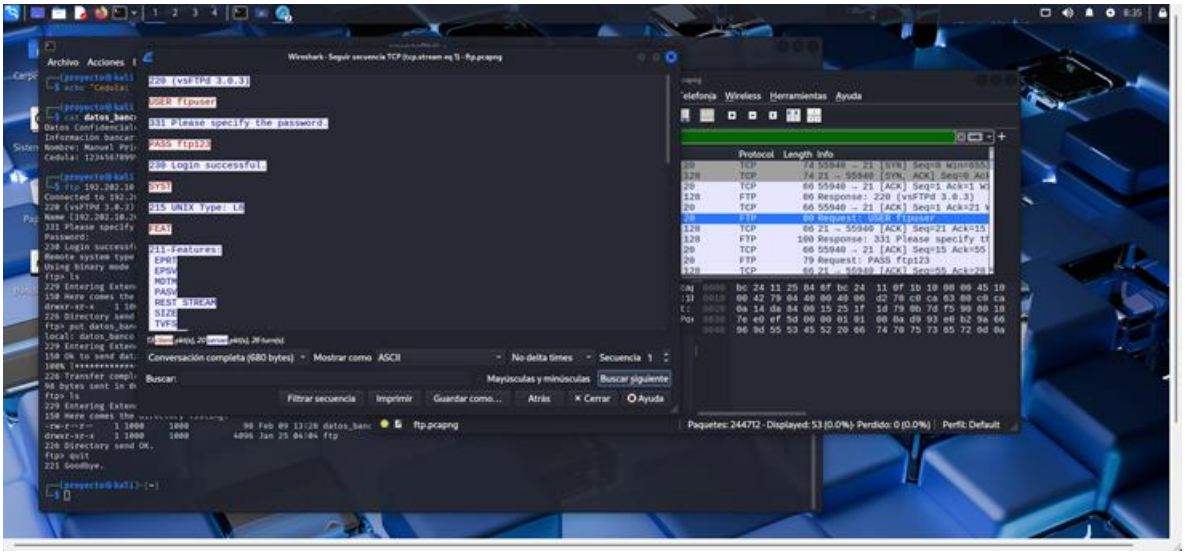


Figura 2: *Captura de credenciales FTP en Wireshark*

En esta captura de Wireshark se puede evidenciar que las credenciales de autenticación FTP son completamente visibles en texto plano. Al analizar el flujo TCP en Wireshark, se revelan las credenciales de autenticación que son ftpuser como nombre de usuario y ftp123 como contraseña.

Intercepción del Contenido del Archivo

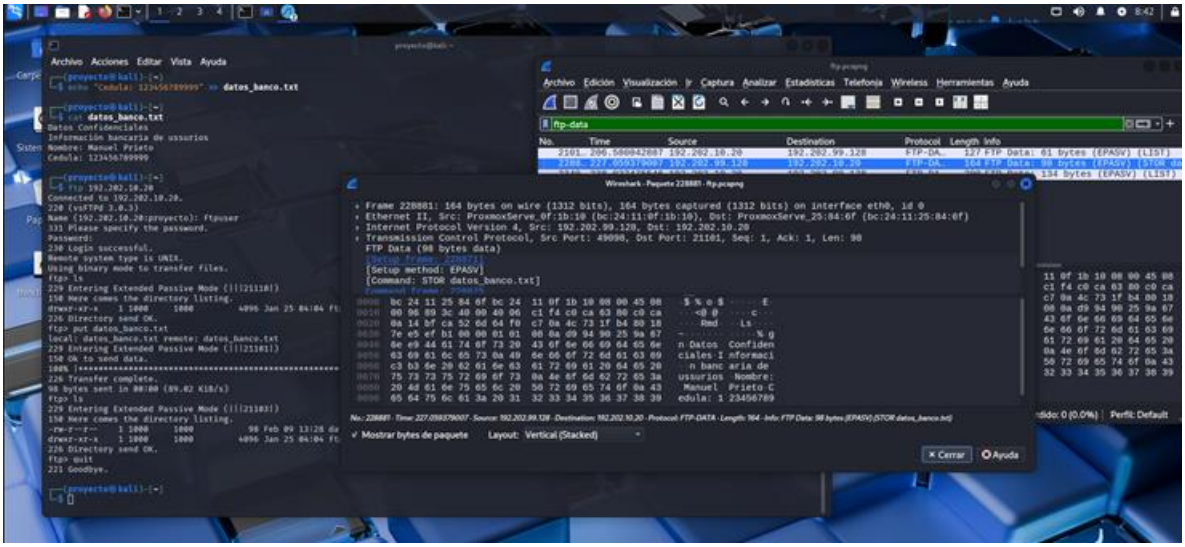


Figura 3: *Vista general del tráfico FTP-DATA*

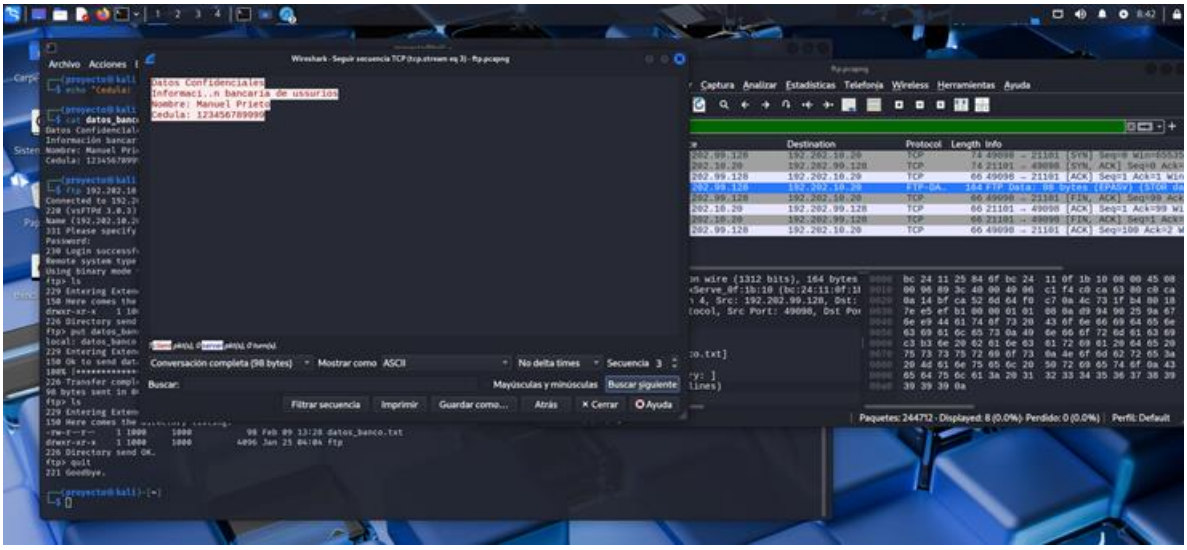


Figura 4: *Contenido del archivo interceptado (Follow TCP Stream)*

El contenido completo del archivo transferido es visible en texto plano, incluyendo toda la información confidencial del cliente como se ve en las images. Las Figuras 3 y 4 muestran cómo el protocolo FTP-DATA transmite el contenido del archivo sin ningún tipo de cifrado. Un atacante con acceso a la red puede Leer completamente el contenido de archivos sensibles, acceder a información personal de clientes (nombres, cédulas, diagnósticos) y obtener datos financieros y administrativos del banco.

2.2 MYSQL SIN TLS

Configuración del análisis

Figura 5: *Consultas realizadas en MySQL*

Se ejecutaron consultas SQL simulando operaciones reales del banco en la base de datos.

Figura 6: *Resultados de consulta con datos de usuarios*

La Figura 6 muestra los resultados de la consulta, revelando información de usuarios del sistema, incluyendo nombres completos de usuarios (Alice Sánchez, Roberto Gómez, Carolina Pérez, David Rodríguez), correos electrónicos corporativos, números telefónicos y direcciones físicas en Bogotá, Medellín, Cali y Barranquilla.

Análisis del tráfico MySQL en Wireshark

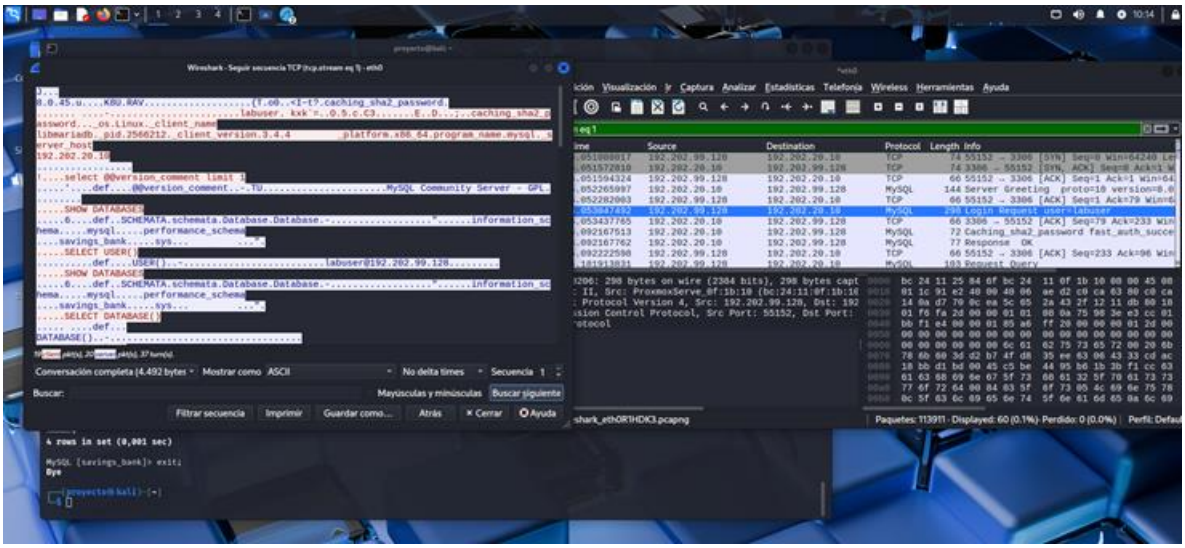


Figura 7: *Paquetes MySQL capturados en Wireshark*

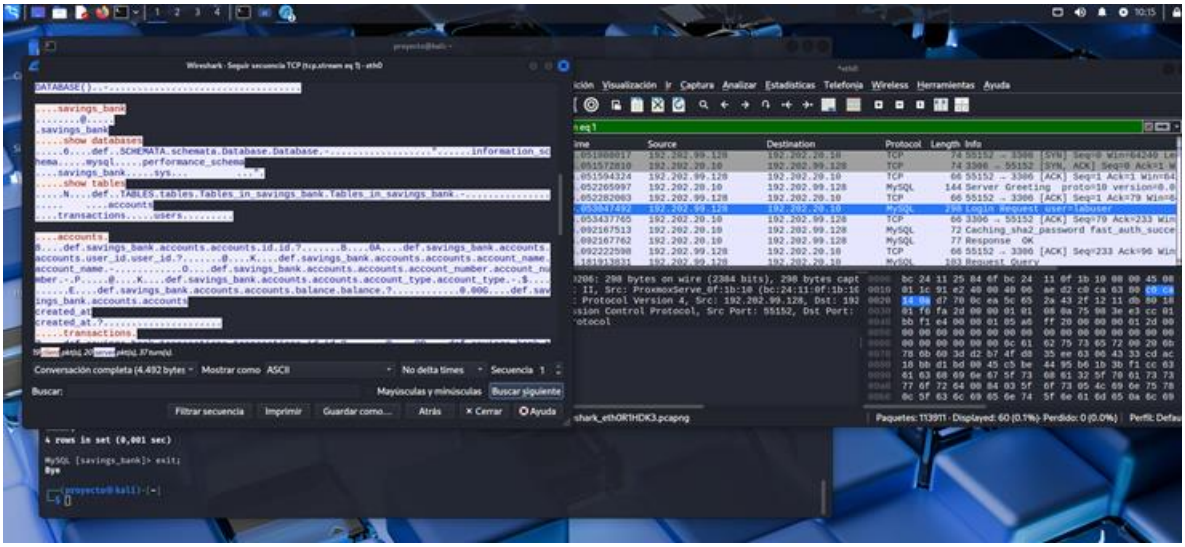


Figura 8: *Contenido del flujo TCP mostrando consultas SQL*

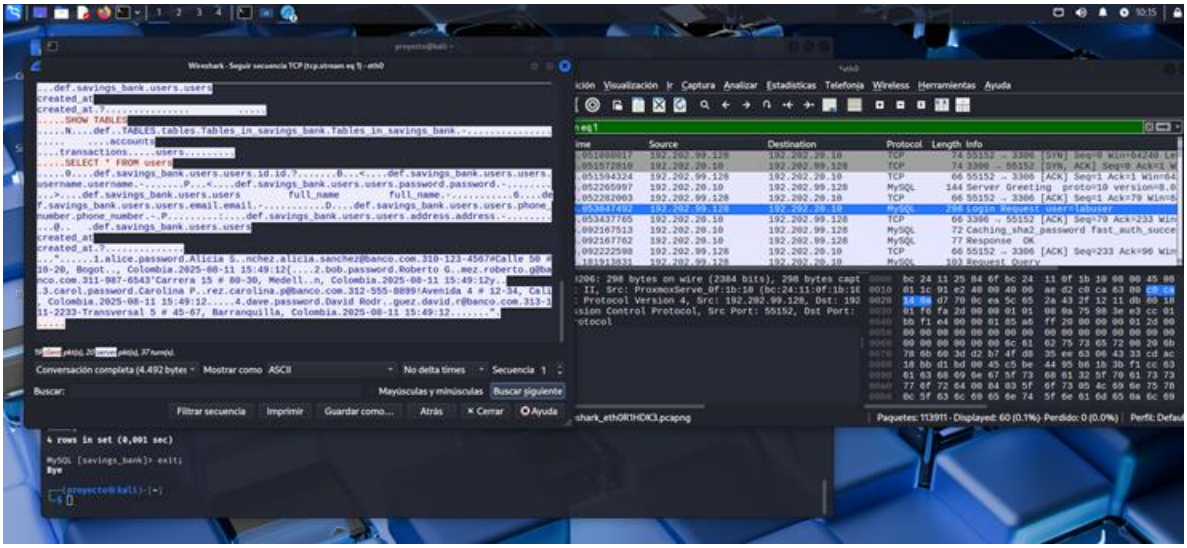


Figura 9: *Datos de usuarios visibles en texto plano*

Todo el tráfico MySQL, incluyendo consultas SQL y resultados, es transmitido sin cifrado TLS/SSL. En las figuras 7, 8 y 9 se evidencia que las consultas SQL son visibles en texto plano; también se puede ver que los resultados de las consultas, incluyendo datos personales, son completamente legibles y No existe indicación de cifrado TLS/SSL en la conexión, y la metadata de las tablas (nombres de columnas, estructuras) también es visible.

3. Implementación de mecanismos de protección

3.1 CIFRADO DE ARCHIVOS CON PYTHON (AES)

El objetivo del mecanismo implementado es proteger la confidencialidad e integridad de archivos transferidos mediante FTP, servicio que transmite información en texto plano y expone datos sensibles ante interceptación de tráfico (en nuestro caso usando analizado usando Wireshark).

Dado que FTP no cifra el canal de transporte (es bastante inseguro), se implementa un esquema de cifrado a nivel de archivo utilizando criptografía simétrica (AES), de modo que el contenido viaje protegido a pesar de la inseguridad del canal.

Este enfoque aplica el principio de defensa en profundidad, lo cual agrega una capa de protección independiente del protocolo de transporte.

3,1,1 DISEÑO

Para este diseño consideramos el siguiente escenario de riesgo:

Un atacante puede:

- Interceptar tráfico en la red.
- Capturar el contenido transmitido por FTP.
- Modificar el archivo durante la transmisión.

Aunque el atacante no tiene acceso a la contraseña utilizada para derivar la clave de cifrado ni a el entorno local del usuario donde se realiza el cifrado/descifrado (código que realizamos en Python).

Los objetivos de seguridad del mecanismo son los siguientes:

- Confidencialidad: impedir que terceros lean el contenido del archivo.
- Integridad: detectar cualquier alteración del archivo cifrado.
- Resistencia a fuerza bruta (básica): dificultar ataques sobre la contraseña

Se utiliza AES-256, con clave de 256 bits, para proporcionar mayor nivel de seguridad, y en modo GCM para proporcionar un cifrado autenticado (AEAD), garantizar la confidencialidad, integridad y detectar modificaciones del texto cifrado; lo cual evitaría ataques donde el atacante modifique el archivo cifrado sin ser detectado.

Utilizamos:

cipher.encrypt_and_digest()

cipher.decrypt_and_verify()

Para asegurar si el archivo fue alterado y si la contraseña es incorrecta (en ambos casos la verificación falla).

Utilizamos también un mecanismo de derivación de clave PBKDF2 con HMAC-SHA256 para no usar directamente esta clave, los parámetros utilizados son los siguientes:

- Hash: SHA-256
- Iteraciones: 200,000
- Salt: 16 bytes aleatorios
- Longitud de clave derivada: 32 bytes (AES-256)

Para evitar ataques de diccionario triviales, prevenir ataques con rainbow tables (usando la sal) e incrementar el costo computacional de fuerza bruta mediante el uso de más iteraciones.

La sal la generamos con:

secrets.token_bytes(16)

Y se almacena junto al archivo cifrado.

La generación de valores aleatorios seguros se realizó usando el módulo “secrets”, específicamente secrets.token_bytes(n) (donde n es 16), es menos predecible que random, y aunque pueden seguir habiendo rastros de patrones en este módulo, para el alcance del laboratorio lo consideramos adecuado.

Estructura del archivo cifrado:

Campo	Tamaño	Descripción
MAGIC	4 bytes	Identificador del formato (SFA1)
Salt	16 bytes	Usado para derivar la clave
Nonce	12 bytes	Valor único para AES-GCM
Tag	16 bytes	Autenticación de Integridad
Ciphertext	Variable	Datos cifrados

Se almacenan de la siguiente forma:

MAGIC-SALT-NONCE-TAG-CIPHERTEXT

Mediante esta estructura podemos validar que el archivo pertenece al sistema, regenerar la clave correctamente durante el descifrado y detectar modificaciones maliciosas.

Flujo Operativo del Sistema:

1) Cifrado y Subida:

- El usuario ingresa contraseña de cifrado.
- Se genera salt aleatorio.
- Se deriva clave con PBKDF2.
- Se genera nonce aleatorio.
- Se cifra el archivo con AES-GCM.
- Se genera tag de autenticación.
- Se construye el archivo cifrado.
- Se transfiere vía FTP.
- Se elimina copia temporal local cifrada.

2) Descarga y Descifrado:

- Se descarga archivo cifrado desde FTP.
- Se valida encabezado MAGIC.
- Se extraen salt, nonce y tag.
- Se solicita contraseña.
- Se deriva la clave nuevamente.
- Se ejecuta decrypt_and_verify.
- Si la verificación es exitosa se genera archivo original.
- Si falla se detecta manipulación o contraseña incorrecta.

3.1.1 EVIDENCIA

```
┌──(proyecto㉿kali)-[~]
└─$ source lab2env/bin/activate

┌──(lab2env)-(proyecto㉿kali)-[~]
└─$ python Descargas/CryptoPythonLab2.py encrypt -i prueba.txt -o salida.txt
Contraseña para cifrado:
[OK] Archivo cifrado guardado en: salida.txt

┌──(lab2env)-(proyecto㉿kali)-[~]
└─$ cat salida.txt
SFA1◆\'◆◆z椹◆(◆◆◆A+K◆ O◆◆G'◆◆'I◆;◆◆&◆◆\]1◆_◆U°◆D◆◆◆QI◆◆

┌──(lab2env)-(proyecto㉿kali)-[~]
└─$
```

```
┌──(lab2env)-(proyecto㉿kali)-[~]
└─$ python Descargas/CryptoPythonLab2.py decrypt -i salida.txt -o final.txt
Contraseña para cifrado:
[OK] Archivo descifrado guardado en: final.txt

┌──(lab2env)-(proyecto㉿kali)-[~]
└─$ cat final.txt
esto es secreto
```

3.1.2 ANÁLISIS

3.2 PROTECCIÓN DEL SERVICIO FTP

Como se evidenció en la sección anterior, el protocolo FTP transmite credenciales y datos en texto plano, lo que representa un riesgo significativo de interceptación en la red. Con el fin de mitigar esta vulnerabilidad, se optó por utilizar el protocolo SFTP, el cual opera sobre SSH y proporciona cifrado del canal de comunicación.

3.2.1 IMPLEMENTACIÓN DE SFTP

Se verificó que el puerto 22 (SSH) se encontraba habilitado en el servidor y se estableció una conexión mediante "*sftp ftpuser@192.202.10.20*" Posteriormente, se realizó la transferencia de un archivo de prueba para validar el funcionamiento del servicio seguro.

```
┌──(proyecto㉿kali)-[~]
└─$ sftp ftpuser@192.202.10.20
The authenticity of host '192.202.10.20 (192.202.10.20)' can't be established.
ED25519 key fingerprint is SHA256:JinVxR/a+bTCoPYzdFkGpuL15jr4xO6WfSxaI3KqKO4.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.202.10.20' (ED25519) to the list of known hosts.
ftpuser@192.202.10.20's password:
Connected to 192.202.10.20.
sftp> put prueba.txt
Uploading prueba.txt to /home/ftpuser/prueba.txt
prueba.txt                                                    100%   16    19.3KB/s   00:00
sftp> ls
datos_banco.txt  ftp              prueba.txt
sftp> exit
```

3.2.2 EVIDENCIA DE CIFRADO

Durante la transferencia mediante SFTP se capturó el tráfico de red y se analizó con Wireshark. A diferencia del protocolo FTP tradicional, no fue posible visualizar credenciales ni comandos en texto plano.

En la captura se observa:

- Establecimiento de conexión SSH.
- Intercambio criptográfico.
- Tráfico clasificado como datos cifrados.

1	0.000000	192.202.99.128	10.10.8.15	SSH	224	Server: Encrypted packet (len=164)
2	0.004301	192.202.99.128	10.10.8.15	SSH	192	Server: Encrypted packet (len=132)
3	0.005174	10.10.8.15	192.202.99.128	TCP	60	63691 → 22 [ACK] Seq=1 Ack=165 Win=253 Len=0
4	0.006319	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
5	0.006426	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
6	0.010747	10.10.8.15	192.202.99.128	TCP	60	63691 → 22 [ACK] Seq=37 Ack=333 Win=253 Len=0
7	0.030157	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
8	0.030274	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
9	0.061359	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
10	0.061437	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
11	0.092269	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
12	0.092369	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
13	0.123113	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
14	0.123217	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
15	0.155578	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
16	0.155679	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
17	0.185640	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
18	0.185790	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
19	0.217158	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
20	0.217250	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
21	0.247071	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
22	0.247197	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
23	0.279919	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)
24	0.280047	192.202.99.128	10.10.8.15	SSH	96	Server: Encrypted packet (len=36)
25	0.302635	10.10.8.15	192.202.99.128	SSH	96	Client: Encrypted packet (len=36)

3.2.3 COMPARACIÓN

La diferencia entre ambos enfoques es significativa:

Característica	FTP	SFTP
Credenciales visibles	SI	NO
Datos en texto plano	SI	NO
Cifrado del canal	NO	SI

El uso de SFTP elimina la exposición de credenciales y contenido transferido, garantizando la confidencialidad de la comunicación.

3.3 PROTECCIÓN DE LA BASE DE DATOS

Como se evidenció en la sección de análisis inicial, la base de datos MySQL permite establecer conexiones sin cifrado cuando se utiliza el parámetro --ssl=0, lo que expone consultas y metadatos en texto plano. Con el fin de mitigar este riesgo, se estableció una conexión utilizando TLS.

3.3.1 ESTABLECIMIENTO DE CONEXIÓN CIFRADA

El servidor de base de datos ya contaba con TLS habilitado mediante un certificado autofirmado. Debido a que el certificado no está firmado por una Autoridad Certificadora (CA) confiable, fue necesario deshabilitar la verificación estricta del certificado desde el cliente. Una vez establecida la conexión, se verificó el uso de cifrado y el resultado indicó el uso de cypher.

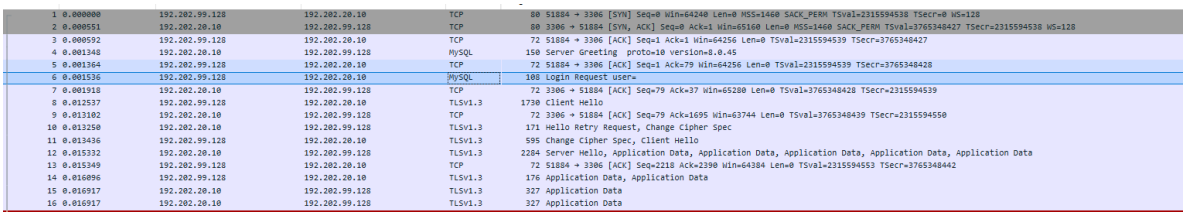
```
┌──(proyecto㉿kali)-[~]
└─$ mysql -h 192.202.20.10 -u root --ssl --ssl-verify-server-cert=0
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 167085
Server version: 8.0.45 MySQL Community Server - GPL

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Support MariaDB developers by giving a star at https://github.com/MariaDB/server
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+-------------------------+
| Variable_name | Value                   |
+---------------+-------------------------+
| Ssl_cipher    | TLS_AES_256_GCM_SHA384 |
+---------------+-------------------------+
1 row in set (0,355 sec)

MySQL [(none)]> show DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| savings_bank       |
| sys                |
+--------------------+
5 rows in set (0,007 sec)
```

3.3.2 EVIDENCIA EN ANÁLISIS DE TRÁFICO

Se realizó una captura de red durante la conexión segura (mysql_tls.pcap) y se analizó con Wireshark. A diferencia de la captura sin TLS, donde era posible visualizar las consultas SQL en texto plano, en esta ocasión el tráfico se clasifica como:

- TLSv1.3 Client Hello
- Server Hello
- Application Data

No es posible observar las consultas SQL ni credenciales en formato legible, ya que la comunicación se encuentra cifrada.



3.3.3 COMPARACIÓN

A continuación, se presenta una comparación entre la conexión establecida sin cifrado y la conexión protegida mediante TLS, con base en el análisis realizado en Wireshark.

Característica	Conexión sin TLS (--ssl=0)	Conexión con TLS (--ssl)
Protocolo detectado en Wireshark	MySQL	TLSv1.3
Handshake criptográfico	No	Sí (Client Hello / Server Hello)
Consultas SQL visibles	Sí (Statement: select ...)	No
Respuestas del servidor visibles	Sí	No
Credenciales expuestas	Usuario visible	No visibles
Tipo de tráfico posterior	Texto plano	Application Data cifrada
Confidencialidad	No garantizada	Garantizada
Cipher utilizado	No aplica	TLS_AES_256_GCM_SHA384

La diferencia es significativa: mientras que en la conexión sin TLS es posible inspeccionar consultas y respuestas en texto plano, en la conexión protegida el tráfico se encapsula bajo TLS 1.3, impidiendo la inspección del contenido por parte de terceros.

4. Análisis Comparativo General

Durante el laboratorio se analizaron tres escenarios de seguridad:

- Servicio FTP
- Base de datos MySQL
- Protección a nivel de aplicación mediante script Python

4.1 COMPARACION POR NIVEL DE SEGURIDAD

Mecanismo	Nivel donde actúa	Qué protege	Depende del canal	Protección si interceptan tráfico
SFTP (SSH)	Transporte	Credenciales y archivos en tránsito	Sí	Alto
MySQL + TLS	Transporte	Consultas y credenciales DB	Sí	Alto
AES en Python	Aplicación	Contenido del archivo	No	Muy alto

4.2 DIFERENCIAS CLAVE ENTRE ENFOQUES

4.2.1 SFTP Y TLS

Ambos protegen el canal de comunicación, si alguien intercepta el tráfico:

- Verá datos cifrados
- No podrá leer consultas ni archivos
- Dependen de que el canal esté correctamente configurado

Pero si el archivo se guarda sin cifrar en el servidor, ya no está protegido.

4.2.2 CIFRADO AES EN PYTHON

Este mecanismo:

- No depende del canal
- Protege el contenido del archivo incluso en reposo
- Añade defensa en profundidad

Incluso si el archivo es robado del servidor, seguirá cifrado

5. Recomendaciones

Con base en los resultados obtenidos durante el laboratorio, se proponen las siguientes recomendaciones para fortalecer la seguridad de los servicios evaluados.

5.1 USO OBLIGATORIO DE PROTOCOLOS SEGUROS

Se recomienda deshabilitar completamente protocolos inseguros como FTP sin cifrado y conexiones MySQL sin TLS. Durante el laboratorio se evidenció que estos modos de operación permiten visualizar credenciales y consultas en texto plano mediante herramientas como Wireshark. En un entorno real, estos servicios deberían configurarse para aceptar únicamente conexiones seguras, utilizando SFTP para transferencia de archivos y TLS obligatorio para la base de datos.

5.2 VALIDACIÓN ADECUADA DE CERTIFICADOS

Aunque en el entorno académico se utilizó un certificado autofirmado, lo cual obligó a desactivar la verificación del certificado para permitir la conexión, esta práctica no es recomendable en ambientes productivos. Es importante utilizar certificados emitidos por una autoridad certificadora confiable y mantener activa la validación del certificado del servidor. De lo contrario, se incrementa el riesgo de ataques de intermediario (Man-in-the-Middle).

5.3 IMPLEMENTACIÓN DE DEFENSA EN PROFUNDIDAD

El cifrado implementado mediante AES en el script Python demuestra que la protección no debe limitarse únicamente al canal de comunicación. Mientras que TLS y SSH protegen la información en tránsito, el cifrado a nivel de aplicación protege el contenido incluso si el archivo es almacenado o interceptado fuera del canal seguro. Se recomienda combinar ambos enfoques para fortalecer la confidencialidad de la información.

5.4 MANTENIMIENTO Y MONITOREO CONTINUO

Finalmente, es importante mantener actualizados los servicios utilizados, incluyendo OpenSSH, MySQL y las bibliotecas criptográficas. Asimismo, se recomienda implementar mecanismos de monitoreo y auditoría que permitan verificar que las conexiones se establecen de forma segura y detectar configuraciones incorrectas o intentos de acceso no autorizado.

6. Conclusiones

El desarrollo del laboratorio permitió evidenciar de manera práctica las diferencias entre comunicaciones inseguras y protegidas en un entorno realista. A través del análisis de tráfico de red se comprobó que tanto el protocolo FTP tradicional como las conexiones MySQL sin TLS transmiten información sensible en texto plano, lo que facilita su interceptación mediante técnicas de sniffing. Esta situación representa un riesgo significativo para la confidencialidad de credenciales, consultas y datos transferidos.

La implementación de SFTP demostró que el uso de SSH como capa de transporte elimina la exposición de credenciales y contenido de archivos, ya que toda la comunicación se encapsula dentro de un canal cifrado. De forma similar, el uso de TLS en la base de datos permitió proteger las consultas SQL y las respuestas del servidor, impidiendo su inspección en herramientas como Wireshark. La verificación del cipher negociado confirmó el uso de criptografía robusta, específicamente TLS 1.3 con AES-256-GCM.

Adicionalmente, el desarrollo del script en Python para cifrado mediante AES en modo GCM aportó una capa adicional de protección a nivel de aplicación. Este mecanismo garantiza que la información permanezca protegida incluso si el archivo es almacenado o interceptado fuera de un canal seguro, reforzando el principio de defensa en profundidad.

En conjunto, el laboratorio demuestra que la seguridad efectiva no depende de una única medida, sino de la combinación de múltiples mecanismos aplicados en distintas capas del sistema. La correcta configuración de protocolos seguros, el uso de cifrado fuerte y la aplicación de buenas prácticas de seguridad son elementos fundamentales para proteger sistemas que manejan información sensible.