

**Universidad Nacional del Sur**

Proyecto Final de Carrera Ingenier´ıa en Computacion´

*Seguridad en redes LAN: utilizando Docker para mejorar la navegaci´on web.*

Salvador Catalfamo

Bah´ıa Blanca – Argentina

2021



**Universidad Nacional del Sur**

Proyecto Final de Carrera Ingenier´ıa en Computacion´

*Seguridad en redes LAN: utilizando Docker para mejorar la navegaci´on web.*

Salvador Catalfamo

Bah´ıa Blanca – Argentina

2021

# **Resumen**

A lo largo de la carrera, y, particularmente, en una de mis materias preferidas, Seguridad en Sistemas, nos han explicado la importancia la proteccio´n de los datos, como as´ı tambi´en la de los canales de comunicaci´on. Mientras se desarrolla una red segura, se debe considerar la confidencialidad, integridad y disponibilidad (CIA).

Tambi´en, en nuestra corta experiencia hemos visto como las organizaciones abordan los temas de seguridad en sus sistemas inform´aticos. Mayormente, se concentran en los equipos que esta´n expuestos a la red pu´blica, dejando de lado los que se encuentran aislados de la misma. Err´oneamente, muchas veces se piensa que es suficiente, sin embargo, puede traer graves inconvenientes. Es por eso que realizaremos un estudio te´orico/pra´ctico sobre las consecuencias de la navegaci´on en redes internas sin ningu´n tipo de cifrado de datos ni certificaciones.

Palabras Clave:

Seguridad e Infraestructura

Docker

Linux

Kali

Ma´quinas Virtuales

Entidad Certificante CA

Protocolo SSL

Navegaci´on segura Redes internas

iii

# **´Indice general**

## **Introduccio´n1**

* 1. Objetivos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .1
  2. Metodolog´ıa de trabajo . . . . . . . . . . . . . . . . . . . . . . . . . .2

## **Navegaci´on web en redes internas3**

* 1. Protocolo HTTP. . . . . . . . . . . . . . . . . . . . . . . . . . . . .3
     1. Arquitectura . . . . . . . . . . . . . . . . . . . . . . . . . . . .4
     2. M´etodos mas importantes del del protocolo HTTP. . . . . .6
     3. C´odigos de respuesta . . . . . . . . . . . . . . . . . . . . . . .7
     4. HTTP con Seguridad SSL (HTTPS). . . . . . . . . . . . . .7

## **Debilidades del protocolo HTTP19**

* 1. Riesgos de intermediarios . . . . . . . . . . . . . . . . . . . . . . . . .19
  2. Confidencialidad del mensaje . . . . . . . . . . . . . . . . . . . . . . .20
  3. Integridad de los mensajes . . . . . . . . . . . . . . . . . . . . . . . .20
  4. Tipos de ataques . . . . . . . . . . . . . . . . . . . . . . . . . . . . .21
     1. Ataques Pasivos . . . . . . . . . . . . . . . . . . . . . . . . . .21
     2. Ataques Activos . . . . . . . . . . . . . . . . . . . . . . . . . .21
  5. Caso de estudio: Interceptando la red para obtener credenciales. . .22
     1. Herramienta Utilizadas . . . . . . . . . . . . . . . . . . . . . .22
     2. Realizaci´on del ataque . . . . . . . . . . . . . . . . . . . . . .23
     3. Preparando Ettercap para el ataque ARP Poisoning . . . . . .25
     4. Nuestro Ettercap ya esta´ listo. Ya podemos empezar con el

ataque ARP Poisoning . . . . . . . . . . . . . . . . . . . . . . 26

v

## **Soluciones estudiadas29**

* 1. Ma´quinas virtuales . . . . . . . . . . . . . . . . . . . . . . . . . . . .29
  2. Virtualizacio´n basada en Contenedores . . . . . . . . . . . . . . . . .30
  3. Docker: nuestro aliado en las pruebas . . . . . . . . . . . . . . . . . .32
     1. Im´agenes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .33
     2. Registros . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .33
     3. Contenedores . . . . . . . . . . . . . . . . . . . . . . . . . . .33
  4. Propuestas: Introduccio´n . . . . . . . . . . . . . . . . . . . . . . . . .34
  5. Propuesta 1: Self-signed Certificates . . . . . . . . . . . . . . . . . . .34
  6. Propuesta 2: Internal CA . . . . . . . . . . . . . . . . . . . . . . . . .35
     1. Caso de estudio: Creando nuestra Entidad Certificante privada 36
  7. Propuesta 3: Certificaci´on con Let’s Encrypt . . . . . . . . . . . . . .42
     1. Pasos a seguir . . . . . . . . . . . . . . . . . . . . . . . . . . .42
  8. Caso de estudio: Buscando credenciales en tra´fico seguro. . . . . . .45

1. **Conclusiones47**
2. **Trabajos a Futuro49**
3. **Glosario 51**
   1. Terminolog´ıa . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
   2. Simbolog´ıa . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51

# **Cap´ıtulo 1**

# **Introduccio´n**

A lo largo de la carrera, y, particularmente, en una de mis materias preferidas, Seguridad en Sistemas, nos han explicado la importancia la proteccio´n de los datos, como as´ı tambi´en la de los canales de comunicaci´on. Mientras se desarrolla una red segura, se debe considerar la confidencialidad, integridad y disponibilidad (CIA).

Tambi´en, en nuestra corta experiencia hemos visto como las organizaciones abordan los temas de seguridad en sus sistemas inform´aticos. Mayormente, se concentran en los equipos que esta´n expuestos a la red pu´blica, dejando de lado los que se encuentran aislados de la misma. Err´oneamente, muchas veces se piensa que es suficiente, sin embargo, puede traer graves inconvenientes. Es por eso que realizaremos un estudio te´orico/pra´ctico sobre las consecuencias de la navegaci´on en redes internas sin ningu´n tipo de cifrado de datos ni certificaciones.

## **1.1.Objetivos**

Aplicar los conocimientos en seguridad en redes para desplegar aplicaciones de red.

Aprender el uso de nuevas herramientas de administraci´on, automatizaci´on y seguridad en sistemas.

Montar un escenario virtual que sirva de pruebas frente a la gesti´on de certificados, administracio´n de la infraestructura y a la deteccio´n de debilidades dentro una red privada.

Concientizar la implementacio´n de medidas de seguridad en redes internas

1

*CAP´ITULO 1. INTRODUCCION´*

Implementar una mejora en la navegacio´n en una red interna utilizando Docker

## **1.2.Metodolog´ıa de trabajo**

El trabajo se dividir´a en tres etapas de trabajo:

* Primera: Se investigar´an y plantear´an escenarios donde la navegacio´n insegura pueda traer problemas asociados dentro de la organizacio´n.
* Segunda: Se plantear´an escenarios de trabajo buscando adquirir experiencia en la utilizaci´on de la herramienta Docker, para poder obtener para´metros que nos permitan realizar comparaciones con tecnolog´ıas similares y sus a´reas de aplicacio´n.
* Tercera: Se investigar´an e implementar´an posibles soluciones para afrontar los aspectos planteados en la primera etapa

# **Cap´ıtulo 2**

# **Navegacio´n web en redes internas**

La conexi´on en red es la tecnolog´ıa clave para una amplia variedad de aplicaciones dentro de una organizaci´on, como correo electro´nico, transferencia de archivos, navegaci´on web, consultas de pa´ginas, etc. Sin embargo, existe una falta significativa de m´etodos de seguridad para estas aplicaciones.

## **2.1.Protocolo HTTP**

El Protocolo de transferencia de hipertexto (HTTP) es un protocolo de solicitud/respuesta *stateless* (sin estado) que utiliza sema´ntica extensible y cargas u´tiles de mensajes autodescriptivos para una interacci´on flexible con sistemas basados en red.

HTTP es un protocolo gen´erico para sistemas de informaci´on. Esta´ disen˜ado para ocultar los detalles de co´mo se implementa un servicio mostrando una interfaz a los clientes que es independiente de los tipos de recursos proporcionados. Del mismo modo, los servidores no necesitan conocer el propo´sito de cada cliente: una solicitud HTTP puede aislada en vez de estar asociada con un tipo espec´ıfico de cliente o una secuencia predeterminada de pasos de la aplicacio´n.

El resultado es un protocolo que se puede utilizar de forma eficaz en muchos contextos diferentes y cuyas implementaciones pueden evolucionar a lo largo del tiempo. Una consecuencia de esta flexibilidad es que el protocolo no se puede definir en t´erminos de lo que ocurre detr´as de la interfaz: estamos limitados a definir la sintaxis de la comunicacio´n, los mensajes y el comportamiento esperado de los

3

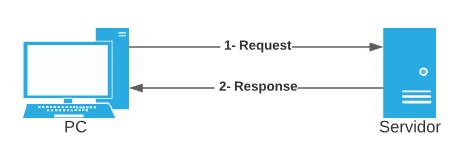


Figura 2.1: Comunicaci´on ba´sica en HTTP

destinatarios. Si la comunicaci´on se considera de forma aislada, las acciones exitosas deben reflejarse correspondientemente. Sin embargo, dado que varios clientes pueden actuar en paralelo y quiz´as con propo´sitos cruzados, no podemos exigir que tales cambios sean observables m´as alla´ del alcance de una u´nica respuesta.

### 2.1.1. Arquitectura

HTTP fue creado para la arquitectura World Wide Web (WWW) y ha evolucionado con el tiempo para soportar las necesidades de escalabilidad de un sistema mundial. Gran parte de esa arquitectura se refleja en la terminolog´ıa y definiciones de sintaxis utilizadas para definir HTTP.

#### Mensajes Cliente/Servidor

HTTP es un protocolo de solicitud/respuesta que opera intercambiando mensajes a trav´es de una “conexi´on” en la capa de sesio´n o transporte. Un “cliente” HTTP es un programa que establece una conexio´n a un servidor con el prop´osito de enviar una o m´as solicitudes HTTP. Un “servidor” HTTP es un programa que acepta conexiones para atender solicitudes HTTP mediante el env´ıo de respuestas HTTP. La mayor´ıa de las comunicaciones HTTP consisten en una solicitud (GET) de algu´n recurso identificado por un URI. En el caso ma´s simple, esto podr´ıa lograrse mediante una u´nica conexi´on entre un usuario y el servidor.

Un cliente env´ıa una solicitud HTTP a un servidor en forma de mensaje de solicitud, comenzando con una l´ınea que incluye un m´etodo, URI y versio´n del protocolo,

seguida de campos de encabezado que contienen modificadores de solicitud, informacio´n del cliente y metadatos de representacio´n, una l´ınea vac´ıa para indicar el final de la seccio´n del encabezado, y finalmente un cuerpo del mensaje que contiene el cuerpo de la carga u´til (si lo hay).

Un servidor responde a la solicitud de un cliente enviando uno o ma´s mensajes de respuesta HTTP, cada uno de los cuales comienza con una l´ınea de estado que incluye la versio´n del protocolo, un co´digo de estado (´exito o error) y una descripci´on en forma de texto asociada al mismo, posiblemente seguida de campos de encabezado con informacio´n del servidor y metadatos de recursos, una l´ınea vac´ıa para indicar el final de la seccio´n del encabezado y finalmente, un cuerpo del mensaje la carga u´til del mismo

#### Ejemplo

El siguiente ejemplo ilustra un intercambio de mensajes t´ıpico para una solicitud GET a la direccio´n “http://www.example.com/hello.txt”:

Client request:

GET /hello.txt HTTP/1.1

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

Server response:

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.

### 2.1.2. M´etodos mas importantes del del protocolo HTTP

El protocolo HTTP contiene varios m´etodos, como por ejemplo PUT, HEAD, DELETE, etc. Sin embargo, para nuestro trabajo explicaremos los dos mas utilizados GET y POST, lo que nos permitira´ tener una base a la hora de presentar el caso de estudio de la seccio´n 3.5.

#### GET

El m´etodo GET solicita al servidor la transferencia de un recurso. GET es el mecanismo principal de recuperacio´n de informaci´on y el foco de casi todas las optimizaciones de rendimiento. Por lo tanto, cuando las personas hablan de recuperar informacio´n identificable a trav´es de HTTP, generalmente se refieren a realizar una solicitud GET.

Se puede pensar que a la hora de solicitar un recuro, este sea un archivo dentro de un directorio, y la respuesta sea el mismo archivo. Sin embargo, no existen tales limitaciones en la pra´ctica. De hecho, se puede configurar un servidor para ejecutar los archivos de la solicitud y enviar la salida en lugar de transferir los archivos directamente. Independientemente de la solicitud, el servidor solo necesita saber co´mo tratar a cada uno de sus recursos.

#### POST

El m´etodo POST solicita que un recurso del servidor sea procesado con los datos que el cliente le env´ıa. Por ejemplo, POST se utiliza para las siguientes funciones (entre otras):

Proporcionar un bloque de datos, como los campos ingresados en un formulario

HTML, a un proceso de manejo de datos.

Publicar un mensaje grupo de noticias, lista de correo, blog o grupo similar de art´ıculos.

Crear un nuevo recurso que au´n no ha sido identificado por el servidor.

Agregar datos a las representaciones existentes de un recurso.

### 2.1.3. Co´digos de respuesta

El co´digo de estado es un nu´mero entero de tres d´ıgitos que da el resultado del intento de comprender y satisfacer la solicitud. Los c´odigos de estado HTTP son extensibles. No se requiere que los clientes HTTP comprendan el significado de todos los c´odigos de estado registrados, aunque se espera una m´ınima comprensi´on.

Por ejemplo, si un cliente recibe un co´digo de estado no reconocido de 471, el cliente puede asumir que hubo algo mal con su solicitud y tratar la respuesta como si hubiera recibido un c´odigo de estado 400 (Solicitud incorrecta). El mensaje de respuesta generalmente contendra´ una representaci´on que explica el estado.

El primer d´ıgito del c´odigo de estado define la clase de respuesta. Los dos u´ltimos d´ıgitos no tienen ninguna funci´on de categorizacio´n. Hay cinco valores para el primer d´ıgito:

1xx (Informativo): Se recibi´o la solicitud, se continua procesando.

2xx (Satisfactoria): La solicitud se recibio´, comprendio´ y acept´o correctamente.

3xx (Redireccionamiento): Se deben realizar m´as acciones para completar la solicitud.

4xx (Error del cliente): La solicitud contiene una sintaxis incorrecta o no se puede cumplir.

5xx (Error del servidor): El servidor no cumplio´ con una solicitud aparentemente va´lida.

### 2.1.4. HTTP con Seguridad SSL (HTTPS)

Con una comprensi´on m´ınima de los conceptos la criptograf´ıa, podemos observar co´mo funciona el protocolo *Secure Sockets Layer* (SSL). Aunque SSL no es un protocolo extremadamente complicado, ofrece varias opciones y variaciones.

El protocolo SSL consiste en un conjunto de mensajes y reglas sobre cua´ndo enviar (y no enviar) cada mensaje. En esta seccio´n, mostraremos cu´ales son esos mensajes, la informacio´n general que contienen y c´omo los sistemas usan los diferentes mensajes en una sesi´on de comunicaciones.

#### Roles SSL

El protocolo *Secure Sockets Layer* define dos roles diferentes para las partes que se comunican. Por un lado tenemos un cliente, y por el otro un servidor. La distincio´n es muy importante, porque SSL requiere que los dos sistemas se comporten de manera muy diferente.

El cliente es el sistema que inicia las comunicaciones seguras; el servidor responde a la solicitud del cliente. En el uso ma´s comu´n de SSL, la navegaci´on web segura, el navegador web es el cliente SSL y el sitio web es el servidor SSL. Para SSL en s´ı, las distinciones m´as importantes entre clientes y servidores son sus acciones durante la negociacio´n de los par´ametros de seguridad.

Dado que el cliente inicia una comunicaci´on, tiene la responsabilidad de proponer un conjunto de opciones SSL para usar en el intercambio. El servidor selecciona entre las opciones propuestas por el cliente y decide qu´e utilizara´n realmente los dos sistemas. Aunque la decisio´n final recae en el servidor, el servidor solo puede elegir entre las opciones que el cliente propuso originalmente.

#### Mensajes SSL

Cuando los clientes y servidores SSL se comunican, lo hacen intercambiando mensajes SSL. Esta secci´on mostrara´ c´omo los sistemas utilizan estos mensajes en sus comunicaciones. La funcio´n ma´s b´asica (y uno de los propo´sitos mas importantes) que realiza un cliente y un servidor SSL es establecer la seguridad a trav´es de un canal para comunicaciones cifradas. Los primeros tres mensajes SYN, SYN ACK y SYN correspondientes al protocolo TCP, Luego, inician los mensajes pertenecientes a la comunicaci´on SSL.

**ClientHello** El mensaje *ClientHello* inicia la comunicacio´n SSL entre las dos partes. El cliente usa este mensaje para pedirle al servidor que comience a negociar los servicios de seguridad usando SSL.

El mensaje este compuesto por ciertos campos:

Versi´on: refiere a la versio´n ma´s alta de SSL que el cliente puede admitir.

RandomNumber: proporciona la semilla para c´alculos criptogra´ficos cr´ıticos.

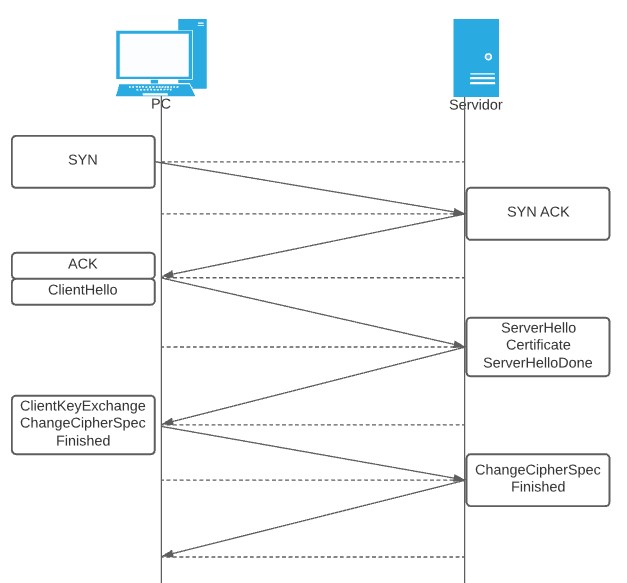


Figura 2.2: Mensajes SSL SessionID: es opcional, y muchas veces no es utilizado.

CipherSuites: permite a un cliente enumerar los diversos servicios criptogra´ficos que el cliente puede admitir

CompressionMethods: es utilizado por el cliente para enumerar todos los diversos m´etodos de compresio´n de datos que puede admitir. Los m´etodos de compresio´n son una parte importante de SSL porque el cifrado tiene una secuencia significativa en la efectividad de cualquier t´ecnica de compresio´n de datos.

**ServerHello** Este mensaje complementa el campo *CipherSuite* del *ServerHello*. Si bien el campo *CipherSuite* indica los algoritmos criptogra´ficos y los taman˜os de clave, este mensaje contiene la informacio´n de la clave pu´blica en s´ı. Tenga en cuenta que el mensaje *ServerKeyExchange* se transmite sin cifrado, por lo que solo se puede incluir de forma segura informacio´n de clave pu´blica. El cliente utilizara´ la clave pu´blica del servidor para cifrar una clave de sesio´n, que las partes utilizara´n para cifrar los datos de la aplicacio´n para la sesi´on.

**ServerKeyExchange** Este mensaje complementa el campo *CipherSuite* del *ServerHello*. Si bien el campo *CipherSuite* indica los algoritmos criptogra´ficos y los taman˜os de clave, este mensaje contiene la informacio´n de la clave pu´blica en s´ı. Tenga en cuenta que el mensaje *ServerKeyExchange* se transmite sin cifrado, por lo que solo se puede incluir de forma segura informaci´on de clave pu´blica. El cliente utilizara´ la clave pu´blica del servidor para cifrar una clave de sesi´on, que las partes utilizara´n para cifrar los datos de la aplicaci´on para la sesio´n.

**ServerHelloDone** El mensaje *ServerHelloDone* le dice al cliente que el servidor ha terminado con sus mensajes iniciales de negociacio´n. El mensaje en s´ı no contiene otra informaci´on, pero es importante para el cliente, porque una vez que el cliente recibe un *ServerHelloDone*, puede pasar a la siguiente fase para establecer las comunicaciones seguras.

**ClientKeyExchange** Cuando el servidor ha terminado su parte de la negociaci´on SSL inicial, el cliente responde con un mensaje *ClientKeyExchange*. As´ı como *ServerKeyExchange* proporciona la informacio´n clave para el servidor, *ClientKeyExchange* le dice al servidor la informacio´n clave del cliente. En este caso, sin embargo, la informacio´n clave es para el algoritmo de cifrado sim´etrico que ambas partes usara´n para la sesi´on. Adema´s, la informacio´n del mensaje del cliente se cifra mediante la clave pu´blica del servidor. Este cifrado protege la informaci´on de la clave a medida que atraviesa la red y permite al cliente verificar que el servidor realmente posee la clave privada correspondiente a su clave pu´blica. De lo contrario, el servidor no podr´a descifrar este mensaje. Esta operacio´n es una protecci´on importante contra un atacante que intercepta mensajes de un servidor leg´ıtimo y finge ser ese servidor reenviando los mensajes a un cliente desprevenido. Dado que un servidor falso no conocera´ la clave privada del servidor real, no podr´a descifrar el mensaje *ClientKeyExchange*. Sin la informacio´n en ese mensaje, la comunicaci´on entre las dos partes no puede tener ´exito.

**ChangeCipherSpec** Una vez que el cliente env´ıa informaci´on clave en un mensaje *ClientKeyExchange*, se completa la negociacio´n SSL preliminar. En ese momento, las partes est´an listas para comenzar a utilizar los servicios de seguridad que han negociado.

El protocolo SSL define un mensaje especial, *ChangeCipherSpec*, para indicar expl´ıcitamente que ahora se deben invocar los servicios de seguridad. Un conjunto de claves protegera´ los datos que el cliente env´ıa al servidor, y un conjunto diferente de claves protegera´ los datos que el servidor env´ıa al cliente. Para cualquier sistema dado, ya sea un cliente o un servidor, SSL define un estado de escritura y un estado de lectura. El estado de escritura define la informacio´n de seguridad de los datos que env´ıa el sistema y el estado de lectura define la informacio´n de seguridad de los datos que recibe el sistema.

**Finished** Inmediatamente despu´es de enviar sus mensajes *ChangeCipherSpec*, cada sistema tambi´en env´ıa un mensaje *Finished*. Los mensajes *Finished* permiten que ambos sistemas verifiquen que la negociacio´n se ha realizado correctamente y que la seguridad no se ha visto comprometida.

Cada mensaje *Finished* contiene un hash criptogra´fico de informacio´n importante sobre la negociacio´n reci´en finalizada. Esto protege contra un atacante que logra insertar mensajes ficticios o eliminar mensajes leg´ıtimos de la comunicacio´n. Si un atacante pudiera hacerlo, los ca´lculos de hash del cliente y del servidor no coincidir´ıan y detectar´ıan el compromiso.

**Finalizacio´n de las comunicaciones seguras** Aunque, en la pra´ctica, rara vez se utiliza (principalmente debido a la naturaleza de las sesiones web), SSL tiene un procedimiento definido para finalizar una comunicaci´on segura entre dos partes. En este procedimiento, los dos sistemas env´ıan cada uno una alerta de cierre especial al otro. El cierre expl´ıcito de una sesio´n protege contra un ataque de truncamiento, en el que un atacante puede comprometer la seguridad al terminar prematuramente una comunicacio´n.

#### Autenticar la identidad del servidor

Anteriormente se explico´ co´mo SSL puede establecer comunicaciones cifradas entre dos partes, lo que puede no agregar mucha seguridad a la comunicacio´n. Con el cifrado solo, ninguna de las partes puede estar realmente segura de la identidad de la otra. La razo´n t´ıpica para usar el cifrado en primer lugar es mantener la informacio´n en secreto de algu´n tercero. Pero si ese tercero pudiera hacerse pasar con ´exito como el destinatario previsto de la informacio´n, entonces el cifrado no servir´ıa de nada. Los datos estar´ıan encriptados, pero el atacante tendr´ıa todos los datos necesarios para desencriptarlos. Para evitar este tipo de ataques, SSL incluye mecanismos que permiten a cada parte autenticar la identidad de la otra. Con estos mecanismos, cada parte puede estar segura de que la otra es genuina y no un atacante enmascarado. En esta seccio´n, veremos c´omo SSL permite que un servidor se autentique.

**Certificate** Al autenticar su identidad, el servidor env´ıa un mensaje de certificado en lugar del mensaje *ServerKeyExchange* descrito anteriormente. El mensaje *Certificate* simplemente contiene una cadena de certificados que comienza con el certificado de clave pu´blica del servidor y termina con el certificado ra´ız de la autoridad certificadora.

El cliente tiene la responsabilidad de asegurarse de que puede confiar en el certificado que recibe del servidor. Esa responsabilidad incluye verificar las firmas del certificado, los tiempos de validez y el estado de revocacio´n. Tambi´en significa asegurarse de que la autoridad de certificacio´n sea una en la que el cliente conf´ıe.

Normalmente, los clientes toman esta determinacio´n conociendo la clave pu´blica de las autoridades de certificacio´n confiables de antemano, a trav´es de algunos medios confiables. Microsoft, por ejemplo, carga su software con claves pu´blicas para autoridades de certificaci´on conocidas.

**ClientKeyExchange** El mensaje *ClientKeyExchange* del cliente tambi´en difiere en la autenticaci´on del servidor, aunque la diferencia no es importante. Cuando solo se va a utilizar cifrado, el cliente cifra la informacio´n en en mensaje *ClientKeyExchange* utilizando la clave pu´blica que el servidor proporciona en su mensaje *ServerKeyExchange*. En este caso, por supuesto, el servidor se est´a autenticando y, por lo tanto, ha enviado un mensaje de certificado en lugar de un *ServerKeyExchange*. El cliente, por lo tanto, encripta su informaci´on usando la clave pu´blica contenida en el certificado del servidor. Este paso es importante porque le permite al cliente asegurarse de que la parte con la que se esta´ comunicando realmente posee la clave privada del servidor. Solo un sistema con la clave privada real podra´ descifrar este mensaje y continuar con ´exito la comunicacio´n.

#### Niveles de validaci´on

Hay tres tipos de certificados SSL disponibles en la actualidad: validacio´n por dominio (DV), validacio´n por organizacio´n (OV) y validacio´n extendida (EV). Los niveles de cifrado son los mismos para cada certificado, lo que difiere son los procesos de investigacio´n y verificaci´on necesarios para obtener el certificado.

**Validaci´on de dominio (DV)** Validaci´on de dominio SSL o DV SSL representa el nivel base para los tipos de SSL. Estos son perfectos para sitios web que solo necesitan cifrado y nada ma´s. Los certificados DV SSL suelen ser econ´omicos y se pueden emitir en cuestio´n de minutos. Eso es porque el proceso de validaci´on est´a completamente automatizado. Simplemente es necesario demostrar que es propietario de su dominio y que el certificado DV es suyo.

**Validaci´on de la organizaci´on (OV)** Validaci´on de organizacio´n u OV SSL representa el t´ermino medio para los tipos de certificados SSL. Para obtener OV SSL,

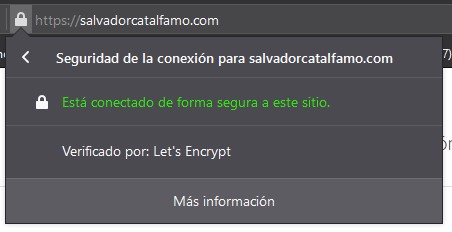


Figura 2.3: Validacio´n de dominio

su empresa u organizaci´on debe someterse a un examen comercial ligero. Esto puede demorar hasta tres d´ıas ha´biles porque alguien tiene que verificar la informacio´n de su empresa. OV SSL muestra los mismos indicadores visuales que DV SSL, pero proporciona una forma de ver la informacio´n comercial verificada en la secci´on de detalles del certificado.

**Extended Validation (EV)** SSL de validaci´on extendida o SSL con EV requiere un exhaustivo examen comercial. Esto puede parecer mucho, pero en realidad no lo es si su empresa tiene registros disponibles pu´blicamente. EV SSL activaba un indicador visual u´nico: el nombre de su organizacio´n verificado que se muestra en el navegador. Esto en la actualidad ya no sucede, por lo que no es posible a simple vista identificarlo.

#### Tipos de certificados

**Dominio Simple** Como sugiere el nombre, un certificado SSL de un solo dominio solo se puede usar en un solo dominio o IP. Este se considera el tipo de certificado SSL predeterminado.

**Multi-Dominio** Este tipo de SSL es un certificado para todos los usos. Permiten cifrar hasta 250 dominios diferentes y subdominios ilimitados. Desafortunadamente,

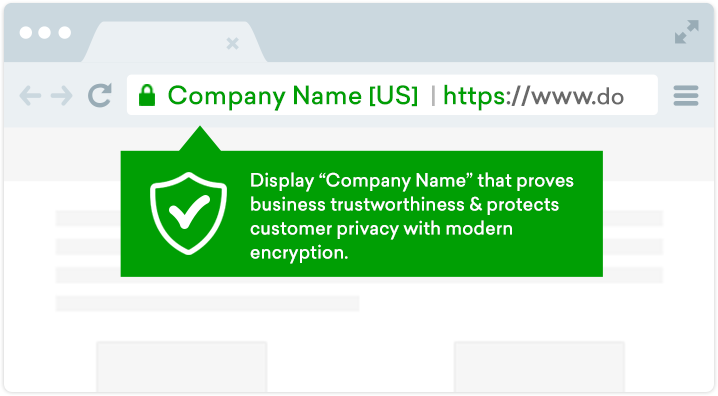


Figura 2.4: Validacio´n extendida (Captura antigua)

no est´a disponible en EV.

**Wildcard** Los *wildcard* esta´n disen˜ados espec´ıficamente para cifrar un dominio y todos los subdominios que lo acompan˜an (tambi´en representado como \*.dominio.com). Desafortunadamente, los *wildcard* solo esta´n disponibles en los niveles DV y OV.

**Multi-Dominio Wildcard** Los *wildcard* multidominio pueden cifrar hasta 250 dominios diferentes y subdominios ilimitados. Desafortunadamente, no esta´ disponible en EV.

#### Validaci´on de propiedad de dominio

Los certificados se utilizan con mayor frecuencia para autenticar nombres de dominio. Por lo tanto, se conf´ıa en las autoridades de certificaci´on (CA) para verificar que un solicitante de un certificado represente leg´ıtimamente el nombre de dominio en el certificado.

Los diferentes tipos de certificados reflejan diferentes tipos de verificaci´on de CA. Los certificados de “Validacio´n de dominio” (DV) son el tipo ma´s comu´n. La u´nica validaci´on que debe realizar la CA en el proceso de emisio´n de un certificado DV es verificar que el solicitante tiene un control efectivo del dominio. La CA no esta´ obligada a intentar verificar la identidad real del solicitante. Esto difiere en los certificados de “Validacio´n de la organizacio´n” y “Validacio´n extendida,” donde el proceso est´a destinado a verificar tambi´en la identidad real del solicitante.

ACME (*Automatic Certificate Management Environment*) permite a un cliente la automatizaci´on de gesti´on de certificados mediante un conjunto de mensajes transmitidos a trav´es de HTTPS. La emisio´n de certificados mediante el protocolo ACME se asemeja al proceso de emisio´n de una CA tradicional, en el que un usuario crea una cuenta, solicita un certificado y demuestra el control de los dominios con el certificado para que la CA emita el certificado solicitado.

ACME utiliza un *framework* de desaf´ıo/respuesta extensible para la validaci´on de dominios. El servidor env´ıa al cliente un conjunto de desaf´ıos, y el cliente responde enviando la respuesta al mismo en una solicitud POST a una URL de desaf´ıo.

Los diferentes desaf´ıos permiten al servidor obtener pruebas de diferentes aspectos del control sobre un dominio. En los desaf´ıos como HTTP y DNS, el cliente demuestra directamente su capacidad para hacer ciertas acciones relacionadas con el dominio. Es de gran utilidad explicar los diferentes tipos de desaf´ıos que se puede ofrecer a un cliente, ya que uno es el mas comu´n, sin embargo, al hablar de redes internas, no lo podremos utilizar, e iremos por la otra opci´on, un tanto menos conocida.

**Desaf´ıo HTTP** Con la validacio´n HTTP, el cliente prueba su control sobre un nombre de dominio al demostrar que puede guardar recursos HTTP en un servidor accesible bajo ese nombre de dominio. El servidor ACME desaf´ıa al cliente solicita´ndole un archivo en una ruta espec´ıfica, con una cadena espec´ıfica como contenido.

Este es el tipo de desaf´ıo m´as comu´n en la actualidad. El servidor le da un token a su cliente ACME y su cliente ACME coloca un archivo en su servidor web en http://SU DOMINIO /.well-known/acme-challenge/TOKEN . Ese archivo contiene el token, m´as una huella digital de la clave de su cuenta.

Una vez que el cliente le dice al servidor que el archivo esta´ listo, el servidor intenta recuperarlo. Al recibir una respuesta, el servidor construye y almacena la autorizacio´n de la clave a partir del valor del “token” de desaf´ıo y la clave de la cuenta del cliente actual.

Dado un par de desaf´ıo/respuesta, el servidor verifica el control del dominio por parte del cliente verificando que el recurso se aprovisiono´ como se esperaba.

Ventajas:

Es f´acil de automatizar sin conocimientos adicionales sobre la configuracio´n de un dominio.

Funciona con servidores web est´andar.

Desventajas:

No funciona si su ISP bloquea el puerto 80 (esto es raro, pero algunos ISP residenciales lo hacen).

Let’s Encrypt no le permite utilizar este desaf´ıo para emitir certificados Wildcard.

Si tiene varios servidores web, debe asegurarse de que el archivo est´e disponible en todos ellos.

Este desaf´ıo esta´ fuera de nuestro alcance, ya que partimos de la premisa de que el tr´afico que queremos proteger nunca saldr´a a Internet, lo que implica que no tendremos ni puertos ni direcciones expuestas para que un servidor externo pueda verificar el recurso mencionado anteriormente.

**Desaf´ıo DNS** Cuando el identificador que se esta´ validando es un nombre de dominio, el cliente puede demostrar el control de ese dominio proporcionando un registro DNS de tipo TXT que contenga un valor designado.

Un cliente cumple este desaf´ıo construyendo una clave de autorizacio´n a partir del valor de un token proporcionado y la clave de la cuenta del cliente. A continuacio´n, el cliente calcula un hash SHA-256 de la clave de autorizaci´on. El registro proporcionado al DNS contiene la codificaci´on de URL base64 de este hash.

Ejemplo:

Si se desea validar el nombre de dominio “www.ejemplo.org”, el cliente proporcionar´ıa el siguiente registro DNS:

\_acme-challenge.www.ejemplo.org: "gfj9Xq...Rg85nM"

Al recibir una respuesta, el servidor construye y almacena la llave de autorizacio´n clave a partir del valor del “token” del desaf´ıo y la clave de la cuenta del cliente actual.

Para validar un desaf´ıo de DNS, el servidor realiza los siguientes pasos:

1. Calcula el hash SHA-256 de la clave de autorizacio´n almacenada.
2. Consulta los registros TXT para el nombre de dominio de validacio´n.
3. Verifica que el contenido de uno de los registros TXT coincida con el valor dehash.

Si todas las verificaciones anteriores tienen ´exito, entonces la validacio´n es exitosa. Si no se encuentra ningu´n registro DNS, o si el registro DNS y el contenido del mismo no pasan estas comprobaciones, la validaci´on falla.

**Cap´ıtulo 3**

# Debilidades del protocolo HTTP

HTTP fue originalmente un protocolo donde la transmisio´n de datos era en texto plano. Los mensajes HTTP se env´ıan a trav´es de Internet sin cifrar y, por lo tanto, cualquiera puede leer el mismo cuando se dirige a su destino. Internet, como su nombre indica, es una red de computadoras (*interconnected computer networks*), no un sistema punto a punto. Hablando espec´ıficamente de una red interna, no sabemos co´mo se enrutan los mensajes y nosotros, como usuarios, no tenemos idea de cu´antas otras partes pueden ver lo mismos. Debido a que HTTP va a trav´es de texto plano, los mensajes se pueden interceptar, leer, e incluso alterar en el camino.

En ´este cap´ıtulo veremos por qu´e el protocolo HTTP es inseguro, adicionando luego un caso de estudio donde se demuestra un pequen˜o ataque.

## 3.1. Riesgos de intermediarios

Un intermediario es alguien que puede acceder al contenido de lo que circula por la red. A esta actividad o ataque se lo conoce como *man-in-the-middle* u hombre en el medio. El intermediario puede realizar ciertos actos: interceptar, modificar o fabricar datos. Desde este punto de vista, la confidencialidad puede verse afectada si alguien intercepta datos, y la integridad puede fallar si alguien o un programa modifica o fabrica datos falsos.

Como mencionamos anteriormente, HTTP no fue pensado para ser seguro, por lo que el protocolo en s´ı mismo no puede resolver este problema. En este proyecto, explotaremos esta vulnerabilidad para demostrar la facilidad con la que un agente

19

puede hacerse de nuestro tra´fico.

## 3.2. Confidencialidad del mensaje

La definicio´n de confidencialidad es sencilla: solo las personas o los sistemas autorizados pueden acceder a los datos protegidos. Sin embargo, como veremos en cap´ıtulos posteriores, garantizar la confidencialidad puede resultar dif´ıcil. HTTP se basa en protocolos de transporte subyacentes para brindar confidencialidad a los mensajes cuando se desee. Por s´ı solo, el protocolo no cifra los mensajes, sin embargo, dado que HTTP se ha disen˜ado para ser independiente del protocolo de transporte, de modo que se puede utilizar en muchas formas diferentes de conexio´n cifrada.

## 3.3. Integridad de los mensajes

Integridad refiere a distintos significados en diferentes contextos. Cuando examinamos la forma en la que usamos este t´ermino, encontramos varios significados diferentes. Por ejemplo, si decimos que hemos conservado la integridad de un elemento, podemos querer decir que el elemento est´a sin modificar, modificado solo de formas aceptables, modificado solo por personas autorizadas, modificado solo por procesos autorizados, etc.

Hablando de datos, se reconocen tres aspectos particulares de la integridad: acciones autorizadas, protecci´on de recursos y detecci´on y correcci´on de errores. La integridad se puede hacer cumplir de la misma manera que la confidencialidad: mediante un control riguroso de qui´en o qu´e puede acceder a qu´e recursos y de qu´e manera.

De la misma manera que en la integridad del mensaje, HTTP no define un mecanismo espec´ıfico para garantizar la integridad de los mensajes, sino que se basa en la capacidad de deteccio´n de errores de los protocolos de transporte subyacentes y en el uso de tramas delimitadas por longitud. Hist´oricamente, la falta de un mecanismo de integridad u´nico se ha justificado por la naturaleza informal de la mayor´ıa de las comunicaciones HTTP. Sin embargo, el predominio de HTTP como mecanismo de acceso a la informacio´n ha dado como resultado la necesidad de verificar la integridad en ciertos entornos.

*3.4. TIPOS DE ATAQUES* 21

## 3.4. Tipos de ataques

Existen principalmente dos tipos de ataques a la red: ataques pasivos y ataques activos. La motivaci´on detra´s de los atacantes pasivos y los atacantes activos son totalmente diferentes. Mientras que la motivaci´on de los atacantes pasivos es simplemente robar informacio´n sensible y analizar el tra´fico para robar mensajes futuros, la motivaci´on de los atacantes activos es detener la comunicaci´on normal entre dos entidades leg´ıtimas.

### 3.4.1. Ataques Pasivos

Los ataques pasivos ocurren cuando se monitorea y analiza informaci´on sensible, posiblemente comprometiendo la seguridad de las empresas y sus clientes.

Los atacantes pasivos est´an principalmente interesados en robar informaci´on confidencial. Esto sucede sin el conocimiento de la v´ıctima. Como tales, los ataques pasivos son dif´ıciles de detectar y, por lo tanto, es dif´ıcil de proteger la red de los mismos. Entre los ataques mas comunes esta´n los ataques de an´alisis y monitoreo de tr´afico, como as´ı tambi´en las escuchas de comunicaciones telefo´nicas.

### 3.4.2. Ataques Activos

Los ataques activos ocurren cuando la informacio´n se modifica, se altera o se destruye por completo. Aqu´ı el intruso inicia instrucciones para perturbar la comunicacio´n regular de la red. Algunos de los ataques activos son los siguientes:

Modificacio´n: el nodo malicioso realiza algunas alteraciones en el enrutamiento. Esto da como resultado que el remitente env´ıe mensajes a trav´es de la ruta larga, lo que provoca un retraso en la comunicacio´n. Este es un ataque a la integridad.

Fabricaci´on: un nodo malicioso genera un mensaje de enrutamiento falso que provoca la generacio´n de informacio´n incorrecta sobre la ruta entre dispositivos. Este es un ataque a la autenticidad.

Spoofing: un nodo malicioso presenta incorrectamente su identidad para que el remitente cambie su topolog´ıa, y por consiguiente el destino de sus mensajes.

Denegacio´n de servicios: un nodo malicioso env´ıa un mensaje al nodo y consume el ancho de banda de la red en co´mputo desperdiciado.

Ataque *Man-in-the-middle*: tambi´en llamado ataque de secuestro, es un ataque en el que el atacante altera y transmite en secreto las comunicaciones entre dos partes leg´ıtimas sin su conocimiento. Estas partes, a su vez, desconocen lo que sucede, pues no perciben un cambio en la comunicacio´n.

## 3.5. Caso de estudio: Interceptando la red para obtener credenciales

### 3.5.1. Herramienta Utilizadas

#### Kali Linux

Kali Linux es una distribucio´n de Linux (basada en Debian) centrada en la seguridad. Es una versio´n renombrada de la famosa distribuci´on de Linux conocida como Backtrack, que ven´ıa con un enorme repositorio de herramientas de pirater´ıa de co´digo abierto, para pruebas de penetracio´n de aplicaciones web, inal´ambricas y de red. Aunque Kali Linux contiene la mayor´ıa de las herramientas de Backtrack, el objetivo principal de Kali Linux es hacerlo porta´til para que pueda instalarse en dispositivos basados en arquitecturas ARM como tabletas y Chromebooks.

El uso de herramientas de pirater´ıa de c´odigo abierto tiene un gran inconveniente: contienen una gran cantidad de dependencias cuando se instalan en Linux y deben instalarse en una secuencia predefinida. Adema´s, los autores de algunas herramientas no han publicado documentacio´n precisa, lo que aumenta la dificultad.

Kali Linux simplifica este proceso; contiene muchas herramientas preinstaladas con todas las dependencias y ya esta´ lista para usar. Esto nos permite tener que prestar ma´s atencio´n al ataque real y no a la instalaci´on de la herramienta. Las actualizaciones para las herramientas instaladas en Kali Linux se publican con mayor frecuencia, lo que le ayuda a mantener las a las mismas actualizadas.

Este sistema operativo contiene las herramientas necesarias para realizar nuestro pequen˜o ataque, las mismas sera´n nombradas a continuaci´on.



Figura 3.1: Escritorio de Kali Linux

#### Wireshark

Wireshark es uno de los analizadores de protocolos de red ma´s populares, es de co´digo abierto y gratuito. Wireshark esta´ preinstalado en Kali y es ideal para la resolucio´n de problemas de red, an´alisis y, para este caso de estudio, una herramienta perfecta para monitorear el tra´fico de posibles objetivos. Wireshark usa un kit de herramientas para implementar su interfaz de usuario y para capturar paquetes. Funciona de manera muy similar a un comando *tcpdump*; sin embargo, actu´a al contener una interfaz gr´afica, tiene opciones integradas de clasificacio´n y filtrado.

#### Ettercap

Ettercap es un paquete completo gratuito y de co´digo abierto para ataques basados en intermediarios. Ettercap se puede utilizar para an´alisis de protocolos de redes inform´aticas y auditor´ıas de seguridad, con funciones de rastreo de conexiones en tiempo real y filtrado de contenido. Ettercap funciona poniendo la interfaz de red del atacante en modo promiscuo y ARP para envenenar las ma´quinas v´ıctimas.

### 3.5.2. Realizacio´n del ataque

La idea principal de esta seccio´n es demostrar que, encontra´ndose en una red

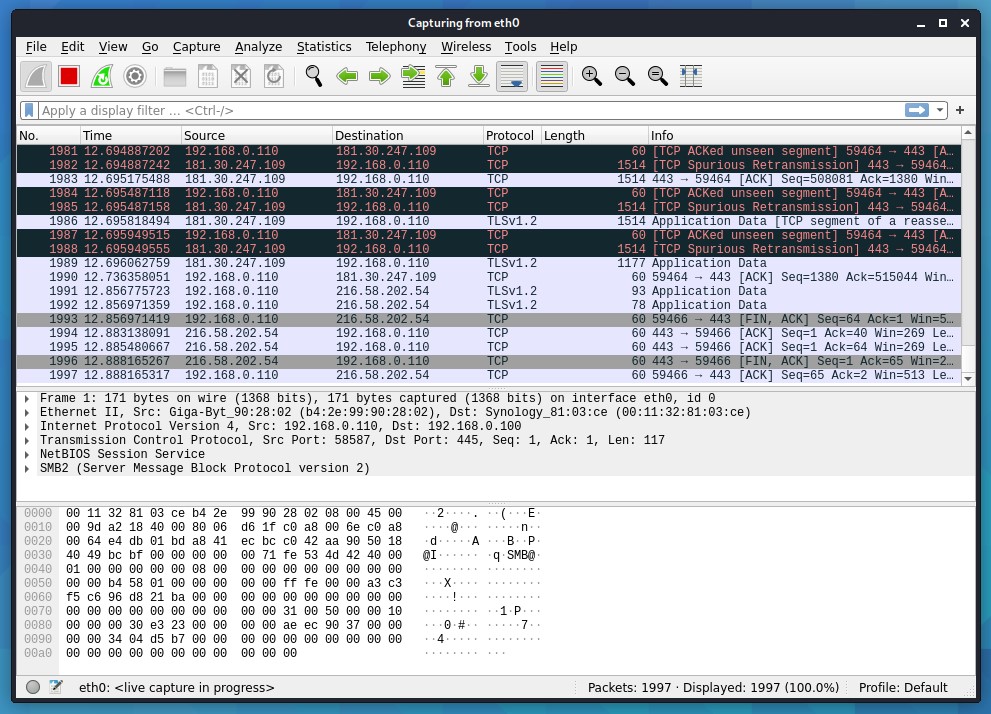


Figura 3.2: Interface del Wireshark

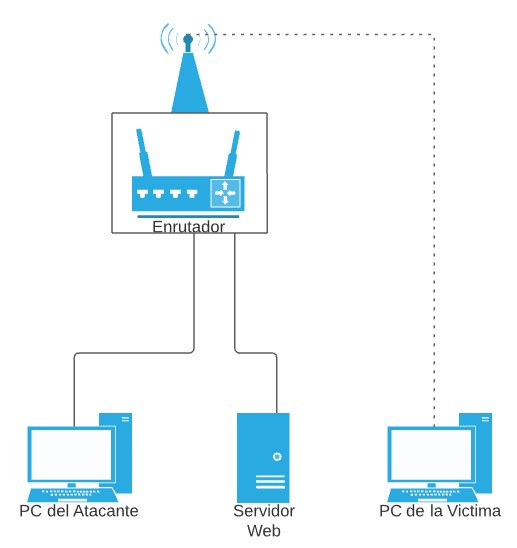
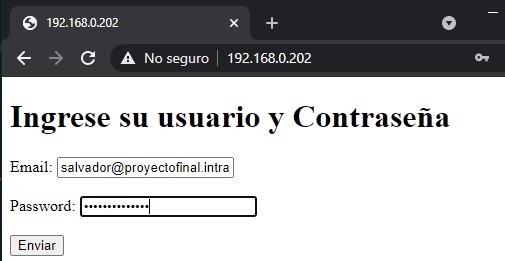


Figura 3.3: Escenario montado Figura 3.4: Formulario montado

interna y con con herramientas ya desarrolladas y libres, es posible realizar un ataque sin necesidad de conocer a fondo la implementaci´on de la misma ni de tener mayores privilegios.

El ataque consiste en crear una pa´gina web con un simple formulario donde se debe completar con usuario y contrasen˜a, y un submit el cual envia esta informacio´n desde el cliente hasta el servidor web. El env´ıo de este formualrio contiene informacio´n confidencial, tales como usuario y contrasen˜a, por lo que en un escenario seguro ningu´n intermediario podr´ıa obtener estos datos. Dado que este tra´fico va a circular por HTTP, demostraremos como nos podemos hacer de las credenciales ingresadas por el usuario.

Recordar que esto fue realizado en una red interna donde son todos equipos de nuestra propiedad.

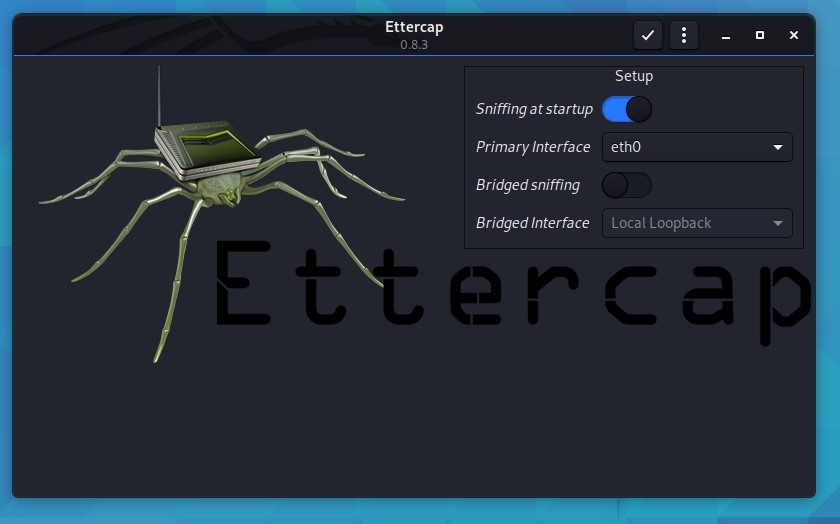
### 3.5.3. Preparando Ettercap para el ataque ARP Poisoning

Lo primero que debemos hacer, en la lista de aplicaciones, es buscar el apartado *Sniffing* y *Spoofing*, ya que es all´ı donde encontraremos las herramientas necesarias para llevar a cabo este ataque. A continuaci´on, abriremos Ettercap.

El siguiente paso es seleccionar la tarjeta de red con la que vamos a trabajar. Para ello, en el menu´ superior de Ettercap seleccionaremos Sniff *>* Unified Sniffing Figura 3.5: Ettercap

y, cuando nos lo pregunte, seleccionaremos nuestra tarjeta de red (por ejemplo, en nuestro caso, eth0).

Luego se debe buscar todos los hosts conectados a nuestra red local. Para ello, seleccionaremos Hosts del menu´ de la parte superior y seleccionaremos la primera opci´on, Hosts List.

All´ı deber´ıan salirnos todos los hosts o dispositivos conectados a nuestra red. Sin embargo, en caso de que no salgan todos, podemos realizar una exploracio´n completa de la red simplemente abriendo el menu´ Hosts y seleccionando la opci´on Scan for hosts. Tras unos segundos, la lista de antes se deber´ıa actualizar mostrando todos los dispositivos, con sus respectivas IPs y MACs, conectados a nuestra red.

### 3.5.4. Nuestro Ettercap ya est´a listo. Ya podemos empezar con el ataque ARP Poisoning

En caso de querer realizar un ataque dirigido contra un solo host, por ejemplo, suplantar la identidad de la puerta de enlace para monitorear las conexiones de la v´ıctima que nos aparece en la lista de dispositivos, antes de empezar con el ataque debemos establecer los dos objetivos.

Para ello, debajo de la lista de hosts podemos ver tres botones, aunque nosotros prestaremos atenci´on a los dos u´ltimos:

Target 1 – Seleccionamos la IP del dispositivo a monitorear, en este caso, la

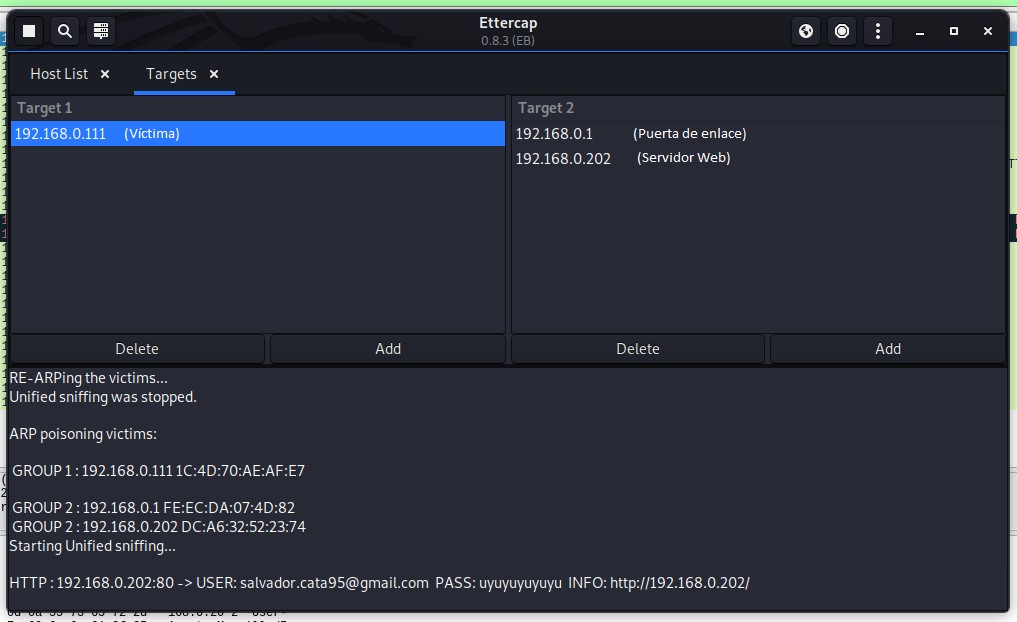


Figura 3.6: Ettercap

computadora de la v´ıctima, y pulsamos sobre dicho boto´n.

Target 2 – Pulsamos la IP que queremos suplantar, en este caso, la de la puerta de enlace y la del servidor web.

Todo listo. Ahora solo debemos elegir el menu´ MITM de la parte superior y, en ´el, escoger la opci´on ARP Poisoning. Nos aparecera´ una pequen˜a ventana de configuracio´n, en la cual debemos asegurarnos de marcar Sniff Remote Connections. Pulsamos sobre Ok y el ataque dar´a lugar. Ahora ya podemos tener el control sobre el host que hayamos establecido como Target 1. Lo siguiente que debemos hacer es, ejecutar Wireshark para capturar todos los paquetes de red y analizarlos en busca de informaci´on interesante.

Como se puede ver, Wireshark nos permite filtrar el tr´afico, y con el simple hecho de decirle que queremos mostrar los requerimientos GET pudimos dar con el paquete que queriamos, en el request podemos ver el usuario “salvador@proyectofinal.test” y la contrasen˜a “PASSWORD SEGURO”

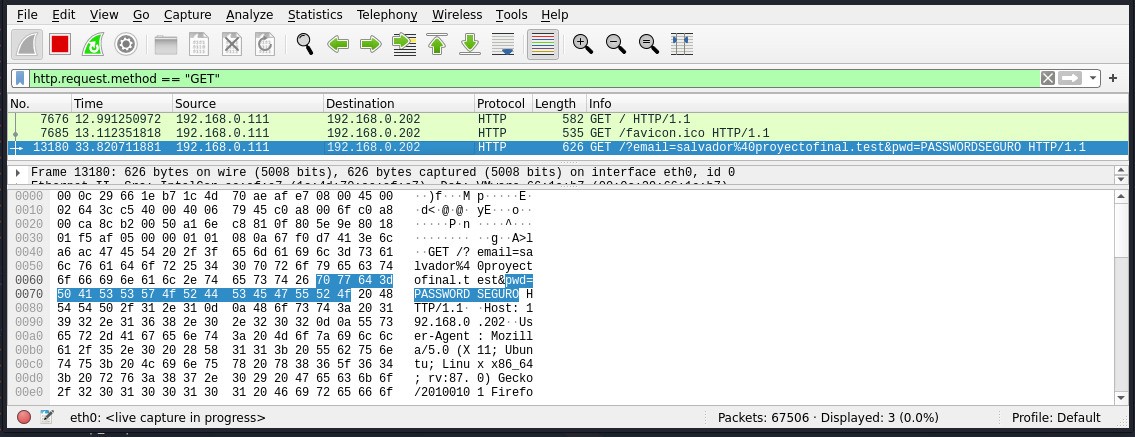


Figura 3.7: Paquetes capturados

**Cap´ıtulo 4**

# Soluciones estudiadas

Luego de haber explicado las debilidades del protocolo HTTP, vamos a explorar las distintas soluciones encontradas, empezando con un pequen˜o marco te´orico, que es lo que nos permitira´ probar estas implementaciones y aplicarlas en un escenario de prueba.

Una herramienta fundamental utilizada es Docker, ya que con ella es posible simular escenarios sin necesidad de tener cada uno de los servidores de manera f´ısica. Para entenderlo un poco mejor, explicaremos lo que es la virtualizaci´on y profundizaremos en la virtualizacio´n basada en contenedores.

## 4.1. M´aquinas virtuales

(AGREGAR GRAFICOS SI O SI, FALTA DEFINICI´ ON de maquinas virtua-´ les) El enfoque basado en VM virtualiza el sistema operativo completo. Con esto nos referimos a la virtualizaci´on de discos, CPU y NIC. En otras palabras, podemos afirmar que se trata de virtualizar la arquitectura de conjunto de instrucciones completa, como ejemplo, la arquitectura *x86*.

Para virtualizar un sistema operativo, se utiliza un software especial, este software se denomina *hipervisor*, y es el encargado, entre otras cosas, de aislar el sistema operativo y los recursos del *hipervisor* de las ma´quinas virtuales y permitir crearlas y gestionarlas.

El VM debe satisfacer tres propiedades:

29

Aislamiento: debe aislar a los invitados (VM) entre s´ı.

Equivalencia: debe comportarse igual, con o sin virtualizaci´on. Esto significa que se deben ejecutar la mayor´ıa de las instrucciones en el hardware f´ısico sin ninguna traduccio´n hacia el hardware virtual.

Rendimiento: Deber´ıa funcionar tan bien como lo hace sin ninguna virtualizacio´n. Esto nuevamente significa que la sobrecarga de ejecutar una VM es m´ınima.

## 4.2. Virtualizacio´n basada en Contenedores

Esta forma de virtualizaci´on no abstrae el hardware, sino que utiliza t´ecnicas dentro del kernel de Linux para aislar las rutas de acceso para diferentes recursos. Establece un l´ımite l´ogico dentro del mismo sistema operativo. Como resultado, obtenemos un sistema de archivos ra´ız separado, un ´arbol de procesos separado, un subsistema de red separado, etc.

El kernel de Linux se compone de varios componentes y funcionalidades; los relacionados con contenedores son los siguientes:

Grupos de control (cgroups)

Espacios de nombres (Namespaces)

Linux con seguridad mejorada (SELinux)

### Cgroups

La funcionalidad de cgroup permite limitar y priorizar recursos, como CPU, RAM, la red, el sistema de archivos, etc. El objetivo principal es no exceder los recursos, para evitar desperdiciar recursos que podr´ıan ser necesarios para otros procesos.

### Espacios de nombres (Namespaces)

La funcionalidad del espacio de nombres permite particionar los recursos del kernel, de modo que un conjunto de procesos ve un conjunto de recursos, mientras que otro conjunto de procesos ve un conjunto diferente de recursos. La caracter´ıstica funciona al tener el mismo espacio de nombres para estos recursos en los distintos

*4.2. VIRTUALIZACION BASADA EN CONTENEDORES´*

conjuntos de procesos, pero esos nombres se refieren a recursos distintos. Algunos nombres de recursos que pueden existir en varios espacios son los ID de proceso, nombres de host, ID de usuario, nombres de archivo y algunos nombres asociados con el acceso a la red y la comunicaci´on entre procesos.

Cuando se inicia un sistema Linux, solo se crea un espacio de nombres. Los procesos y recursos se unir´an al mismo espacio de nombres, hasta que se cree un espacio de nombres diferente, se le asignen recursos y los procesos se unan a ´el.

### SELinux

SELinux es un m´odulo del kernel de Linux que proporciona un mecanismo para hacer cumplir la seguridad del sistema, con pol´ıticas espec´ıficas. B´asicamente, SELinux puede limitar el acceso de los programas a archivos y recursos de red. La idea es limitar los privilegios de los programas y demonios al m´ınimo, de modo que pueda limitar el riesgo de que el sistema se detenga.

La contenerizacio´n utiliza los recursos directamente y no necesita de un emulador en absoluto, cuantos menos recursos, mayor eficiencia. Se pueden ejecutar diferentes aplicaciones en el mismo host: aisladas a nivel de kernel y aisladas por espacios de nombres y cgroups. El kernel (es decir, el SO) lo comparten todos los contenedores, como se muestra en el siguiente diagrama: (DIAGRAMA)

### Contenedores

Cuando hablamos de contenedores, nos referimos indirectamente a dos conceptos principales: una imagen de contenedor y una imagen de contenedor en ejecucio´n. Una imagen de contenedor es la definicio´n del contenedor, en donde el software restante se instala como capas adicionales, como se muestra en el siguiente diagrama: (DIAGRAMA)

Una imagen de contenedor suele estar formada por varias capas. La primera capa est´a dada por la imagen base, que proporciona las funcionalidades centrales del sistema operativo, con todas las herramientas necesarias para comenzar. Los equipos de desarrolladores a menudo trabajan construyendo sus propias capas sobre estas ima´genes base. Los usuarios tambi´en pueden crear ima´genes de aplicaciones ma´s avanzadas, que no solo tienen un sistema operativo, sino que tambi´en incluyen tiempos de ejecucio´n, herramientas de depuracio´n y bibliotecas

Los contenedores brindan aislamiento al aprovechar las tecnolog´ıas del kernel, como cgroups, espacios de nombres del kernel y SELinux, que se han probado y utilizado durante an˜os para brindar aislamiento de aplicaciones. Dado que utilizan un kernel compartido y un host de contenedor, se reduce la cantidad de recursos necesarios para el contenedor en s´ı y son ma´s livianos en comparacio´n con las ma´quinas virtuales.

Por lo mencionado anteriormente, podemos afirmar que los contenedores brindan una agilidad incomparable que no es factible con las VM, y una prueba de esto es que solo se necesitan unos segundos para iniciar un nuevo contenedor. Adem´as, los mismos admiten un modelo ma´s flexible en lo que respecta a la utilizacio´n de la CPU y los recursos de memoria, y permiten modos de ra´faga de recursos, de modo que las aplicaciones pueden consumir ma´s recursos cuando es requerido, dentro de los l´ımites definidos.

## 4.3. Docker: nuestro aliado en las pruebas

Docker es un motor de c´odigo abierto que automatiza la implementacio´n de aplicaciones en contenedores. Fue escrito por el equipo de Docker y publicado por ellos bajo la licencia Apache 2.0. Docker agrega un motor de implementaci´on de aplicaciones sobre un entorno de ejecucio´n de contenedores virtualizado. Esta´ disen˜ado para proporcionar un entorno ligero y ra´pido para ejecutar nuestro co´digo, as´ı como un flujo de trabajo eficiente para llevar ese co´digo desde una computadora porta´til a su entorno de prueba y luego a produccio´n. De hecho, se puede comenzar con Docker en un host m´ınimo que no ejecute nada ma´s que un kernel de Linux compatible y un binario de Docker.

Con Docker, los desarrolladores se preocupan por sus aplicaciones que se ejecutan dentro de los contenedores. Docker esta´ disen˜ado para mejorar la coherencia al garantizar que el entorno en el que los desarrolladores escriben el co´digo coincida con los entornos en los que se implementan sus aplicaciones. Esto reduce el riesgo de funcionamiento distinto de una aplicacio´n dependiendo de donde de ejecute

*4.3. DOCKER: NUESTRO ALIADO EN LAS PRUEBAS*

### 4.3.1. Im´agenes

Las im´agenes son los componentes ba´sicos del mundo de Docker. El uso comu´n es lanzando los contenedores a partir de ima´genes. Tienen un formato en capas, que se forman paso a paso utilizando una serie de instrucciones.

Se puede considerar a las ima´genes como el “co´digo fuente” de los contenedores. Son muy porta´tiles y se pueden compartir, almacenar y actualizar. En este proyecto, utilizaremos im´agenes para crear nuestra propia entidad certificante.

### 4.3.2. Registros

Docker almacena las ima´genes que se crean en registros. Hay dos tipos de registros: pu´blicos y privados. Docker opera el registro pu´blico de im´agenes, llamado Docker Hub. Se puede crear una cuenta en Docker Hub y usarla para compartir y almacenar nuestras propias im´agenes. Tambi´en es posible almacenar las ima´genes que desee y mantenerlas privadas en Docker Hub. Estas ima´genes pueden incluir co´digo fuente u otra informacio´n de propiedad que se quiera mantener segura o solo compartir con otros miembros de su equipo u organizaci´on.

### 4.3.3. Contenedores

Docker permite construir e implementar contenedores dentro de los cuales se puede empaquetar aplicaciones y servicios. Como acabamos de mencionar, los contenedores se lanzan a partir de ima´genes y estos pueden contener uno o m´as procesos en ejecuci´on.

Un contenedor Docker es:

Un formato de imagen.

Un conjunto de operaciones esta´ndar.

Un entorno de ejecucio´n.

Se puede hacer una analog´ıa entre los contenedores de Docker y los contenedores de env´ıo est´andar, utilizado para transportar mercanc´ıas a nivel mundial. En lugar de enviar mercanc´ıas, los contenedores de Docker env´ıan software. Cada contenedor contiene una imagen de software, su “carga”, y, al igual que su contraparte f´ısica, permite realizar un conjunto de operaciones. Por ejemplo, se puede crear, iniciar, detener, reiniciar y destruir. Como un contenedor de env´ıo, Docker no se preocupa por el contenido del contenedor cuando realiza estas acciones; por ejemplo, si un contenedor es un servidor web, una base de datos o un servidor de aplicaciones. Cada contenedor se carga igual que cualquier otro. A Docker tampoco le importa do´nde env´ıa su contenedor: se puede compilar en una computadora porta´til, cargarlo en un registro, luego descargarlo en un servidor f´ısico o virtual, probarlo, implementarlo en un clu´ster de una docena de hosts y ejecutarlo.

## 4.4. Propuestas: Introduccio´n

Se han investigado tres alternativas enfocadas en redes internas para mejorar la seguridad de las mismas, luego de eso se eligi´o la que mas ventajas nos ofrecio´. Vimos tres alternativas posibles: Los certificados auto-firmados (*Self-signed Certificates*), implementar una entidad certificante interna, y la utilizacio´n de un certificado emitido por una entidad certificante conocida.

## 4.5. Propuesta 1: Self-signed Certificates

Los certificados autofirmados son los menos u´tiles de los tres. Firefox facilita su uso de forma segura; crea una excepcio´n en la primera visita, despu´es de lo cual el certificado autofirmado se considera v´alido en las conexiones posteriores. Otros navegadores hacen que haga clic en una advertencia de certificado cada vez. A menos que est´e comprobando la huella digital del certificado cada vez, no es posible hacer que ese certificado autofirmado sea seguro. Incluso con Firefox, puede resultar dif´ıcil utilizar estos certificados de forma segura.

(IMAGEN CON ADVERTENCIA DE CERTIFICADO AUTOFIRMADO)

Por ejemplo, para solicitar un certificado SSL de una CA de confianza como Verisign o GoDaddy, se debe enviar una Solicitud de firma de certificado (CSR) y te dan un certificado a cambio, que firmaron con su certificado ra´ız y clave privada. Todos los navegadores tienen una copia (o acceden a una copia desde el sistema operativo) del certificado ra´ız, por lo que el navegador puede verificar que su certificado fue firmado por una CA confiable.

Cuando generamos un certificado autofirmado, generamos nuestro propio certificado ra´ız y clave privada. Debido a que genera un certificado autofirmado, el navegador no conf´ıa en ´el. Esta´ autofirmado. No ha sido firmado por una CA. Todos los certificados que generamos y firmamos sera´n de confianza inherente.

La principal dificultad es que los usuarios siempre encontrara´n una advertencia donde el navegador diga que se encuentra en un sitio con un certificado autofirmado. En la mayor´ıa de los casos, no verificar´an que el certificado es el correcto, por lo que generara´ desconfianza en los usuarios.

En pr´acticamente todos los casos, un enfoque mucho mejor es utilizar una CA privada, que es nuestra pr´oxima propuesta. Requiere un poco ma´s de trabajo por adelantado, pero una vez que la infraestructura est´a establecida y la clave ra´ız se distribuye de manera segura a todos los usuarios, dichas implementaciones son tan seguras como el resto del ecosistema PKI.

## 4.6. Propuesta 2: Internal CA

(FALTA MAS MARCO TEORICO) Aca´ se tiene que decir: C¸omo se explico´´ anteriormente, una entidad de certificacion es ....Esta alternativa implica establecer una entidad de certificacio´n interna a la red privada. Esto se hace mediante un servidor dedicado que certifique los certificados que circulen internamente. Como ventaja se tiene que ...

Ventajas de la autoridad de certificacio´n interna (CA)

La administraci´on simplificada y fa´cil es la principal ventaja de utilizar una autoridad de certificacio´n (CA) interna. No es necesario depender de una entidad externa para los certificados.

En un entorno de Microsoft Windows, la Autoridad de certificaci´on (CA) interna se puede integrar en Active Directory. Esto simplifica au´n m´as la gesti´on de la estructura de la CA.

No hay ningu´n costo por certificado cuando utiliza una Autoridad de certificacio´n (CA) interna.

Desventajas de la autoridad certificadora (CA) interna

Implementar una autoridad certificadora (CA) interna es m´as complicado que utilizar una autoridad certificadora (CA) externa.

La seguridad y la responsabilidad de la infraestructura de clave pu´blica (PKI) esta´ completamente sobre el hombro de la organizacio´n.

Los usuarios externos normalmente no confiara´n en un certificado digital firmado por una Autoridad de Certificaci´on (CA) interna.

La sobrecarga de gestio´n de certificados de la Autoridad de certificacio´n (CA) interna es mayor que la de la Autoridad de certificacio´n (CA) externa.

La desventaja por la que decidimos ir por una mejor opcio´n fue que, se debe establecer individualmente en cada uno de los hosts pertenecientes a la red privada que nuestra entidad certificante es de confianza, lo que puede llevar a cabo un gran trabajo de los administradores, y, aun as´ı, pueden suceder que en nuestra red se conecten agentes externos a nuestra organizacio´n, por lo que no podremos realizar la configuraci´on mencionada.

### 4.6.1. Caso de estudio: Creando nuestra Entidad Certificante privada

#### Servidor dns con Docker

Se eligio el servidor dns CoreDNS ya que es amigable con Docker, sucede que, por cada versi´on del programa, se generan las ima´genes de docker correspondientes. Estas im´agenes son publicas y oficiales, lo que da confiabilidad y seguridad extra a la hora de utilizarlas. La configuraci´on esta formada por los siguientes componentes: los archivos de configuraci´on de CoreDNS son el corefile y los sitios que nosotros deseemos, en nuestro caso salvadorcatalfamo.intra

El Corefile es el archivo de configuraci´on de CoreDNS. Este define:

Qu´e servidores escuchan en qu´e puertos y qu´e protocolo.

Para qu´e zona tiene autoridad cada servidor.

Qu´e *plugins* (complementos) se cargan en un servidor.

El formato es el siguiente

ZONE: [PORT] {

[PLUGIN] ...

}

ZONE: define la zona de este servidor. El puerto por defecto es el 53, o bien el valor que se le indique con el flag -dns.port.

PLUGIN: define los complementos que queremos cargar. Cada plugin puede tener varias propiedades, por lo que tambi´en podr´ıan tener argumentos de entrada.

Nuestro archivo de configuraci´on es el siguiente:

.:53 { forward . 8.8.8.8 9.9.9.9 log

}

salvadorcatalfamo.intra:53 { file /etc/coredns/salvadorcatalfamo log reload 10s

}

A grandes rasgos, lo que indica esta configuracio´n es que va a existir una zona “salvadorcatalfamo.intra”, que estar´a definida por el archivo que se encuentra en “/etc/coredns/salvadorcatalfamo”. Por otro lado, el tr´afico restante ser´a fordwardeado a servidores DNS externos (8.8.8.8 y 9.9.9.9). Adem´as se establecieron algunos plugins de logeo y de refresco de configuraci´on.

Nuestro archivo “/etc/coredns/salvadorcatalfamo” contiene la siguiente informacio´n.

salvadorcatalfamo.intra. IN SOA ns1.salvadorcatalfamo.intra. ...

pagina1.salvadorcatalfamo.intra. IN A 192.168.0.124

Esto, en principio es suficiente para nuestro sitio interno, y contiene las direcciones ip de los servidores web y entidades certificantes.

Por el lado de Docker, se utiliz´o un archivo Docker.compose.yml, y un Dockerfile. El fichero Docker-compose sirve para (COMPLETAR) En nuestro caso, se definio´ de la siguiente manera

version: ’3.1’ services:

coredns: build: .

container\_name: coredns restart: always expose:

* ’53’
* ’53/udp’ ports:
* ’53:53’
* ’53:53/udp’ volumes:
* ’./config:/etc/coredns’

Por otro lado, el fichero dockerfile esta compuesto por las siguientes l´ıneas

FROM coredns/coredns:1.7.0

ENTRYPOINT ["/coredns"]

CMD ["-conf", "/etc/coredns/Corefile"]

En conjunto, establecen la imagen de CoreDNS que se utilizara´, los archivos de configuracio´n y los puertos que se expondr´an, entre otras configuraciones.

#### Creacio´n de nuestra CA en Docker

La estrategia para crear nuestra CA ser´a seguir los pasos que se deber´ıan realizar en un servidor habitual, pero partiendo desde una imagen de Docker de Ubuntu (de stock), y luego realizando un commit de estas configuraciones. Luego, archivos importantes como el el certificado root y la llave privada debera´n resguardarse, o simplemente resguardar el contenedor creado.

Se utilizo´ esta estrategia ya que no hab´ıa im´agenes oficiales que nos sirva para tal fin, por el simple hecho de que u´nicamente se requiere tener instalado OpenSSL y configurado.

Como primer paso, corremos una imagen del sistema operativo Ubuntu

docker run -it -v $PWD/ca:/root/ca ubuntu

Hay que ver las cosas que se hacen una vez, y las cosas que deben estar configuradas

apt-get update apt-get install ntp apt-get install openssl

Establecer el hostname al contenedor, hay una linea con la ip del contenedor y el nombre del mismo, que es utilizado como hostname, en nuestro caso

172.17.0.2 080dec560726

Lo cambiamos por un hostname con el dominio incluido

172.17.0.2 ca.salvadorcatalfamo.intra

Creamos las carpetas para mejor organizaci´on

mkdr newcerts mkdir certs mkdir crl mkdir private mkdir requests

Creamos un archivo vac´ıo y un archivo que contiene el primer numero de serie para los certificados

touch index.txt echo ’1234’ > serial

Luego hay que crear la llave privada y el certificado root, en este caso nos pedira´ una contrasen˜a, si este servidor se usar´a en un ambiente de produccio´n, debera´ ser una contrasen˜a compleja.

openssl genrsa -aes256 -out private/cakey.pem

Una vez que generamos la llave privada, la misma ser´a utilizada como entrada en la creacio´n de nuestro certificado root. Nos pedir´a algunos datos de localizacio´n y relacionados a la organizacio´n

openssl req -new -x509 -key /root/ca/private/cakey.pem -out cacert.pem -days 3650

Cambiamos los permisos de los archivos que creamos chmod 600 -R /root/ca

Realizamos unas modificaciones en el archivo de configuracio´n vim /usr/lib/ssl/openssl.cnf

Lo u´nico que se tiene que cambiar es la siguiente l´ınea

(IMAGEN O PONER LA LINEA)

Por la direcci´on de los certificados, en nuestro caso el directorio /root/ca Luego cambiamos algunas configuraciones opcionales de pol´ıticas.

Una vez que realizamos estos pasos, estamos listos para realizar el commit de la imagen, con esto, todos los pasos que realizamos (instalar los paquetes, modificar los archivos de configuraci´on, etc) no son necesarios que se ejecuten nuevamente.

Para realizar un commit, y que nuestro contenedor sea fa´cilmente identificable, deberemos seguir los siguientes pasos docker ps -a #identificamos el ultimo contenedor utilizado

(IMAGEN DOCKER-PS-A)

docker commit {id\_del\_contenedor}

docker image ls #Identificamos la imagen recien creada, no tendr´a ni repositorio ni t

(IMAGEN DOCKER IMAGE LS) docker image ls #Identificamos la imagen recien creada, con repositorio y tag

(IMAGEN DOCKER IMAGE LS 2)

Ahora cada vez que queramos correr nuestra ca, lo haremos de la siguiente manera

docker run -it -v $PWD/ca:/root/ca ca:1.0

Em el comando anterior, estamos asumiendo que queremos compartir los archivos de la CA con el servidor host.

#### Nginx con Docker

Para probar nuestro certificado, utilizaremos una imagen oficial de Nginx, los archivos de configuracio´n son los siguientes: docker-compose.yml

web:

image: nginx volumes:

* ./pagina1:/usr/share/nginx/html:ro ports:
* "80:80" environment:
* NGINX\_HOST=pagina1.salvadorcatalfamo.intra
* NGINX\_PORT=80

En el archivo mostrado, le decimos a Docker que utilice la imagen Nginx”, que nuestros archivos fuentes van a estar en el directorio ./pagina1 y que exponga el puerto 80, entre otras cosas.

Para ejecutar este contenedor, se debe ejecutar el siguiente comando: docker-compose up -d

#### Creando nuestro certificado

Para firmar un certificado, el servidor donde se alojara´ la web debe realizar una solicitud, donde nuestra CA retornara´ el certificado firmado. Desde el servidor web en cuesti´on, se debe crear una llave privada privada: openssl genrsa -aes256 -out webserver.pem 2048

Luego, se debera´ crear la solicitud de firma de certificado:

openssl req -new -key webserver.pem -out webserver.csr

Luego enviamos esta solicitud y la firmamos en nuestra CA, esta solicitud la vamos a colocar en el directorio /root/ca/requests openssl ca -in webserver.csr -out webserber.crt

#### Configuracio´n del certificado en el servidor web

Una vez que tenemos este certificado, lo colocamos en en el servidor web y lo configuramos. Para el caso de Nginx, se debe editar el archivo de configuracio´n correspondiente a nuestra web, que en este caso es “pagina1.salvadorcatalfamo.intra” con el fin de que el mismo pueda localizar correctamente los certificados firmados recientemente. Adicionalmente, se puede obligar a que cada requerimiento sea redirigido a una conexio´n segura mediante SSL.

Ahora vamos a ver el navegador, podemos ver que aunque tenga el certificado instalado, no es confiable ya que nuestra entidad certificante no esta configurada como confiable.

(HAY CAPTURAS) ...

Luego de establecer en nuestra computadora, y, particularmente en nuestro navegador Mozilla, que la entidad certificante creada es confiable, es posible ver que nuestra conexio´n es segura, como muestra la siguiente captura.

Luego, ya podemos ver que nuestra pa´gina aparece como una web confiable.

## 4.7. Propuesta 3: Certificacio´n con Let’s Encrypt

Esta estrategia consiste en generar un certificado wildcard, y utilizarlo en cada sitio de la organizacio´n Para esto se debe tener un dominio (en mi caso, salvadorcatalfamo.com) y demostrar la propiedad del mismo. Como se vio anteriormente, hay dos maneras que utiliza Let’s Encrypt para demostrar la propiedad de un dominio, pero la que nos sirve en el caso de una red interna es la que intervienen los registros DNS. La verificacio´n que ofrecen con este tipo de certificado es la m´ınima (DV) y el proceso es explicado a continuaci´on.

### 4.7.1. Pasos a seguir

#### Obtener un dominio

El primer paso es conseguir un dominio, en mi caso ya ten´ıa uno: salvadorcatalfamo.com. Este dominio apunta a mi ip pu´blica. La configuracio´n DNS es la siguiente:

*4.7. PROPUESTA 3: CERTIFICACION CON LET’S ENCRYPT´*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo** | **Nombre** | **Contenido** | **Prioridad** | **TTL** |
| A | salvadorcatalfamo.com | 181.228.121.12 | 0 | 14400 |
| NS | salvadorcatalfamo.com | ns1.donweb.com | 0 | 14400 |
| SOA | salvadorcatalfamo.com | ns2.donweb.com | 0 | 14400 |
| SOA | salvadorcatalfamo.com | ns3.hostmar.com root.hostmar.com  2021010700 28800 7200 2000000 86400 ns2.donweb.com | 0 | 14400 |

#### Instalando Let’s Encrypt en el servidor

sudo add-apt-repository ppa:certbot/certbotsudo apt-get update

sudo apt-get install python-certbot-nginx

#### Instalando Nginx

sudo apt-get update sudo apt-get install nginx

#### Obteniendo un certificado SSL de tipo wildcard desde Let’s Encrypt

sudo certbot --server https://acme-v02.api.letsencrypt.org/directory

-d \*.salvadorcatalfamo.com --manual --preferred-challenges dns-01 certonly

#### Configuracio´n DNS

Luego de ejecutar el comando anterior, Let’s Encrypt nos da un contenido que se debe agregar a un registro DNS. El tipo de registro es TXT y se muestra en la siguiente tabla.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo** | **Nombre** | **Contenido** | **Prioridad** | **TTL** |
| TXT | acme-challenge.salvadorcatalfamo.com | 11UZJD27bPDb jFs6f... | 0 | 14400 |

#### Configuracio´n de Nginx para servir a subdominios

Se debe modificar el siguiente archivo de configuracio´n /etc/nginx/sites-available/example.com como se muestra a continuacio´n

server { listen 80; listen [::]:80; server\_name \*.example.com; return 301 https://$host$request\_uri;

} server { listen 443 ssl;

server\_name \*.example.com; ssl\_certificate /etc/letsencrypt/live/example.com/fullc ssl\_certificate\_key /etc/letsencrypt/live/example.com/privkey.pem; include /etc/letsencrypt/options-ssl-nginx.conf;

ssl\_dhparam /etc/letsencrypt/ssl-dhparams.pem; root /var/www/example.com; index index.html; location / {

try\_files $uri $uri/ =404;

}

}

#### Test la configuraci´on y reinicio del servicio

La configuraci´on se puede testear con el siguiente comando:

sudo nginx -t

Si tiene ´exito, se debe volver a cargar Nginx usando

sudo /etc/init.d/nginx reload

Ahora Nginx contiene un certificado de tipo wildcard, un certificado SSL con respaldo de una entidad certificante como Let’s Encrypt

(IMAGEN)

*4.8. CASO DE ESTUDIO: BUSCANDO CREDENCIALES EN TRAFICO SEGURO´* 45

Le hemos visto un gran potencial a esta solucio´n, aunque tambi´en es poco implementada. Tenemos la gran ventaja de no tener que instalar ningu´n requerimiento en las computadoras dentro de la organizacio´n. Con esto, no se mostrar´an mensajes de seguridad en los navegadores, no importa cual sea el programa, ya que Let’s Encrypt es una entidad de confianza para diversos navegadores y sistemas operativos. Todo esto y la seguridad extra al saber que nuestros datos van por un canal seguro gracias al protocolo SSL.

Como desventajas tenemos que no tenemos la posibilidad de obtener la validaci´on extendida (EV), ya que no est´a disponible actualmente. Por otro lado, que las llaves privadas est´en en diversos servidores a la vez, implica que se deben tener mayores recaudos a la hora de utilizarlos, ya que se debe asegurar el control de las mismas. Aunque se puede llegar a pensar, administrar los certificados y las llaves puede llegar a ser un gran desaf´ıo para los administradores de sistema, sin embargo, hay muchas herramientas que nos proveen automatizaci´on y monitoreo para realizar esta clase de tareas.

## 4.8. Caso de estudio: Buscando credenciales en tr´afico seguro

Luego de ver las diversas soluciones propuestas, una parte importante de nuestro proyecto fue verificar que verdaderamente aumenta la seguridad cuando nuestro tra´fico va encriptado. Para este caso de estudio, se utilizo´ el mismo formulario propuesto en la seccio´n 3.5, lo u´nico que con servidores en distintas direcciones.

Dado que el proceso de capturar el tr´afico en una red interna fue explicado previamente, se van a mostrar u´nicamente los paquetes capturados desde la primera solicitud hasta el env´ıo del formulario

(IMAGEN)

En esta captura podemos observar que:

No es posible determinar, a diferencia de nuestro caso de estudio, a simple vista cual es el paquete en el cual se env´ıa la informacio´n critica.

Viendo el contenido de cada uno de los paquetes mostrados, tampoco es posible ver las credenciales completadas en el formulario, que obviamente son de

46 *CAP´ITULO 4. SOLUCIONES ESTUDIADAS*

nuestro conocimiento.

Se establece una conexio´n segura a trav´es del protocolo TCP.

**Cap´ıtulo 5**

# Conclusiones

* explicacion de navegacion web
* concientizacio´n
* uso de docker agil
* posibles soluciones
* demostracion de las soluciones

48 *CAP´ITULO 5. CONCLUSIONES*

**Cap´ıtulo 6**

# Trabajos a Futuro

Como mencionamos al principio del proyecto, en la redes internas, ademas de los datos que circulan producto de la navegaci´on web, tambi´en circula otro tipo de trafico, como correos electro´nicos, archivos completos de configuraci´on o bien multimedia, como as´ı tambi´en las consultas DNS que se realizar a estos servidores dedicados. En diversas ocasiones, este tipo de tr´afico va sin cifrar, y, hablando particularmente del correo electro´nico, es mas comu´n de lo que parece, ya que tampoco fue disen˜ado pensando en la seguridad. El estudio de todo el circuito que conlleva desde el env´ıo de un correo, hasta la recepci´on quedara´ pendiende para trabajos futuros, ya que puede llegar a tomar tanto como este proyecto mismo.

Con respecto a las propuestas de solucio´n fue la creaci´on de una entidad certificante interna. La gestio´n de la creaci´on de los certificados fue completamente manual, esto, es fa´cilmente automatizable, obviamente estableciendo est´andares de seguridad, tales como encriptaci´on, autenticacio´n y logeo de los certificados generados.

50 *CAP´ITULO 6. TRABAJOS A FUTURO*

**Ap´endice A**

# Glosario

## A.1. Terminolog´ıa

|  |  |
| --- | --- |
| **T´ermino en ingl´es** | **Traducci´on utilizada** |

|  |  |
| --- | --- |
| argument | argumento |
| argumentative system | sistema argumentativo |
| assumption | suposicio´n |
| atom | a´tomo |
| backing | fundamentos |
| blocking defeater | derrotador de bloqueo |
| burden of proof | peso de la prueba |
| claim | afirmacio´n |

## A.2. Simbolog´ıa

|  |  |  |
| --- | --- | --- |
| **S´ımbolo** | **P´agina** | **Significado** |

¬*h* 103 negacio´n fuerte del ´atomo *h*

52 *APENDICE A. GLOSARIO´*

# Bibliograf´ıa

1. Bondarenko, A., Dung, P. M., Kowalski, R., and Toni, F. An abstract

argumentation–theoretic approach to default reasoning. *Artificial Intelligence 93*, 1–2 (1997), 63–101.

1. Capobianco, M. El Rol de las Bases de Dial´ectica en la Argumentacio´n Rebatible. tesis de licenciatura, July 1999.
2. Capobianco, M., Chesnevar, C. I., and Simari, G. R.˜ An argumentative formalism for implementing rational agents. In *Proceedings del 2do Workshop en Agentes y Sistemas Inteligentes (WASI), 7mo Congreso Argentino de Ciencias de la Computaci´on (CACIC)* (El Calafate, Santa Cruz, Oct. 2001), Universidad Nacional de la Patagonia Austral, pp. 1051–1062.
3. Chesnevar, C. I.˜ *Formalizaci´on de los Procesos de Argumentaci´on Rebatible como Sistemas Deductivos Etiquetados*. PhD thesis, Departamento de Ciencias de la Computacio´n, Universidad Nacional del Sur, Bah´ıa Blanca, Argentina, Jan. 2001.
4. Davis, R. E. *Truth, Deduction, and Computation*. Computer Science Press, 1989.
5. Garc´ıa, A. J. La Programacio´n en L´ogica Rebatible: su definicio´n teo´rica y computacional. Master’s thesis, Departamento de Ciencias de la Computacio´n, Universidad Nacional del Sur, Bah´ıa Blanca, Argentina, July 1997.
6. Haenni, R. Modeling uncertainty with propositional assumption-based systems. In *Applications of uncertainty formalisms*, A. Hunter and S. Parsons, Eds.

Springer-Verlag, 1998, pp. 446–470.