Universidad Nacional del Sur

Proyecto Final de Carrera
Ingeniería en Computación

Seguridad en redes LAN: utilizando docker para mejorar la infraestructura.

Salvador Catalfamo

Bahía Blanca – Argentina
2021

Universidad Nacional del Sur

Proyecto Final de Carrera
Ingeniería en Computación

Seguridad en redes LAN: utilizando docker para mejorar la infraestructura.

Salvador Catalfamo

Bahía Blanca – Argentina
2021

Resumen

A lo largo de la carrera, y, particulamente, en una de mis materias preferidas, Seguridad en Sistemas, nos han explicado la importancia la protección de los datos, como asi también la de los canales de comunicación. Mientras se desarrolla una red segura, se debe considerar la confidencialidad, integridad y disponibilidad (CIA).

También, en nuestra corta experiencia hemos visto como las organizaciones abordan los temas de seguridad en sus sistemas informáticos. Mayormente, se concentran en los equipos que están expuestos a la red pública, dejando de lado los que se encuentran aislados de la misma. Erróneamente, muchas veces se piensa que es suficiente, sin embargo, puede traer graves inconvenientes. Es por eso que realizaremos un estudio teórico/práctico sobre las consecuencias de la navegación en redes internas sin ningún tipo de cifrado de datos ni certificaciones.

Palabras Clave:

Seguridad e Infraestructura

Docker

Linux

Kali

Máquinas Virtuales

Índice general

Capítulo 1

Introducción

A lo largo de la carrera, y, particulamente, en una de mis materias preferidas, Seguridad en Sistemas, nos han explicado la importancia la protección de los datos, como asi también la de los canales de comunicación. Mientras se desarrolla una red segura, se debe considerar la confidencialidad, integridad y disponibilidad (CIA).

También, en nuestra corta experiencia hemos visto como las organizaciones abordan los temas de seguridad en sus sistemas informáticos. Mayormente, se concentran en los equipos que están expuestos a la red pública, dejando de lado los que se encuentran aislados de la misma. Erróneamente, muchas veces se piensa que es suficiente, sin embargo, puede traer graves inconvenientes. Es por eso que realizaremos un estudio teórico/práctico sobre las consecuencias de la navegación en redes internas sin ningún 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 administración, automatización y seguridad en sistemas.
- Montar un escenario virtual que sirva de pruebas frente a la gestión de certificados, administración de la infraestructura y a la detección de debilidades dentro una red privada.
- Concientizar la implementacion de medidas de seguridad en redes internas

- Implementar una mejora en la navegacion en una red interna utilizando docker

1.2. Metodología de trabajo

El trabajo se dividirá en tres etapas de trabajo:

- Primera: Se investigarán y plantearán escenarios donde la navegación insegura pueda traer problemas asociados dentro de la organización.

- Segunda: Se plantearán escenarios de trabajo buscando adquirir experiencia en la utilización de la herramienta Docker, para poder obtener parámetros que nos permitan realizar comparaciones con tecnologías similares y sus áreas de aplicación.

- Tercera: Se investigarán e implementarán posibles soluciones para afrontar los aspectos planteador en la primera etapa

Capítulo 2

¿Qué circula por una red interna? (HACER ESTE TITULO MAS ESPECIFICO DE LA NAVEGACION Web)

La conexión en red es la tecnología clave para una amplia variedad de aplicaciones dentro de una organización, como correo electrónico, transferencia de archivos, navegación web, consultas de páginas, etc. Sin embargo, existe una falta significativa de métodos 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 semántica extensible y cargas útiles de mensajes autodescriptivos para una interacción flexible con sistemas basados en red. HTTP es un protocolo genérico para sistemas de información. Está diseñado para ocultar los detalles de cómo 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 propósito de cada cliente: una solicitud HTTP puede aislada en vez de estar asociada con un tipo específico

de cliente o una secuencia predeterminada de pasos de la aplicación. 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 términos de lo que ocurre detrás de la interfaz: estamos limitados a definir la sintaxis de la comunicación, los mensajes y el comportamiento esperado de los destinatarios. Si la comunicación se considera de forma aislada, las acciones exitosas deben reflejarse correspondientemente. Sin embargo, dado que varios clientes pueden actuar en paralelo y quizás con propósitos cruzados, no podemos exigir que tales cambios sean observables más allá del alcance de una única 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 terminología y definiciones de sintaxis utilizadas para definir HTTP.

Mensajes Cliente/Servidor

HTTP es un protocolo de solicitud/respuesta que opera intercambiando mensajes a través de una "conexión" en la capa de sesión o transporte. Un "cliente" HTTP es un programa que establece una conexión a un servidor con el propósito de enviar una o más solicitudes HTTP. Un "servidor" HTTP es un programa que acepta conexiones para atender solicitudes HTTP mediante el envío de respuestas HTTP. La mayoría de las comunicaciones HTTP consisten en una solicitud (GET) de algún recurso identificado por un URI. En el caso más simple, esto podría lograrse mediante una única conexión entre un usuario y el servidor.

Un cliente envía una solicitud HTTP a un servidor en forma de mensaje de solicitud, comenzando con una línea que incluye un método, URI y versión del protocolo, seguida de campos de encabezado que contienen modificadores de solicitud, información del cliente y metadatos de representación, una línea vacía para indicar

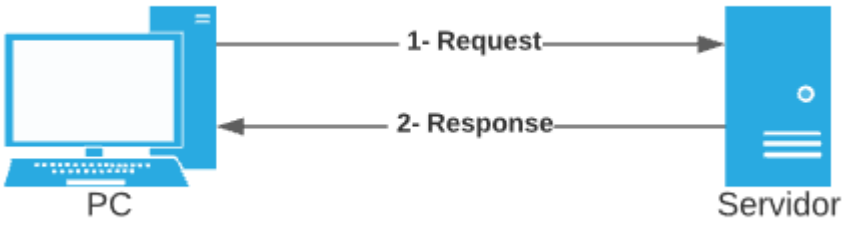

Figura 2.1: Comunicación básica en HTTP

el final de la sección del encabezado, y finalmente un cuerpo del mensaje que contiene el cuerpo de la carga útil (si lo hay). Un servidor responde a la solicitud de un cliente enviando uno o más mensajes de respuesta HTTP, cada uno de los cuales comienza con una línea de estado que incluye la versión del protocolo, un código de estado (éxito o error) y una descripción en forma de texto asociada al mismo, posiblemente seguida de campos de encabezado con información del servidor y metadatos de recursos, una línea vacía para indicar el final de la sección del encabezado y finalmente, un cuerpo del mensaje la carga útil del mismo

Ejemplo

El siguiente ejemplo ilustra un intercambio de mensajes típico para una solicitud GET a la direccion "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. Métodos mas importantes del del protocolo HTTP

El protocolo HTTP contiene varios método, comopor ejemplo PUT, HEAD, DELETE, etc. Sin embargo, para nuestro trabajo explicaremos los dos mas utilizados GET y POST, lo que nos permitirá tener una base a la hora del caso de estudio de la sección 3.6

GET

El método GET solicita al servidor la transferencia de un recurso. GET es el mecanismo principal de recuperación de información y el foco de casi todas las optimizaciones de rendimiento. Por lo tanto, cuando las personas hablan de recuperar información identificable a través 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 práctica. 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 cómo tratar a cada uno de sus recursos.

POST

El método POST solicita que un recurso del servidor sea procesado con los datos que el cliente le envía. 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 en un tablón de anuncios, grupo de noticias, lista de correo, blog o grupo similar de artículos.
- Crear un nuevo recurso que aún no ha sido identificado por el servidor.
- Agregar datos a las representaciones existentes de un recurso.

2.1.3. Códigos de respuesta

El código de estado es un número entero de tres dígitos que da el resultado del intento de comprender y satisfacer la solicitud.

Los códigos de estado HTTP son extensibles. No se requiere que los clientes HTTP comprendan el significado de todos los códigos de estado registrados, aunque se espara una mínima comprensión.

Por ejemplo, si un cliente recibe un código 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 código de estado 400 (Solicitud incorrecta). El mensaje de respuesta generalmente contendrá una representación que explica el estado.

El primer dígito del código de estado define la clase de respuesta. Los dos últimos dígitos no tienen ninguna función de categorización. Hay cinco valores para el primer dígito:

o 1xx (Informativo): Se recibió la solicitud, se continua procesando

o 2xx (satisfactoria): la solicitud se recibió, comprendió y aceptó correctamente

o 3xx (redireccionamiento): se deben realizar más acciones para completar la solicitud

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

o 5xx (error del servidor): el servidor no cumplió con una solicitud aparentemente válida

2.1.4. HTTP con Seguridad SSL

Con una comprensión mínima de los conceptos la criptografía, podemos observar cómo 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 cuándo enviar (y no enviar) cada mensaje. En este capítulo, consideramos cuáles son esos mensajes, la información general que contienen y cómo los sistemas usan los diferentes mensajes en una sesión de comunicaciones.

SSL Roles

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 distinción 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 más común de ssl, la navegación web segura, el navegador web es el cliente ssl y el sitio web es el servidor ssl. Para SSL en sí, las distinciones más importantes entre clientes y servidores son sus acciones durante la negociación de los parámetros de seguridad.

Dado que el cliente inicia una comunicación, 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 qué utilizarán realmente los dos sistemas. Aunque la decisión 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. Este capítulo mostrará cómo los sistemas utilizan estos mensajes en sus comunicaciones. La función más básica (y uno de los propósitos mas importantes) que realiza un cliente y un servidor SSL es establecer la seguridad a través de un canal para comunicaciones cifradas. Los primeros tres mensajes SYN, SYN ACK y

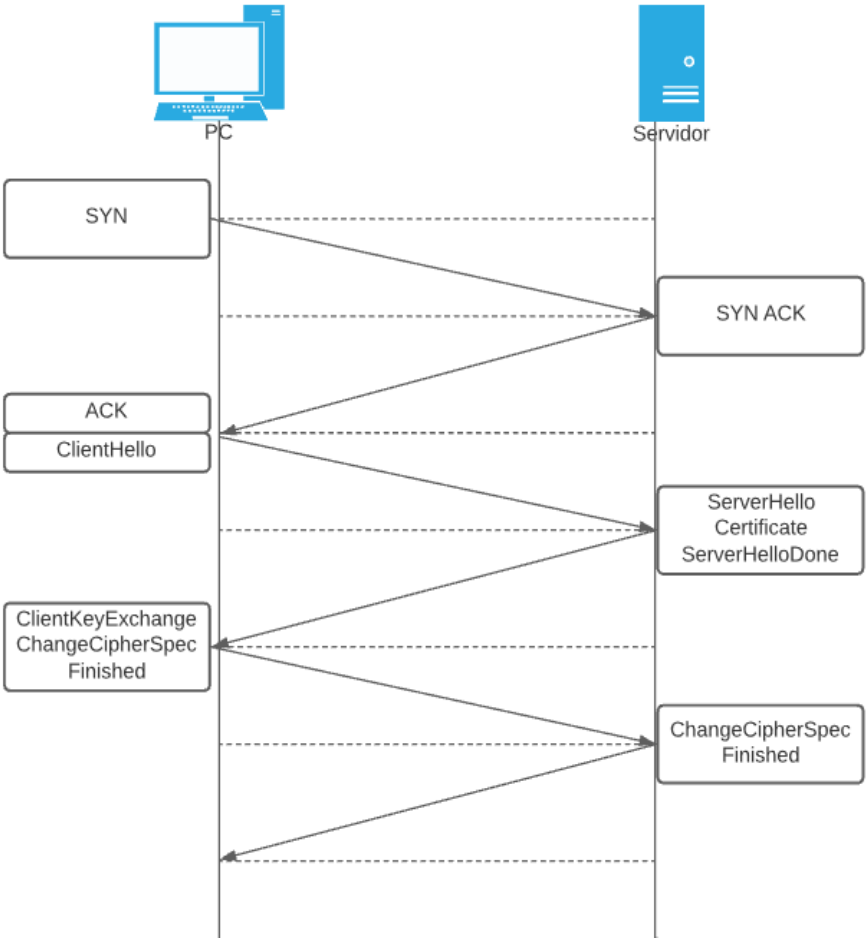

Figura 2.2: Mensajes SSL (grafico beta)

SYN corresponden al protocolo TCP, Luego, inician los mensajes pertenecientes a la comunicación SSL.

ClientHello El mensaje ClientHello inicia la comunicación 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:

- Versión: refiere a la versión más alta de SSL que el cliente puede admitir.
- RandomNumber: proporciona la semilla para cálculos criptográficos críticos.

- SessionID: es opcional, y muchas veces no es utilizado.

- CipherSuites: permite a un cliente enumerar los diversos servicios criptográficos que el cliente puede admitir

- CompressionMethods: es utilizado por el cliente para enumerar todos los diversos métodos de compresión de datos que puede admitir. Los métodos de compresión son una parte importante de ssl porque el cifrado tiene una secuencia significativa en la efectividad de cualquier técnica de compresión de datos.

ServerHello Este mensaje complementa el campo CipherSuite del ServerHello. Si bien el campo CipherSuite indica los algoritmos criptográficos y los tamaños de clave, este mensaje contiene la información de la clave pública en sí. Tenga en cuenta que el mensaje ServerKeyExchange se transmite sin cifrado, por lo que solo se puede incluir de forma segura información de clave pública. El cliente utilizará la clave pública del servidor para cifrar una clave de sesión, que las partes utilizarán para cifrar los datos de la aplicación para la sesión.

ServerKeyExchange Este mensaje complementa el campo CipherSuite del ServerHello. Si bien el campo CipherSuite indica los algoritmos criptográficos y los tamaños de clave, este mensaje contiene la información de la clave pública en sí. Tenga en cuenta que el mensaje ServerKeyExchange se transmite sin cifrado, por lo que solo se puede incluir de forma segura información de clave pública. El cliente utilizará la clave pública del servidor para cifrar una clave de sesión, que las partes utilizarán para cifrar los datos de la aplicación para la sesión.

ServerHelloDone El mensaje ServerHelloDone le dice al cliente que el servidor ha terminado con sus mensajes iniciales de negociación. El mensaje en sí no contiene otra información, 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 negociación ssl inicial, el cliente responde con un mensaje ClientKeyExchange. Así como ServerKeyExchange proporciona la información clave para el servidor, ClientKeyExchange le dice al servidor la información clave del cliente. En este caso, sin embargo, la información clave es para el algoritmo de cifrado simétrico que ambas partes usarán para la sesión. Además, la información del mensaje del cliente se cifra mediante la clave pública del servidor. Este cifrado protege la información 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 pública. De lo contrario, el servidor no podrá descifrar este mensaje. Esta operación es una protección importante contra un atacante que intercepta mensajes de un servidor legítimo y finge ser ese servidor reenviando los mensajes a un cliente desprevenido. Dado que un servidor falso no conocerá la clave privada del servidor real, no podrá descifrar el mensaje ClientKeyExchange. Sin la información en ese mensaje, la comunicación entre las dos partes no puede tener éxito.

ChangeCipherSpec Una vez que el cliente envía información clave en un mensaje ClientKeyExchange, se completa la negociación ssl preliminar. En ese momento, las partes están listas para comenzar a utilizar los servicios de seguridad que han negociado.

El protocolo ssl define un mensaje especial, ChangeCipherSpec, para indicar explícitamente que ahora se deben invocar los servicios de seguridad. Un conjunto de claves protegerá los datos que el cliente envía al servidor, y un conjunto diferente de claves protegerá los datos que el servidor envía 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 información de seguridad de los datos que envía el sistema y el estado de lectura define la información de seguridad de los datos que recibe el sistema.

GRAFIQUITO Figures 3-2 and 3-3

Finished Inmediatamente después de enviar sus mensajes ChangeCipherSpec, cada sistema también envía un mensaje “Finalizado”. Los mensajes “Finalizado” permiten que ambos sistemas verifiquen que la negociación se ha realizado correcta-

mente y que la seguridad no se ha visto comprometida.

Cada mensaje Finalizado contiene un hash criptográfico de información importante sobre la negociación recién finalizada. Esto protege contra un atacante que logra insertar mensajes ficticios o eliminar mensajes legítimos de la comunicación. Si un atacante pudiera hacerlo, los cálculos de hash del cliente y del servidor no coincidirían y detectarían el compromiso.

Finalización de las comunicaciones seguras Aunque, en la práctica, rara vez se utiliza (principalmente debido a la naturaleza de las sesiones web), ssl tiene un procedimiento definido para finalizar una comunicación segura entre dos partes. En este procedimiento, los dos sistemas envían cada uno una alerta de cierre especial al otro. El cierre explícito de una sesión protege contra un ataque de truncamiento, en el que un atacante puede comprometer la seguridad al terminar prematuramente una comunicación.

Autenticar la identidad del servidor

A nteriormente se explicó cómo SSL puede establecer comunicaciones cifradas entre dos partes, lo que puede no agregar mucha seguridad a la comunicación. Con el cifrado solo, ninguna de las partes puede estar realmente segura de la identidad de la otra. La razón típica para usar el cifrado en primer lugar es mantener la información en secreto de algún tercero. Pero si ese tercero pudiera hacerse pasar con éxito como el destinatario previsto de la información, entonces el cifrado no serviría de nada. Los datos estarían encriptados, pero el atacante tendría 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 sección, veremos cómo SSL permite que un servidor se autentique.

Certificado Al autenticar su identidad, el servidor envía un mensaje de certificado en lugar del mensaje ServerKeyExchange descrito anteriormente. El mensaje Certificado simplemente contiene una cadena de certificados que comienza con el certificado de clave pública del servidor y termina con el certificado raíz 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 revocación. También significa asegurarse de que la autoridad de certificación sea una en la que el cliente confíe. Normalmente, los clientes toman esta determinación conociendo la clave pública de las autoridades de certificación confiables de antemano, a través de algunos medios confiables. Microsoft, por ejemplo, carga su software con claves públicas para autoridades de certificación conocidas.

ClientKeyExchange El mensaje ClientKeyExchange del cliente también difiere en la autenticación del servidor, aunque la diferencia no es importante. Cuando solo se va a utilizar cifrado, el cliente cifra la información en en mensaje ClientKeyExchange utilizando la clave pública que el servidor proporciona en su mensaje ServerKeyExchange. En este caso, por supuesto, el servidor se está autenticando y, por lo tanto, ha enviado un mensaje de certificado en lugar de un ServerKeyExchange. El cliente, por lo tanto, encripta su información usando la clave pública contenida en el certificado del servidor. Este paso es importante porque le permite al cliente asegurarse de que la parte con la que se está comunicando realmente posee la clave privada del servidor. Solo un sistema con la clave privada real podrá descifrar este mensaje y continuar con éxito la comunicación.

Niveles de validación

Hay tres tipos de certificados SSL disponibles en la actualidad: Validación extendida (EV SSL), validación por organización (OV SSL) y validación por dominio (DV SSL). Los niveles de cifrado son los mismos para cada certificado, lo que difiere son los procesos de investigación y verificación necesarios para obtener el certificado.

Validación de dominio (DV) Validación 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 más. Los certificados DV SSL suelen ser económicos y se pueden emitir en cuestión de minutos. Eso es porque el proceso de validación está completamente automatizado. Simplemente es necesario demostrar que es propietario de su dominio y que el certificado DV es suyo.

IMAGEN

Validación de la organización (OV) Validación de organización u OV SSL representa el término medio para los tipos de certificados SSL. Para obtener OV SSL, su empresa u organización debe someterse a un examen comercial ligero. Esto puede demorar hasta tres días hábiles porque alguien tiene que verificar la información de su empresa. OV SSL muestra los mismos indicadores visuales que DV SSL, pero proporciona una forma de ver la información comercial verificada en la sección de detalles del certificado.

Extended Validation (EV) SSL de validación extendida o SSL con EV requiere un exhaustivo examen comercial de Comodo. Esto puede parecer mucho, pero en realidad no lo es si su empresa tiene registros disponibles públicamente. EV SSL activa un indicador visual único: el nombre de su organización verificado que se muestra en el navegador.

IMAGEN

Certificate Functionality

Single Domain 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. El tipo de DV está disponible en todos los niveles de validación.

Multi-Domain Este tipo de SSL es un certificado para todos los usos. Permiten cifrar hasta 250 dominios diferentes y subdominios ilimitados. Desafortunadamente, no está disponible en EV.

Wildcard Los Wildcard están diseñados específicamente para cifrar un dominio y todos los subdominios que lo acompañan. Subdominios ilimitados. Desafortunadamente, los Wildcard solo están disponibles en los niveles DV y OV.

Multi-Domain Wildcard Los Wildcard multidominio pueden cifrar hasta 250 dominios diferentes y subdominios ilimitados. Desafortunadamente, no está disponible en EV.

Identifier Validation Challenges

(CAMBIAR LA INTRO, NO ME GUSTA LO DE IDENTIFICADOR, RELACIONARLO MAS CON UN DOMINIO)

ACME uses an extensible challenge/response framework for identifier validation. The server presents a set of challenges in the authorization object it sends to a client (as objects in the "challenges" array), and the client responds by sending a response object in a POST request to a challenge URL.

Different challenges allow the server to obtain proof of different aspects of control over an identifier. In some challenges, like HTTP and DNS, the client directly proves its ability to do certain things related to the identifier. The choice of which challenges to offer to a client under which circumstances is a matter of server policy.

HTTP Challenge Con la validación 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 desafía al cliente solicitándole un archivo en una ruta específica, con una cadena específica como contenido.

Este es el tipo de desafío más común 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, más una huella digital de la clave de su cuenta.

Una vez que el cliente le dice al servidor que el archivo está listo, el servidor intenta recuperarlo. Al recibir una respuesta, el servidor construye y almacena la autorización de la clave a partir del valor del "token" de desafío y la clave de la cuenta del cliente actual.

Dado un par de desafío/respuesta, el servidor verifica el control del dominio por parte del cliente verificando que el recurso se aprovisionó como se esperaba.

Ventajas:

- Es fácil de automatizar sin conocimientos adicionales sobre la configuración de un dominio.
- Funciona con servidores web estándar.

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 desafío para emitir certificados Wildcard.
- Si tiene varios servidores web, debe asegurarse de que el archivo esté disponible en todos ellos.

(EXPLICAR POR QUE NO PUEDO USAR ESTE CHALENGE EN ESTE PROYEXTO)

Desafío DNS Cuando el identificador que se está 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 para el nombre de dominio.

Un cliente cumple este desafío construyendo una clave de autorización a partir del valor de un token proporcionado y la clave de la cuenta del cliente. A continuación, el cliente calcula un hash (REVISAR HASH, la palabra es digest) SHA-256 de la clave de autorización. El registro proporcionado al DNS contiene la codificación de URL base64 de este hash(REVISAR).
Ejemplo:

Si se desea validar el nombre de dominio "www.ejemplo.org", el cliente proporcionaría el siguiente registro DNS:

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

Al recibir una respuesta, el servidor construye y almacena la llave de autorización clave a partir del valor del "token" del desafío y la clave de la cuenta del cliente actual.
Para validar un desafío de DNS, el servidor realiza los siguientes pasos:

1. Calcula el hash (REVISAR) SHA-256 de la clave de autorización almacenada
2. Consulta los registros TXT para el nombre de dominio de validación.

3. Verifica que el contenido de uno de los registros TXT coincida con el valor de hash.

Si todas las verificaciones anteriores tienen éxito, entonces la validación es exitosa. Si no se encuentra ningún registro DNS, o si el registro DNS y el contenido del mismo no pasan estas comprobaciones, la validación falla.

Capítulo 3

Debilidades del protocolo http

En éste capítulo se verá por qué los protocolos http y smtp son inseguros, introducción al snnifin, spoofing, arp attack

(Agregar una intro)

3.1. Riesgos de intermediarios

Un intermediario es alguien que puede ver el 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. Los intermediarios pueden tener acceso a información relacionada con la seguridad, información personal sobre usuarios individuales y organizaciones, o contenido propietario de usuarios y proveedores de contenido. HTTP en sí mismo no puede resolver este problema.

En este proyecto, explotaremos esta vulnerabilidad para demostrar la facilidad con la que un agente puede hacerse de nuestro tráfico.

3.2. División de respuesta

La división de respuestas aprovecha una vulnerabilidad en los servidores (generalmente dentro de un servidor de aplicaciones) donde un atacante puede enviar datos codificados dentro de algún parámetro de la solicitud que luego se decodifica y se repite dentro de cualquiera de los campos de encabezado en la respuesta. Si los datos decodificados están diseñados para que parezca que la respuesta ha finalizado

y ha comenzado una respuesta posterior, la respuesta se ha dividido y el atacante controla el contenido de la segunda respuesta aparente. Luego, el atacante puede realizar cualquier otra solicitud en la misma conexión persistente y engañar a los destinatarios (incluidos los intermediarios) para que crean que la segunda mitad de la división es una respuesta autorizada a la segunda solicitud.

3.3. Integridad de los mensajes

HTTP no define un mecanismo específico para garantizar la integridad de los mensajes, sino que se basa en la capacidad de detección de errores de los protocolos de transporte subyacentes y en el uso de tramas delimitadas por longitud. Históricamente, la falta de un mecanismo de integridad único se ha justificado por la naturaleza informal de la mayoría de las comunicaciones HTTP. Sin embargo, el predominio de HTTP como mecanismo de acceso a la información ha dado como resultado la necesidad de veririficar la integridad en ciertos entornos.

3.4. Confidencialidad del mensaje

De la misma manera que en la integridad del mensaje, HTTP se basa en protocolos de transporte subyacentes para brindar confidencialidad a los mensajes cuando se desee. Por sí solo, el protocolo no cifra los mensajes, sin embargo, dado que HTTP se ha diseñado para ser independiente del protocolo de transporte, de modo que se puede utilizar en muchas formas diferentes de conexión cifrada.

3.5. Tipos de ataques

Existen principalmente dos tipos de ataques a la red: ataques pasivos y ataques activos. La motivación detrás de los atacantes pasivos y los atacantes activos son totalmente diferentes. Mientras que la motivación de los atacantes pasivos es simplemente robar información sensible y analizar el tráfico para robar mensajes futuros, la motivación de los atacantes activos es detener la comunicación normal entre dos entidades legítimas.

3.5.1. Passive Attacks

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

Los atacantes pasivos están principalmente interesados en robar información confidencial. Esto sucede sin el conocimiento de la víctima. Como tales, los ataques pasivos son difíciles de detectar y, por lo tanto, es dificil de proteger la red de los mismos. Entre los ataques mas comunes estan los ataques de análisis y monitoreo de tráfico, como así también las escuchas de comunicaciones telefónicas.

3.5.2. Active Attacks

Los ataques activos ocurren cuando la información se modifica, se altera o se destruye por completo. Aquí el intruso inicia instrucciones para perturbar la comunicación regular de la red. Algunos de los ataques activos son los siguientes:

- Modificación: el nodo malicioso realiza algunas alteraciones en la enrutación. Esto da como resultado que el remitente envíe mensajes a través de la ruta larga, lo que provoca un retraso en la comunicación. Este es un ataque a la integridad.
- Wormhole(Agujero de gusano): este ataque también se denomina ataque de túnel. Un atacante recibe un paquete en un punto. Luego lo envia a otro nodo malicioso en la red. Esto hace que se piense que se encontró un camino más corto en la red.
- Fabricación: un nodo malicioso genera un mensaje de enrutamiento falso que provoca la generación de información 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 topología, y por consiguiente el destino de sus mensajes.
- Denegación de servicios: un nodo malicioso envía un mensaje al nodo y consume el ancho de banda de la red en cómputo desperdiciado.
- Ataque Man-in-the-middle: también llamado ataque de secuestro, es un ataque en el que el atacante altera y transmite en secreto las comunicaciones entre dos partes legítimas sin su conocimiento. Estas partes, a su vez, desconocen lo que sucede, pues no perciben un cambio en la comunicación.

3.6. Caso de estudio: Interceptando la red para obtener credenciales

3.6.1. Herramienta Utilizadas

Kali Linux

Kali Linux es una distribución de Linux (basada en Debian) centrada en la seguridad. Es una versión renombrada de la famosa distribución de Linux conocida como Backtrack, que venía con un enorme repositorio de herramientas de piratería de código abierto, para pruebas de penetración de aplicaciones web, inalámbricas y de red. Aunque Kali Linux contiene la mayoría de las herramientas de Backtrack, el objetivo principal de Kali Linux es hacerlo portátil para que pueda instalarse en dispositivos basados en arquitecturas ARM como tabletas y Chromebooks.

El uso de herramientas de piratería de código abierto tiene un gran inconveniente: contienen una gran cantidad de dependencias cuando se instalan en Linux y deben instalarse en una secuencia predefinida. Además, los autores de algunas herramientas no han publicado documentación precisa, lo que aumenta la dificultad.

Kali Linux simplifica este proceso; contiene muchas herramientas preinstaladas con todas las dependencias y ya está lista para usar. Esto nos permite tener que prestar más atención al ataque real y no a la instalación 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.

(EXPLICAR COMO Y DONDE LO INSTALE) (DECIR QUE FUE EL SISTEMA OPERATIVO UTILIZADO QUE CONTIENEN LAS SIGUIENTES HERRAMIENTAS)

Wireshark

Wireshark es uno de los analizadores de protocolos de red más populares, es de código abierto y gratuito. Wireshark está preinstalado en Kali y es ideal para la resolución de problemas de red, análisis y, para este caso de estudio, una herramienta perfecta para monitorear el tráfico 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, actúa al contener una interfaz gráfica, tiene opciones integradas de clasificación y filtrado.

(HAY MAS, CON GRAFICOS EN) Joseph Muniz, Aamir Lakhani - Pruebas de penetración web con Kali Linux-Packt Publishing (2013)

Ettrcap

Ettercap es un paquete completo gratuito y de código abierto para ataques basados en intermediarios. Ettercap se puede utilizar para análisis de protocolos de redes informáticas y auditorías 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 máquinas víctimas.

(HAY MAS, CON GRAFICOS EN) Joseph Muniz, Aamir Lakhani - Web Penetration Testing with Kali Linux-Packt Publishing (2013) Aca yo segui un tutorial, buscarlo

3.6.2. Realización del ataque

La idea principal de esta sección es demostrar que, encontrandose en una red interna y con con herramientas ya desarrolladas y libres es posible realizar un ataque sin necesidad de conocer a fondo la implementacion de la misma ni de tener mayores privilegios

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

IMAGEN de le red

Tiene que estar: -Router

-Origen de la pagina

-Consumidor de la pagina

-El atacante

3.6.3. Preparando Ettercap para el ataque ARP Poisoning

Lo primero que debemos hacer, en la lista de aplicaciones, es buscar el apartado «9. Sniffing y Spoofing«, ya que es allí donde encontraremos las herramientas

necesarias para llevar a cabo este ataque.

IMAGEN Kali Linux Spoofing

A continuación, abriremos «Ettercap» y veremos una ventana similar a la siguiente.

IMAGEN Kali Linux Ettercap

El siguiente paso es seleccionar la tarjeta de red con la que vamos a trabajar. Para ello, en el menú superior de Ettercap seleccionaremos «Sniff ¿Unified Sniffing» y, cuando nos lo pregunte, seleccionaremos nuestra tarjeta de red (por ejemplo, en nuestro caso, eth0).

IMAGEN Kali Linux - Ettercap - Tarjeta de red

El siguiente paso es buscar todos los hosts conectados a nuestra red local. Para ello, seleccionaremos «Hosts» del menú de la parte superior y seleccionaremos la primera opción, «Hosts List«.

IMAGEN Kali Linux - Ettercap - Lista de hosts

Aquí deberían salirnos todos los hosts o dispositivos conectados a nuestra red. Sin embargo, en caso de que no salgan todos, podemos realizar una exploración completa de la red simplemente abriendo el menú «Hosts» y seleccionando la opción «Scan for hosts«. Tras unos segundos, la lista de antes se debería actualizar mostrando todos los dispositivos, con sus respectivas IPs y MACs, conectados a nuestra red.

IMAGEN Kali Linux - Ettercap - Lista de hosts 2

3.6.4. Nuestro Ettercap ya está 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 monitorizar las conexiones del iPad 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 atención a los dos últimos:

Target 1 – Seleccionamos la IP del dispositivo a monitorizar, en este caso, el iPad, y pulsamos sobre dicho botón. Target 2 – Pulsamos la IP que queremos suplantar, en este caso, la de la puerta de enlace.

IMAGEN Objetivos Ettercap

Todo listo. Ahora solo debemos elegir el menú «MITM» de la parte superior y, en él, escoger la opción «ARP Poisoning«.

IMAGEN Kali Linux - Ettercap - Ataques MITM

Nos aparecerá una pequeña ventana de configuración, en la cual debemos asegurarnos de marcar «Sniff Remote Connections«.

IMAGEN Comenzar MITM ARP Poisoning

Pulsamos sobre «Ok» y el ataque dará lugar. Ahora ya podemos tener el control sobre el host que hayamos establecido como «Target 1«. Lo siguiente que debemos hacer es, por ejemplo, ejecutar Wireshark para capturar todos los paquetes de red y analizarlos en busca de información interesante o recurrir a los diferentes plugins que nos ofrece Ettercap, como, por ejemplo, el navegador web remoto, donde nos cargará todas las webs que visite el objetivo.

Plugins Ettercap

Capítulo 4

Soluciones estudiadas

Luego de haber explicado las debilidades del protocolo http, vamos a explorar las distintas soluciones encontradas, empezando con un pequeño marco teórico, que es lo que nos permetirá 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 física. Para entenderlo un poco mejor, explicaremos lo que es la virtualización y profundizaremos en la virtualización basada en contenedores.

4.1. Máquinas virtuales

(AGREGAR GRAFICOS SI O SI, FALTA DEFINICION de maquinas virtuales) El enfoque basado en VM virtualiza el sistema operativo completo. Con esto nos referimos a la virtualización 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 máquinas virtuales y permitir crearlas y gestionarlas.

El VMM debe satisfacer tres propiedades:

- Aislamiento: debe aislar a los invitados (VM) entre sí.
- Equivalencia: debe comportarse igual, con o sin virtualización. Esto significa que se deben ejecutar la mayoría de las instrucciones en el hardware físico sin ninguna traducción hacia el hardware virtual.
- Rendimiento: Debería funcionar tan bien como lo hace sin ninguna virtualización. Esto nuevamente significa que la sobrecarga de ejecutar una VM es mínima.

4.2. Container-Based Virtualization

Esta forma de virtualización no abstrae el hardware, sino que utiliza técnicas dentro del kernel de Linux para aislar las rutas de acceso para diferentes recursos. Establece un límite lógico dentro del mismo sistema operativo. Como resultado, obtenemos un sistema de archivos raíz separado, un árbol 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 podrían 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 característica funciona al tener el mismo espacio de nombres para estos recursos en los distintos 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 comunicación entre procesos.

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

SELinux

SELinux es un módulo del kernel de Linux que proporciona un mecanismo para hacer cumplir la seguridad del sistema, con políticas específicas. Básicamente, 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 mínimo, de modo que pueda limitar el riesgo de que el sistema se detenga.

La contenerización 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 ejecución. Una imagen de contenedor es la definición 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 está 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 imágenes base. Los usuarios también pueden crear imágenes de aplicaciones más avanzadas, que no solo tienen un sistema operativo, sino que también incluyen tiempos de ejecución, herramientas de depuración y bibliotecas

Los contenedores brindan aislamiento al aprovechar las tecnologías del kernel, como cgroups, espacios de nombres del kernel y SELinux, que se han probado y utilizado durante años 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 sí y son más livianos en comparación con las máquinas 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. Además, los mismos admiten un modelo más flexible en lo que respecta a la utilización de la CPU y los recursos de memoria, y permiten modos de ráfaga de recursos, de modo que las aplicaciones pueden consumir más recursos cuando es requerido, dentro de los límites definidos.

4.3. Docker: nuestro aliado en las pruebas

Docker es un motor de código abierto que automatiza la implementación 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 implementación de aplicaciones sobre un entorno de ejecución de contenedores virtualizado. Está diseñado para proporcionar un entorno ligero y rápido para ejecutar nuestro código, así como un flujo de trabajo eficiente para llevar ese código desde una computadora portátil a su entorno de prueba y luego a producción. De hecho, se puede comenzar con Docker en un host mínimo que no ejecute nada más 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 está diseñado para mejorar la coherencia al garantizar que el entorno en el que los desarrolladores escriben el código coincida con los entornos en los que se implementan sus aplicaciones. Esto reduce el riesgo de funcionamiento distinto de una aplicación dependiendo de donde de ejecute

4.3.1. Imagenes

Las imágenes son los componentes básicos del mundo de Docker. El uso común es lanzando los contenedores a partir de imágenes. Tienen un formato en capas, que se forman paso a paso utilizando una serie de instrucciones.

Se puede considerar a las imágenes como el çódigo fuente"de los contenedores. Son muy portátiles y se pueden compartir, almacenar y actualizar. En este proyecto, utilizaremos imágenes para crear nuestra propia entidad certificante.

4.3.2. Registros

Docker almacena las imágenes que se crean en registros. Hay dos tipos de registros: públicos y privados. Docker opera el registro público de imágenes, llamado Docker Hub. Se puede crear una cuenta en Docker Hub y usarla para compartir y almacenar nuestras propias imágenes. También es posible almacenar las imágenes que desee y mantenerlas privadas en Docker Hub. Estas imágenes pueden incluir código fuente u otra información de propiedad que se quiera mantener segura o solo compartir con otros miembros de su equipo u organización.

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 imágenes y estos pueden contener uno o más procesos en ejecución.

Un contenedor Docker es:

- Un formato de imagen.
- Un conjunto de operaciones estándar.
- Un entorno de ejecución.

Se puede hacer una analogía entre los contenedores de Docker y los contenedores de envío estándar, utilizado para transportar mercancías a nivel mundial. En lugar de enviar mercancías, los contenedores de Docker envían software. Cada contenedor

contiene una imagen de software, su çarga", y, al igual que su contraparte física, permite realizar un conjunto de operaciones. Por ejemplo, se puede crear, iniciar, detener, reiniciar y destruir. Como un contenedor de envío, 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 dónde envía su contenedor: se puede compilar en una computadora portátil, cargarlo en un registro, luego descargarlo en un servidor físico o virtual, probarlo, implementarlo en un clúster de una docena de hosts y ejecutarlo.

4.4. Propuestas: Introducción

Se han investigado tres alternativas enfocadas en redes internas mara mejorar la seguridad de las mismas, luego de eso se eligió la que mas ventajas nos ofreció. Vimos tres alternativas posibles: Los certificados auto-firmado (Self-signed Certificates), implementar una entidad certificante interna, y la utilizacion de un certificado emitido por una entidad certificante conocida

4.5. Propuesta 1: Self-signed Certificates

Los certificados autofirmados son los menos útiles de los tres. Firefox facilita su uso de forma segura; crea una excepción en la primera visita, después de lo cual el certificado autofirmado se considera válido en las conexiones posteriores. Otros navegadores hacen que haga clic en una advertencia de certificado cada vez. A menos que esté comprobando la huella digital del certificado cada vez, no es posible hacer que ese certificado autofirmado sea seguro. Incluso con Firefox, puede resultar difícil utilizar estos certificados de forma segura.

Por ejemplo, para solicitar un certificado SSL de una CA de confianza como Verisign o GoDaddy, se debe enviae una Solicitud de firma de certificado (CSR) y te dan un certificado a cambio, que firmaron con su certificado raíz y clave privada. Todos los navegadores tienen una copia (o acceden a una copia desde el sistema operativo) del certificado raíz, 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 raíz y clave privada. Debido a que genera un certificado autofirmado, el navegador no confía en él. Está autofirmado. No ha sido firmado por una CA. Todos los certificados que generamos y firmamos serán de confianza inherente.

La principal dificultad es que los usuarios siempre encontrarán una advertencia donde el navegador diga que se encuentra en un sitio con un certificado autofirmado. En la mayoría de los casos, no verificarán que el certificado es el correcto, por lo que generará desconfianza en los usuarios.

En prácticamente todos los casos, un enfoque mucho mejor es utilizar una CA privada, que es nuestra próxima propuesta. Requiere un poco más de trabajo por adelantado, pero una vez que la infraestructura está establecida y la clave raíz 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: Çomo se explicó anteriormente, una entidad de certificacion es ....Esta alternativa implica establecer una entidad de certificacion interna a la red privada. Esto se hace mediente un servidor dedicado que certifique los certificados que circulen internamente. Como ventaja se tiene que ...

Ventajas de la autoridad de certificación interna (CA)

• La administración simplificada y fácil es la principal ventaja de utilizar una autoridad de certificación (CA) interna. No es necesario depender de una entidad externa para los certificados.

• En un entorno de Microsoft Windows, la Autoridad de certificación (CA) interna se puede integrar en Active Directory. Esto simplifica aún más la gestión de la estructura de la CA.

• No hay ningún costo por certificado cuando utiliza una Autoridad de certificación (CA) interna.

Desventajas de la autoridad certificadora (CA) interna

• Implementar una autoridad certificadora (CA) interna es más complicado que utilizar una autoridad certificadora (CA) externa.

• La seguridad y la responsabilidad de la infraestructura de clave pública (PKI) está completamente sobre el hombro de la organización.

• Las partes / usuarios externos normalmente no confiarán en un certificado digital firmado por una Autoridad de Certificación (CA) interna.

• La sobrecarga de gestión de certificados de la Autoridad de certificación (CA) interna es mayor que la de la Autoridad de certificación (CA) externa.

La desventaja por la que decidí ir por una mejor opcion 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 asi, pueden suceder que en nuestra red se conecten agentes externos a nuestra organizacion, por lo que no podremos realizar la configuracion 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 versión del programa, se generan las imágenes de docker correspondientes. Estas imágenes son publicas y oficiales, lo que da confiabilidad y seguridad extra a la hora de utilizarlas. La configuración esta formada por los siguientes componentes: los archivos de configuración de CoreDNS son el corefile y los sitios que nosotros deseemos, en nuestro caso salvadorcatalfamo.intra

El Corefile es el archivo de configuración de CoreDNS. Este define:

- Qué servidores escuchan en qué puertos y qué protocolo.
- Para qué zona tiene autoridad cada servidor.
- Qué 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 también podrían tener argumentos de entrada.
Nuestro archivo de configuración 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 configuración es que va a existir una zona "salvadorcatalfamo.intra", que estará definida por el archivo que se encuentra en "/etc/coredns/salvadorcatalfamo". Por otro lado, el tráfico restante será fordwardeado a servidores dns externos (8.8.8.8 y 9.9.9.9). Además se establecieron algunos plugins de logeo y de refresco de configuración.

Nuestro archivo "/etc/coredns/salvadorcatalfamo" contiene la siguiente información.

```
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 webs y entidades certificantes.

Por el lado de Docker, se utilizó un archivo Docker.compose.yml, y un Dockerfile. El fichero Docker-compose sirve para .... En nuestro caso, se definió 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 líneas

```
FROM coredns/coredns:1.7.0

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

En conjunto, establecen la imagen de CoreDNS que se utilizará, los archivos de configuración y los puertos que se expondrán, entre otras configuraciones.

Creación de nuestra CA en Docker

La estrategia para crear nuestra CA será seguir los pasos que se deberían 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 deberan resguardarse, o simplemente resguardar el contenedor creado.

Se utilizó esta estrategia ya que no habia habia imágenes oficiales que nos sirva para tal fin, por el simple hecho de que únicamente 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
```

Setear 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 organizacion

```
mkdr newcerts
mkdir certs
mkdir crl
mkdir private
mkdir requests
```

Creamos un archivo vacio 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 contraseña, si este servidor se usará en un ambiente de producción, deberá ser una contraseña compleja.

```
openssl genrsa -aes256 -out private/cakey.pem
```

Una vez que generamos la llave privada, la misma será utilizada como entrada en la creación de nuestro certificado root. Nos pedirá algunos datos de localización y relacionados a la organización

```
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 configuración

```
vim /usr/lib/ssl/openssl.cnf
```

Lo único que se tiene que cambiar es la siguiente línea

Por la direcciona de los certificados, en nuestro caso el directorio /root/ca Luego cambiamos algunas configuraciones opcionales de políticas.

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 configuración, etc) no son necesarios que se ejecuten nuevamente.

Para realizar un commit, y que nuestro contenedor sea fácilmente 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 tendrá ni repositorio ni
```

(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 querramos 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 configuración 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 alojará la web debe realizar una solicitud, donde nuestra CA retornará el certificado firmado. Desde el servidor web en cuestión, se debe crear una llave privada privada:

```
openssl genrsa -aes256 -out webserver.pem 2048
```

Luego, se deberá 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
```

Configuración 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 ngixn, se debe editar el archivo de configuración correspondiente a nuestra web, que en este caso es "pagina1.salvadorcatalfamo.intraçon 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 conexión 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, particulamente en nuestro navegador Mozilla, que la entidad certificante creada es confiable, es posible ver que nuestra conexión es segura, como muestra la siguiente captura.
Luego, ya podemos ver que nuestra página aparece como una web confiable.

4.7. Propuesta 3: Cerfification with let's encrypt

Esta estrategia consiste en generar un certificado wildrau, y utilizarlo en cada sitio de la organizacion Para esto se debe tener un dominio, en mi caso, salvadorcatalfamo.com, tener la propiedad del dominio implica poder manejar registros dns del mismo, que es lo que requiere letsencript para poder verificar el mismo La verificacion es la minima, que es la de que dice cque soy el dueño del dominio y la verificacion de que cada sitio es mio es la del dns, donde se hace la verificacion con el registro dns Entonces, formalmente los requisitos solución tener el dominio agregar el registro txt al dns solicitar la llave publica y privada colocarla en cada sitio

4.7.1. Pasos a seguir

Get a Domain

The firs step is getting a Domain, in my case, I had one: salvadorcatalfamo.com. This domain points to mi public ip address. For that, I had to create some DNS

records:

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

Installing Let's Encrypt on the server

```
sudo add-apt-repository ppa:certbot/certbotsudo
apt-get update
sudo apt-get install python-certbot-nginx
```

Installing Nginx

```
sudo apt-get update
sudo apt-get install nginx
```

Obtaining wildcard ssl certificate from Let's Encrypt

```
sudo certbot --server https://acme-v02.api.letsencrypt.org/directory
-d *.salvadorcatalfamo.com --manual --preferred-challenges dns-01 certonly
```

Deploy a DNS TXT record provided by Let's Encrypt certbot after running the above command

Then I added an entry to my dns records

Tipo	Nombre	Contenido	Prioridad	TTL
TXT	_acme-challenge.salvadorcatalfamo.com	11UZJD27bPDb_jFs6f...	0	14400

Configuring Nginx to serve wildcard subdomains

Create a config file sudo touch /etc/nginx/sites-available/example.com
Open the file sudo vi /etc/nginx/sites-available/example.com
Add the following code in the file

```
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/full
   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;
   }
 }
```

The above server block is listening on port 80 and redirects the request to the server block below it that is listening on port 443.

Test and restart Nginx

EXTEDER Test Nginx configuration using sudo nginx -t If it's success reload Nginx using

```
sudo /etc/init.d/nginx reload
```

Nginx is now setup to handle wildcard subdomains.

Ver una posible automatizacion, aunque creo que será con puppet

ventajas No mas mensajes de errores Seguridad de encriptacion privacidad, etc etc

Desventajas Tal vez la cantidad de dominios gratis Tal vez la duracion de validez del certificado Tal vez la confiabilidad

4.8. Caso de estudio: Buscando credenciales en tráfico seguro

Luego de ver las diversas soluciones propuestas, una parte importante de nuestro proyecto fue verificar que verdaderamente aumenta la seguridad cuando nuestro tráfico va encriptado. Para este caso de estudio, se utilizó el mismo formulario propuesto en la sección 3.6, lo único que con servidores en distintas direcciones.

Dado que el proceso de capturar el tráfico en una red interna fue explicado previamente, se van a mostrar únicamente los paquetes capturados desde la primera solicitud hasta el envío 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 envía la información 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 nuestro conocimiento.
- Se establece una conexión segura a traves del protocolo TCP.

Capítulo 5

Conclusiones

Apéndice A

Glosario

A.1. Terminología

Término en inglés	**Traducción utilizada**
argument	argumento
argumentative system	sistema argumentativo
assumption	suposición
atom	átomo
backing	fundamentos
blocking defeater	derrotador de bloqueo
burden of proof	peso de la prueba
claim	afirmación

A.2. Simbología

Símbolo	**Página**	**Significado**
$\neg h$	103	negación fuerte del átomo h

Bibliografía

[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.

[2] Capobianco, M. El Rol de las Bases de Dialéctica en la Argumentación Rebatible. tesis de licenciatura, July 1999.

[3] Capobianco, M., Chesñevar, 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 Computación (CACIC)* (El Calafate, Santa Cruz, Oct. 2001), Universidad Nacional de la Patagonia Austral, pp. 1051–1062.

[4] Chesñevar, C. I. *Formalización de los Procesos de Argumentación Rebatible como Sistemas Deductivos Etiquetados*. PhD thesis, Departamento de Ciencias de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, Jan. 2001.

[5] Davis, R. E. *Truth, Deduction, and Computation*. Computer Science Press, 1989.

[6] García, A. J. La Programación en Lógica Rebatible: su definición teórica y computacional. Master's thesis, Departamento de Ciencias de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, July 1997.

[7] 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.