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
2020

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
2020

Resumen

A lo largo de la carrera, 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, 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, se piensa que esto es suficiente, sin embargo, puede traer graves inconvenientes. Por la red interna circulan credenciales, mails, o información sin cifrar crítica, que, si se llegase a filtrar, traerían severos problemas a la organización. Es por eso que realizaremos un estudio teórico/práctico sobre las consecuencias de la navegación y transferencia de información en redes internas sin cifrado de datos ni certificaciones. El trabajo se dividirá en tres secciones, en primera instancia, se demostrará, con herramientas de fácil acceso y, sin un amplio conocimiento, como se puede obtener la información que pasa por la red. En segunda instancia, se utilizarán las herramientas Docker y Let's Encript para paliar esta situación, y, por último, se analizarán los resultados obtenidos.

1.1. Objetivos

Hence, not only the nodes but the communication channel should also be secured. While developing a secure network, Confidentiality, Integrity, Availability (CIA) needs to be considered

- 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 implementará un servidor dedicado a ciertos aspectos de la seguridad:

1. Proxy que se encargue de los certificados ssl de los hosts pertenecientes a la red LAN, obtención y validación diaria de los mismos contra la autoridad de certificación (CA) Let's Encript.

2. Recopilación automática de datos sobre los inicios de sesión en cada uno de los hosts pertenecientes a la organización.

3. Recopilación automática de datos sobre los permisos que poseen los usuarios de la organización.

Capítulo 2

(Ajustar) ¿Qué circula por una red interna?

2.1. Introducción

Network Technology is the key technology for a wide variety of applications such as email, file transfer, web browsing, online transactions, form fill up for various governmental or private activities, cab booking, etc. However, there is a significant lack of easy implementation of security methods for these applications.

2.2. Protocolos asociados a la web

2.2.1. ¿Que es el protocolo HTTP?

The Hypertext Transfer Protocol (HTTP) is a stateless application-level request/response protocol that uses extensible semantics and self-descriptive message payloads for flexible interaction with network-based hypertext information systems. HTTP is a generic interface protocol for information systems. It is designed to hide the details of how a service is implemented by presenting a uniform interface to clients that is independent of the types of resources provided. Likewise, servers do not need to be aware of each client's purpose: an HTTP request can be considered in isolation rather than being associated with a specific type of client or a predetermined sequence of application steps. The result is a protocol that can be

used effectively in many different contexts and for which implementations can evolve independently over time.

One consequence of this flexibility is that the protocol cannot be defined in terms of what occurs behind the interface. Instead, we are limited to defining the syntax of communication, the intent of received communication, and the expected behavior of recipients. If the communication is considered in isolation, then successful actions ought to be reflected in corresponding changes to the observable interface provided by servers. However, since multiple clients might act in parallel and perhaps at cross-purposes, we cannot require that such changes be observable beyond the scope of a single response.

2.2.2. Arquitectura

HTTP was created for the World Wide Web (WWW) architecture and has evolved over time to support the scalability needs of a worldwide hypertext system. Much of that architecture is reflected in the terminology and syntax productions used to define HTTP.

Client/Server Messaging

HTTP is a stateless request/response protocol that operates by exchanging messages across a reliable transport- or session-layer çonnection". An HTTP çlientïs a program that establishes a connection to a server for the purpose of sending one or more HTTP requests. An HTTP "serverïs a program that accepts connections in order to service HTTP requests by sending HTTP responses. Most HTTP communication consists of a retrieval request (GET) for a representation of some resource identified by a URI. In the simplest case, this might be accomplished via a single bidirectional connection (===) between the user agent (UA) and the origin server (O).

A client sends an HTTP request to a server in the form of a request message, beginning with a request-line that includes a method, URI, and protocol version, followed by header fields containing request modifiers, client information, and representation metadata, an empty line to indicate the end of the header section, and

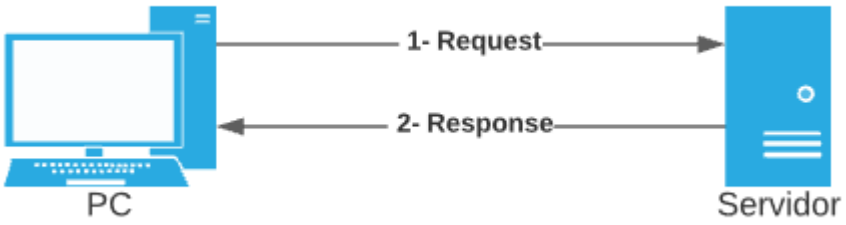

Figura 2.1: Simple comunication method

finally a message body containing the payload body (if any, Section 3.3). A server responds to a client's request by sending one or more HTTP response messages, each beginning with a status line that includes the protocol version, a success or error code, and textual reason phrase (Section 3.1.2), possibly followed by header fields containing server information, resource metadata, and representation metadata (Section 3.2), an empty line to indicate the end of the header section, and finally a message body containing the payload body (if any, Section 3.3).

Ejemplo

The following example illustrates a typical message exchange for a GET request (Section 4.3.1 of [RFC7231]) on the URI "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.

2.2.3. Métodos mas importantes del del protocolo HTTP

head delete CONNECT OPTIONS TRACE PUT

GET

The GET method requests transfer of a current selected representation for the target resource. GET is the primary mechanism of information retrieval and the focus of almost all performance optimizations. Hence, when people speak of retrieving some identifiable information via HTTP, they are generally referring to making a GET request.

It is tempting to think of resource identifiers as remote file system pathnames and of representations as being a copy of the contents of such files. In fact, that is how many resources are implemented . However, there are no such limitations in practice. The HTTP interface for a resource is just as likely to be implemented as a tree of content objects, a programmatic view on various database records, or a gateway to other information systems. Even when the URI mapping mechanism is tied to a file system, an origin server might be configured to execute the files with the request as input and send the output as the representation rather than transfer the files directly. Regardless, only the origin server needs to know how each of its resource identifiers corresponds to an implementation and how each implementation manages to select and send a current representation of the target resource in a response to GET.

POST

The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics. For example, POST is used for the following functions (among others):

o Providing a block of data, such as the fields entered into an HTML form, to a data-handling process; o Posting a message to a bulletin board, newsgroup, mailing list, blog, or similar group of articles;

o Creating a new resource that has yet to be identified by the origin server; and

o Appending data to a resource's existing representation(s).

2.2.4. Response Status Codes

The status-code element is a three-digit integer code giving the result of the attempt to understand and satisfy the request.

HTTP status codes are extensible. HTTP clients are not required to understand the meaning of all registered status codes, though such understanding is obviously

desirable. However, a client MUST understand the class of any status code, as indicated by the first digit, and treat an unrecognized status code as being equivalent to the x00 status code of that class, with the exception that a recipient MUST NOT cache a response with an unrecognized status code.

For example, if an unrecognized status code of 471 is received by a client, the client can assume that there was something wrong with its request and treat the response as if it had received a 400 (Bad Request) status code. The response message will usually contain a representation that explains the status.

The first digit of the status-code defines the class of response. The last two digits do not have any categorization role. There are five values for the first digit:

o 1xx (Informational): The request was received, continuing process

o 2xx (Successful): The request was successfully received, understood, and accepted

o 3xx (Redirection): Further action needs to be taken in order to complete the request

o 4xx (Client Error): The request contains bad syntax or cannot be fulfilled o 5xx (Server Error): The server failed to fulfill an apparently valid request

2.2.5. HTTPS con SSL

With an understanding of some of the key concepts of cryptography, we can now look closely at the operation of the Secure Sockets Layer (ssl) protocol. Although ssl is not an extremely complicated protocol, it does offer several options and variations. The ssl protocol consists of a set of messages and rules about when to send (and not to send) each one. In this chapter, we consider what those messages are, the general information they contain, and how systems use the different messages in a communications session.

SSL Roles

The Secure Sockets Layer protocol defines two different roles for the communicating parties. One system is always a client, while the other is a server. The distinction is very important, because ssl requires the two systems to behave very differently. The client is the system that initiates the secure communications; the server res-

ponds to the client's request. In the most common use of ssl, secure Web browsing, the Web browser is the ssl client and the Web site is the ssl server. For ssl itself, the most important distinctions between clients and servers are their actions during the negotiation of security parameters. Since the client initiates a communication, it has the responsibility of proposing a set of ssl options to use for the exchange. The server selects from the client's proposed options, deciding what the two systems will actually use. Although the final decision rests with the server, the server can only choose from among those options that the client originally proposed.

SSL Messages

When ssl clients and servers communicate, they do so by exchanging ssl messages. this chapter will show how systems use these messages in their communications. The most basic function that an ssl client and server can perform is establishing a channel for encrypted communications. This section looks at these steps in more detail by considering each message in the exchange.

GRAFIQUITO DE LOS MENSAJES

ClientHello The ClientHello message starts the ssl communication between the two parties. The client uses this message to ask the server to begin negotiating security services by using ssl.

The Version field of the ClientHello message contains the highest version number of ssl that the client can support. The current ssl version is 3.0, and it is by far the most widely deployed on the Internet.

The RandomNumber field, as you might expect, contains a random number. This random value, along with a similar random value that the server creates, provides the seed for critical cryptographic calculations. The ssl specification suggests that four of this field's 32 bytes consist of the time and date.

The next field in the ClientHello message is SessionID. Although all ClientHello messages may include this field, in this example, the field is meaningless and would be empty. The CipherSuites field allows a client to list the various cryptographic services that the client can support, including exact algorithms and key sizes. The server actually makes the final decision as to which cryptographic services will be used for the communication, but it is limited to choosing from this list.

The CompressionMethods field is, in theory, similar to the CipherSuites field. In it, the client may list all of the various data compression methods that it can support. Compression methods are an important part of ssl because encryption has significant consequences on the effectiveness of any data compression techniques. Encryption changes the mathematical properties of information in a way that makes data compression virtually impossible.

ServerHello When the server receives the ClientHello message, it responds with a ServerHello. The Version field is the first example of a server making a final decision for the communications. The ClientHello's version simply identifies which ssl versions the client can support. The ServerHello's version, on the other hand, determines the ssl version that the communication will use. The RandomNumber field of the ServerHello is essentially the same as in the ClientHello, though this random value is chosen by the server. Along with the client's value, this number seeds important cryptographic calculations. The SessionID field of a ServerHello may contain a value, unlike the ClientHello's field just discussed. The value in this case uniquely identifies this particular ssl communication, or session. The main reason for explicitly identifying a particular ssl session is to refer to it again later. The CipherSuite field (note that the name is singular, not plural, as in the case of a ClientHello) determines the exact cryptographic parameters, specifically algorithms and key sizes, to be used for the session. The server must select a single cipher suite from among those listed by the client in its ClientHello message. The CompressionMethod field is also singular for a ServerHello. In theory, the server uses this field to identify the data compression to be used for the session. Again, the server must pick from among those listed in the ClientHello. Current ssl versions have not defined any compression methods, however, so this field has no practical utility.

ServerKeyExchange In this example, the server follows its ServerHello message with a ServerKeyExchange message. This message complements the CipherSuite field of the ServerHello. While the CipherSuite field indicates the cryptographic algorithms and key sizes, this message contains the public key information itself. The exact format of the key information depends on the particular public key algorithm used. For the rsa algorithm, for example, the server includes the modulus and public

exponent of the server's rsa public key. Note that the ServerKeyExchange message is transmitted without encryption, so that only public key information can be safely included within it. The client will use the server's public key to encrypt a session key, which the parties will use to actually encrypt the application data for the session.

ServerHelloDone The ServerHelloDone message tells the client that the server has finished with its initial negotiation messages. The message itself contains no other information, but it is important to the client, because once the client receives a ServerHelloDone, it can move to the next phase of establishing the secure communications.

ClientKeyExchange When the server has finished its part of the initial ssl negotiation, the client responds with a ClientKeyExchange message. Just as the ServerKeyExchange provides the key information for the server, the ClientKeyExchange tells the server the client's key information. In

this case, however, the key information is for the symmetric encryption algorithm both parties will use for the session. Furthermore, the information in the client's message is encrypted using the public key of the server. This encryption protects the key information as it traverses the network, and it allows the client to verify that the server truly possesses the private key corresponding to its public key. Otherwise, the server won't be able to decrypt this message. This operation is an important protection against an attacker that intercepts messages from a legitimate server and pretends to be that server by forwarding the messages to an unsuspecting client. Since a fake server won't know the real server's private key, it won't be able to decrypt the ClientKeyExchange message. Without the information in that message, communication between the two parties cannot succeed.

ChangeCipherSpec After the client sends key information in a ClientKeyExchange message, the preliminary ssl negotiation is complete. At that point, the parties are ready to begin using the security services they have negotiated. The ssl protocol defines a special message— ChangeCipherSpec—to explicitly indicate that the security services should now be invoked. Since the transition to secured communication is critical, and both parties have to get it exactly right, the ssl specification

is very precise in describing the process. First, it identifies the set of information that defines security services. That information includes a specific symmetric encryption algorithm, a specific message integrity algorithm, and specific key material for those algorithms. The ssl specification also recognizes that some of that information (in particular, the key material) will be different for each direction of communication. In other words, one set of keys will secure data the client sends to the server, and a different set of keys will secure data the server sends to the client. (In principle, the actual algorithms could differ as well, but ssl does not define a way to negotiate such an option.) For any given system, whether it is a client or a server, ssl defines a write state and a read state. The write state defines the security information

for data that the system sends, and the read state defines the security information for data that the system receives. The ChangeCipherSpec message serves as the cue for a system to begin using its security information. Before a client or server sends a ChangeCipherSpec message, it must know the complete security information it is about to activate. As soon as the system sends this message, it activates its write state. Similarly, as soon as a system receives a ChangeCipherSpec from its peer, the system activates its read state.

GRAFIQUITO Figures 3-2 and 3-3

Finished Immediately after sending their ChangeCipherSpec messages, each system also sends a Finished message. The Finished messages allow both systems to verify that the negotiation has been successful and that security has not been compromised. Two aspects of the Finished message contribute to this security. First, as the previous subsection explained, the Finished message itself is subject to the negotiated cipher suite. That means that it is encrypted and authenticated according to that suite. If the receiving party cannot successfully decrypt and verify the message, then clearly something has gone awry with the security negotiation. The contents of the Finished message also serve to protect the security of the ssl negotiation. Each Finished message contains a cryptographic hash of important information about the just-finished negotiation. Notice that protected data includes the exact content of all handshake messages used in the exchange (though ChangeCipherSpec messages are not considered “handshake” messages in the strict sense of the word, and thus are not included). This protects against an attacker who manages to insert fictitious

messages or remove legitimate messages from the communication. If an attacker were able to do so, the client's and server's hash calculations would not match, and they would detect the compromise.

Ending Secure Communications Although as a practical matter it is rarely used (primarily due to the nature of Web sessions), ssl does have a defined procedure for ending a secure communication between two parties. In this procedure, the two systems each send a special ClosureAlert to the other. Explicitly closing a session protects against a truncation attack, in which an attacker is able to compromise security by prematurely terminating a communication.

Authenticating the Server's Identity

previously it was explained how ssl can establish encrypted communications between two parties, that may not really add that much security to the communication. With encryption alone neither party can really be sure of the other's identity. The typical reason for using encryption in the first place is to keep information secret from some third party. But if that third party were able to successfully masquerade as the intended recipient of the information, then encryption would serve no purpose. The data would be encrypted, but the attacker would have all the data necessary to decrypt it. To avoid this type of attack, ssl includes mechanisms that allow each party to authenticate the identity of the other. With these mechanisms, each party can be sure that the other is genuine, and not a masquerading attacker. In this section, we'll look at how ssl enables a server to authenticate itself. A natural question is, of course, if authenticating identities is so important, why don't we always authenticate both parties? Aca un ejemplo que sirva en una red interna The answer lies in the nature of Web commerce. When you want to purchase something using your Web browser, it's very important that the Web site you're browsing is authentic. You wouldn't want to send your credit card number to some imposter posing as your favorite merchant. The merchant, on the other hand, has other means for authenticating your identity. Once it receives a credit card number, for example, it can validate that number. Since the server doesn't need ssl to authenticate your identity, the ssl protocol allows for server authentication only.

Certificate When authenticating your identity, the server sends a Certificate message in place of the ServerKeyExchange message previously described. The Certificate message simply contains a certificate chain that begins with the server's public key certificate and ends with the certificate authority's root certificate. The client has the responsibility to make sure it can trust the certificate it receives from the server. That responsibility includes verifying the certificate signatures, validity times, and revocation status. It also means ensuring that the certificate authority is one that the client trusts. Typically, clients make this determination by knowing the public key of trusted certificate authorities in advance, through some trusted means. Netscape and Microsoft, for example, preload their browser software with public keys for well-known certificate authorities. Web servers that want to rely on this trust mechanism can only obtain their certificates (at least indirectly) from one of these wellknown authorities.

ClientKeyExchange The client's ClientKeyExchange message also differs in server authentication, though the difference is not major. When encryption only is to be used, the client encrypts the information in the ClientKeyExchange using the public key the server provides in its ServerKeyExchange message. In this case, of course, the server is authenticating itself and, thus, has sent a Certificate message instead of a ServerKeyExchange. The client, therefore, encrypts its ClientKeyExchange information using the public key contained in the server's certificate. This step is important because it allows the client to make sure that the party with whom it is communicating actually possesses the server's private key. Only a system with the actual private key will be able to decrypt this message and successfully continue the communication.

Certificate Functionality

Single Domain As the name suggests, a single domain SSL certificate can only be used on a single domain or IP. This is considered the default SSL certificate type. The DV SSL type is available at all validation levels.

Multi-Domain This SSL type is a jack-of-all-trades certificate. Multi-Domain Wildcards can encrypt up to 250 different domains and unlimited sub-domains.

Unfortunately, it's not available in EV.

Wildcard Wildcards are specifically designed to encrypt one domain and all of its accompanying sub-domains. Unlimited sub-domains. Unfortunately, Wildcards are only available at the DV and OV levels.

Multi-Domain Wildcard These are the jack-of-all-trades certificates. Multi-Domain Wildcards can encrypt up to 250 different domains and unlimited sub-domains. Unfortunately, it's not available in EV.

Validation Level

There are three types of SSL Certificate available today; Extended Validation (EV SSL), Organization Validated (OV SSL) and Domain Validated (DV SSL). The encryption levels are the same for each certificate, what differs is the vetting and verification processes needed to obtain the certificate.

Domain Validation (DV) Domain Validation SSL or DV SSL represents the base-level for SSL types. These are perfect for websites that just need encryption and nothing more. DV SSL certificates are typically inexpensive and they can be issued within minutes. That's because the validation process is fully automated. Just prove you own your domain and the DV SSL certificate is yours.

Organization Validation (OV) Organization Validation SSL or OV SSL represents the middle ground for SSL certificate types. To obtains OV SSL, your company or organization must undergo light business vetting. This can take up to three business days because someone has to verify your business information. OV SSL displays the same visual indicators as DV SSL, but provides a way for your customers to check your verified business information in the certificate details section.

Extended Validation (EV) Extended Validation SSL or EV SSL requires extensive business vetting by Comodo. This may sound like a lot, but it's really not if your business has publicly available records. EV SSL activates a unique visual indicator – your verified organization name shown in the browser.

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 çhallenges.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 With HTTP validation, the client in an ACME transaction proves its control over a domain name by proving that it can provision HTTP resources on a server accessible under that domain name. The ACME server challenges the client to provision a file at a specific path, with a specific string as its content.

This is the most common challenge type today. The server gives a token to your ACME client, and your ACME client puts a file on your web server at http://YOUR_DOMAIN /.well-known/acme-challenge/TOKEN . That file contains the token, plus a thumbprint of your account key.

Once the client tells to the server that the file is ready, the server tries retrieving it.On receiving a response, the server constructs and stores the key authorization from the challenge "token"value and the current client account key.

Given a challenge/response pair, the server verifies the client's control of the domain by verifying that the resource was provisioned as expected.

(TAL VEZ PARA LA PRESENTACION)

Pros:

It's easy to automate without extra knowledge about a domain's configuration. It allows hosting providers to issue certificates for domains CNAMEd to them. It works with off-the-shelf web servers.

Cons:

It doesn't work if your ISP blocks port 80 (this is rare, but some residential

ISPs do this). Let's Encrypt doesn't let you use this challenge to issue wildcard certificates. If you have multiple web servers, you have to make sure the file is available on all of them.

(EXPLICAR POR QUE NO PUEDO USAR ESTE DESAFIO)

DNS Challenge When the identifier being validated is a domain name, the client can prove control of that domain by provisioning a TXT resource record containing a designated value for a specific validation domain name.

A client fulfills this challenge by constructing a key authorization from the "token"value provided in the challenge and the client's account key. The client then computes the SHA-256 digest of the key authorization.

The record provisioned to the DNS contains the base64url encoding of this digest. The client constructs the validation domain name by prepending the label _acme-challenge to the domain name being validated, then provisions a TXT record with the digest value under that name. For example, if the domain name being validated is "www.example.org", then the client would provision the following DNS record: _acme-challenge.www.example.org. 300 IN TXT "gfj9Xq...Rg85nM"

On receiving a response, the server constructs and stores the key authorization from the challenge "token"value and the current client account key.

To validate a DNS challenge, the server performs the following steps:

1. Compute the SHA-256 digest of the stored key authorization

2. Query for TXT records for the validation domain name

3. Verify that the contents of one of the TXT records match the digest value

If all of the above verifications succeed, then the validation is successful. If no DNS record is found, or DNS record and response payload do not pass these checks, then the validation fails.

The client SHOULD de-provision the resource record(s) provisioned for this challenge once the challenge is complete, i.e., once the "status"field of the challenge has the value "valid.ºr ïnvalid".

2.3. Protocolos asociados a la consulta de un sitio (DNS)

2.3.1. ¿Que es el protocolo DNS?

Capítulo 3

Network Security Attacks: basic concepts

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

3.1. El problema del protocolo http

(Agregar una intro)

3.1.1. Establishing Authority

HTTP relies on the notion of an authoritative response: a response that has been determined by (or at the direction of) the authority identified within the target URI to be the most appropriate response for that request given the state of the target resource at the time of response message origination. Providing a response from a non-authoritative source, such as a shared cache, is often useful to improve performance and availability, but only to the extent that the source can be trusted or the distrusted response can be safely used.

Unfortunately, establishing authority can be difficult. For example, phishing is an attack on the user's perception of authority, where that perception can be misled by presenting similar branding in hypertext, possibly aided by userinfo obfuscating the authority component (see Section 2.7.1). User agents can reduce the impact of

phishing attacks by enabling users to easily inspect a target URI prior to making an action, by prominently distinguishing (or rejecting) userinfo when present, and by not sending stored credentials and cookies when the referring document is from an unknown or untrusted source.

When a registered name is used in the authority component, the "httpÜRI scheme (Section 2.7.1) relies on the user's local name resolution service to determine where it can find authoritative responses. This means that any attack on a user's network host table, cached names, or name resolution libraries becomes an avenue for attack on establishing authority. Likewise, the user's choice of server for Domain Name Service (DNS), and the hierarchy of servers from which it obtains resolution results, could impact the authenticity of address mappings; DNS Security Extensions are one way to improve authenticity.

Furthermore, after an IP address is obtained, establishing authority for an "httpÜRI is vulnerable to attacks on Internet Protocol routing.

3.1.2. Risks of Intermediaries

By their very nature, HTTP intermediaries are men-in-the-middle and, thus, represent an opportunity for man-in-the-middle attacks. Compromise of the systems on which the intermediaries run can result in serious security and privacy problems. Intermediaries might have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers. A compromised intermediary, or an intermediary implemented or configured without regard to security and privacy considerations, might be used in the commission of a wide range of potential attacks.

Intermediaries that contain a shared cache are especially vulnerable to cache poisoning attacks. Implementers need to consider the privacy and security implications of their design and coding decisions, and of the configuration options they provide to operators (especially the default configuration).

Users need to be aware that intermediaries are no more trustworthy than the people who run them; HTTP itself cannot solve this problem.

3.1.3. Attacks via Protocol Element Length

Because HTTP uses mostly textual, character-delimited fields, parsers are often vulnerable to attacks based on sending very long (or very slow) streams of data, particularly where an implementation is expecting a protocol element with no predefined length.

3.1.4. Response Splitting

Response splitting (a.k.a, CRLF injection) is a common technique, used in various attacks on Web usage, that exploits the line-based nature of HTTP message framing and the ordered association of requests to responses on persistent connections [Klein]. This technique can be particularly damaging when the requests pass through a shared cache.

Response splitting exploits a vulnerability in servers (usually within an application server) where an attacker can send encoded data within some parameter of the request that is later decoded and echoed within any of the response header fields of the response. If the decoded data is crafted to look like the response has ended and a subsequent response has begun, the response has been split and the content within the apparent second response is controlled by the attacker. The attacker can then make any other request on the same persistent connection and trick the recipients (including intermediaries) into believing that the second half of the split is an authoritative answer to the second request.

For example, a parameter within the request-target might be read by an application server and reused within a redirect, resulting in the same parameter being echoed in the Location header field of the response. If the parameter is decoded by the application and not properly encoded when placed in the response field, the attacker can send encoded CRLF octets and other content that will make the application's single response look like two or more responses.

3.1.5. Request Smuggling

Request smuggling ([Linhart]) is a technique that exploits differences in protocol parsing among various recipients to hide additional requests (which might otherwise

be blocked or disabled by policy) within an apparently harmless request. Like response splitting, request smuggling can lead to a variety of attacks on HTTP usage.

3.1.6. Message Integrity

HTTP does not define a specific mechanism for ensuring message integrity, instead relying on the error-detection ability of underlying transport protocols and the use of length or chunk-delimited framing to detect completeness. Additional integrity mechanisms, such as hash functions or digital signatures applied to the content, can be selectively added to messages via extensible metadata header fields. Historically, the lack of a single integrity mechanism has been justified by the informal nature of most HTTP communication. However, the prevalence of HTTP as an information access mechanism has resulted in its increasing use within environments where verification of message integrity is crucial.

User agents are encouraged to implement configurable means for detecting and reporting failures of message integrity such that those means can be enabled within environments for which integrity is necessary. For example, a browser being used to view medical history or drug interaction information needs to indicate to the user when such information is detected by the protocol to be incomplete, expired, or corrupted during transfer. Such mechanisms might be selectively enabled via user agent extensions or the presence of message integrity metadata in a response. At a minimum, user agents ought to provide some indication that allows a user to distinguish between a complete and incomplete response message (Section 3.4) when such verification is desired.

3.1.7. Message Confidentiality

HTTP relies on underlying transport protocols to provide message confidentiality when that is desired. HTTP has been specifically designed to be independent of the transport protocol, such that it can be used over many different forms of encrypted connection, with the selection of such transports being identified by the choice of URI scheme or within user agent configuration.

3.2. Tipes of Attacks

There are mainly two types of network attacks – passive attack and active attack

Passive: This type of attack happens when sensitive information is monitored and analyzed, possibly compromising the security of enterprises and their customers. In short, network intruder intercepts data traveling through the network. • Active: This type of attack happens when information is modified, altered or demolished entirely by a hacker. Here the interloper starts instructions to disturb the network's regular process.

So the motives behind passive attackers and active attackers are totally different. Whereas the motive of passive attackers is simply to steal sensitive information and to analyze the traffic to steal future messages, the motive of active attackers is to stop normal communication between two legitimate entities.

3.2.1. Passive Attacks

Passive attackers aremainly interested in stealing sensitive information. This happens without the knowledge of the victim. As such passive attacks are difficult to detect and thereby secure the network. The following are some of the passive attacks that are in existence [7].

- Traffic Analysis: Attacker senses the communication path between the sender and receiver.
- Monitoring: Attacker can read the confidential data, but he cannot edit or modify the data.
- Eavesdropping: This type of attack occurs in the mobile ad-hoc network where basically the attacker finds out some secret or confidential information from communication.

3.2.2. Active Attacks

The active attacks happen in such a manner so as to notify the victims that their systems have been compromised. As a result, the victim stops communication with the other party. Some of the active attacks are as follows [8].

- Modification: Some alterations in the routing route is performed by the malicious node. This results in causing the sender to send messages through the long route, which causes communication delay. This is an attack on integrity as shown in Fig. 2.

- Wormhole: This attack is also called a tunneling attack. A packet is received by an attacker at one point. He then tunnels it to another malicious node in the network. This causes a beginner to assume that he found the shortest path in the network as shown in Fig. 3.

- Fabrication: A malicious node generates a false routing message that causes the generation of incorrect information about the route between devices. This is an attack on authenticity as shown in Fig. 4.

- Spoofing: A malicious node miss-present his identity so that the sender changes his topology as shown in Fig. 5.

- Denial of services: A malicious node sends a message to the node and consumes the bandwidth of the network as given in Fig.

- Man-in-the-middle: Attack—Also called a hijacking attack, it is an attack where the attacker secretly alters and relays the communications between two legitimate parties without their knowledge. These parties in turn are unaware of the secret hacker consider that they are doing direct communication with each other [13]. Figure 12 depicts this attack.

Capítulo 4

Virtualización

(AGREGAR GRAFICOS SI O SI) Virtualization provides abstraction on top of the actual resources we want to virtualize. The level at which this abstraction is applied changes the way that different virtualization techniques look.At a higher level, there are two major virtualization techniques based on the level of abstraction. • Virtual machine (VM)-based • Container-based Apart from these two virtualizing techniques, there are other techniques, such as unikernels, which are lightweight single-purpose VMs. IBM is currently attempting to run unikernels as processes with projects like Nabla. In this book, we will mainly look at VM-based and container-based virtualizations only.

4.1. Máquinas virtuales

VM-Based VirtualizationThe VM-based approach virtualizes the complete OS. The abstraction it presents to the VM are virtual devices like virtual disks, virtual CPUs, and virtual NICs. In other words, we can state that this is virtualizing the complete ISA (instruction set architecture); as an example, the x86 ISA.With virtual machines, multiple OSes can share the same hardware resources, with virtualized representations of each of the resources available to the VM. For example, the OS on the virtual machine (also called the guest) can continue to do I/O operations on a disk (in this case, it's a virtual disk), thinking that it's the only OS running on the physical hardware (also called the host), although in actuality, it is shared by multiple virtual machines as well as by the host OS.VMs are very similar to other

processes in the host OS. VMs execute in a hardware-isolated virtual address space and at a lower privilege level than the host OS. The primary difference between a process and a VM is the ABI (Application Binary Interface) exposed by the host to the VM. In the case of a process, the exposed ABI has constructs like network sockets, FDs, and so on, whereas with a full-fledged OS virtualization, the ABI will have a virtual disk, a virtual CPU, virtual network cards, and so on.

Hypervisors A special piece of software is used to virtualize the OS, called the hypervisor. The hypervisor itself has two parts: • Virtual Machine Monitor (VMM): Used for trapping and emulating the privileged instruction set (which only the kernel of the operating system can perform). The VMM has to satisfy three properties (Popek and Goldberg, 1973): • Isolation : Should isolate guests (VMs) from each other. • Equivalency : Should behave the same, with or without virtualization. This means we run the majority (almost all) of the instructions on the physical hardware without any translation, and so on. • Performance : Should perform as good as it does without any virtualization. This again means that the overhead of running a VM is minimal.

Device Model The device model of the hypervisor handles the I/O virtualization again by trapping and emulating and then delivering interrupts back to the specific virtual machine. The device model handles: •Memory Virtualization •Shadow Page Table •CPU Virtualization •IO Virtualization

4.2. Container-Based Virtualization

Container-Based VirtualizationThis form of virtualization doesn't abstract the hardware but uses techniques within the Linux kernel to isolate access paths for different resources. It carves out a logical boundary within the same operating system. As an example, we get a separate root file system, a separate process tree, a separate network subsystem, and so on.

The Linux kernel is made up of several components and functionalities; the ones related to containers are as follows: Control groups (cgroups) Namespaces Security-Enhanced Linux (SELinux) Cgroups The cgroup functionality allows for limiting and prioritizing resources, such as CPUs, RAM, the network, the filesystem, and so on. The main goal is to not exceed the resources—to avoid wasting resources

that might be needed for other processes. Namespaces The namespace functionality allows for partitioning of kernel resources, such that one set of processes sees one set of resources, while another set of processes sees a different set of resources. The feature works by having the same namespace for these resources in the various sets of processes, but having those names refer to distinct resources. Examples of resource names that can exist in multiple spaces (so that the named resources will be partitioned) are process IDs, hostnames, user IDs, filenames, and some names associated with network access and inter-process communication. When a Linux system boots; that is, only one namespace is created. Processes and resources will join the same namespace, until a different namespace is created, resources assigned to it, and processes join it. SELinux SELinux is a module of the Linux kernel that provides a mechanism to enforce the security of the system, with specific policies. Basically, SELinux can limit programs from accessing files and network resources. The idea is to limit the privileges of programs and daemons to a minimum, so that it can limit the risk of system halt. The preceding functionalities have been around for many years. Namespaces were first released in 2002, and cgroups in 2005, by Google (cgroups were first named process containers, and then cgroups). For example, SunSolaris 5.10, released at the beginning of 2005, provided support for Solaris containers. Nowadays, Linux containers are the new buzzword, and some people think they are a new means of virtualization. Containerization uses resources directly, and does not need an emulator at all; the fewer resources, the more efficiency. Different applications can run on the same host: isolated at the kernel level and isolated by namespaces and cgroups. The kernel (that is, the OS) is shared by all containers, as shown in the following diagram: Containers When we talk about containers, we are indirectly referring to two main concepts—a container image and a running container image. A container image is the definition of the container, wherein all software stacks are installed as additional layers, as depicted by the following diagram: A container image is typically made up of multiple layers. The first layer is given by the base image, which provides the OS core functionalities, with all of the tools needed to get started. Teams often work by building their own layers on these base images. Users can also build on more advanced application images, which not only have an OS, but which also include language runtimes, debugging tools, and libraries, as shown in the following diagram: Base images are built from the same utilities

and libraries that are included in an OS. A good base image provides a secure and stable platform on which to build applications. Red Hat provides base images for Red Hat Enterprise Linux. These images can be used like a normal OS. Users can customize them for their applications as necessary, installing packages and enabling services to start up just like a normal Red Hat Enterprise Linux Server. Containers provide isolation by taking advantage of kernel technologies, like cgroups, kernel namespaces, and SELinux, which have been battle-tested and used for years at Google and the US Department of Defense, in order to provide application isolation. Since containers use a shared kernel and container host, they reduce the amount of resources required for the container itself, and are more lightweight when compared to VMs. Therefore, containers provide an unmatched agility that is not feasible with VMs; for example, it only takes a few seconds to start a new container. Furthermore, containers support a more flexible model when it comes to CPU utilization and memory resources, and allow for resource burst modes, so that applications can consume more resources when required, within the defined boundaries.

4.3. Docker

4.3.1. Introduction

Docker is an open-source engine that automates the deployment of applications into containers. It was written by the team at Docker, Inc (formerly dotCloud Inc, an early player in the Platform-as-a-Service (PAAS) market), and released by them under the Apache 2.0 license. Docker adds an application deployment engine on top of a virtualized container execution environment. It is designed to provide a lightweight and fast environment in which to run your code as well as an efficient workflow to get that code from your laptop to your test environment and then into production. Docker is incredibly simple. Indeed, you can get started with Docker on a minimal host running nothing but a compatible Linux kernel and a Docker binary.

With Docker, Developers care about their applications running inside containers, and Operations cares about managing the containers. Docker is designed to enhance consistency by ensuring the environment in which your developers write code matches the environments into which your applications are deployed. This reduces the

risk of "worked in dev, now an ops problem.

4.3.2. Docker client and server

Docker is a client-server application. The Docker client talks to the Docker server or daemon, which, in turn, does all the work. You'll also sometimes see the Docker daemon called the Docker Engine. Docker ships with a command line client binary, docker, as well as a full RESTful API to interact with the daemon: dockerd. You can run the Docker daemon and client on the same host or connect your local Docker client to a remote daemon running on another host. You can see Docker's architecture depicted here:

4.3.3. Docker images

Images are the building blocks of the Docker world. You launch your containers from images. Images are the "build" part of Docker's life cycle. They have a layered format, using Union file systems, that are built step-bystep using a series of instructions. For example: Add a file. Run a command. Open a port. You can consider images to be the "source code" for your containers. They are highly portable and can be shared, stored, and updated. In the book, we'll learn how to use existing images as well as build our own images.

4.3.4. Registries

Docker stores the images you build in registries. There are two types of registries: public and private. Docker, Inc., operates the public registry for images, called the Docker Hub. You can create an account on the Docker Hub and use it to share and store your own images. The Docker Hub also contains, at last count, over 10,000 images that other people have built and shared. Want a Docker image for an Nginx web server, the Asterisk open source PABX system, or a MySQL database? All of these are available, along with a whole lot more. You can also store images that you want to keep private on the Docker Hub. These images might include source code or other proprietary information you want to keep secure or only share with other members of your team or organization. You can also run your own private registry,

and we'll show you how to do that in Chapter 4. This allows you to store images behind your firewall, which may be a requirement for some organizations.

4.3.5. Containers

Docker helps you build and deploy containers inside of which you can package your applications and services. As we've just learned, containers are launched from images and can contain one or more running processes. You can think about images as the building or packing aspect of Docker and the containers as the running or execution aspect of Docker. A Docker container is: An image format. A set of standard operations. An execution environment. Docker borrows the concept of the standard shipping container, used to transport goods globally, as a model for its containers. But instead of shipping goods, Docker containers ship software. Each container contains a software image – its 'cargo' – and, like its physical counterpart, allows a set of operations to be performed. For example, it can be created, started, stopped, restarted, and destroyed. Like a shipping container, Docker doesn't care about the contents of the container when performing these actions; for example, whether a container is a web server, a database, or an application server. Each container is loaded the same as any other container. Docker also doesn't care where you ship your container: you can build on your laptop, upload to a registry, then download to a physical or virtual server, test, deploy to a cluster of a dozen Amazon EC2 hosts, and run. Like a normal shipping container, it is interchangeable, stackable, portable, and as generic as possible. With Docker, we can quickly build an application server, a message bus, a utility appliance, a CI test bed for an application, or one of a thousand other possible applications, services, and tools. It can build local, self-contained test environments or replicate complex application stacks for production or development purposes. The possible use cases are endless.

Capítulo 5

Herramienta Utilizadas

5.1. Kali Linux

BackTrack is one of the most famous Linux distribution systems, as can be proven by the number of downloads that reached more than four million as of BackTrack Linux 4.0 pre final.

Kali Linux Version 1.0 was released on March 12, 2013. Five days later, Version 1.0.1 was released, which fixed the USB keyboard issue. In those five days, Kali has been downloaded more than 90,000 times.

Kali Linux is security-focused Linux distribution based on Debian. It's a rebranded version of the famous Linux distribution known as Backtrack, which came with a huge repository of open source hacking tools for network, wireless, and web application penetration testing. Although Kali Linux contains most of the tools from Backtrack, the main aim of Kali Linux is to make it portable so that it could be installed on devices based on the ARM architectures such as tablets and Chromebook, which makes the tools available at your disposal with much ease.

Using open source hacking tools comes with a major drawback: they contain a whole lot of dependencies when installed on Linux and they need to be installed in a predefined sequence. Moreover, authors of some tools have not released accurate documentation, which makes our life difficult.

Kali Linux simplifies this process; it contains many tools preinstalled with all the dependencies and is in ready to use condition so that you can pay more attention

for the actual attack and not on installing the tool. Updates for tools installed in Kali Linux are more frequently released, which helps you to keep the tools up to date. A non-commercial toolkit that has all the major hacking tools preinstalled to test realworld networks and applications is a dream of every ethical hacker and the authors of Kali Linux make every effort to make our life easy, which enables us to spend more time on finding the actual flaws rather than building a toolkit.

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

5.2. Wireshark

Wireshark is one of the most popular, free, and open source network protocol analyzers. Wireshark is preinstalled in Kali and ideal for network troubleshooting, analysis, and for this chapter, a perfect tool to monitor traffic from potential targets with the goal of capturing session tokens. Wireshark uses a GTK+ widget toolkit to implement its user interface and pcap to capture packets. It operates very similarly to a tcpdump command; however, acting as a graphical frontend with integrated sorting and filtering options. (HAY MAS, CON GRAFICOS EN) Joseph Muniz, Aamir Lakhani - Web Penetration Testing with Kali Linux-Packt Publishing (2013)

5.3. Ettrcap

Ettercap is a free and open source comprehensive suite for man-in-the-middle-based attacks. Ettercap can be used for computer network protocol analysis and security auditing, featuring sniffing live connections, content filtering, and support for active and passive dissection of multiple protocols. Ettercap works by putting the attacker's network interface into promiscuous mode and ARP for poisoning the victim machines.

Capítulo 6

Caso de estudio: Sniffing de la red para obtener credenciales

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

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

6.1. Diagrama de explicacion

IMAGEN de le red

Tiene que estar: -Router

-Origen de la pagina

-Consumidor de la pagina

-El atacante

6.2. 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

6.3. 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 7

Mejorando la seguridad en la navegación

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 ons ofrecio. 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

7.1. Self-signed Certificates

Self-signed certificates are the least useful of the three. Firefox makes it easier to use them safely; you create an exception on the first visit, after which the self-signed certificate is treated as valid on subsequent connections. Other browsers make you clickthrough a certificate warning every time.Unless you're actually checking the certificate fingerprint every time, it is not possible to make that self-signed certificate safe. Even with Firefox, it might be difficult to use self-signed certificates safely.

For example, to request an SSL certificate from a trusted CA like Verisign or GoDaddy, you send them a Certificate Signing Request (CSR), and they give you a certificate in return, that they signed using their root certificate and private key. All browsers have a copy (or access a copy from the operating system) of them root certificate, so the browser can verify that your certificate was signed by a trusted CA.

When we generate a self-signed certificate, we generate our own root certificate and private key. Because you generate a self-signed certificate the browser doesn't trust it. It's self-signed. It hasn't been signed by a CA. All certificates that we generate and sign will be inherently trusted.

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 mayoria de los casos, no verificarán que el certificado es el correcto, por lo que generará desconfianza en los usuarios.

Today, self-signed certificates are considered insecure because there is no way for average users to differentiate them from self-signed MITM certificates. In other words, all self-signed certificates look the same. But, we can use a secure DNS to pin the certificate, thus allowing our user agent to know that they are using the right one. MITM certificates are easily detected.

In virtually all cases, a much better approach is to use a private CA. It requires a little more work upfront, but once the infrastructure is in place and the root key is safely distributed to all users, such deployments are as secure as the rest of the PKI ecosystem.

7.2. Internal CA

Aca se tiene que decir: Çomo se explicó anteriormente, una entidad de certificacion es ....[Es]ta 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 ...

Advantages internal Certificate Authority (CA)

• Simplified and ease of management is the main advantage of using internal Certificate Authority (CA). There is no need to depend an external entity for certificates.

• In a Microsoft Windows environment, internal Certificate Authority (CA) can be integrated in Active Directory. This further simplifies the management of the CA structure.

• There is no cost per certificate wen you are using an internal Certificate Authority (CA).

Disadvantages of internal Certificate Authority (CA)

• Implementing internal Certificate Authority (CA) is more complicated than using external Certificate Authority (CA).

• The security and accountability of Public Key Infrastructure (PKI) is completely on the organization's shoulder.

• External parties/users normally will not trust a digital certificate signed by an internal Certification Authority (CA).

• The certificate management overhead of internal Certification Authority (CA) is higher than that of external Certification Authority (CA).

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 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 configuración mencionada

7.3. 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

7.3.1. 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/
   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

Capítulo 8

Conclusiones y Resultados Obtenidos

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.