HyperionDev

PKI and Man-in-the-Middle Attacks

Task

Visit our website

Introduction

Welcome to the task on public key infrastructure (PKI) and man-in-the-middle attacks. In this task, we will investigate authentication, authorisation, access controls, vulnerabilities and exploits, and threat intelligence as they relate to PKI and man-in-the-middle attacks.

Encryption and cryptography

Extra resource

We will start off with an extra resource: **video explaining symmetric and asymmetric encryption** to give you a basic understanding of the principles underlying PKI.

Let's revise and expand your understanding of the basics.

Encryption is the process of converting plaintext (i.e., unencrypted data) into **ciphertext** (i.e., encrypted data) to protect the confidentiality of the data. The process of converting ciphertext back into plaintext is known as **decryption**.

Cryptography studies techniques for secure communication in the presence of third parties. It involves using mathematical algorithms and protocols to convert plaintext into ciphertext in a way that makes it difficult for unauthorised parties to read the original message.

Several types of encryption algorithms can be used, including **symmetric-key algorithms** (e.g., **Advanced Encryption Standard(AES)**) and **asymmetric-key algorithms** (e.g., **Rivest–Shamir–Adleman(RSA)**).

Symmetric-key algorithms use the same key for encryption and decryption, whereas asymmetric-key algorithms use a public key for encryption and a private key for decryption.

Encryption and cryptography are essential tools for protecting the confidentiality of sensitive information as they allow users to communicate securely even when third parties may intercept their messages. They are widely used to protect users' privacy and the confidentiality of their sensitive data in applications such as email, messaging, and online transactions.

Symmetric- and asymmetric-key algorithms

Symmetric-key and asymmetric-key algorithms are two different types of encryption algorithms that are used to secure communication and protect the confidentiality of data.

- **Asymmetric-key algorithms**, also known as public-key algorithms, use a pair of keys, one public and one private. The public key encrypts the data, whereas the private key decrypts it. This means that the sender can use the recipient's public key to encrypt the message, and the recipient can use their private key to decrypt it. Some examples of asymmetric-key algorithms include **RSA**, **Diffie–Hellman**, and **ElGamal**.

  A common example of this technology is a document signing service such as **Docusign,** which enables a user to digitally sign an important document.

- **Symmetric-key algorithms**, also known as secret-key algorithms, use the same key for encryption and decryption. This means that the sender and recipient of a message must both have a copy of the same key to communicate securely. Some examples of symmetric-key algorithms include **AES**, **Data Encryption Standard(DES)**, and **Blowfish**.

Symmetric-key algorithms are generally faster and more efficient than asymmetric-key algorithms, but they require the sender and recipient to securely exchange a copy of the key in advance. Asymmetric-key algorithms do not require the key to be exchanged in advance, but they are generally slower and less efficient than symmetric-key algorithms.

Public key infrastructure

Secure communication can be achieved using PKI by combining encryption, digital certificates, and public and private keys. PKI is essentially a system used to securely exchange information over the Internet using a combination of public and private keys.

To ensure the authenticity of the keys and to prevent man-in-the-middle attacks (see below), PKI also involves using digital certificates, which are issued by a trusted third party known as a **certificate authority** (CA).

However, PKI is not infallible. One type of attack that can be used to exploit it is the **man-in-the-middle** attack. In a man-in-the-middle attack, an attacker intercepts communication between two parties and poses as one of the parties to gain access to sensitive information. This type of attack can have serious consequences, as it can compromise the confidentiality, integrity, and authenticity of the communication.

As we continue through this task, we will explore the various aspects of PKI and man-in-the-middle attacks, including how they work, how to detect and prevent them, and what organisations can do to protect themselves. By the end of the task, you should have a thorough understanding of these concepts and be able to apply them to real-world scenarios.

Authentication

Authentication verifies that a person or device is authentic, i.e., who or what it claims to be. In the context of PKI and man-in-the-middle attacks, authentication is critical because it allows users to verify the identity of the parties they are communicating with and ensures that their messages are not intercepted by an unauthorised party.

Several different methods can be used for authentication in a PKI system, including:

1. **Password-based authentication:** This involves using a username and password to verify the identity of a user.
2. **Multi-factor authentication (MFA):** This involves using an additional piece of information, such as a code sent to a cell/mobile phone, in addition to a password to verify the identity of a user.
3. **Public-key authentication:** This involves using a public and private key pair to verify the identity of a user. The private key is kept secret by the user, while the public key is made available to anyone who needs to verify the user's identity.

In a PKI system, the authenticity of a user's identity can also be verified through digital certificates, which are issued by a trusted third party, as mentioned earlier. A digital certificate contains information about the user's identity, as well as the public key that can be used to verify their identity.

The flow of trust within PKI

There are three key components of the trust model for PKI:

1. Certificate authority (CA)
2. Registration authority (RA)
3. Validation authority (VA)

The flow of trust within a PKI system typically works as follows:

1. An individual or organisation, known as the "sender", wants to use a digital certificate to establish their identity online.

2. The sender contacts an RA and provides identification documents and other information to prove their identity.
3. The RA verifies the identity of the subscriber and forwards the information to the CA.
4. The trusted CA verifies the identity of the subscriber and issues a **digital certificate** that contains the subscriber's name, public key, and other information.
5. The sender can now use the digital certificate to establish their identity online.
6. When a recipient wants to communicate with the sender or access a secure website that requires the use of a digital certificate, they use the sender's digital certificate to verify their identity.
7. The recipient contacts a VA and provides the digital certificate.
8. The VA checks the certificate's authenticity by validating its digital signature and confirming that it was issued by a trusted CA. This step is crucial to ensure that the certificate is valid and has not been tampered with.
9. If the VA confirms that the digital certificate is authentic and was issued by a trusted CA, it approves the use of the certificate. This approval allows the recipient to trust the sender's identity, ensuring secure and verified communication.

Have a look at the schematic below of the relationship between the components:

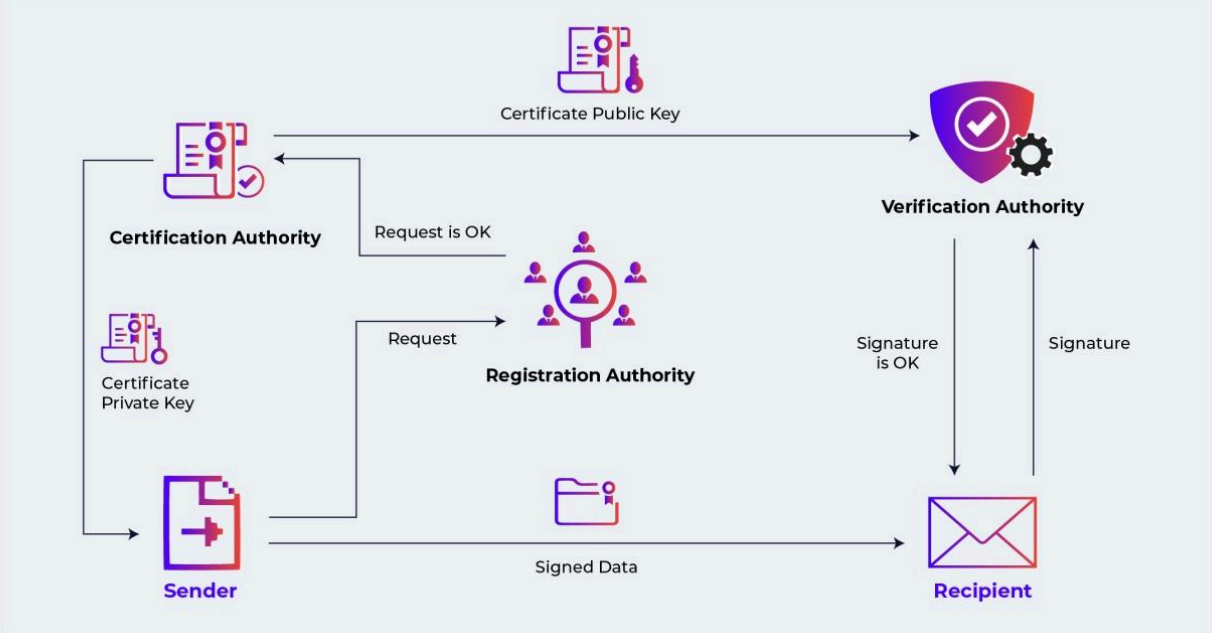

Diagram of the component relationships within PKI (AppViewX, n.d.)

Authorisation and access controls

Authorisation refers to the process of granting or denying access to specific resources or actions based on the privileges or permissions of the user or device. In the context of PKI and man-in-the-middle attacks, authorisation is important because it allows organisations to control who can access sensitive information and perform specific actions within the system.

Several different methods can be used for authorisation in a PKI system, including:

1. **Role-based access control (RBAC):** This involves assigning users to specific roles within the organisation and granting access to certain resources or actions based on their role. For example, a user assigned the role of "admin" might be able to create and delete user accounts, while a user with the role of "guest" might only have access to view specific files.
2. **Attribute-based access control (ABAC):** This involves granting or denying access based on specific attributes of the user or the resource being accessed. For example, an organisation might only grant access to a certain file to users who are members of a certain department or have a certain level of clearance.
3. **Access control list (ACL):** This involves specifying a list of users or groups allowed to access a certain resource or perform specific actions. An ACL can grant or deny access to specific resources or actions based on the user's identity or other attributes.

Vulnerabilities, exploits, and threat intelligence

A **vulnerability** is a weakness in a computer system or network that can be exploited by attackers to gain unauthorised access or perform other malicious actions. An **exploit** is a technique or piece of software or code used by attackers to take advantage of any vulnerability.

In the context of PKI and man-in-the-middle attacks, **vulnerabilities** and **exploits** can pose a serious threat to the security of a system. For example, suppose an attacker can find and exploit a vulnerability in a PKI system. In that case, they may be able to intercept communication between two parties and pose as one of the parties to gain access to sensitive information.

Here is a simple example of a man-in-the-middle attack involving Internet banking (containing very sensitive data):

1. Alice wants to log in to her online bank account.

2. The attacker, James, intercepts the communication between Alice and the bank's website.
3. James creates a fake login page that looks like the real bank's login page and sends it to Alice, which looks as if the genuine website's page is being loaded.
4. Alice enters her login information into the fake page, not realising that it is not the real bank's website.
5. James receives Alice's login information and uses it to log in to Alice's real bank account. (Can you see why most banks have adopted multi-factor authentication (MFA) techniques, such as sending a confirmation code to the account holder's cell/mobile phone, to perform certain actions?)

It is important for organisations to regularly identify and address vulnerabilities in their systems to reduce the risk of successful **exploitation**. This can be achieved through regular testing and security assessments, as well as by applying **patches** and **updates** as soon as they become available.

In addition to identifying and addressing vulnerabilities, it is also important for organisations to stay informed about "zero-day threats", which are vulnerabilities and exploits that may not have a fix yet. It may be necessary to implement an interim resolution to protect against the threat, e.g., isolating vulnerable servers until a patch has been released.

This can be done through **threat intelligence**, which involves collecting, analysing, and disseminating information about potential threats. By staying up to date on the latest threats, organisations can take proactive measures to protect themselves and reduce the risk of successful exploitation.

Man-in-the-middle cyber attacks

As previously mentioned, man-in-the-middle attacks are a type of cyber attack in which an attacker intercepts communication between two parties and poses as one of the parties to gain access to sensitive information or perform other malicious actions.

The leading cause of man-in-the-middle attacks is the lack of proper authentication and encryption in the communication between the two parties. When the parties do not verify each other's identity and the communication is not encrypted, an attacker can intercept the communication and pose as one of the parties without being detected. There are several ways that man-in-the-middle attacks can be prevented, including:

1. **Implementing robust authentication methods:** By using methods such as MFA or public-key authentication, organisations can verify the identity of the parties

involved in the communication and reduce the risk of successful man-in-the-middle attacks.

2. **Using encryption:** By encrypting the communication between the two parties, organisations can protect the confidentiality of the information and prevent an attacker from being able to read the original message.

3. **Using digital certificates:** By using digital certificates issued by a trusted CA, organisations can verify the authenticity of the keys used in the communication and prevent an attacker from being able to impersonate one of the parties.

4. **Implementing network security measures:** By implementing firewalls and intrusion-detection systems, organisations can monitor and prevent unauthorised access to their networks.

Internet and network security

We often need to communicate on open platforms such as the Internet but still require security and confidentiality. PKI and cryptography enable information to be exchanged on insecure platforms without compromising security or confidentiality. The following sections delve into key components and practices essential for maintaining robust Internet and network security. Understanding these elements is crucial for protecting sensitive information and ensuring safe communication online.

Certificate authority

As explained earlier, a CA is a trusted third party that issues digital certificates for secure communication over the Internet. Digital certificates are used to establish the identity of a website or individual and to ensure that the data transmitted between two parties is secure and cannot be intercepted by an unauthorised third party.

CAs operate by issuing digital certificates that contain a public key and the identity of the certificate owner. When a user's web browser connects to a website, the website presents its digital certificate to the browser. The browser then checks the certificate against a list of trusted CAs to verify that the certificate was issued by a trusted CA. If the certificate is valid, the browser establishes a secure connection with the website, allowing the two parties to communicate securely.

CAs play a crucial role in the security of Internet communication, as they act as a trusted intermediary between users and websites. By issuing digital certificates, CAs enable secure communication over the Internet and help to protect against attacks such as man-in-the-middle attacks. CAs are also responsible for revoking digital certificates when they are no longer valid or when the identity of the certificate owner can no longer be verified.

Digital signatures

A digital signature is a type of electronic signature used to authenticate the identity of the sender of a message or document, ensuring the integrity of the message or document. Digital signatures use public-key cryptography to create a secure, encrypted link between the sender of a message and the message itself.

Digital signatures help with cyber security in several ways:

1. **Authentication:** Digital signatures allow the recipient of a message or document to verify the sender's identity. This helps to prevent impersonation and other types of cyber attacks in which an attacker poses as someone else.

2. **Integrity:** Digital signatures help to ensure that the message or document has not been tampered with during transmission. If the message or document is modified in any way, the digital signature will be invalidated, alerting the recipient to the fact that the message or document has been compromised.

3. **Non-repudiation:** Digital signatures provide a way for the sender of a message or document to prove that they sent it. This can help to prevent disputes over whether or not the sender sent a particular message or document.

4. **Legal standing:** In some cases, digital signatures may be given legal standing, allowing them to be used in legal proceedings as evidence of the authenticity of a message or document.

SSL, TLS, and HTTPS

Secure Sockets Layer (SSL), Transport Layer Security (TLS), and Hypertext Transfer Protocol Secure (HTTPS) are all technologies that are used to establish secure, encrypted communication over the Internet. These technologies help to protect against man-in-the-middle attacks and other types of cyber attacks by encrypting the data transmitted between two parties.

At a high level, SSL and TLS are protocols to establish secure communication between two parties over the Internet. These protocols use a combination of public-key cryptography and symmetric-key cryptography to create a secure, encrypted link between the two parties.

HTTPS is a secure version of HTTP that is used to transmit data over the Internet. When a user's web browser connects to a website using HTTPS, the website presents a digital certificate to the browser. The browser then checks the certificate against a list of trusted CAs to verify that the certificate was issued by a trusted CA. If the certificate is valid, the browser establishes a secure connection with the website using SSL or TLS.

Network monitoring

Network monitoring can help prevent, mitigate, or detect a man-in-the-middle attack by providing visibility into network activity and alerting security personnel to unusual or suspicious activity.

There are several ways in which network monitoring can help detect, prevent, and mitigate a man-in-the-middle attack:

1. **Detection:** Network monitoring tools can continuously monitor network traffic for signs of a man-in-the-middle attack, such as unexpected traffic patterns or unauthorised connections. If a man-in-the-middle attack is detected, the network monitoring system can alert security personnel, allowing them to take appropriate action to mitigate the attack.

2. **Prevention:** Network monitoring tools can be configured to block or alert when suspicious traffic or connections occur, helping to prevent a man-in-the-middle attack from taking place.

3. **Mitigation:** If a man-in-the-middle attack is detected, network monitoring tools can provide the necessary data to help security personnel understand the scope and nature of the attack, allowing them to take appropriate steps to mitigate the attack.

Incident response

Incident response entails the identification, response, and management of the aftermath of a security incident or breach. Incident response is relevant for man-in-the-middle attacks because it provides a structured approach for responding to and mitigating the impact of an attack.

When an organisation experiences a man-in-the-middle attack, it is important to have a well-defined incident response plan (IRP) in place to ensure that the attack is responded to in a rapid, organised, and effective manner. The IRP should outline the steps that should be taken to contain the attack, restore normal operations, and mitigate any potential damage.

Some of the key steps that are typically included in an IRP for a man-in-the-middle attack include:

1. **Identification:** The first step in responding to a man-in-the-middle attack is to notice that an attack is taking place. This may involve monitoring for unusual traffic patterns or analysing network logs for signs of an attack.

2. **Containment:** Once the attack has been identified, the next step is to contain it to prevent it from spreading further. This may involve shutting down affected systems or disconnecting affected devices from the network.

3. **Analysis:** Once the attack has been contained, the next step is to analyse it to understand its scope and nature. This may involve reviewing network logs, analysing traffic patterns, or examining system configurations.

4. **Recovery:** After the attack has been analysed, the next step is to recover from the attack by restoring normal operations and repairing any damage that may have been caused.

5. **Mitigation:** Finally, it is important to mitigate the risk of future attacks by implementing stronger security controls and improving incident response processes.

Overall, incident response is an important part of managing a man-in-the-middle attack and minimising its impact on an organisation.

Web server hardening and testing

A web server is a server that stores and hosts websites. When a user enters a URL into a web browser, the browser retrieves the corresponding IP address, sends an HTTP request to the web server, and the server responds with the requested page or an error message if the page is unavailable. This process is known as the request-response model between a client and a server.

A web server is essentially a computer system running an operating system and software (such as an HTTP server) that is responsible for processing, storing, and serving website content to users. The web server can be linked to a database to run multiple applications, enabling dynamic content generation and complex interactions with users.

Static vs. dynamic web servers

A web server can be categorised as either static or dynamic:

- **Static web servers**: A static web server serves pre-rendered, unchanging content like HTML files, images, and other static resources. The content is delivered exactly as stored on the server, without any modification or generation at the time of request. Although **NGINX** can serve both static and dynamic content, it is often used to serve static files efficiently.

- **Dynamic web servers**: A dynamic web server generates content on-the-fly, often using application logic and database interactions. This allows for more interactive

and personalised web experiences. A dynamic web server typically includes an application server (such as Apache, NGINX with a back-end like PHP, or Python frameworks) that processes requests, generates HTML or other content dynamically, and sends it to the user's browser. Apache is a popular example of a web server capable of serving dynamic content.

Understanding the types of web servers is important, but securing them is paramount. Hardening web servers through measures like strong authentication, regular software updates, and secure configurations helps prevent unauthorised access and malicious attacks, such as man-in-the-middle (MitM) attacks. Additionally, implementing PKI with valid SSL/TLS certificates ensures encrypted communication and server identity verification, further mitigating MitM risks.

The importance of web servers

An essential component of a vulnerability management program is hardening web servers and guaranteeing server security. Attackers may use web server vulnerabilities to their advantage in order to access the systems that host web servers and carry out unauthorised operations.

Web server hardening involves:

- Modifying the configuration file to eliminate server misconfigurations.
- Managing SSL/TLS certificates and settings to ensure secure communication between the client and server.
- Restricting access permissions to the web server installation directory.
- Using an Intrusion Detection System (IDS) and a firewall with regularly updated signatures enhances security measures. The IDS monitors network or system activities for suspicious behaviour, while the firewall uses these updated signatures to effectively filter incoming and outgoing traffic, safeguarding against potential threats.

Apache

Apache HTTP Server is a free and open-source web server that delivers web content through the Internet. This means that the code is freely available for collaboration, and as mentioned before, it's a dynamic web server. Using the **Transmission Control Protocol/Internet Protocol** (TCP/IP), Apache serves as a means of client-server network communication. Although Apache is capable of handling many different protocols, HTTPS is the most widely used and is the one for which Apache is most well-known.

However, this very openness also makes Apache a target for attackers. By hardening the Apache configuration and following security best practices, we can significantly

reduce the risk of successful attacks and protect the websites hosted on the server. This includes steps like disabling unused modules, using strong passwords, and keeping the software up to date.

Apache modules

Apache is popular with developers who create and configure their own modules to add new features. Server modules are used to extend the functionality of the Apache web server. These modules are collections of libraries that enhance the server's capabilities, security, and performance. The Apache server operates on a modular architecture, meaning it starts with only essential features. Additional capabilities, such as supporting different programming languages or handling specific protocols like HTTPS or PHP, can be added by loading modules into Apache.

During the setup of Apache, it comes with a foundational set of modules already included to handle basic functions. If administrators need more advanced features, they can load additional modules using the LoadModule directive in the configuration files. This flexibility allows for dynamic loading of modules, meaning they can be added or removed without recompiling the server. However, if the server is not configured to load modules dynamically, adding new modules would require recompiling Apache from its original source code, which is the human-readable form of the software before it is compiled into an executable form.

An important aspect of configuring Apache is its ability to conditionally apply settings based on the presence of specific modules. This is done by placing configuration settings inside conditional blocks that check if particular modules are loaded. This allows administrators to customise Apache's behaviour and features according to the installed modules, ensuring a tailored and efficient server setup.

The `-l` command-line option can be used to view the modules that are currently compiled into the Apache server.

Configuration file

The Apache server is configured via configuration files, which control its behaviour using various modules. By default, Apache responds to requests based on the IP addresses and settings specified in its configuration files. The main configuration file is typically named httpd.conf, and its location is determined during the server's compile time but can be overridden using the -f command-line flag. Apache is primarily configured through **directives** written in plain text within the configuration file. These directives specify server operation parameters and client handling rules. Note that directives placed in the main configuration file apply to the entire server, but specific settings for particular sections can be included in separate configuration blocks.

Key directives and their functions:

- `ServerRoot` specifies where the subdirectories **conf** and **logs** can be found. This directive must be included when starting Apache with the `-f` (file) option. However, you do not need this directive if you use the `-d` (directory) option.
- `ServerName` directive is used to define or set the name of your domain name or subdomains.
- `Listen` is the directive used to define the port Apache listens on and also any specific IP address to listen on.
- `LoadModule` is the directive used to inform the Apache server of a module to be loaded.
- `KeepAlive` is used to prevent clients from consuming too much of the web server's resources.
- `ServerRoot` directive sets the directory in which all the web server's global configurations are defined and set.
- `DocumentRoot` directive specifies where Apache2 should look for the files that are requested by the clients and the files that should be rendered and processed by the web server.

To edit the Apache configuration file, use a text editor with root privileges. For example, on Linux, you can use:

```
sudo nano /etc/apache2/apache2.conf
```

or

```
sudo vi /etc/apache2/apache2.conf
```

Lines beginning with `#` are comments and do not affect the server's operation. Comments are used to explain the configuration settings. To enable a directive, remove the `#` at the beginning of the line.

Syntax

In Apache configuration files, each directive occupies a single line, ensuring clarity and organisation. Directives can extend across multiple lines by using a backslash (`'\'`) at the end of a line, indicating that the directive continues on the next line without any intervening characters or whitespace.

While directives themselves are case-insensitive – meaning `Listen`, `LISTEN`, and `listen` are treated the same – the arguments they receive are often case-sensitive. For

example, specifying file paths or module names must match exactly as expected by the server or module.

It is crucial to note that any line following a directive cannot contain comments. This ensures that configurations remain clear and unambiguous to Apache's parser.

To enhance readability and organisation, directives can be indented. Apache ignores any leading blank lines or whitespace before a directive, allowing administrators to structure configuration files in a visually appealing and logically structured manner.

Understanding these nuances ensures that Apache configuration files are correctly formatted and interpreted by the server, maintaining reliable and efficient operation.

Apache web server hardening

One essential component of web-based applications is the web server. Due to its frequent placement at the network's edge, the Apache web server is among the most attack-prone services. A hacker may be able to better prepare for an application attack by using default settings, which provide a lot of sensitive information. The lack of security measures leaves web servers vulnerable to various threats and attacks. One aspect of this is that it’s relatively easy for cybercriminals to launch man-in-the-middle attacks, intercepting HTTP requests and responses between the web browser and server, allowing the bad actor to modify the content, inject malicious scripts, or steal sensitive information without the user’s knowledge.

Apache web server hardening techniques include enabling HTTPS encryption, installing firewalls, keeping Apache updated, using a web application firewall, limiting access to sensitive directories, using security headers, and disabling unused network services and modules.

OpenSSL

SSL is an encryption-based Internet security protocol. It was initially created to guarantee data integrity, privacy, and authentication in online conversations by ensuring data is not transmitted in plain text. The TLS encryption that is currently in use was modelled after SSL. The URL of a website that uses SSL/TLS begins with “https” instead of “http.”

OpenSSL contains an open-source implementation of the SSL and TLS protocols. OpenSSL is an open-source command line tool that is commonly used to generate private keys, create certificate signing requests, install SSL/TLS certificates, and identify certificate information. It is a software library that is used by applications to enable secure communications over networks while preventing eavesdropping and identifying the person on the other end. Internet servers, including the vast majority of HTTPS websites, use it extensively. It can be installed on Windows, Linux, and macOS.

Within a PK), OpenSSL plays a crucial role by providing tools for certificate management, essential for defending against man-in-the-middle attacks. It also strengthens web server security by facilitating secure communication (HTTPS) and authenticating server identities.

How it works

There are a few required steps that apply to all servers and email clients in order to install an SSL certificate on your website. In situations where you do not have an online control panel or want to streamline the entire procedure, OpenSSL is very useful. Since it works on different operating systems, the steps may slightly differ, but the **basic steps** that apply to all operating systems include:

1. Installation: Install OpenSSL on your system if it's not already installed. If already installed, check the version to confirm.
2. Generate RSA key pair: This will create a private key.
3. Creating Certificate Signing Request (CSR): Using the private key from the previous step, your CSR will include details you want in your certificate such as domain name and common name (CN).
4. Creating a self-signed certificate: This is optional and is a certificate not signed by a CA but by its own private key. The certificate generated is typically X.509, and it contains an identity and a public key. It binds an identity (such as an individual or hostname) to a public key with a digital signature.
5. Certificate signing: If you have access to a CA you can send the CSR file created in step 3.
6. Checking certificate details: This displays information such as the issuer, validity period, public key, etc. using a specific command.
7. Creating digital signatures: OpenSSL can create and verify digital signatures using public-key cryptography algorithms like RSA.
8. Testing SSL/TLS connections: This can be done once you have configured the certificates to test the HTTP/HTTPS connections.

OpenSSL usage

The OpenSSL command line tool is used as a testing and analysis tool for cryptography. As such, it's applicable to:

- Establishment of crucial parameters

- Creation of X.509 certificates, CSRs, and Certificate Revocation Lists(CRLs)
- Message digest computation (hashing)
- Encryption and decryption using various algorithms including RSA, AES, and DES.
- Testing of clients and servers, including SSL, TLS, and Datagram Transport Layer Security(DTLS) protocols..
- Assessments of Quick UDP Internet Connections (QUIC) for customers
- Handling of encrypted or signed S/MIME mail

In order to make configurations and edits you would need **httpd** config files, as mentioned in the previous section. These files are:

- **httpd.conf**: This is the main Apache HTTP server configuration file and contains configuration directives that give the server its instructions. You get this file once you have installed Apache. Typically found in the **conf** directory of your Apache installation (**C:\Apache24\conf\httpd.conf** on Windows or **/etc/apache2/httpd.conf** on some Unix-based systems).
- **httpd-ssl.conf**: This is the Apache server configuration file providing SSL support, which contains configuration directives to instruct the server how to serve pages over an HTTPS connection.
- **apache2.conf**: This is the main configuration file on Debian-based systems like Ubuntu that is similar to **httpd.conf** on other systems. Typically found in **/etc/apache2/apache2.conf** on the Linux OS.
- **sites-available.conf and sites-enabled.conf**: These provide configurations for virtual hosts.
- **“extra” directory**: Contains additional configuration files for specific purposes, such as **httpd-ssl.conf** for SSL settings, **httpd-vhosts.conf** for virtual hosts, and **httpd-default.conf** for default settings. This is found in **C:\Apache24\conf\extra** in the Windows OS.

Key configuration directives

To effectively set up and manage your Apache web server, it's essential to understand key configuration directives that govern its behaviour and functionality. These directives allow you to customise various aspects of Apache's operation to suit your specific needs

- `ServerRoot`**:** Specifies the directory where the server's configuration, error, and log files are located.

- `DocumentRoot:` Defines the directory from which Apache will serve files.
- `Listen:` Configures the port that Apache will listen on for incoming requests.
- `ServerName:` Specifies the hostname and port number of the server.
- `LoadModule:` Loads a specified module into the server at runtime.
- `Include`: Includes other configuration files, allowing modular configurations.
- `<Directory>:` Applies directives to a particular directory and its contents.
- `<VirtualHost>:` Configures settings for virtual hosts, allowing multiple websites to be hosted on a single server.

Commands

OpenSSL is a command line utility that has many applicable **commands**. Some common commands used to operate OpenSSL are:

- `openssl version`: To get the current version of OpenSSL currently installed.
- `openssl genrsa -out private.key 2048`: To generate a private key using 2048 bits and the RSA algorithm.
- `openssl req -new -key private.key -out csr.pem`: To create a CSR.
- `openssl req -x509 -days 365 -key private.key -in csr.pem -out certificate.crt`: To create a self-signed certificate that expires in 365 days.
- `openssl x509 -in certificate.crt -text -noout`: To check certificate details.

Note that the commands above are explained in detail in the OpenSSL toolkit installation guide. There are also some Apache commands required to restart the service after every change is made to the configuration file. This is important so that the changes can be loaded and take effect. These commands include:

- `httpd -k start` (Windows) and `sudo systemctl start apache2/httpd` (Linux): To start the Apache service.
- `httpd -k stop` (Windows) and `sudo systemctl stop apache2/httpd` (Linux): To stop the service.
- `httpd -k restart` (Windows) and `sudo systemctl restart apache2/httpd` (Linux): To restart the service once changes are made.
- `sudo systemctl status apache2/httpd`: To check if the service is running or not.
- `sudo systemctl reload apache2/httpd`: To reload the service.

- `sudo apachectl configtest`: To run a configuration test for syntax errors in case you get an error.

Apache setup instructions

Installing, configuring, and using Apache differs depending on the operating system you are running. In your folder for this task, you will find walkthrough instructions for three different operating systems: Windows, macOS, and Ubuntu Linux. Access the guide that aligns with your operating system and work through it before attempting the task.

Practical task

In this task, you will put into practice what you have learned so far about web server hardening. First, create a Google doc called **server_hardening**; you will enter all your answers into this document and submit it for marking at the end of the task.

1. Follow the instructions for your specific operating system:
    - **Windows:** Install the Apache service on your PC. Confirm that it's running, and use your Edge browser to browse to it via the HTTP protocol. Take a screenshot and add it to the **server_hardening** file you just created.
    - **macOS:** Confirm the Apache web server on your Mac is running by browsing to it using the HTTP protocol. Take a screenshot and add it to the **server_hardening** file you just created.
    - **Ubuntu Linux:** Install the Chrome browser on Ubuntu if you do not already have it installed (if this is not possible based on your system architecture – i.e., your system uses ARM architecture – install the Chromium browser instead). Then, install the Apache server on Ubuntu. Confirm that it's running by browsing it via the HTTP protocol. Take a screenshot and add it to the **server_hardening** file you just created.
2. Create a CA. Paste the content of this file into the **server_hardening** Google doc.
3. Secure the Apache server. Paste the content of this file into the **server_hardening** Google doc.
4. Browse to your Apache web server using the browser that is appropriate for your operating system (Windows: Edge; Mac: Safari; Linux: Chrome or Chromium) and using an SSL connection.

- Do you receive a warning message? If so, take a screenshot and paste it into the **server_hardening** Google doc.

5. Follow the instructions for your specific operating system:

    - **Windows:** Install the self-signed certificate into your Microsoft Certificate Manager's "Trusted Root Certificate Authorities" folder. Bring up another Edge web browser instance and navigate to the Apache web server using an SSL connection. The warning should be gone. Take a snapshot of what you see in your browser and paste it into the **server_hardening** Google doc.

    - **macOS:** Install the self-signed certificate into your Mac's Keychain. Bring up another Safari web browser instance and navigate to the Apache web server using an SSL connection. The warning should be gone. Take a snapshot of what you see in your browser and paste it into the **server_hardening** Google doc.

    - **Ubuntu Linux:** Add Apache's web certificate to the Chrome or Chromium database and then browse to your Apache web server's SSL service by using an SSL connection. The warning should be gone. Take a screenshot and place it in your **server_hardening** Google doc.

6. Analyse the certificate and answer the following in your **server_hardening** Google doc.

    - Who issued the certificate?

    - When does the certificate expire?

Save your Google doc as a PDF and be sure to place files for submission inside your task folder and click "Request review" on your dashboard.

Share your thoughts

HyperionDev strives to provide internationally excellent course content that helps you achieve your learning outcomes.

Do you think we've done a good job or do you think the content of this task, or this course as a whole, can be improved?

Share your thoughts anonymously using this **form**.

Reference list

AppViewX. (n.d.). *What is public key infrastructure (PKI)?*
https://www.appviewx.com/education-center/pki/