Session 2:

Threats and OWASP Principles & Introduction to Secure design

Defense in Depth

Also known as layered defense, defense in depth is a security principle where single points of complete compromise are eliminated or mitigated by the incorporation of a series or multiple layers of security safeguards and risk-mitigation countermeasures.

Have diverse defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense will hopefully prevent a full breach.

Fail Safe

A security principle that aims to maintaining confidentiality, integrity and availability by defaulting to a secure state, rapid recovery of software resiliency upon design or implementation failure. In the context of software security, fail secure is commonly used interchangeably with fail safe, which comes from physical security terminology.

Unless a subject is given explicit access to an object, it should be denied access to that object, aka Fail-Safe Defaults.

Least Privilege

A security principle in which a person or process is given only the minimum level of access rights (privileges) that is necessary for that person or process to complete an assigned operation. This right must be given only for a minimum amount of time that is necessary to complete the operation.

Limits the damage in case of exploited vulnerability.

In order to apply this principle, proper granularity of privileges and permissions should be established.

Separation of Duties

Also known as the compartmentalization principle, or separation of privilege, separation of duties is a security principle which states that the successful completion of a single task is dependent upon two or more conditions that need to be met and just one of the conditions will be insufficient in completing the task by itself.

Economy of Mechanism

This in layman terms is the Keep It Simple, Stupid principle because the likelihood of a greater number of vulnerabilities increases with the complexity of the software architectural design and code.

By keeping the software design and implementation details simple, the attack-ability or attack surface of the software is reduced.

Complete Mediation

A security principle that ensures that authority is not circumvented in subsequent requests of an object by a subject, by checking for authorization (rights and privileges) upon every request for the object.

In other words, the access requests by a subject for an object is completely mediated every time.

"All accesses to objects must be checked to ensure that they are allowed."

Performance v/s Security issue:

- Results of access check are often cached
- What if permissions have changed since the last check?
- Mechanisms to invalidate or flush caches after a change are often missing

Open Design

The open design security principle states that the implementation details of the design should be independent of the design itself, which can remain open, unlike in the case of security by obscurity wherein the security of the software is dependent upon the obscuring of the design itself.

When software is architected using the open design concept, the review of the design itself will not result in the compromise of the safeguards in the software.

"The security of a mechanism should not depend on the secrecy of its design or implementation."

If the details of the mechanism leaks then it is a catastrophic failure for all the users at once.

If the secrets are abstracted from the mechanism, e.g. inside a key, then leakage of a key affects only one user.

Least Common Mechanism

The security principle of least common mechanisms disallows the sharing of mechanisms that are common to more than one user or process if the users and processes are at different levels of privilege. For example, the use of the same function to retrieve the bonus amount of an exempt employee and a non-exempt employee will not be allowed. In this case the calculation of the bonus is the common mechanism.

Psychological acceptability

A security principle that aims at maximizing the usage and adoption of the security functionality in the software by ensuring that the security functionality is easy to use and at the same time transparent to the user. Ease of use and transparency are essential requirements for this security principle to be effective.

Security mechanisms should not make the resource more difficult to access than if the security mechanism were not present.

Problem: Users looks for ways to defeat the mechanisms and "prop the doors open".

Weakest Link

This security principle states that the resiliency of your software against hacker attempts will depend heavily on the protection of its weakest components, be it the code, service or an interface.

Leveraging Existing Components

This is a security principle that focuses on ensuring that the attack surface is not increased and no new vulnerabilities are introduced by promoting the reuse of existing software components, code and functionality.

Session 3

Web server: introduction a secure setup of apache

Introduction

A Web server is a server that is responsible for accepting HTTP requests from web clients and serving them HTTP responses, usually in the form of web pages containing static (text, images etc) and dynamic (scripts) content. The Apache Web server has been the most popular and widely used Web server for the last decade. It is used by approximately 50% of all websites. Apache is cross-platform, lightweight, robust, and used in small companies as well as large corporations. Apache is also free and open-source. The Apache Web server has almost endless possibilities, due to its great modularity, which allows it to be integrated with numerous other applications. One of the most popular bundles is the LAMP Web server application stack, which includes the Apache Web server alongside MySQL, PHP, Perl, and Python.

Being able to configure and secure the Apache Web server is one of the most important tasks for a (Linux) system administrator. Almost every company has some sort of a website that advertises it, including intranet pages that are used by the company's workers. The Web interface is used for many tasks beside pure browsing, including tasks as simple as meal orders and shift rosters, but also important tasks like administration of databases. In most cases, a local web server is setup to accommodate these needs. If you are working for a company that hosts public websites, the task becomes even more complicated. Web sites are used to serve content to billions of users daily. Whoever controls this content - controls the World Wide Web, from news and blogs to financial transactions. Web servers are hubs of information and power. Misconfigured or compromised

servers can expose a large number of people to undesired content and potentially incur huge damages to involved parties. Running a Web site is much more than opening a port and serving a few HTML pages. There are tremendous network usability and security considerations that must continuously be met, evaluated and improved in order to maintain a safe and effective Web server. In this Part of the Book, we will learn how to properly setup and run the Apache Web server, including the secure (HTTPS) server.

What is Apache Web Server?

Apache HTTP Server is a free and open-source web server that delivers web content through the internet. It is commonly referred to as Apache and after development, it quickly became the most popular HTTP client on the web. It's widely thought that Apache gets its name from its development history and process of improvement through applied patches and modules but that was corrected back in 2000. It was revealed that the name originated from the respect of the Native American tribe for its resiliency and durability.

Now, before we get too in depth on Apache, we should first go over what a web application is and the standard architecture usually found in web apps.

Web Application Architecture

Apache is just one component that is needed in a web application stack to deliver web content. One of the most common web application stacks involves LAMP, or Linux, Apache, MySQL, and PHP.

Linux is the operating system that handles the operations of the application. Apache is the web server that processes requests and serves web assets and content via HTTP. MySQL is the database that stores all your information in an easily queried format. PHP is the programming language that works with apache to help create dynamic web content.

While actual statistics may vary, it's fair to say a large portion of web applications run on some form of the LAMP stack because it is easy to build and also free to use. For the most part, web applications tend to generally have similar architecture and structure even though they serve many different functions and purposes. Most web applications also benefit from Firewalls, Load Balancers, Web Servers, Content Delivery Networks, and Database Servers.

Firewalls help protect the web application from both external threats and internal vulnerabilities depending on where the firewalls are configured. Load Balancers help distribute traffic across the web servers which handle the HTTP(S) requests (this is where Apache comes in) and application servers (servers that handle the functionality and workload of the web app.) We also have Database Servers, which handle asset storage and backups. Depending on your infrastructure, your database and application can both live on the same server although it's recommended to keep those separate.

Web Server Landscape

The internet is comprised of many different technologies and not all of them are the same. While Apache is arguably one of the most popular web servers out there on the net, there are many other players and the landscape is always changing. Back in the late 90s and early 2000s, Apache's dominance was very strong, serving over 50% of the internet's active websites. Microsoft's IIS (Internet Information Services) was also an option but not nearly as popular.

Today, Apache still serves a large portion of the active websites but their share of the field has shrunk from 50% to just under 40% as of 2018 and NGINX, a relatively new player to the web server playing field, is in second place with roughly 35% and Microsoft IIS hovering around 8-10%. Every year there's a new crop of web applications with new stacks and servers so the landscape is always changing.

Why Apache?

Apache is considered open source software, which means the original source code is freely available for viewing and collaboration. Being open source has made Apache very popular with developers who have built and configured their own modules to apply specific functionality and improve on its core features. Apache has been around since 1995 and is responsible as a core technology that helped spur the initial growth of the internet in its infancy.

One of the pros of Apache is its ability to handle large amounts of traffic with minimal configuration. It scales with ease and with its modular functionality at its core, you can configure Apache to do what you want, how you want it. You can also remove unwanted modules to make Apache more lightweight and efficient.

Some of the most popular modules that can be added are SSL, Server-Side Programming Support (PHP), and Load Balancing configs to handle large amounts of traffic. Apache can also be deployed on Linux, MacOS, and Windows. If you learn how to configure Apache on Linux, you can administer Apache on Windows and Mac. The only difference would be directory paths and installation processes.

How does Apache Work?

Apache functions as a way to communicate over networks from client to server using the TCP/IP protocol. Apache can be used for a wide variety of protocols, but the most common is HTTP/S. HTTP/S or Hyper Text Transfer Protocol (S stands for Secure) is one of the main protocols on the web, and the one protocol Apache is most known for.

HTTP/S is used to define how messages are formatted and transmitted across the web, with instructions for browsers and servers on how to respond to various requests and commands. Hypertext Transfer Protocol Secure is usually through port 443 with the unsecured protocol being through port 80.

The Apache server is configured via config files in which modules are used to control its behavior. By default, Apache listens to the IP addresses configured in its config files that are being requested. This is where one of Apaches many strengths come into play.

With the Listen directive, Apache can accept and route specific traffic to certain ports and domains based on specific address-port combination requests. By default, Listen runs on port 80 but Apache can be bound to different ports for different domains, allowing for many different websites and domains to be hosted and a single server. You can have domain1.com listening on port 80, domain2.com on port 8080 and domain3.com on port 443 using HTTPS all on Apache.

Once a message reaches its destination or recipient, it sends a notice, or ACK message, basically giving acknowledgment to the original sender that their data has successfully arrived. If there's an error in receiving data, or some packets were lost in transit, the destination host or client sends a Not Acknowledged, or NAK message, to inform the sender that the data needs to be retransmitted.

Basic Setup

In this chapter, we will setup a Web server that will serve pages on our internal network. In this chapter, we will perform the most basic setup with the minimum number of steps required to get the server running. Later, we will slowly expand, introducing new features and options.

Verify installation

First, we'll verify that Apache is indeed installed: rpm -q httpd

If you get an empty prompt or a message saying the package is not installed, you will need to download and install it. If the shell displays the package name and version, you're good to go.

Package files

Rule no. 1: don't panic! The list before you might seem intimidating at the moment, but that is simply because you are not yet familiar with Apache. But don't worry. For now, treat the list as a reference only. At this stage, you don't need to know anything or remember anything. We will slowly cover everything, step by step. Now, let us overview the location and purpose of the files used by the Apache server. Please note that the list is partial and includes only the most important entries. We will slowly expand this list as we go through the Part.

File name	Description
/usr/sbin/httpd	server binary
/etc/httpd	directory containing server configuration files
/etc/httpd/conf	directory containing main configuration files
/etc/httpd/conf.d	directory containing configuration files for individually packaged modules, like ssl, php, perl etc
/etc/httpd/logs	symbolic link to /var/log/httpd
/etc/httpd/modules	symbolic link to /usr/lib/httpd/modules
/etc/httpd/run	symbolic link to /var/run
/usr/lib/httpd/modules	server modules
/var/log/httpd	server log
/var/run	runtime variables
/var/run/httpd.pid	server process ID
/var/www/html	public html files

Conclusion

Throughout the last few decades, Apache has proven to be a staple in many popular stacks and the backbone of the early internet year. While its popularity is declining and the options of web server choices are increasing, Apache still plays a pivotal role in many technology stacks and companies system infrastructure. Even with new technologies and servers coming out nonstop, Apache is still a technology every developer should learn how to handle and configure.