**CIS Apache HTTP Server 2.4 Benchmark: Hardening Recommendations**

* Minimize Apache Modules: It is very critical to have minimal Apache installation based on business requirements.
* Enable the Log Config Module: Logging is critical to monitor usage and potential abuse.
* Disable WebDAV Modules: Disabling the WebDAV modules will increase the security posture pf the web server. This is done by reducing the amount of potentially vulnerable code paths.
* Disable Status Module: Reduces the amount of info that can be gained by the mod\_status.
* Disable Autoindex Module: Reduces the information the module will give to a potential attacker.
* Disable Proxy Modules: A web server should not also be a proxy server this can lead to an attack therefore we will disable the proxy server on our webserver.
* Disable User Directories Modules: Limits the amount of data available to people on the internet.
* Disable Info Module: Having this module enabled can leak sensitive information.
* Run Apache Web Server as a non-root user: By default, Apache is configured to run with nobody or daemon. Don’t set User (or Group) to root unless you know exactly what you are doing, and what the dangers are.
* Give the Apache User Account an Invalid Shell: Service accounts like apache account represent a risk if they can be used to get a login in shell to the system.
* Lock the Apache User Account: This is a defense in depth measure, the Apache user account should be locked to prevent logins, use sudo instead.
* Set Ownership and Group ID on Apache Directories and Files: Restricting and securing these files and directories reduces the risk of unauthorized modifications.
* Restrict Other Write Access on Apache Directories and Files: Without restriction, other write access allows access for modifications as part of malicious attacks.
* Secure Core Dump Directory: Core dumps can contain sensitive information, so access should be restricted.
* Secure the Lock, Pid, and ScoreBoard Files: Without being secured, a denial of service attack could prevent the service from starting by creating a file with the same name.
* Restrict Group Write Access for the Apache and Document Root Directories and Files: Prevent unauthorized modifications associated with file uploads or command execution.
* Deny Access to OS Root Directory: Addresses potential for default access to be made available to clients unintentionally.
* Restrict Override for the OS Root Directory and All Directories: Reduces risk of changes by unintended sources.
* Restrict Options for the OS Root Directory and Web Root Directory, and minimize options for Other Directories: Options should be restricted to the minimal level, and a directive of None is recommended.
* Remove Default HTML Content: Historically exploited for different levels of access to the server.
* Remove Default CGI Content printenv and test-cgi: Known for improperly accepting user-input.
* Limit HTTP Request Methods: Reduces unauthorized modification of resources on the server.
* Disable HTTP TRACE Method: Unnecessary and easily abused.
* Restrict HTTP Protocol Versions: Increases ability to filter and deny malicious attacks.
* Restrict Access to .ht\* files: Usage of default access files should not be allowed, and a FileMatch directive is recommended to hide files created in error from web clients.
* Restrict File Extensions: Establish a list of trusted file extensions, and only allow access to those.
* Deny IP Address Based Requests: Blocks potential malware and network scanning.
* Restrict Listen Directive and Browser Frame Options: Only listen to appropriate IP Addresses/interfaces, and load frames with the same origin as the associated page.
* Configure the Error Log, Syslog Facility for Error Logging, and Access Log: Crucial to monitoring for anomalous behavior, server errors, and resource usage.
* Log Storage and Rotation: Do not store on the root partition and use log rotation utilities.
* Apply Applicable Patches: New fixes and security patches are added in every release. Always upgrade to the latest stable version of Apache.
* Install and Enable ModSecurity and OWASP ModSecurity Core Rule Set: Enables a customizable web application firewall rule set ad additional baseline security defense.
* Install mod\_ssl and/or mod\_nss: Allows for planning for SSL/TLS from the beginning.
* Install a Valid Trusted Certificate: Automatically communicates your site’s authenticity.
* Protect the Server's Private Key: Used to decrypt all SSL communications.
* Restrict Weak and Medium Strength SSL/TLS Ciphers: Allow only strong ciphers, so you close all the doors who try to handshake on lower cipher suites.
* Disable the TLS v1.0, SSL v2.0 & v3.0 Protocol, SSL Insecure Renegotiation, and ensure SSL Compression is not Enabled: SSL 2.0 & 3.0, reportedly suffers from several cryptographic flaws.
* Enable OCSP Stapling: Improves overhead of making an SSL connection by having the SSL server “staple” an OCSP response to the certificate.
* Enable HTTP Strict Transport Security: Protects from HTTP downgrade attacks.
* Set ServerToken to 'Prod': Limits data returned during scans for software installed on the host.
* Set ServerSignature to 'Off': The Off setting, which is the default, suppresses the footer line. The On setting simply adds a line with the server version number and ServerName of the serving virtual host.
* Information Leakage via Default Apache Content and ETag: Increases difficulty for hackers to obtain server and software versions.
* Set TimeOut to 10 or less: Reduces risk of DoS attacks caused by flooding with requests.
* Set the KeepAlive directive to On, MaxKeepAliveRequests to 100 or greater, and KeepAliveTimeout Low: Mitigates the risk of Denial of Service attacks and frees resources up more quickly.
* Set Timeout Limits for Request Headers and Body: Reduces risk of Denial of Service attacks resulting from slow requests.
* Set Request Limits for Line, Fields, Fieldsize, and Body: Placing limits on requests can prevent an unexpectedly large request from passing through, but limits must be high enough to account for any valid application requests.
* Enable SELinux in Enforcing Mode: Can enforce rules on files and processes in a Linux system.
* Run Apache Processes in the httpd\_t Confined Context and ensure the httpd\_t Type is Not in Permissive Mode: Permissive mode is helpful for testing but causes SELinux to allow all access.
* Enable the AppArmor Framework and ensure Apache AppArmor Profile is in Enforce Mod: Only the confined process running in enforce mode will prevent attacks.

**References**

CIS Apache HTTP Server 2.4 Benchmark. (2018, July 13). Retrieved December, 2018, from https://www.cisecurity.org/benchmark/apache\_http\_server/

https://geekflare.com/10-best-practices-to-secure-and-harden-your-apache-web-server/