CIS Center for Internet Security®

CIS Benchmarks™

CIS NGINX Benchmark

v3.0.0 - 12-24-2025

Terms of Use

Please see the below link for our current terms of use:

https://www.cisecurity.org/cis-securesuite/cis-securesuite-membership-terms-of-use/

For information on referencing and/or citing CIS Benchmarks in 3rd party documentation (including using portions of Benchmark Recommendations) please contact CIS Legal (legalnotices@cisecurity.org) and request guidance on copyright usage.

NOTE: It is ***NEVER*** acceptable to host a CIS Benchmark in ***ANY*** format (PDF, etc.) on a 3rd party (non-CIS owned) site.

Table of Contents

Overview

All CIS Benchmarks™ (Benchmarks) focus on technical configuration settings used to maintain and/or increase the security of the addressed technology, and they should be used in **conjunction** with other essential cyber hygiene tasks like:

- Monitoring the base operating system and applications for vulnerabilities and quickly updating with the latest security patches.
- End-point protection (Antivirus software, Endpoint Detection and Response (EDR), etc.).
- Logging and monitoring user and system activity.

In the end, the Benchmarks are designed to be a key **component** of a comprehensive cybersecurity program.

Important Usage Information

All Benchmarks are available free for non-commercial use from the CIS Website. They can be used to manually assess and remediate systems and applications. In lieu of manual assessment and remediation, there are several tools available to assist with assessment:

- CIS Configuration Assessment Tool (CIS-CAT® Pro Assessor)
- CIS Benchmarks™ Certified 3rd Party Tooling

These tools make the hardening process much more scalable for large numbers of systems and applications.

NOTE: Some tooling focuses only on the Benchmark Recommendations that can be fully automated (skipping ones marked **Manual**). It is important that ***ALL*** Recommendations (**Automated** and **Manual**) be addressed since all are important for properly securing systems and are typically in scope for audits.

Key Stakeholders

Cybersecurity is a collaborative effort, and cross functional cooperation is imperative within an organization to discuss, test, and deploy Benchmarks in an effective and efficient way. The Benchmarks are developed to be best practice configuration guidelines applicable to a wide range of use cases. In some organizations, exceptions to specific Recommendations will be needed, and this team should work to prioritize the problematic Recommendations based on several factors like risk, time, cost, and labor. These exceptions should be properly categorized and documented for auditing purposes.

Apply the Correct Version of a Benchmark

Benchmarks are developed and tested for a specific set of products and versions and applying an incorrect Benchmark to a system can cause the resulting pass/fail score to be incorrect. This is due to the assessment of settings that do not apply to the target systems. To assure the correct Benchmark is being assessed:

- **Deploy the Benchmark applicable to the way settings are managed in the environment:** An example of this is the Microsoft Windows family of Benchmarks, which have separate Benchmarks for Group Policy, Intune, and Stand-alone systems based upon how system management is deployed. Applying the wrong Benchmark in this case will give invalid results.

- **Use the most recent version of a Benchmark**: This is true for all Benchmarks, but especially true for cloud technologies. Cloud technologies change frequently and using an older version of a Benchmark may have invalid methods for auditing and remediation.

Exceptions

The guidance items in the Benchmarks are called recommendations and not requirements, and exceptions to some of them are expected and acceptable. The Benchmarks strive to be a secure baseline, or starting point, for a specific technology, with known issues identified during Benchmark development are documented in the Impact section of each Recommendation. In addition, organizational, system specific requirements, or local site policy may require changes as well, or an exception to a Recommendation or group of Recommendations (e.g. A Benchmark could Recommend that a Web server not be installed on the system, but if a system's primary purpose is to function as a Webserver, there should be a documented exception to this Recommendation for that specific server).

In the end, exceptions to some Benchmark Recommendations are common and acceptable, and should be handled as follows:

- The reasons for the exception should be reviewed cross-functionally and be well documented for audit purposes.
- A plan should be developed for mitigating, or eliminating, the exception in the future, if applicable.
- If the organization decides to accept the risk of this exception (not work toward mitigation or elimination), this should be documented for audit purposes.

It is the responsibility of the organization to determine their overall security policy, and which settings are applicable to their unique needs based on the overall risk profile for the organization.

Remediation

CIS has developed Build Kits for many technologies to assist in the automation of hardening systems. Build Kits are designed to correspond to Benchmark's "Remediation" section, which provides the manual remediation steps necessary to make that Recommendation compliant to the Benchmark.

When remediating systems (changing configuration settings on deployed systems as per the Benchmark's Recommendations), please approach this with caution and test thoroughly.

The following is a reasonable remediation approach to follow:

- CIS Build Kits, or internally developed remediation methods should never be applied to production systems without proper testing.
- Proper testing consists of the following:
  - Understand the configuration (including installed applications) of the targeted systems. Various parts of the organization may need different configurations (e.g., software developers vs standard office workers).
  - Read the Impact section of the given Recommendation to help determine if there might be an issue with the targeted systems.
  - Test the configuration changes with representative lab system(s). If issues arise during testing, they can be resolved prior to deploying to any production systems.
  - When testing is complete, initially deploy to a small sub-set of production systems and monitor closely for issues. If there are issues, they can be resolved prior to deploying more broadly.
  - When the initial deployment above is completes successfully, iteratively deploy to additional systems and monitor closely for issues. Repeat this process until the full deployment is complete.

Summary

Using the Benchmarks Certified tools, working as a team with key stakeholders, being selective with exceptions, and being careful with remediation deployment, it is possible to harden large numbers of deployed systems in a cost effective, efficient, and safe manner.

NOTE: As previously stated, the PDF versions of the CIS Benchmarks™ are available for free, non-commercial use on the CIS Website. All other formats of the CIS Benchmarks™ (MS Word, Excel, and Build Kits) are available for CIS SecureSuite® members.

CIS-CAT® Pro is also available to CIS SecureSuite® members.

Target Technology Details

This document, CIS NGINX Benchmark, provides prescriptive guidance for establishing a secure configuration posture for NGINX version 1.28.x running on a systemd based Linux system

This guide was tested against NGINX version 1.28.0 with default installation location, using the packages installed using the Red Hat Enterprise Linux 9 package managers.

This Benchmark was written using commands for and tested on, Red Hat Enterprise Linux release 9.0. For other versions of Linux, please substitute the distribution-specific commands for the equivalent commands on the Linux distribution you are using.

To obtain the latest version of this guide, please visit https://benchmarks.cisecurity.org. If you have questions or comments or have identified ways to improve this guide, please write to us at feedback@cisecurity.org.

Intended Audience

This document is intended for system and application administrators, security specialists, auditors, and help desk and platform deployment personnel who plan to develop, deploy, assess, or secure solutions that incorporate NGINX.

Consensus Guidance

This CIS Benchmark™ was created using a consensus review process comprised of a global community of subject matter experts. The process combines real world experience with data-based information to create technology specific guidance to assist users to secure their environments. Consensus participants provide perspective from a diverse set of backgrounds including consulting, software development, audit and compliance, security research, operations, government, and legal.

Each CIS Benchmark undergoes two phases of consensus review. The first phase occurs during initial Benchmark development. During this phase, subject matter experts convene to discuss, create, and test working drafts of the Benchmark. This discussion occurs until consensus has been reached on Benchmark recommendations. The second phase begins after the Benchmark has been published. During this phase, all feedback provided by the Internet community is reviewed by the consensus team for incorporation in the Benchmark. If you are interested in participating in the consensus process, please visit https://workbench.cisecurity.org/.

Typographical Conventions

The following typographical conventions are used throughout this guide:

Convention	Meaning
`Stylized Monospace font`	Used for blocks of code, command, and script examples. Text should be interpreted exactly as presented.
`Monospace font`	Used for inline code, commands, UI/Menu selections or examples. Text should be interpreted exactly as presented.
`<Monospace font in brackets>`	Text set in angle brackets denote a variable requiring substitution for a real value.
Italic font	Used to reference other relevant settings, CIS Benchmarks and/or Benchmark Communities. Also, used to denote the title of a book, article, or other publication.
Bold font	Additional information or caveats things like **Notes**, **Warnings**, or **Cautions** (usually just the word itself and the rest of the text normal).

Recommendation Definitions

The following defines the various components included in a CIS recommendation as applicable. If any of the components are not applicable it will be noted, or the component will not be included in the recommendation.

Title

Concise description for the recommendation's intended configuration.

Assessment Status

An assessment status is included for every recommendation. The assessment status indicates whether the given recommendation can be automated or requires manual steps to implement. Both statuses are equally important and are determined and supported as defined below:

Automated

Represents recommendations for which assessment of a technical control can be fully automated and validated to a pass/fail state. Recommendations will include the necessary information to implement automation.

Manual

Represents recommendations for which assessment of a technical control cannot be fully automated and requires all or some manual steps to validate that the configured state is set as expected. The expected state can vary depending on the environment.

Profile

A collection of recommendations for securing a technology or a supporting platform. Most benchmarks include at least a Level 1 and Level 2 Profile. Level 2 extends Level 1 recommendations and is not a standalone profile. The Profile Definitions section in the benchmark provides the definitions as they pertain to the recommendations included for the technology.

Description

Detailed information pertaining to the setting with which the recommendation is concerned. In some cases, the description will include the recommended value.

Rationale Statement

Detailed reasoning for the recommendation to provide the user a clear and concise understanding on the importance of the recommendation.

Impact Statement

Any security, functionality, or operational consequences that can result from following the recommendation.

Audit Procedure

Systematic instructions for determining if the target system complies with the recommendation.

Remediation Procedure

Systematic instructions for applying recommendations to the target system to bring it into compliance according to the recommendation.

Default Value

Default value for the given setting in this recommendation, if known. If not known, either not configured or not defined will be applied.

References

Additional documentation relative to the recommendation.

CIS Critical Security Controls® (CIS Controls®)

The mapping between a recommendation and the CIS Controls is organized by CIS Controls version, Safeguard, and Implementation Group (IG). The Benchmark in its entirety addresses the CIS Controls safeguards of (v7) “5.1 - Establish Secure Configurations” and (v8) '4.1 - Establish and Maintain a Secure Configuration Process” so individual recommendations will not be mapped to these safeguards.

Additional Information

Supplementary information that does not correspond to any other field but may be useful to the user.

Profile Definitions

The following configuration profiles are defined by this Benchmark:

- **Level 1 - Webserver**

  Items in this profile intend to:

  - be practical and prudent;
  - provide a clear security benefit; and
  - not inhibit the utility of the technology beyond acceptable means.

- **Level 1 - Proxy**

  Items in this profile intend to:

  - be practical and prudent;
  - provide a clear security benefit; and
  - not inhibit the utility of the technology beyond acceptable means.

- **Level 1 - Loadbalancer**

  Items in this profile intend to:

  - be practical and prudent;
  - provide a clear security benefit; and
  - not inhibit the utility of the technology beyond acceptable means.

- **Level 2 - Webserver**

  This profile extends the "Level 1" profile. Items in this profile exhibit one or more of the following characteristics:

  - are intended for environments or use cases where security is paramount
  - acts as defense in depth measure
  - may negatively inhibit the utility or performance of the technology

- **Level 2 - Proxy**

  This profile extends the "Level 1" profile. Items in this profile exhibit one or more of the following characteristics:

  - are intended for environments or use cases where security is paramount
  - acts as defense in depth measure

    - may negatively inhibit the utility or performance of the technology

- **Level 2 - Loadbalancer**

  This profile extends the "Level 1" profile. Items in this profile exhibit one or more of the following characteristics:

    - are intended for environments or use cases where security is paramount
    - acts as defense in depth measure
    - may negatively inhibit the utility or performance of the technology

Acknowledgements

This Benchmark exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this guide:

Contributor
Alexander Sennhauser
James Scott
Greg MacLean

Editor
Eric Pinnell
Krishna Rayavaram
Stephan Wenderlich

Recommendations

1 Initial Setup

This section contains recommendations for the installation and maintenance of an NGINX server.

1.1 Installation

1.1.1 Ensure NGINX is installed (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

An NGINX installation must be present on the system. To ensure support for modern security standards (such as TLS 1.3 and HTTP/3) and to mitigate known vulnerabilities, the installed version must be `1.28.0` or later and compiled with the necessary modules.

Rationale:

NGINX must be installed and operational to serve as the target for this benchmark's security controls. Enforcing a minimum version and feature set ensures the platform is capable of supporting the required security configurations.

Impact:

Upgrading NGINX to a newer version may introduce configuration syntax changes or deprecated directives. Administrators should test the configuration syntax `nginx -t` before restarting the service.

Audit:

1. Verify Installation and Version: Run the following command to display the installed NGINX version:

```
# Note: It's an uppercase V
nginx -V
```

2. Evaluation:

- **Verify Output:** The command must return an installed version (e.g., `nginx version: nginx/1.28.0`). If the command is not found, NGINX is not installed.
- **Check Version:** Ensure the version number is 1.28.0 or higher.

Remediation:

Install or upgrade NGINX to version `1.28.0` or later.

Note: Official packages from nginx.org (see recommendation 1.2.1) typically include these modules by default. Custom builds must explicitly enable them.

Default Value:

NGINX is not installed by default.

References:

1. https://nginx.org/en/docs/install.html
2. https://nginx.org/en/linux_packages.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.2 Ensure Authorized Software is Currently Supported Ensure that only currently supported software is designated as authorized in the software inventory for enterprise assets. If software is unsupported, yet necessary for the fulfillment of the enterprise's mission, document an exception detailing mitigating controls and residual risk acceptance. For any unsupported software without an exception documentation, designate as unauthorized. Review the software list to verify software support at least monthly, or more frequently.	●	●	●
v7	2.2 Ensure Software is Supported by Vendor Ensure that only software applications or operating systems currently supported by the software's vendor are added to the organization's authorized software inventory. Unsupported software should be tagged as unsupported in the inventory system.	●	●	●

1.2 Configure Software Updates

1.2.1 Ensure package manager repositories are properly configured (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Package repositories must be trustworthy, properly configured, and maintained to ensure the system receives timely security patches, bug fixes, and support for modern protocols. While Operating System (OS) vendors provide NGINX packages, these versions are often frozen at older release points ("stable" but stale). Access to critical modern features like HTTP/3 (QUIC) and the latest TLS updates typically requires using the official repositories maintained by NGINX/F5.

Rationale:

If a system's package manager repositories are misconfigured or outdated, critical security patches may not be applied in a timely manner. Furthermore, relying solely on default OS repositories often restricts the web server to legacy versions that lack support for modern security standards (e.g., HTTP/3). Using the official nginx.org repositories ensures access to the latest stable and mainline versions directly from the source, reducing the risk of running obsolete software. Conversely, adding untrusted third-party repositories can introduce compromised software or dependency conflicts.

Impact:

Switching from OS-provided packages to upstream (nginx.org) packages alters the update lifecycle. Administrators become responsible for tracking upstream changes rather than relying on the OS vendor's backporting policy. However, this is often necessary to meet modern security and performance requirements.

Audit:

To verify that package manager repositories are configured correctly and point to a trusted source (either the OS vendor or official NGINX), run the following commands:

Red Hat / Rocky-Linux:

```
dnf repolist -v | grep -i nginx
```

Debian / Ubuntu:

```
apt-cache policy nginx
```

Evaluation:

- Verify that the repository URL points to a trusted domain (e.g., nginx.org, rhel..., ubuntu...).
- Ensure that the repository is enabled.
- Check that no unknown or untrusted third-party repositories are configured for NGINX.

Remediation:

Configure your package manager to use a trusted repository that meets your version requirements.

To enable the official NGINX repository (Recommended for HTTP/3 support): Follow the instructions at nginx.org/en/linux_packages.html for your specific distribution. This typically involves adding the NGINX signing key and creating a repository configuration file.

Default Value:

By default, systems are configured with the OS vendor's repositories, which typically contain older versions of NGINX and not always all needed features.

References:

1. https://nginx.org/en/linux_packages.html

Additional Information:

Package update and installation commands and repository structures vary by Linux distribution. The examples provided are based on Red Hat Enterprise Linux 9. Please substitute with the appropriate commands for your environment. Note that HTTP/3 (QUIC) support was introduced in NGINX 1.25.0 (Mainline) and is not available in many OS-bundled versions.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	7.3 Perform Automated Operating System Patch Management Perform operating system updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.	●	●	●
v8	7.4 Perform Automated Application Patch Management Perform application updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.	●	●	●
v7	3.4 Deploy Automated Operating System Patch Management Tools Deploy automated software update tools in order to ensure that the operating systems are running the most recent security updates provided by the software vendor.	●	●	●
v7	3.5 Deploy Automated Software Patch Management Tools Deploy automated software update tools in order to ensure that third-party software on all systems is running the most recent security updates provided by the software vendor.	●	●	●

1.2.2 Ensure the latest software package is installed (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

As new security vulnerabilities are discovered, the corresponding fixes are implemented by your NGINX software package provider. Installing the latest software version ensures these fixes are available on your system.

Rationale:

Up-to-date software provides the best possible protection against exploitation of security vulnerabilities, such as the execution of malicious code.

Impact:

Updating the NGINX package requires a service reload or restart to apply the changes. This may cause a brief interruption or configuration errors if the new version deprecates existing syntax.

Audit:

To verify your NGINX package is up to date, run the following command (example):

Redhat:

```
dnf info nginx
```

Remediation:

To install the latest NGINX package, run the following command (example):

Redhat:

```
dnf update nginx -y
```

References:

1. https://nginx.org/en/linux_packages.html

Additional Information:

Package update and installation commands are based on Red Hat Enterprise Linux 9. If using a different Linux distribution, please substitute with the appropriate command(s) and consider your organization patch management policy.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	7.3 Perform Automated Operating System Patch Management Perform operating system updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.	●	●	●
v8	7.4 Perform Automated Application Patch Management Perform application updates on enterprise assets through automated patch management on a monthly, or more frequent, basis.	●	●	●
v7	3.4 Deploy Automated Operating System Patch Management Tools Deploy automated software update tools in order to ensure that the operating systems are running the most recent security updates provided by the software vendor.	●	●	●
v7	3.5 Deploy Automated Software Patch Management Tools Deploy automated software update tools in order to ensure that third-party software on all systems is running the most recent security updates provided by the software vendor.	●	●	●

2 Basic Configuration

2.1 Minimize NGINX Modules

2.1.1 Ensure only required dynamic modules are loaded (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

NGINX functionality is provided by modules. These modules are either compiled statically into the NGINX binary or loaded dynamically at runtime via the `load_module` directive.

- **Static Modules:** These are fixed at compile time. When using official pre-built packages (e.g., from nginx.org or OS vendors), a standard set of modules is included and cannot be removed without recompiling NGINX.
- **Dynamic Modules:** These are separate `.so` files that can be loaded on demand. To reduce the attack surface and complexity, only strictly required **dynamic modules** should be loaded. Additionally, administrators should be aware of the active **static modules** to avoid configuring unused features unintentionally.

Rationale:

Minimizing the loaded code reduces the potential attack surface. While static modules in pre-built packages cannot be removed, ensuring that no unnecessary **dynamic modules** are loaded prevents the execution of unneeded code. Furthermore, understanding which **static modules** are present helps administrators avoid enabling risky features (like `autoindex` or `stub_status`) in the configuration if they are not needed.

Impact:

Removing a required dynamic module or misinterpreting the availability of a static module can cause the NGINX service to fail on restart or break specific application features.

Audit:

1. Audit Dynamic Modules (Actionable):

Run the following command to check for actively loaded dynamic modules:

```
nginx -T 2>/dev/null | grep "load_module"
```

Evaluation:

- If the output is empty, no dynamic modules are loaded (PASS).
- If output exists (e.g., `load_module modules/ngx_http_geoip_module.so;`), verify that each listed module is required for the application's business logic.

2. Audit Static Modules (Informational):

Run the following command to list all modules compiled into the binary:

```
nginx -V 2>&1 | grep -oEi '\-\-(with|without)-[^ ]*'
```

Evaluation:

Review the `--with-...` flags to understand the server's capabilities. Ensure that risky modules present in the build (e.g., `http_stub_status_module`) are not enabled in any `server` or `location` block **unless authorized**.

Remediation:

For Dynamic Modules:

Open the main configuration file (`/etc/nginx/nginx.conf`) or the relevant include file (e.g., in `/etc/nginx/modules-enabled/`). Comment out or remove the `load_module` directive for any module **that is not strictly necessary**.

For Static Modules:

Since static modules cannot be removed from pre-built packages, ensure their directives are not used in your configuration. If a specific static module poses a critical risk to your environment, you must switch to a custom build or a different package flavor that excludes it.

Default Value:

Official pre-built packages (like `nginx-stable` or `nginx-mainline`) are "feature-rich" builds containing most standard modules statically. This is a trade-off for ease of maintenance. Security hardening for these packages relies on configuration discipline (not enabling unused modules) rather than binary minimization.

References:

1. https://nginx.org/en/docs/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.6 Allowlist Authorized Libraries Use technical controls to ensure that only authorized software libraries, such as specific .dll, .ocx, .so, etc., files, are allowed to load into a system process. Block unauthorized libraries from loading into a system process. Reassess bi-annually, or more frequently.		●	●
v7	2.8 Implement Application Whitelisting of Libraries The organization's application whitelisting software must ensure that only authorized software libraries (such as *.dll, *.ocx, *.so, etc) are allowed to load into a system process.			●

2.2 Account Security

2.2.1 Ensure that NGINX is run using a non-privileged, dedicated service account (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The NGINX master process typically runs as `root` to bind to privileged ports and manage resources, but it spawns worker processes to handle the actual client traffic. The `user` directive in the main configuration designates the operating system account under which these worker processes run.

Running worker processes under a non-privileged, dedicated service account limits the damage an attacker can cause in the event the NGINX process is compromised. This account should be exclusively dedicated to NGINX, have no login capabilities, and possess no elevated system privileges.

Rationale:

If an attacker successfully exploits a vulnerability in a worker process (e.g., via a buffer overflow or Remote Code Execution), they inherit the permissions of the user account running that process. Using a privileged account like `root` significantly increases the risk of lateral movement. A dedicated, locked-down service account ensures that an attacker cannot access other services, modify sensitive system files, or easily escalate privileges, effectively reducing the impact of the compromise.

Impact:

Changing the NGINX user requires that ownership and permissions for all runtime directories (logs, caches, PID files) are updated to match the new account. Incorrect permissions will prevent NGINX from starting or writing necessary logs. Additionally, the dedicated user must remain strictly unprivileged; adding it to groups like `sudo` or `wheel` would negate the security benefit.

Audit:

1. Identify Configured User:

Inspect the running configuration to find the user directive:

```
nginx -T 2>/dev/null | grep -i "^user"
```

Evaluation: Verify that a specific user is defined (e.g., `user nginx;` or `user www-data;`). If missing, NGINX might run as `nobody` or the user used at compile time.

2. Verify User Privileges:

Check the UID and group membership of the identified user (e.g., `nginx`):

```
id nginx
```

Evaluation:

- Ensure **uid is not 0** (root).
- Ensure the user **is not** a member of privileged groups (like `root`, `wheel`, `sudo`).

3. Check Sudo Access:

Ensure the user cannot execute commands via `sudo`:

```
sudo -l -U nginx
```

Evaluation: Output should indicate "`User nginx is not allowed to run sudo`".

Remediation:

1. Create/Harden User (if missing):

If no user exists, create a system user with a no login shell:

```
useradd -r -d /var/cache/nginx -s /sbin/nologin nginx
```

2. Configure NGINX:

Set the user directive in the main context of `nginx.conf`:

```
user nginx;
```

3. Lock User Account:

Ensure the account cannot be used for login:

```
usermod -s /sbin/nologin nginx
usermod -L nginx
```

Default Value:

Official packages usually create and configure a dedicated user (e.g., `nginx` or `www-data`). If compiled from source without arguments, the user defaults to `nobody`.

References:

1. https://nginx.org/en/docs/ngx_core_module.html#user

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	5.4 Restrict Administrator Privileges to Dedicated Administrator Accounts Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.	●	●	●
v7	4.3 Ensure the Use of Dedicated Administrative Accounts Ensure that all users with administrative account access use a dedicated or secondary account for elevated activities. This account should only be used for administrative activities and not internet browsing, email, or similar activities.	●	●	●

2.2.2 Ensure the NGINX service account is locked (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The NGINX service account must not have a usable password and should be explicitly locked in the system's shadow file to prevent direct login or password-based privilege escalation.

Rationale:

As a defense-in-depth measure, the NGINX service account should be explicitly locked. This prevents password-based logins and blocks adversaries from using the account for lateral movement, even if they manage to change the account's shell configuration.

In a properly hardened environment, there is no operational need for any user to log in as `nginx`. Administrative tasks requiring the NGINX identity should be performed using `sudo` (e.g., `sudo -u nginx`), which utilizes the administrator's credentials rather than the service account's password.

Impact:

Locking the service account has minimal operational impact. The account is not intended for human interaction, and all administrative tasks requiring the NGINX user context should already be performed using `sudo -u nginx` rather than password-based authentication.

Audit:

1. Identify the User:

```
nginx -T 2>/dev/null | grep -i "^user"
```

(Note the user, e.g., `nginx`)

2. Check Lock Status:

Run the following command for the identified user:

```
passwd -S nginx
```

Evaluation:

Verify that the output indicates a locked status:

- **RHEL**: Shows `LK` or `Password locked`.
- **Debian/Ubuntu:** Shows `L`.

Remediation:

Lock the account using the `passwd` command:

```
passwd -l nginx
```

(Replace `nginx` with the actual service user found in step 1)

Default Value:

System users created by official packages are typically created without a password (locked by default).

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

2.2.3 Ensure the NGINX service account has an invalid shell (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The NGINX service account must be configured with an invalid login shell to prevent interactive access.

Rationale:

The NGINX service account is strictly for running daemon processes. Assigning it a valid login shell (like `/bin/bash`) unnecessarily expands the attack surface. If an attacker compromises the account credentials (or adds an SSH key), a valid shell facilitates interactive system access. Setting the shell to `/sbin/nologin` or `/bin/false` ensures that even with valid credentials, the system immediately rejects a login attempt.

Impact:

None. Service accounts do not require interactive login capabilities for normal operation.

Audit:

1. Identify the User:

```
nginx -T 2>/dev/null | grep -i "^user"
```

(Note the user, e.g., `nginx`)

2. Verify Shell:

Run the following command to inspect the configured shell for the identified user:

```
getent passwd nginx
```

(Replace `nginx` with the actual user found in step 1)

Evaluation:

Examine the last field of the output (the shell).

- **PASS:** The shell is set to `/sbin/nologin`, `/bin/nologin`, or `/bin/false`.
- **FAIL:** The shell is set to `/bin/bash`, `/bin/sh`, or any other interactive shell listed in `/etc/shells`.

Example Output (PASS):

```
nginx:x:999:988:nginx user:/nonexistent:/usr/sbin/nologin
```

Remediation:

Change the login shell for the identified user to `/sbin/nologin`:

```
usermod -s /sbin/nologin nginx
```

(Replace `nginx` with the actual user)

Default Value:

Official packages typically configure the user with `/sbin/nologin` or `/bin/false` by default.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 Establish and Maintain a Secure Configuration Process Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.	●	●	●
v7	5.1 Establish Secure Configurations Maintain documented, standard security configuration standards for all authorized operating systems and software.	●	●	●

2.3 Permissions and Ownership

2.3.1 Ensure NGINX directories and files are owned by root (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The NGINX configuration directory and all contained files must be owned by the `root` user and group to prevent unauthorized modification.

Rationale:

The NGINX configuration **controls** the security posture of the web server. If a non-privileged user (including the `nginx` worker user) can modify these files, they can trivially escalate privileges (e.g., by loading a malicious module or changing the `user` directive to `root`). Ensuring that **only** `root` **owns** these files guarantees that configuration changes require administrative privileges.

Impact:

None. The NGINX master process runs as `root` and can read these files. The worker processes (running as `nginx`) do not need write access to the configuration.

Audit:

1. Identify Configuration Directory:

Run `nginx -V` and look for `--conf-path` to find the main configuration file location (e.g., `/etc/nginx/nginx.conf`). The directory containing this file is the target.

2. Verify Ownership:

Run the following command to audit the ownership of the configuration directory and its contents:

```
find /etc/nginx -name "*" \( -not -user root -o -not -group root \) -exec ls -ld {} \;
```

(Replace `/etc/nginx` with the actual configuration path in case it is different)

Evaluation:

- **PASS:** The command produces no output. All files are owned by `root:root`.
- **FAIL:** The command lists files owned by other users (e.g., `nginx` or a developer account).

Remediation:

Set the ownership of the NGINX configuration directory and files to `root`:

```
chown -R root:root /etc/nginx
```

(Replace `/etc/nginx` with the actual configuration path in case it is different)

Note: Ensure that this does not break access to specific files if you have a custom setup where external processes need write access.

Default Value:

The default ownership and group for nginx is `root`.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

2.3.2 Ensure access to NGINX directories and files is restricted (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The NGINX configuration directory (`/etc/nginx` or equivalent) and its contents should have restrictive permissions to enforce the principle of least privilege.

- **Directories** should be accessible only by the `root` user and the `root` group (and potentially read/execute by the group), but not by others.
- **Files** should be readable/writable by `root` and readable by the group, but inaccessible to others.

Rationale:

Restrictive file permissions prevent unauthorized users on the system from viewing sensitive configuration details, such as backend IP addresses, routing logic, or loaded module paths. By removing "world" access (permissions for "other"), we ensure that only administrators (via `sudo`) can interact with the web server configuration. This is a fundamental defense against information disclosure.

Impact:

Setting permissions to `640` (files) and `750` (directories) prevents non-privileged users from listing the configuration directory or reading configuration files.

Audit:

1. Identify Configuration Directory:

Run `nginx -V` and identify the `--conf-path` to determine the configuration root (e.g., `/etc/nginx`).

2. Audit Directory Permissions:

Run the following command to find directories with loose permissions:

```
find /etc/nginx -type d -exec stat -Lc "%n %a" {} +
```

Evaluation:

Verify that the output shows permissions of `750` (`drwxr-x---`) or more restrictive (e.g., `700`).

- **Standard:** `755` (`drwxr-xr-x`) allows world read/execute.
- **Hardened Target:** `750` or `700`.

3. Audit File Permissions:

Run the following command to find files with loose permissions:

```
find /etc/nginx -type f -exec stat -Lc "%n %a" {} +
```

Evaluation:

Verify that the output shows permissions of `640` (`-rw-r-----`) or more restrictive (e.g., `600`).

- **Standard:** `644` (`-rw-r--r--`) allows world read.
- **Hardened Target:** `640` or `600`.

Remediation:

To restrict access to the NGINX configuration directory and files, execute the following commands:

1. Restrict Directories (`750`):

Allow owner (`root`) full access, group read/execute, deny others.

```
find /etc/nginx -type d -exec chmod 750 {} +
```

2. Restrict Files (`640`): Allow owner (`root`) read/write, group read, deny others.

```
find /etc/nginx -type f -exec chmod 640 {} +
```

Note: Private keys (e.g., `.key` files) require even stricter permissions (`400` or `600`) and should be addressed separately or manually verified here.

Default Value:

Default permissions are typically `755` for directories and `644` for files (world-readable).

Additional Information:

Critical: If your NGINX setup relies on the worker process (`nginx` user) reading configuration files directly (uncommon for standard configs, but possible with certain includes or SSL certificates not handled by the master process), ensure the `nginx` user is in the group that owns the files, **OR** use `chmod 644` if absolutely necessary. However, the Master Process (`root`) handles parsing, so `640`/`root:root` is usually safe.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

2.3.3 Ensure the NGINX process ID (PID) file is secured (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `PID` file stores the main process ID of the nginx process. This file should be protected from unauthorized modification.

Rationale:

The `PID` file should be owned by `root` and the group `root`. It should also be readable to everyone, but only writable by `root` (permissions `644`). This will prevent unauthorized modification of the `PID` file, which could cause a denial of service.

Impact:

None. The PID file is managed by the master process (`root`). Restricting write access prevents other users from tampering with the file, but read access is generally safe and required for monitoring.

Audit:

1. Identify PID File Location:

Run `nginx -V` and look for the `--pid-path` argument to confirm the location (e.g., `/run/nginx.pid` or `/var/run/nginx.pid`).

2. Verify Ownership and Permissions:

Run the following command (substituting the identified path):

```
stat -Lc "%U:%G %a" /run/nginx.pid
```

Evaluation:

- **Ownership:** Must be `root:root`.
- **Permissions:** Must be `644` (`rw-r--r--`) or more restrictive.

Remediation:

Set the correct ownership and permissions for the `PID` file (replace path as needed):

```
chown root:root /run/nginx.pid
chmod 644       /run/nginx.pid
```

Default Value:

The `PID` file is owned by `root` and has permissions `644` by default when building using `(dnf or apt)`.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

2.4 Network Configuration

2.4.1 Ensure NGINX only listens for network connections on authorized ports (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

NGINX should be configured to listen only on authorized ports and protocols. While traditional HTTP/1.1 and HTTP/2 use TCP ports `80` and `443`, modern HTTP/3 (QUIC) utilizes UDP port `443`. Ensuring that NGINX binds only to approved interfaces and ports minimizes the attack surface.

Rationale:

Limiting listening ports to authorized values ensures that no hidden or unintended services are exposed via NGINX. It also enforces strict control over which protocols (TCP vs. UDP) are accessible, which is particularly important with the introduction of UDP-based HTTP/3 traffic alongside traditional TCP traffic.

Impact:

Disabling unused ports reduces the risk of unauthorized access. However, administrators must be aware that disabling UDP port `443` will break HTTP/3 connectivity, forcing clients to fall back to slower TCP-based HTTP/2 or HTTP/1.1.

Audit:

1. Inspect Configuration:

Run the following command to inspect all listen directives in the loaded configuration:

```
nginx -T 2>/dev/null | grep -r "listen"
```

Evaluation:

Review the output for unauthorized ports. A modern secure configuration typically includes:

- `listen 80;` (TCP) - Often used only for redirecting to HTTPS.
- `listen 443 ssl;` (TCP) - For HTTP/1.1 and HTTP/2.
- `listen 443 quic;` (UDP) - For HTTP/3 (QUIC).

Example Output:

```
server {

    listen  80;
    listen 443 ssl;
    listen 443 quic reuseport; # HTTP/3 (UDP)
    ...
}
```

Ensure that no other ports (e.g., `8080`, `8443`) are open unless explicitly authorized for internal services or management interfaces.

2. Verify System Listening Ports:

Optionally, verify what the process is actually binding to on the OS level:

```
netstat -tulpen | grep -i nginx
```

- Look for `tcp` lines for standard traffic.
- Look for `udp` lines (e.g., `*:443`) if HTTP/3 is enabled.

Remediation:

Remove or comment out any `listen` directives that bind to unauthorized ports.

For HTTP/3 (QUIC) Support: Ensure that you explicitly authorize and configure UDP port `443` in addition to TCP port `443`.

```
server {

    # Standard HTTPS (TCP)
    listen 443 ssl;

    # HTTP/3 (UDP)
    listen 443 quic reuseport;

    # ... SSL/TLS configuration ...
}
```

Default Value:

By default, NGINX often listens only on TCP port `80`. Modern secure defaults should listen on TCP `80` (for redirect), TCP `443`, and optionally UDP `443` (for HTTP/3).

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.10 Apply Secure Design Principles in Application Architectures Apply secure design principles in application architectures. Secure design principles include the concept of least privilege and enforcing mediation to validate every operation that the user makes, promoting the concept of "never trust user input." Examples include ensuring that explicit error checking is performed and documented for all input, including for size, data type, and acceptable ranges or formats. Secure design also means minimizing the application infrastructure attack surface, such as turning off unprotected ports and services, removing unnecessary programs and files, and renaming or removing default accounts.		●	●
v7	9.2 Ensure Only Approved Ports, Protocols and Services Are Running Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.		●	●

2.4.2 Ensure requests for unknown host names are rejected (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

NGINX routes incoming requests to the appropriate virtual host by matching the `Host` header (HTTP/1.1) or `:authority` pseudo-header (HTTP/2, HTTP/3) against the `server_name` directives in your configuration. If no explicit match is found, NGINX falls back to the first defined `server` block **or** the one marked as `default_server`. Without a properly configured catch-all block that rejects unknown hostnames, your server will respond to arbitrary domain names that happen to point to your IP address, potentially exposing internal applications or enabling Host Header attacks.

Rationale:

When NGINX receives a request, it selects the virtual host based on the `Host` header (or `:authority` in HTTP/2/3). If requests for unknown host names are not explicitly rejected, your applications may be served for arbitrary domains that simply point to your IP. This behavior can be abused in Host Header attacks and makes it harder to distinguish legitimate traffic from automated scans or misrouted requests in your logs.

Impact:

Clients accessing the server directly via IP address or an unconfigured **CNAME** will be rejected. This is intended behavior but requires that all valid domains are explicitly defined in their own `server` blocks.

Audit:

1. Review Configuration:

Check for the existence of a default `server` block that handles unknown hosts.

```
nginx -T 2>/dev/null | grep -Ei "listen.*default_server|ssl_reject_handshake"
```

Evaluation:

- Ensure a `server` block exists with `listen ... default_server`.
- Verify it contains `return 444;` (closes connection) or a `4xx` error code.
- For HTTPS/TLS: Verify `ssl_reject_handshake on;` is used to prevent certificate leakage.

2. Functional Test:

Send a request with an invalid Host header and verify the connection is rejected or returns an error.

```
# Test HTTPS (expect connection reset or 4xx)
curl -k -v https://127.0.0.1 -H 'Host: invalid.example.com'
```

Remediation:

Configure a "Catch-All" default `server` block as the **first** block in your configuration (or explicitly marked with `default_server`).

Configuration Example (Modern Standard with TLS/HTTP3):

```
server {

    # Listen on standard ports for IPv4 and IPv6
    listen      80  default_server;
    listen [::]:80  default_server;

    # Listen for HTTPS (TCP) and QUIC (UDP)
    listen      443 ssl  default_server;
    listen [::]:443 ssl  default_server;
    listen      443 quic default_server;
    listen [::]:443 quic default_server;

    # Reject SSL Handshake for unknown domains (Prevents cert leakage)
    ssl_reject_handshake on;

    # Catch-all name
    server_name _;

    # Close connection without response (Non-standard code 444)
    return 444;

}
```

After adding this block, ensure all your valid applications have their own `server` blocks with explicit `server_name` directives.

Default Value:

By default, if no `default_server` is defined, NGINX uses the first `server` block configuration it finds, potentially serving your application for any incoming request regardless of the Host header.

References:

1. https://nginx.org/en/docs/http/request_processing.html
2. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Host
3. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/17-Testing_for_Host_Header_Injection
4. https://portswigger.net/web-security/host-header

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	4.1 Establish and Maintain a Secure Configuration Process Establish and maintain a secure configuration process for enterprise assets (end-user devices, including portable and mobile, non-computing/IoT devices, and servers) and software (operating systems and applications). Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.	●	●	●
v7	5.1 Establish Secure Configurations Maintain documented, standard security configuration standards for all authorized operating systems and software.	●	●	●

2.4.3 Ensure keepalive_timeout is 10 seconds or less, but not 0 (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Persistent connections are leveraged by all modern browsers to facilitate greater web performance. The keep-alive timeout limits the time a persistent connection may remain open. Setting the keep-alive timeout allows this timeout to be controlled on the server side.

Rationale:

Setting a keep-alive timeout on the server side helps mitigate denial of service attacks that establish too many persistent connections, exhausting server resources.

Audit:

To check the current setting for the `keepalive_timeout` directive, issue the below command. You should also manually check your nginx configuration for include statements that may be located outside the `/etc/nginx` directory. If none of these are present, the value is set at the default.

```
grep -ir keepalive_timeout /etc/nginx
```

The output of the command should contain something similar to the following:

```
keepalive_timeout 10;
```

Remediation:

Find the `HTTP` or server block of your nginx configuration, and add the `keepalive_timeout` directive. Set it to `10` seconds or less, but not `0`. This example command sets it to `10` seconds:

```
keepalive_timeout 10;
```

Default Value:

By default, this timeout is dictated by the user agent and varies. It is not set on the server side by default.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html#keepalive_timeout

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

2.4.4 Ensure send_timeout is set to 10 seconds or less, but not 0 (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `send_timeout` directive sets a timeout for transmitting a response to the client between two successive write operations.

Rationale:

Setting the `send_timeout` directive on the server side helps mitigate slow `HTTP` denial of service attacks by ensuring write operations taking up large amounts of time are closed.

Audit:

To check the current setting for the `send_timeout` directive, issue the below command. You should also manually check your nginx configuration for include statements that may be located outside the `/etc/nginx` directory. If none of these are present, the value is set at the default.

```
grep -ir send_timeout /etc/nginx
```

The output of the command should be similar to the following:

```
send_timeout  10;
```

Remediation:

Find the `HTTP` or server block of your nginx configuration, and add the `send_timeout` directive. Set it to `10` seconds or less, but not `0`.

```
send_timeout   10;
```

Default Value:

`send_timeout 60s;`

References:

1. https://www.owasp.org/index.php/SCG_WS_nginx
2. https://nginx.org/en/docs/http/ngx_http_core_module.html#send_timeout

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

2.5 Information Disclosure

2.5.1 Ensure server_tokens directive is set to `off` (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `server_tokens` directive is responsible for displaying the NGINX version number and operating system version on error pages and in the `Server HTTP` response header field. This information should not be displayed.

Rationale:

Attackers can conduct reconnaissance on a website using these response headers, then target attacks for specific known vulnerabilities associated with the underlying technologies. Hiding the version will slow down and deter some potential attackers.

Impact:

None. Disabling server tokens does not affect functionality. It merely removes the version string from error pages and headers. Note that determined attackers can still fingerprint NGINX via other methods, but removing the banner raises the bar for opportunistic scanners.

Audit:

In the NGINX configuration file `nginx.conf`, verify the `server_tokens` directive is set to `off`. To do this, check the response headers for the server header by issuing this command:

```
curl -I 127.0.0.1 | grep -i server
```

The output should not contain the server header providing your server version, such as the below:

```
Server: nginx/1.28.0
```

Remediation:

Disable version disclosure globally by adding the directive to the `http` block in `/etc/nginx/nginx.conf`:

```
http {
    ...
    server_tokens        off;
    ...
}
```

Default Value:

The default value of `server_tokens` is `on`.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

2.5.2 Ensure default error and index.html pages do not reference NGINX (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Default error pages (e.g., `404`, `500`) and the default welcome page often contain NGINX branding or signatures. These pages should be removed or replaced with generic or custom-branded pages that do not disclose the underlying server technology.

Rationale:

Standard NGINX error pages visually identify the server software, even if headers are suppressed. By gathering information about the underlying technology stack, attackers can tailor their exploits to known vulnerabilities of NGINX. Replacing default pages with generic or branded content removes this information leakage vector and increases the effort required for successful reconnaissance.

Impact:

Creating and maintaining custom error pages requires additional administrative effort. Ensure that custom error pages are simple and do not themselves introduce vulnerabilities.

Audit:

1. Verify Config:

Check if `error_page` directives are active:

```
nginx -T 2>/dev/null | grep -i "error_page"
```

2. Verify Content (Functional Test):

Trigger an error (e.g., request a non-existent page) and inspect the body:

```
curl -k https://127.0.0.1/non-existent-page | grep -i "nginx"
```

Evaluation:

- **PASS:** The output does not contain "nginx" (or `grep` returns nothing).
- **FAIL:** The HTML body contains "nginx".

Remediation:

Instead of editing the default files (which may be overwritten by package updates), configure NGINX to use custom error pages.

1. Create Custom Error Pages:

Create a directory (e.g., `/var/www/html/errors`) and place generic HTML files there (e.g., `404.html`, `50x.html`) **without** NGINX branding.

2. Configure NGINX:

Add the `error_page` directive to your `http` or `server` blocks:

```
error_page 404 /404.html;
error_page 500 502 503 504 /50x.html;

location = /50x.html {
    root /var/www/html/errors;
    internal;
}
```

Default Value:

Default error pages identify the server as NGINX.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html#error_page

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

2.5.3 Ensure hidden file serving is disabled (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Hidden files and directories (starting with a dot, e.g., `.git`, `.env`) often contain sensitive metadata, version control history, or environment configurations. Serving these files should be globally disabled.

Rationale:

Version control systems (Git, SVN) and editors create hidden files that may unintentionally be deployed to the web root. If accessible, files like `.git/config` or `.env` can leak database credentials, source code, and infrastructure details, leading to full system compromise. Blocking requests to any path starting with a dot (`.`) neutralizes this risk.

Impact:

Blocking all dot-files will break Let's Encrypt / Certbot validation (`.well-known/acme-challenge`) unless explicitly allowed. Ensure the exception rule is placed **before** the deny rule or is more specific.

Audit:

1. Check Configuration:

Search the loaded configuration for hidden file protection rules:

```
nginx -T 2>/dev/null | grep "location.*\\\."
```

Evaluation:

- Look for a block like `location ~ /\. { deny all; ... }`.

2. Functional Test (Recommended):

Try to access a dummy hidden file:

```
curl -k -I https://127.0.0.1/.git/HEAD
```

Evaluation:

- **PASS:** Returns `403 Forbidden` or `404 Not Found`.
- **FAIL:** Returns `200 OK` (if file exists) or the content of the file.

Remediation:

To restrict access to hidden files, add the configuration block below inside each server block.

Option A: Direct Configuration

Place this block directly into your `server` contexts:

```
# Allow Let's Encrypt validation (must be before the deny rule)
location ^~ /.well-known/acme-challenge/ {
    allow all;
    default_type "text/plain";
}

# Deny access to all other hidden files
location ~ /\. {
    deny all;
    return 404;
}
```

Option B: Using a Shared Snippet (Recommended)

Create a reusable snippet file (e.g., `/etc/nginx/snippets/deny-hidden.conf`) containing the rules above, and include it in your `server` blocks:

1. Create `/etc/nginx/snippets/deny-hidden.conf` with the content from Option A.
2. Security Check: Ensure the new file has restrictive permissions (Owner: `root:root`, Mode: `640`) as described in Recommendation 2.3.2.
3. Add the include directive to your server blocks:

```
server {
    # Modern HTTP/3 (QUIC) and HTTP/2 Setup

    listen 443 ssl;             # TCP for HTTP/1.1 & HTTP/2
    listen 443 quic reuseport; # UDP for HTTP/3
    http2 on;                   # Explicitly enable HTTP/2 (since NGINX
1.25.1)

    server_name example.com;

    include /etc/nginx/snippets/deny-hidden.conf;

    # ... rest of configuration
}
```

Default Value:

This protection is not set by default. NGINX will serve any hidden file if it exists in the web root.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html#location
2. https://nginx.org/en/docs/http/ngx_http_access_module.html#deny
3. https://capec.mitre.org/data/definitions/139.html
4. https://cwe.mitre.org/data/definitions/538.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

2.5.4 Ensure the NGINX reverse proxy does not enable information disclosure (Manual)

Profile Applicability:

- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

When NGINX acts as a reverse proxy, it forwards headers sent by the upstream application (e.g., "`X-Powered-By: Custom_APP`" or "`Server: Apache/2.4`"). These headers should be stripped before the response reaches the client to prevent information disclosure about the backend infrastructure.

Rationale:

Attackers conduct reconnaissance by inspecting response headers to identify the technologies used in the backend (e.g., specific versions of PHP, Java/Tomcat, or Python frameworks). Knowing the exact version allows attackers to target specific CVEs associated with that software stack. Removing these headers reduces the information available for targeted attacks.

Audit:

1. Configuration Check:

Search the loaded configuration for header hiding directives:

```
nginx -T 2>/dev/null | grep -Ei "(proxy|fastcgi)_hide_header"
```

Evaluation:

- Verify that directives exist to hide `X-Powered-By` and `Server`.

2. Functional Check (Recommended):

Send a request to a proxied endpoint and inspect the response headers:

```
curl -k -I https://127.0.0.1 | grep -Ei "^(Server|X-Powered-By)"
```

Evaluation:

- **PASS:** The output does not contain backend details (e.g., `X-Powered-By: PHP/8.2`). If Server is present, it should only be `Server: nginx` (controlled by Control 2.5.1).
- **FAIL:** The output contains backend information or unmasked version numbers.

Remediation:

Configure NGINX to strip the sensitive headers. The directive depends on the upstream protocol (HTTP Proxy vs. FastCGI).

For Standard Reverse Proxy (`proxy_pass`):

Add the following directives to your `http`, `server`, or `location` block:

```
proxy_hide_header X-Powered-By;
proxy_hide_header Server;
```

For PHP/FastCGI (`fastcgi_pass`):

If you are using FastCGI (e.g., for PHP-FPM), use the `fastcgi_hide_header` directive instead:

```
fastcgi_hide_header X-Powered-By;
```

Default Value:

By default, NGINX passes all headers received from the upstream server to the client unchanged.

References:

1. https://nginx.org/en/docs/http/ngx_http_proxy_module.html
2. https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_hide_header

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

3 Logging

3.1 Ensure detailed logging is enabled (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

System logging must be configured to meet organizational security and privacy policies. Detailed logs provide the necessary context (event source, timestamp, user, network data) for incident response and forensic analysis. Modern logging strategies favor structured formats (JSON) over unstructured text to facilitate parsing by SIEM solutions.

Note: Sensitive information (e.g., session tokens, PII in query strings) should be excluded or masked in logs to prevent data leaks.

Rationale:

Detailed logs are the foundation of effective incident response. CIS Control 8.5 ("Collect Detailed Audit Logs") recommends capturing event sources, dates, users, timestamps, and network addresses. Traditional text logs require complex, fragile Regex parsing that breaks easily when formats change. Structured logging (JSON) solves this by providing a self-describing format that is natively ingested by modern analysis tools (SIEM), ensuring that critical forensic data is always indexable and searchable.

Impact:

Enabling detailed JSON logging increases the volume of log data. Ensure that your log rotation policies (`logrotate`) and disk space monitoring are adjusted to handle the increased storage requirements.

Audit:

1. Verify Log Format Configuration:

Inspect the `log_format` directives in your configuration:

```
nginx -T 2>/dev/null | grep -i "log_format"
```

Evaluation:

- Confirm that a detailed format (preferably JSON) is defined.
- Verify that the format includes critical fields: `$time_iso8601`, `$remote_addr`, `$remote_user`, `$request`, `$status`, `$http_user_agent`.

2. Verify Access Log Usage:

Check that the defined format is actually used by the `access_log` directive:

```
nginx -T 2>/dev/null | grep "access_log"
```

Evaluation:

- The `access_log` directive should reference the detailed format name (e.g., `access_log /var/log/nginx/access.json main_access_json;`).

Remediation:

Define a detailed log format in the `http` block of `/etc/nginx/nginx.conf`. It is highly recommended to use JSON format for compatibility with modern SIEM tools.

Recommended Configuration (JSON):

```
http {
    log_format main_access_json escape=json '{'
      '"timestamp":          "$time_iso8601",'
      '"remote_addr":        "$remote_addr",'
      '"remote_user":        "$remote_user",'
      '"server_name":        "$server_name",'
      '"request_method":     "$request_method",'
      '"request_uri":        "$request_uri",'
      '"status":             $status,'
      '"body_bytes_sent":    $body_bytes_sent,'
      '"http_referer":       "$http_referer",'
      '"http_user_agent":    "$http_user_agent",'
      '"x_forwarded_for":    "$http_x_forwarded_for",'
      '"request_id":         "$request_id"'
    '}';

    # Apply the format globally or per server
    access_log /var/log/nginx/access.json main_access_json;
}
```

Legacy Configuration (Text-based):

If JSON is not feasible, ensure the text format captures all necessary fields:

```
log_format main_detailed '$remote_addr - $remote_user [$time_local] '
                         '"$request" $status $body_bytes_sent '
                         '"$http_referer" "$http_user_agent" '
                         '"$http_x_forwarded_for"';
```

Default Value:

By default, NGINX uses the `combined` log format, which is a standard text format but lacks details (e.g., request processing time, upstream information).

References:

1. https://nginx.org/en/docs/http/ngx_http_log_module.html#log_format
2. https://nginx.org/en/docs/varindex.html

Additional Information:

- **Load Balancers & Proxies:** Since NGINX often sits behind other proxies (LBs, CDNs), the `$remote_addr` variable may only show the LB's IP. Ensure you log `$http_x_forwarded_for` (as shown in the JSON example) to capture the true client IP.
- **SIEM Integration:** The `escape=json` parameter automatically handles escaping of special characters, preventing broken JSON structures.
- **Policy Compliance:** Consult your internal Logging & Monitoring Policy to determine exactly which data points are required for retention and which sensitive fields must be excluded.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.		●	●
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.		●	●

3.2 Ensure access logging is enabled (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `access_log` directive enables the logging of client requests. While NGINX enables this by default, it allows granular control per server or location context. Based on enterprise requirements, the log should be enriched with relevant variables or converted to structured JSON format for modern SIEM integration. Refer to Recommendation 3.1 for detailed configuration of log formats and variables. Ensure that access logging is active for all critical services.

Rationale:

Access logs are the primary record of system usage, detailing who accessed what resources and when and general troubleshooting. Without active access logs, incident responders are blind to web-based attacks (such as SQL injection, XSS probing, or Brute Force attempts) and auditors cannot verify compliance or user activity. Disabling logs globally (`access_log off;`) effectively destroys the forensic chain of custody for security events.

Impact:

Enabling detailed access logging increases disk space usage significantly. Without proper log rotation (e.g., `logrotate`) and monitoring, log files can rapidly consume available disk space, potentially causing the server to stop processing requests or crash. Ensure sufficient storage capacity and retention policies are in place.

Audit:

1. Verify Configuration:

Inspect the fully loaded configuration for log settings:

```
nginx -T 2>/dev/null | grep -i "access_log"
```

Evaluation:

- **Destination Check:** Verify that `access_log` directives point to a valid local file path (e.g., `/var/log/nginx/access.json`) for ingestion by log shippers.
- **Status Check:** Identify any instances of `access_log off;`.
- **Pass:** If `access_log off;` is absent, or strictly limited to non-critical assets (e.g., `location = /favicon.ico`, static assets, or internal health checks).
- **Fail:** If `access_log off;` is applied globally in the `http` block or to `server` blocks handling business logic.

Remediation:

Enable access logging in the `http` block to set a secure global default, or configure it explicitly within specific `server` blocks. It is recommended to use the detailed log format defined in Recommendation 3.1.

Configuration Example:

```
http {

    # Enable global logging using the detailed JSON format from Rec 3.1
    access_log /var/log/nginx/access.json main_access_json;

    server {

        # Inherits the global log setting, or can be overridden:
        access_log /var/log/nginx/example.com.access.json main_access_json;

        location / {
            # ...
        }

        # Exception: Disable logging for favicon to reduce noise (Optional)
        location = /favicon.ico {
            access_log    off;
            log_not_found off;
        }
    }
}
```

Default Value:

Access logging is enabled by default, typically logging to `logs/access.log` or `/var/log/nginx/access.log` using the standard `combined` format.

References:

1. https://nginx.org/en/docs/http/ngx_http_log_module.html#access_log
2. https://nginx.org/en/docs/http/ngx_http_core_module.html#log_not_found

Additional Information:

Embedded Variables for NGINX

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.		●	●
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.		●	●

3.3 Ensure error logging is enabled and set to the info logging level (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `error_log` directive configures logging for server errors and operational messages. Unlike access logs, error logs capture diagnostic information about failed requests, upstream connection issues, and configuration errors. The log level determines the verbosity of these messages and should be set to capture sufficient detail (typically `notice` or `info`) without overwhelming the storage system.

Rationale:

While access logs capture incoming request patterns, error logs provide the internal system context required to diagnose why a request failed. They are essential for identifying:

1. Upstream Failures: Connection timeouts or refused connections to backend servers (e.g., application server is down).

2. Process Anomalies: Unexpected worker process terminations or restarts, which may indicate resource exhaustion or exploitation attempts.

3. Configuration Errors: Invalid request handling that NGINX rejects before logging to access logs (e.g., header size limits exceeded).

Without error logs, an administrator sees a "`500 Internal Server Error`" in the access log but has no way to determine the root cause.

Impact:

Setting the log level to `info` (or even `debug`) can generate a significant volume of log data, especially on busy servers or during denial-of-service attacks. This increases disk I/O and storage requirements. Ensure that log rotation (e.g., via `logrotate`) is configured and storage usage is monitored to prevent disk exhaustion.

Audit:

1. Verify Configuration:

Check the fully loaded configuration for error log settings:

```
nginx -T 2>/dev/null | grep -i "error_log"
```

Evaluation:

- **Presence:** Verify that `error_log` is defined globally in the `main` context (or `http` block).
- **Destination:** Ensure it points to a valid local file (e.g., `/var/log/nginx/error.log`) accessible for ingestion by log shippers.
- **Level:** Confirm the level is set according to your internal "Monitoring and Logging" policy.
- **Fail:** If `error_log` points to `/dev/null` or the level is set to `crit`, `alert`, or `emerg` (which suppresses too many relevant warnings).

Remediation:

Configure the `error_log` directive in the `main` context (at the top of `nginx.conf`) to capture operational events.

Configuration Example:

```
# Log errors to a specific file with the 'notice' level
error_log /var/log/nginx/error.log notice;

http {
    # ...
}
```

Note: The specific logging level should be aligned with the organization's "Monitoring and Logging" Policy, balancing the need for forensic detail against storage and processing costs. Typically, `info` or `notice` is recommended.

Default Value:

By default, NGINX logs errors to `logs/error.log` with the severity level `error`. This configuration misses `warn`, `notice`, and `info` events.

References:

1. https://nginx.org/en/docs/ngx_core_module.html#error_log
2. https://docs.nginx.com/nginx/admin-guide/monitoring/logging/

Additional Information:

Unlike access logs, NGINX Open Source uses a hardcoded text format for error logs and does not support custom `log_format` definitions (e.g., JSON, additional variables).

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.5 Collect Detailed Audit Logs Configure detailed audit logging for enterprise assets containing sensitive data. Include event source, date, username, timestamp, source addresses, destination addresses, and other useful elements that could assist in a forensic investigation.		●	●
v7	6.3 Enable Detailed Logging Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.		●	●

3.4 Ensure proxies pass source IP information (Manual)

Profile Applicability:

- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

When NGINX acts as a reverse proxy or load balancer, it terminates the client connection and opens a new connection to the upstream application server. By default, the upstream server sees the NGINX server's internal IP address as the source, obscuring the original client IP. Standard HTTP headers like `X-Forwarded-For` and `X-Real-IP` must be explicitly configured to pass the original client's IP address and protocol information to the backend application.

Rationale:

Visibility of the true client IP address is essential for security auditing, incident response, and access control within the backend application. Without forwarding this information:

1. **Forensics:** Application logs will show all traffic coming from the NGINX proxy IP, making it impossible to trace malicious activity to a specific attacker.
2. **Access Control:** Application-level IP allow/deny lists or rate limits will fail or mistakenly block the entire proxy.
3. **Compliance:** Accurate logging of the user origin is often a regulatory requirement.

Impact:

Enabling these headers allows the backend application to see the original client IP. However, if NGINX simply appends to an existing `X-Forwarded-For` header sent by a malicious client, the backend might be tricked into trusting a spoofed IP at the beginning of the list.

Audit:

1. Verify Configuration:

Check the active configuration for proxy header directives in proxied locations:

```
nginx -T 2>/dev/null | grep -E "proxy_set_header (X-Real-IP|X-Forwarded-For)"
```

Evaluation:

- **Presence:** Verify that `proxy_set_header X-Forwarded-For` **and** `proxy_set_header X-Real-IP` are present in `location` blocks that use `proxy_pass` (or `grpc_pass`, `fastcgi_pass`).
- **Correctness:**
  - `X-Forwarded-For` should typically use `$proxy_add_x_forwarded_for` (to preserve the chain) or `$remote_addr` (if NGINX is the first trusted hop).
  - `X-Real-IP` should use `$remote_addr`.
- **Scope:** Check that this is applied to all relevant proxied locations.

Remediation:

Configure NGINX to forward client IP information in your `server` or `location` blocks where `proxy_pass` is used.

Configuration Example:

```
location / {

    # Use 'https' for Zero Trust environments (requires proxy_ssl_verify
configuration)
    # Use 'http'  for standard TLS offloading (upstream traffic is
unencrypted)
    proxy_pass <protocol>://example_backend_application;

    # Standard header: Appends the client IP to the list of proxies
    proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;

    # NGINX-specific header: Sets the direct client IP (useful for apps
expecting a single value)
    proxy_set_header X-Real-IP         $remote_addr;

    # Recommended: Forward the protocol (http vs https)
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

Default Value:

By default, NGINX does not add these headers. The upstream server receives requests appearing to originate from the NGINX server's IP address.

References:

1. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Forwarded-For
2. https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_set_header

Additional Information:

Privacy: Users' privacy should be considered when forwarding client IP addresses. Ensure your organization's privacy policy discloses the collection and processing of IP address information.

Security Trust: Any information in the `X-Forwarded-For` header supplied by the client (before reaching your trusted infrastructure) is untrusted and can be easily spoofed. Backend applications and security controls (like rate limiting) must be configured to trust only the IPs appended by your own NGINX instance or known trusted proxies, ignoring the client-supplied part of the chain.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	8.11 Conduct Audit Log Reviews Conduct reviews of audit logs to detect anomalies or abnormal events that could indicate a potential threat. Conduct reviews on a weekly, or more frequent, basis.		●	●
v7	6.7 Regularly Review Logs On a regular basis, review logs to identify anomalies or abnormal events.		●	●

4 Encryption

4.1 TLS / SSL Configuration

4.1.1 Ensure HTTP is redirected to HTTPS (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Browsers and clients establish encrypted connections with servers by leveraging HTTPS. Requests leveraging HTTP are unencrypted. Unencrypted requests should be redirected so they are encrypted. Any listening HTTP port on your web server should redirect to a server profile that uses encryption. The default HTTP (unencrypted) port is `80`.

Rationale:

Redirecting user agent traffic to HTTPS helps to ensure all user traffic is encrypted. Modern browsers alert users that your website is insecure when HTTPS is not used. This can decrease user trust in your website and ultimately result in decreased use of your web services. Redirection from HTTP to HTTPS couples security with usability; users are able to access your website even if they lack the security awareness to use HTTPS over HTTP when requesting your website.

Impact:

Use of HTTPS does result in a performance reduction in traffic to your website, however, due to the increased value of the security, many businesses consider this to be a cost of doing business.

Audit:

To verify your server listening configuration, check your web server or proxy configuration file. The default web server configuration file is `/etc/nginx/conf.d/default.conf`, and the default proxy configuration file is `/etc/nginx/nginx.conf`. The configuration file should return a statement redirecting to HTTPS. This should be similar to the code below, where cisecurity.org is used as an example.

```
server {
    listen 80;

    server_name cisecurity.org;

    return 301 https://$host$request_uri;
}
```

Remediation:

Edit your web server or proxy configuration file to redirect all unencrypted listening ports, such as port `80`, using a redirection through the return directive (cisecurity.org is used as an example server name).

```
server {
    listen 80;

    server_name cisecurity.org;

    return 301 https://$host$request_uri;
}
```

Default Value:

NGINX is not configured to use HTTPS or redirect to it by default.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.2 Ensure a trusted certificate and trust chain is installed (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Certificates and their trust chains are needed to establish the identity of a web server as legitimate and trusted. Certificate authorities validate a web server's identity and that you are the owner of that web server domain name.

Rationale:

Without a certificate and full trust chain installed on your web server, modern browsers will flag your web server as untrusted.

Audit:

Run this command to find the file location of your certificate:

```
grep -ir ssl_certificate /etc/nginx/
```

The output of your command should look similar to the below output. If there is no output, you do not have a certificate installed.

Web Server:

```
/etc/nginx/nginx.conf:    ssl_certificate /etc/nginx/cert.pem;
/etc/nginx/nginx.conf:    ssl_certificate_key /etc/nginx/nginx.key;
```

Open the file to the right of the `ssl_certificate` directive using the following command:

```
cat /etc/nginx/cert.pem
```

The output of your command should look similar to the below. It should include the full certificate chain.

```
-----BEGIN CERTIFICATE-----
Insert Your Web Server Certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Insert Your Certificate Authority Intermediate Certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Insert Your Certificate Authority Root Certificate
-----END CERTIFICATE-----
```

Remediation:

Use the following procedure to install a certificate and its signing certificate chain onto your web server, load balancer, or proxy.

Step 1: Create the server's private key and a certificate signing request.

The following command will create your certificate's private key with 4096-bit key strength. It will also output your certificate signing request to the `nginx.csr` file in your present working directory.

```
openssl req -new -newkey rsa:4096 -keyout nginx.key -out nginx.csr
```

Enter the below information about your private key:

```
Country Name (2 letter code) [XX]: Your Country
State or Province Name (full name) []: Your State
Locality Name (eg, city) [Default City]: Your City
Organization Name (eg, company) [Default Company Ltd]: Your City
Organizational Unit Name (eg, section) []: Your Organizational Unit
Common Name (eg, your name or your server's hostname) []: Your server's DNS
name
Email Address []: Your email address
```

Step 2: Obtain a signed certificate from your certificate authority.

Provide your chosen certificate authority with your certificate signing request. Follow your certificate authority's signing procedures in order to obtain a certificate and the certificate's trust chain. A full trust chain is typically delivered in `.pem` format.

Step 3: Install certificate and signing certificate chain on your web server.

Place the `.pem` file from your certificate authority into the directory of your choice. Locate your created key file from the command you used to generate your certificate signing request. Open your website configuration file and edit your encrypted listener to leverage the ssl_certificate and `ssl_certificate_key` directives for a web server as shown below. You should also inspect include files inside your `nginx.conf`. This should be part of the server block.

```
server {
    listen              443 ssl http2;
    listen              [::]:443 ssl http2;
    ssl_certificate     /etc/nginx/cert.crt;
    ssl_certificate_key /etc/nginx/nginx.key;
    ...
    }
```

After editing this file, you must restart the nginx systemd service for these changes to take effect. This can be done with the following command:

```
sudo systemctl restart nginx
```

Default Value:

No certificate is installed by default.

References:

1. https://nginx.org/en/docs/http/configuring_https_servers.html#chains
2. https://support.globalsign.com/customer/portal/articles/1290470-install-certificate---nginx

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.3 Ensure private key permissions are restricted (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The server's and potentially its vhost's private keys should be protected from unauthorized access by limiting access based on the principle of least privilege.

Rationale:

A server's private key file should be restricted to `400` permissions. This ensures only the owner of the private key file can access it. This is the minimum necessary permissions for the server to operate. If the private key file is not protected, an unauthorized user with access to the server may be able to find the private key file and use it to decrypt traffic sent to your server.

Audit:

Verify the permissions on the key file are `400`. This can be found by running the following command. You should replace `/etc/nginx/nginx.key` with the location of your key file.

```
find /etc/nginx/ -name '*.key' -exec stat -Lc "%n %a" {} +
```

The output should show mode 400 or more restrictive

Example:

```
/etc/nginx/nginx.key 400
```

Remediation:

Run the following command to remove excessive permissions on key files in the `/etc/nginx/ directory`.

Note: The directory `/etc/nginx/` should be replaced with the location of your key file.

```
find /etc/nginx/ -name '*.key' -exec chmod u-wx,go-rwx {} +
```

Default Value:

The default permissions on the server's private key are `644` or `-rw-r--r--`.

Additional Information:

Important Note: This recommendation should be applied to both the keys of your server certificate and the key of your client certificate if you are looking to mutually authenticate a proxy server.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.3 Configure Data Access Control Lists Configure data access control lists based on a user's need to know. Apply data access control lists, also known as access permissions, to local and remote file systems, databases, and applications.	●	●	●
v7	14.6 Protect Information through Access Control Lists Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.	●	●	●

4.1.4 Ensure only modern TLS protocols are used (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Only modern TLS protocols should be enabled in NGINX for all client connections and upstream connections. Removing legacy TLS and SSL protocols (SSL 3.0, TLS 1.0, 1.1 and 1.2), and enable stable TLS protocols (TLS 1.3), ensures users are able to take advantage of strong security capabilities and protects them from insecure legacy protocols.

Rationale:

Why disable SSL 3.0: The POODLE Vulnerability allowed attackers to exploit SSL 3.0 to obtain cleartext information by exploiting weaknesses in CBC in 2014. SSL 3.0 is also no longer FIPS 140-2 compliant.

Why disable TLS 1.0: TLS 1.0 was deprecated from use when PCI DSS Compliance mandated that it not be used for any applications processing credit card numbers in June 2018. TLS 1.0 does not make use of modern protections, and almost all user agents that do not support TLS 1.2 or higher are no longer supported by their vendor.

Why disable TLS 1.1: Because of the increased security associated with higher versions of TLS, TLS 1.0 should be disabled. Modern browsers will begin to flag TLS 1.1 as deprecated in early 2019.

Why disable TLS 1.2: While robust for its time, TLS 1.2's complexity allows for weak configurations, including cipher suites that lack Perfect Forward Secrecy. TLS 1.3 eliminates this risk by mandating PFS and removing outdated cryptographic primitives. Acknowledging this, NIST SP 800-52 Rev. 2 allows for TLS 1.2 to be disabled if it is not required for interoperability, positioning TLS 1.3 as the sole recommended protocol for modern, secure environments.

Why enable TLS 1.3: TLS 1.3 improves security by removing several insecure cipher suites by default and adding several more secure algorithms. All public-key exchange mechanisms support perfect forward secrecy in this version of TLS. Additionally, TLS 1.3 makes drastic performance improvements by removing a full round trip in the TLS handshake.

Impact:

Disabling certain TLS may not allow legacy user agents to connect to your server. Disabling negotiation of specific protocols with your backend server may also limit your ability to connect with legacy servers. You should always consider if you need to support legacy user agents or servers when selecting your TLS protocols.

Audit:

You can verify which SSL/TLS protocols your server uses by issuing the below command to see the configured cipher suites on the server. If anything older than TLS 1.3 is implemented or nothing appears, this recommendation is not implemented.

```
grep -ir ssl_protocol /etc/nginx
```

Note: Depending on your configuration, you may see different results. The directive `ssl_protocols` should always be part of your server block. If your NGINX server is also a proxy or load balancer, you should also check for the presence of the `proxy_ssl_protocols` directive as part of the location block of your nginx configuration. This ensures your proxy follows a specific set of negotiation rules for encrypting traffic with your upstream server.

Remediation:

Run the following commands to change your `ssl_protocols` if they are already configured. This remediation advice assumes your nginx configuration file does not include server configuration outside of `/etc/nginx/nginx.conf`. You may have to also inspect the include files in your `nginx.conf` to ensure this is properly implemented.

Web Server:

```
sed -i "s/ssl_protocols[^;]*;/ssl_protocols TLSv1.3;/" /etc/nginx/nginx.conf
```

Proxy:

```
sed -i "s/proxy_ssl_protocols[^;]*;/proxy_ssl_protocols TLSv1.3;/"
/etc/nginx/nginx.conf
```

If your `ssl_protocols` are not already configured, this can be accomplished manually by opening your web server or proxy server configuration file and manually adding the directives.

Web Server:

```
server {
    ssl_protocols TLSv1.3;
}
```

Proxy:

```
location / {
      proxy_pass          cisecurity.org;
      proxy_ssl_protocols TLSv1.3;
    }
```

Default Value:

By default, NGINX does not specify the TLS protocol and accepts all TLS versions, except for TLS 1.3, which must be enabled by an administrator to take effect.

Defaults: ssl_protocols TLSv1.0 TLSv1.1 TLSv1.2 proxy_ssl_protocols TLSv1.0 TLSv1.1 TLSv1.2

References:

1. https://webkit.org/blog/8462/deprecation-of-legacy-tls-1-0-and-1-1-versions/
2. https://www.cloudflare.com/learning/ssl/why-use-tls-1.3/
3. https://tools.ietf.org/html/rfc8446
4. https://doi.org/10.6028/NIST.SP.800-52r2

Additional Information:

Note: TLS configuration should always be set to your organizational policy. This recommendation is designed to meet best practices.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.5 Disable weak ciphers (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `ssl_protocols` directive must be used to disable weak protocols and exclusively enable TLS 1.3. In a TLS 1.3-only configuration, the `ssl_ciphers` directive is no longer necessary, as the protocol itself mandates a small, non-negotiable set of highly secure AEAD ciphers (**A**uthenticated **E**ncryption with **A**ssociated **D**ata). This approach simplifies configuration and eliminates the risk of choosing weak or insecure cipher suites.

The `ssl_prefer_server_ciphers` directive should be set to off for TLS 1.3. Since all available ciphers are secure, allowing the client (user agent) to choose the most performant cipher for its hardware provides a performance benefit without compromising security.

In a reverse proxy setup, it is critical to ensure that any upstream services also support TLS 1.3. If an upstream server requires an older protocol, the `proxy_ssl_protocols` and `proxy_ssl_ciphers` directives must be configured to match the upstream's requirements, but this should be treated as a temporary exception to be remediated.

Rationale:

Weak cryptographic ciphers can lead to the compromise of sensitive data. In modern TLS configurations, the most effective way to disable all weak ciphers is to exclusively enable the TLS 1.3 protocol. The TLS 1.3 specification removes all previously known weak and legacy cipher suites, mandating the use of a small set of highly secure Authenticated Encryption (AEAD) ciphers. This approach is simpler and less error-prone than maintaining a complex denylist or allowlist of ciphers for older protocols.

Impact:

Impact of Mandating TLS 1.3:

Enforcing an exclusive TLS 1.3 configuration enhances security but will intentionally cause connection failures for legacy clients. This primarily affects:

- Unsupported User Agents: Clients on outdated operating systems (e.g., Android < 10, iOS < 12.2, Windows < 10 without updates) and very old browsers (e.g., Internet Explorer) that do not support TLS 1.3 will be unable to connect. These clients are typically outside their vendor's security lifecycle and pose an independent risk.

- Legacy Upstream Services: In a reverse proxy scenario, if NGINX is configured for TLS 1.3 exclusively, it will be unable to establish a connection to an upstream (backend) server that only supports TLS 1.2 or older. This will cause service disruptions. A thorough assessment of backend compatibility is required before enforcement.

Audit:

Use the following procedure to verify that only `ssl_protocols TLSv1.3;` is used.

```
nginx -T 2>/dev/null | grep -E '^\s*ssl_protocols'
```

The output must show `ssl_protocols TLSv1.3;` exclusively for every relevant configuration block (e.g., at the `http` or `server` level).

Example output:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
# configuration file /etc/nginx/nginx.conf:
        ssl_protocols TLSv1.3;
# configuration file /etc/nginx/conf.d/default.conf:
        ssl_protocols TLSv1.3;
```

Remediation:

Set `ssl_protocols TLSv1.3;`. The `ssl_ciphers` directive is not required for a TLS 1.3-only configuration, as the secure defaults of the underlying crypto library (e.g., OpenSSL) will be used.

Default Value:

These directives are not specified by default and are set to the default of `HIGH:!aNULL:!MD5`.

References:

1. https://mozilla.github.io/server-side-tls/ssl-config-generator/
2. https://nginx.org/en/docs/http/ngx_http_ssl_module.html

Additional Information:

Note: TLS configuration should always be set to your organizational policy. This recommendation is designed to meet best practices and offers several profiles your organization may align to.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.6 Ensure awareness of TLS 1.3 new Diffie-Hellman parameters (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

This control is not applicable to environments exclusively using TLS 1.3.

Rationale:

The TLS 1.3 protocol (RFC 8446) deprecates the use of custom finite-field Diffie-Hellman (DHE) groups, which were configured via the `ssl_dhparam` directive in NGINX. Instead, TLS 1.3 exclusively uses a set of pre-defined, standardized, and secure elliptic curve (ECDHE) and finite-field (FFDHE) groups for its key exchange mechanism. This design eliminates the risk associated with weak or misconfigured custom DH parameters. As such, the `ssl_dhparam` directive has no effect in a TLS 1.3-only configuration.

Impact:

None for a TLS 1.3-only configuration.

Audit:

Verify that the configuration does not rely on TLS 1.2 **or older protocols** that would require this setting. This control is implicitly passed if control 4.1.4 ("Ensure only modern TLS protocols are used") is configured for TLS 1.3 exclusively.

Remediation:

No remediation is necessary. Ensure `ssl_protocols TLSv1.3;` is set. The `ssl_dhparam` directive should be removed as it is obsolete.

References:

1. https://datatracker.ietf.org/doc/html/rfc8446#page-95
2. https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_dhparam

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.7 Ensure Online Certificate Status Protocol (OCSP) stapling is enabled (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

OCSP stapling allows a server to efficiently deliver certificate revocation information to the client, improving performance and privacy. The server caches the OCSP response from the Certificate Authority (CA), eliminating the need for the client to make a separate connection. For robust security, certificates should be issued with the OCSP Must-Staple extension, which transforms the traditional "soft-fail" behavior into a "hard-fail," ensuring clients always receive and validate a current revocation status.

Rationale:

OCSP stapling is a critical mechanism for distributing certificate revocation status. Without it, clients might not be aware that a server's certificate has been compromised, allowing for potential man-in-the-middle attacks. The OCSP Must-Staple extension is essential as it mitigates the inherent weakness of optional OCSP ("soft-fail"), where a browser might proceed with a connection if it doesn't receive a staple. By enforcing a "hard-fail", Must-Staple ensures that a compromised certificate can be reliably blocked.

Impact:

If OCSP Must-Staple is used, a misconfiguration on the server (e.g., a firewall blocking outbound OCSP queries, an incorrect DNS resolver) will cause clients to reject the certificate and refuse to connect, leading to a service outage. This "hard-fail" behavior is intentional for security but requires diligent configuration and monitoring of the OCSP stapling mechanism.

Audit:

Run the following command to inspect the fully loaded NGINX configuration for the four required directives:

```
nginx -T 2>/dev/null | grep -E
'^\s*(ssl_stapling|ssl_stapling_verify|ssl_trusted_certificate|resolver)'
```

Verify that the output contains the following directives with appropriate values within the relevant `http` or `server` blocks:

- `ssl_stapling on;`
- `ssl_stapling_verify on;`
- `ssl_trusted_certificate /path/to/your/chain.pem;`
- `resolver A.B.C.D;`

If any of these four directives are missing, this recommendation is not fully implemented.

Remediation:

Follow this procedure to enable a robust OCSP stapling configuration:

1. When issuing a certificate, request that the OCSP Must-Staple extension be included.
2. Edit your NGINX configuration to include all four necessary directives. The `ssl_trusted_certificate` must point to a file containing your root and intermediate certificates. The resolver must be set to one or more trusted DNS resolvers.

```
# Example for a server block
server {
    # ... other directives ...

    # OCSP Stapling
    ssl_stapling             on;
    ssl_stapling_verify      on;

    # Path to the certificate chain (Root CA + Intermediates) for
verification
    ssl_trusted_certificate  /etc/nginx/ssl/full_chain.pem;

    # DNS resolver for NGINX to query the CA's OCSP server
    resolver                 8.8.8.8  1.1.1.1  valid=300s;
}
```

Default Value:

OCSP stapling is not enabled by default.

References:

1. https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_stapling
2. https://datatracker.ietf.org/doc/html/rfc6066
3. https://datatracker.ietf.org/doc/html/rfc7633

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.8 Ensure HTTP Strict Transport Security (HSTS) is enabled (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

HTTP Strict Transport Security (HSTS) is a critical security header that instructs browsers to communicate with a domain exclusively over HTTPS. A comprehensive HSTS policy must include the `includeSubDomains` directive to apply the policy to all current and future subdomains. For maximum protection, the policy should also contain the `preload` directive, allowing the domain to be submitted to browser-pre-load lists. This ensures that even the very first connection to the domain is made securely. The `max-age` should be set to a long duration, typically two years (`63072000` seconds), to ensure browsers enforce this policy persistently.

Rationale:

HSTS is the primary mechanism to mitigate protocol downgrade attacks and cookie hijacking. By enforcing HTTPS, it prevents attackers from intercepting requests and manipulating them. The `includeSubDomains` directive is vital as it closes a significant gap where an attacker could otherwise target a non-secure subdomain. The `preload` directive provides protection by removing the initial window of opportunity for an attack on a user's first visit, as the browser already knows to use HTTPS before making any connection.

Impact:

Once an HSTS policy with a long `max-age` is set, there is effectively "no going back." If any part of your site or any subdomain cannot be served over HTTPS, users with a cached HSTS policy will be unable to access it. Enabling `includeSubDomains` requires a commitment that all subdomains of the domain will support HTTPS. Submitting a domain to the HSTS preload list is a long-term commitment and removal is a slow, manual process. Careful planning and testing with short `max-age` values are essential before full deployment.

Audit:

Run the following command to inspect the fully loaded NGINX configuration for the `Strict-Transport-Security` header:

```
nginx -T 2>/dev/null | grep -i 'Strict-Transport-Security'
```

Verify that the output includes a header with the following components:

- A `max-age` directive of at least `31536000` (one year), with `63072000` (two years) being the recommended value.
- The `includeSubDomains` directive.
- The `always` parameter at the end of the `add_header` directive.

Example output:

```
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains"
always;
```

Remediation:

It is critical to deploy HSTS incrementally to avoid locking users out.

Step 1: Initial Rollout (Low `max-age`)

Add the HSTS header with a very short `max-age` to test for any issues. Verify that all parts of your site, including all subdomains, function correctly over HTTPS.

```
# Test with 5 minutes
add_header Strict-Transport-Security "max-age=300; includeSubDomains" always;
```

Step 2: Increase `max-age`

Once confident, gradually increase the `max-age`.

```
# Increase to 1 week
add_header Strict-Transport-Security "max-age=604800; includeSubDomains"
always;
```

Step 3: Full Deployment (Long `max-age` and Preload)

After thorough testing (e.g., one month), set the `max-age` to the recommended final value of two years. Add the `preload` directive if you intend to submit your site to the HSTS preload list.

```
# Final configuration (2 years)
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains;
preload" always;
```

If preloading is desired, submit your domain at hstspreload.org.

Default Value:

HSTS headers are not set by default.

References:

1. https://hstspreload.org
2. https://tools.ietf.org/html/rfc6797

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.9 Ensure upstream server traffic is authenticated with a client certificate (Manual)

Profile Applicability:

- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

In a reverse proxy configuration, NGINX acts as a client when communicating with an upstream server. To secure this server-to-server connection based on a Zero Trust principle, **m**utual **TLS** (mTLS) must be used. This is achieved by configuring NGINX to present its own client certificate to the upstream server. The upstream server then authenticates NGINX based on this certificate, ensuring that only trusted proxies can access backend services.

Rationale:

Authenticating the proxy's connection to the upstream server via a client certificate provides strong, cryptographic proof of identity. This is vastly superior to weaker authentication methods like IP whitelisting, which can be spoofed. In a modern microservices or cloud environment, mTLS is a cornerstone of network security, as it prevents unauthorized services from making requests to sensitive backends, thereby mitigating lateral movement attacks.

Impact:

Implementing mTLS introduces operational overhead for certificate management. You must have a process (often an internal **P**ublic **K**ey **I**nfrastructure, or PKI) for issuing, renewing, and revoking these client certificates. If the client certificate used by NGINX expires, or if the CA certificate on the upstream server expires, the connection between NGINX and the upstream will fail, leading to a service outage.

Audit:

Run the following command to inspect the fully loaded NGINX configuration for the required directives:

```
nginx -T 2>/dev/null | grep -E
'^\s*(proxy_ssl_certificate|proxy_ssl_certificate_key)'
```

Verify that the output includes both `proxy_ssl_certificate` and `proxy_ssl_certificate_key` directives with the correct paths within the relevant location block.

Note: A complete audit is two-sided. You must also verify that the upstream server is configured to require and validate client certificates against a trusted CA. This part of the audit is outside the scope of the NGINX configuration itself.

Remediation:

Implementing mTLS requires configuration on both the NGINX proxy (the client) and the upstream server (the server). This example assumes you have a simple internal CA.

Prerequisite: Create a CA

```
# Create CA Key
openssl genrsa -out my-ca.key 4096
# Create CA Certificate
openssl req -x509 -new -nodes -key my-ca.key -sha256 -days 3650 -out my-
ca.crt
```

Step 1: Configure the Upstream Server to Require Client Certificates

The upstream server must be configured to request and verify client certificates against your CA. (If your upstream is also NGINX, the config would look like this):

```
# On the Upstream Server's configuration
server {
    listen 443 ssl;
    # ... other ssl directives ...

    ssl_client_certificate  /path/to/my-ca.crt; # The CA to verify against
    ssl_verify_client on;                       # Require a valid client cert
}
```

Step 2: Create and Sign a Client Certificate for NGINX

On your NGINX proxy, create a key and a certificate signing request (CSR).

```
# Create a key for the NGINX proxy
openssl genrsa -out      nginx-client.key 4096
# Create CSR
openssl req    -new -key nginx-client.key -out nginx-client.csr
```

Sign this CSR with your CA to create the client certificate:

```
openssl x509  -req -in nginx-client.csr -CA my-ca.crt -CAkey my-ca.key -
CAcreateserial -out nginx-client.crt -days 365
```

Step 3: Configure NGINX to Present its Client Certificate

In your NGINX reverse proxy configuration, use the generated client certificate and key.

```
# In your reverse proxy's location block
location /api/ {
    proxy_pass                https://your-upstream-server;

    # Present this client cert to the upstream
    proxy_ssl_certificate     /etc/nginx/ssl/nginx-client.crt;
    proxy_ssl_certificate_key /etc/nginx/ssl/nginx-client.key;
}
```

Default Value:

This is not authenticated by default.

References:

1. https://docs.nginx.com/nginx-instance-manager/system-configuration/secure-traffic/
2. https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_ssl_certificate

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.10 Ensure the upstream traffic server certificate is trusted (Manual)

Profile Applicability:

- Level 2 - Proxy
- Level 2 - Loadbalancer

Description:

When acting as a reverse proxy, NGINX must be configured to function as a secure TLS client. This requires validating the identity of the upstream server by verifying its certificate against a **trusted Certificate Authority** (CA). Furthermore, NGINX must ensure that the hostname of the upstream server matches the name (Subject Name / SAN) within the certificate itself.

Rationale:

Without proper validation, NGINX blindly trusts the identity of the upstream server, making it vulnerable to man-in-the-middle (MitM) attacks within the internal network. An attacker could impersonate a legitimate backend service and intercept sensitive traffic. By enforcing certificate validation (`proxy_ssl_verify`) and hostname verification (`proxy_ssl_name`), NGINX guarantees that it is communicating with the intended, authentic upstream server.

Impact:

Enabling strict upstream verification introduces a dependency on certificate lifecycle management. If the upstream server's certificate expires or is misconfigured, or if the CA certificate file on the NGINX server is outdated, NGINX will refuse to connect to the upstream, resulting in a service outage. This behavior is intentional and secures the connection at the cost of requiring careful certificate management.

Audit:

Run the following command to inspect the fully loaded NGINX configuration for the three required directives within the relevant proxy `location` block:

```
nginx -T 2>/dev/null | grep -E
'^\s*(proxy_ssl_verify|proxy_ssl_trusted_certificate|proxy_ssl_name)'
```

Verify that the output contains the following directives:

- `proxy_ssl_verify on;`
- `proxy_ssl_trusted_certificate /path/to/ca.crt;`
- `proxy_ssl_name your-upstream-hostname.com;`

If any of these three directives are missing, this recommendation is not fully implemented. Additionally, manually verify that the certificate referenced by `proxy_ssl_trusted_certificate` is the correct, valid CA certificate for your upstream services.

Remediation:

To securely configure upstream validation, you must obtain the CA certificate that signed your upstream server's certificate.

1. Place the CA certificate (e.g., `upstream_ca.crt`) in a secure directory on the NGINX server (e.g., `/etc/nginx/ssl/`).
2. In the `location` block that proxies traffic, add the following three directives. The `proxy_ssl_name` must match the hostname used in the `proxy_pass` directive.

```
location /api/ {
    proxy_pass                      https://your-upstream-hostname.com;
    # 1. Enable verification
    proxy_ssl_verify                on;

    # 2. Specify the CA to trust for verification
    proxy_ssl_trusted_certificate  /etc/nginx/ssl/upstream_ca.crt;

    # 3. Verify the certificate's name matches the server's hostname
    proxy_ssl_name                  your-upstream-hostname.com;
}
```

Default Value:

By default, `proxy_ssl_verify` is set to `off`. This means NGINX **does not validate the upstream server's certificate**, creating a significant security risk equivalent to a browser ignoring all certificate warnings.

References:

1. https://docs.nginx.com/nginx/admin-guide/security-controls/securing-http-traffic-upstream/
2. https://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_ssl_trusted_certificate

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.11 Ensure Secure Session Resumption is Enabled (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

TLS 1.3 introduces a secure session resumption mechanism using **P**re-**S**hared **K**eys (PSKs) that significantly improves performance for returning clients by reducing the handshake latency. This modern mechanism should be enabled to enhance user experience without compromising security.

Rationale:

Unlike older TLS versions, the TLS 1.3 resumption mechanism preserves **P**erfect **F**orward **S**ecrecy (PFS). It accomplishes this by combining the PSK with a **fresh** Ephemeral Diffie-Hellman key exchange (ECDHE) for **every** resumed session. This ensures that a compromise of the resumption key does not compromise any past or future session keys. Disabling this feature provides no security benefit and negatively impacts performance.

Impact:

Enabling session resumption has a positive performance impact. There are no significant negative security implications when using a TLS 1.3-only configuration.

Audit:

Run the following command to verify that `ssl_session_tickets` **is not** explicitly turned `off`.

```
# This command should produce NO output.
grep -ir "ssl_session_tickets" /etc/nginx/ | grep -i "off"
```

If the command produces any output containing `ssl_session_tickets off;`, this recommendation is not implemented.

Remediation:

Ensure that `ssl_session_tickets` is not set to `off`. The recommended approach is to remove the directive entirely, as the default value is `on`.

If the directive is present, either remove it or set it to `on`:

```
# REMOVE this line from your configuration:
# ssl_session_tickets off;

# OR, if you want to be explicit, ensure it is set to ON (optional):
ssl_session_tickets  on;
```

Default Value:

`ssl_session_tickets` is enabled (`on`) by default. This is the desired and secure state for TLS 1.3.

References:

1. https://datatracker.ietf.org/doc/html/rfc8446
2. https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_tickets
3. https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

4.1.12 Ensure HTTP/3.0 is used (Manual)

Profile Applicability:

- Level 2 - Webserver

Description:

HTTP/2 is the established standard for web communication, offering significant performance benefits over HTTP/1.1 through multiplexing. For 2025 and beyond, HTTP/3 should also be enabled. HTTP/3 operates over the **QUIC** protocol, which is built on **UDP**, to solve head-of-line blocking, reduce connection setup time, and improve performance on unreliable networks. Both protocols require a secure TLS 1.3 environment to function.

Rationale:

Enabling HTTP/2 provides a baseline of modern performance via stream multiplexing. Enabling HTTP/3 provides a further competitive advantage by mitigating TCP's head-of-line blocking and offering a faster, more reliable connection handshake, which is especially beneficial for mobile users. A server supporting **both** protocols can serve the vast majority of modern clients with the best possible performance and security. The strong encryption requirements of both protocols naturally align with a TLS 1.3-only policy.

Impact:

HTTP/2 has no significant negative impact as it is universally supported by modern clients. Enabling HTTP/3 has operational considerations:

1. **NGINX Build:** Your NGINX binary must be compiled with HTTP/3 and **QUIC** support. Standard OS packages may not include this. The repository of NGINX itself has the **http_v3** module since NGINX version 1.25.0

- Run this command and check if the **http_v3** module is present

```
nginx -V 2>&1 | tr ' ' '\n' | grep --color=auto 'with-'
```

2. **Firewall Configuration: You must allow UDP** traffic on port 443, as HTTP/3 uses the **QUIC** protocol over **UDP**. This is a common oversight that will cause HTTP/3 to fail.

Audit:

Run the following command to inspect the fully loaded NGINX configuration:

```
nginx -T 2>/dev/null | grep -E '^\s*(listen|add_header.*Alt-Svc)'
```

Verify the following in the output for your primary `server` block:

1. The TCP `listen` directive **includes** the `http2` parameter: `listen 443 ssl http2;`
2. A **UDP** `listen` directive with the `quic` parameter exists: `listen 443 quic reuseport;`
3. An `Alt-Svc` header is being sent to advertise HTTP/3 availability: `add_header Alt-Svc 'h3=":443"; ma=63072000';`

If any of these are missing, this recommendation is not fully implemented.

Remediation:

Prerequisite: Ensure your NGINX version is compiled with the `--with-http_v3_module` flag.

1. Open your NGINX server configuration file.
2. In the main `server` block for your HTTPS site, add or modify the directives to enable HTTP/2, HTTP/3, and advertise its availability.
3. Ensure your firewall **allows UDP** traffic on port `443`.

```
server {
    # 1. Enable HTTP/2 on the standard TCP listener
    listen          443      ssl http2;
    listen          [::]:443 ssl http2;

    # 2. Enable HTTP/3 on the UDP listener
    listen          443      quic reuseport;
    listen          [::]:443 quic reuseport;

    # ... other ssl directives like ssl_certificate ...

    # 3. Advertise HTTP/3 availability to browsers
    # The max-age (ma) is in seconds (e.g., 2 years)
    add_header      Alt-Svc 'h3=":443"; ma=63072000';

    # Required for HTTP/3
    ssl_early_data  on;
}
```

Default Value:

By default, NGINX only enables HTTP/1.1. HTTP/2 and HTTP/3 must be explicitly configured.

References:

1. https://datatracker.ietf.org/doc/html/rfc9114
2. https://datatracker.ietf.org/doc/html/rfc9000
3. https://datatracker.ietf.org/doc/html/rfc9001
4. https://datatracker.ietf.org/doc/html/rfc7838
5. https://nginx.org/en/docs/quic.html
6. https://nginx.org/en/docs/http/ngx_http_v3_module.html
7. https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_early_data

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	3.10 Encrypt Sensitive Data in Transit Encrypt sensitive data in transit. Example implementations can include: Transport Layer Security (TLS) and Open Secure Shell (OpenSSH).		●	●
v7	14.4 Encrypt All Sensitive Information in Transit Encrypt all sensitive information in transit.		●	●

5 Request Filtering and Restrictions

5.1 Access Control

5.1.1 Ensure allow and deny filters limit access to specific IP addresses (Manual)

Profile Applicability:

- Level 2 - Webserver
- Level 2 - Proxy
- Level 2 - Loadbalancer

Description:

Access control based on IP addresses is a fundamental defense-in-depth mechanism. By using NGINX's `allow` and `deny` directives, access to the entire server or specific `location` blocks can be restricted to trusted network sources, such as internal subnets, specific hosts, or VPN ranges. This is particularly effective for protecting non-public administrative interfaces or internal APIs from the public internet.

Rationale:

Applying the principle of least privilege at the network layer is a highly effective security measure. By explicitly defining which IP addresses or CIDR ranges are permitted to access sensitive resources and implicitly denying all others with `deny all;`, the attack surface is significantly reduced. This prevents unauthorized network segments from even attempting to exploit potential application-layer vulnerabilities.

Impact:

A misconfigured IP filter list can lead to service denial for legitimate users or services. In dynamic environments where IP addresses can change (e.g., cloud instances without static IPs), this can be a particular challenge. Maintaining accurate and up-to-date IP allow-lists requires operational discipline.

Audit:

Run the following command to inspect the fully loaded NGINX configuration for `allow` and `deny` rules:

```
nginx -T 2>/dev/null | grep -E '^\s*(allow|deny)'
```

Then, manually review the active rules within the `http`, `server`, or `location` blocks. Verify that the configured IP addresses and CIDR ranges align with the documented list of trusted sources and that a `deny all;` rule is present to enforce a default-deny policy.

Remediation:

Identify the specific `location` block you wish to protect (e.g., an admin login page or internal stats). Compile a list of trusted source IP addresses and network ranges. Add `allow` directives for each trusted source, followed by a final `deny all;` directive. NGINX processes rules in order, and stops at the first match.

```
location /admin_login/ {
    # Allow a specific monitoring server
    allow 192.168.1.100;

    # Allow the internal office network range
    allow 10.20.30.0/24;

    # Deny all other access to this location
    deny all;

    # ... other directives for the admin location, e.g., proxy_pass ...
}
```

Default Value:

By default, no IP-based restrictions are configured. NGINX will process requests from any source IP address unless `allow` or `deny` directives are specified.

References:

1. https://nginx.org/en/docs/http/ngx_http_access_module.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	13.10 Perform Application Layer Filtering Perform application layer filtering. Example implementations include a filtering proxy, application layer firewall, or gateway.			●
v7	9.5 Implement Application Firewalls Place application firewalls in front of any critical servers to verify and validate the traffic going to the server. Any unauthorized traffic should be blocked and logged.			●

5.1.2 Ensure only approved HTTP methods are allowed (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

Following the principle of least functionality, an NGINX server should be configured to reject any HTTP methods that are not explicitly required by the application. While standard web browsing typically only needs `GET`, `POST`, and `HEAD`, modern RESTful APIs might require methods like `PUT`, `PATCH`, or `DELETE`. Any method not essential for the application's functionality should be blocked at the web server level.

Rationale:

Disabling unused HTTP methods mitigates the risk of unintended server interaction and can prevent certain classes of web application attacks. For example, if an attacker finds a way to bypass application-layer authentication, an enabled but unused `PUT` or `DELETE` method on the web server could potentially lead to unauthorized file modification or deletion. By explicitly denying such methods, NGINX ensures that requests never even reach the backend application and therefore significantly reducing the attack surface.

Impact:

An overly restrictive filter can block legitimate application functionality. Before implementing these restrictions, it is crucial to coordinate with application developers to get a definitive list of all required HTTP methods for every application endpoint. Incorrectly blocking a required method (e.g., `PUT` for a file upload feature) will cause parts of the application to fail.

Audit:

1. Configuration Inspection:

Run the following command to analyze the fully loaded NGINX configuration for any method-limiting directives:

```
nginx -T 2>/dev/null | grep -E '(\$request_method|limit_except)'
```

Review the configuration to ensure that either a `limit_except` block or an `if ($request_method)` block is correctly implemented for the relevant location.

2. Active Testing:

Send a request with a non-approved method (e.g., `OPTIONS` or `DELETE`) using `curl` and verify that the server responds with the correct status code, **not a** `200 OK` or `404 Not Found`.

```
# Send a disallowed OPTIONS request
curl -X OPTIONS -I https://example.loc/api.html
```

Expected Output: The server should return either `HTTP/1.1 405 Not Allowed` or `HTTP/1.1 444 Connection Closed Without Response`. Any other `2xx` or `4xx` code indicates a potential misconfiguration.

Remediation:

There are two recommended methods to restrict HTTP verbs.

Method 1 (Preferred): Using `limit_except` This directive is designed for this purpose and is considered the cleanest approach. It restricts all methods **except** for the ones listed.

```
location /api_login/ {

    # Only allow GET, HEAD, and POST methods for this location.
    limit_except GET HEAD POST {
        deny all;
    }

    # ... other directives ...
}
```

Method 2 (Alternative): Using an `if` condition This method offers more flexibility, such as **returning a non-standard status code** like `444`, which simply closes the connection without sending a response header.

```
location / {

    # If the request method is NOT one of GET, HEAD, or POST
    if ($request_method !~ ^(GET|HEAD|POST)$) {
        # --> close the connection immediately.
        return 444;
    }

    # ... other directives ...
}
```

Default Value:

All methods are allowed.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html#limit_except

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.10 Apply Secure Design Principles in Application Architectures Apply secure design principles in application architectures. Secure design principles include the concept of least privilege and enforcing mediation to validate every operation that the user makes, promoting the concept of "never trust user input." Examples include ensuring that explicit error checking is performed and documented for all input, including for size, data type, and acceptable ranges or formats. Secure design also means minimizing the application infrastructure attack surface, such as turning off unprotected ports and services, removing unnecessary programs and files, and renaming or removing default accounts.		●	●
v7	9.2 Ensure Only Approved Ports, Protocols and Services Are Running Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.		●	●

5.2 Request Limits

5.2.1 Ensure timeout values for reading the client header and body are set correctly (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

To protect against slow clients holding connections open indefinitely, NGINX supports several timeout directives. The most important ones for client-facing connections are:

- `client_header_timeout`: Sets the maximum time the server will wait for a client to send the request header.
- `client_body_timeout`: Sets the maximum time allowed between sequential read operations when receiving the request body. This timer does not apply to the total time of the transfer.
- `send_timeout`: Sets the maximum time allowed between sequential write operations when sending a response to the client.

If any of these timeouts are reached, the server closes the connection, freeing up resources.

Rationale:

Aggressively low timeout values are a primary defense against slow-read Denial of Service (DoS) attacks. These attacks attempt to exhaust server resources by opening many connections and keeping them alive for as long as possible by sending data extremely slowly. By setting low timeouts, NGINX efficiently closes these malicious connections, preserving resources for legitimate users.

Impact:

Setting these values too low can terminate legitimate connections too early. For example, a user uploading a large file over a slow mobile connection could be cut off if `client_body_timeout` is too aggressive. The values must be carefully evaluated to achieve a balance between security (low values) **and** functionality (higher values for specific use cases such as file uploads).

Audit:

Run the following command to inspect the fully loaded NGINX configuration for the relevant directives:

```
nginx -T 2>/dev/null | grep -E
'^\s*(client_header_timeout|client_body_timeout|send_timeout)'
```

Verify that these values are explicitly set to a reasonably low, non-default value (e.g., 10–20 seconds) in the main `http` or `server` context. If the application requires long-running connections, verify that `client_body_timeout` is overridden to a higher value in the specific `location` block that handles uploads.

Remediation:

Set reasonably low timeout values globally in your `http` block. If specific locations **require** longer timeouts (e.g., for file uploads), override them within that `location` block.

Example Configuration:

```
http {

    # Set a global default of 15 seconds, which overrides the default of 60s.

    client_header_timeout 15s;
    client_body_timeout   15s;
    send_timeout          15s;

    server {
        # ... other settings ...

        # This location handles large file uploads and needs a longer
timeout.
        location /upload {

            client_body_timeout 300s; # Allow 5 minutes between read
operations for uploads
            # ...
        }
    }
}
```

Default Value:

The default value for `client_header_timeout`, `client_body_timeout`, and `send_timeout` is 60 seconds. This is often considered **too high for public-facing servers**, because it allows slow or malicious clients to uphold connections and resources for an extended period.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html#client_header_timeout
2. https://nginx.org/en/docs/http/ngx_http_core_module.html#client_body_timeout
3. https://nginx.org/en/docs/http/ngx_http_core_module.html#send_timeout

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.2.2 Ensure the maximum request body size is set correctly (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `client_max_body_size` directive defines the maximum permissible size for a client request body, as indicated by the `Content-Length` header. If a request exceeds this size, NGINX will immediately reject it with a `413 Request Entity Too Large` error, preventing the oversized request from being processed further or passed to a backend application.

Rationale:

Limiting the request body size is a crucial defense against resource exhaustion DoS attacks and prevents oversized, potentially malicious payloads from reaching application backends. By setting a logical default limit and only increasing it for specific application endpoints that require it (e.g., file uploads), the principle of least functionality is enforced, significantly reducing the attack surface.

Impact:

Setting this value too low is a common cause of application failure. Legitimate user actions, such as uploading files or submitting large forms, will be blocked with a `413` error **if** they exceed the configured limit. It is essential that this value is determined based on application requirements and not set to an arbitrary low number.

Audit:

Run the following command to inspect the fully loaded NGINX configuration for the `client_max_body_size` directive:

```
nginx -T 2>/dev/null | grep -E '^\s*(client_max_body_size)'
```

Verify that a global `client_max_body_size` is set in the `http` block. For any location blocks that override this value (e.g., for uploads), **manually** verify that the increased limit is documented and aligns with the application's functional requirements.

Remediation:

Define a restrictive global limit in the `http` block. For specific application endpoints that need to accept larger request bodies, override this directive within the corresponding `location` block.

Example Configuration:

```
http {

    # Set a restrictive global default of 2 Megabytes. This prevents
unexpected large requests on most endpoints.
    client_max_body_size 2M;

    server {
        # ...

        # This location handles API requests with potentially large JSON
payloads.
        location /api/v1/data {
            client_max_body_size 10M; # Allow up to 10MB
            # ...
        }

        # This location is for large file uploads.
        location /uploads {
            client_max_body_size 50M; # Allow up to 50MB
            # ...
        }
    }
}
```

Default Value:

The default value is 1 megabyte (`1m`). It's recommended to set this value **explicitly** in the `http` block to serve as a documented, global default. For many API endpoints or static content locations, even this default may be more permissive than necessary.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html#client_max_body_size

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.2.3 Ensure the maximum buffer size for URIs is defined (Manual)

Profile Applicability:

- Level 1 - Webserver
- Level 1 - Proxy
- Level 1 - Loadbalancer

Description:

The `large_client_header_buffers` directive allocates the **maximum number** and **size** of buffers for reading the **entire** client request header, which includes **both** the request line (e.g., `GET /path HTTP/1.1`) **and all** subsequent header fields (e.g., `Host:`, `Cookie:`, `Authorization:`). If a client's request header exceeds the total allocated buffer space, NGINX immediately rejects the request with a `400 Bad Request` error. Also, the client's request line cannot exceed the size of **one** buffer, otherwise the `414 Request-URI Too Large` error will be returned to the client.

Rationale:

While NGINX itself is not vulnerable to buffer overflows from large headers, this directive serves two critical purposes:

1. Denial of Service (DoS) Mitigation: It prevents malicious clients from attempting to exhaust server memory by sending excessively large headers.

2. Downstream Protection: It acts as a gatekeeper, ensuring that malformed or oversized headers **do not** reach backend applications that might be more fragile or could exhibit undefined behavior when processing them.

The default value is intentionally generous to handle legitimate modern web traffic, and lowering it without reason can be counterproductive.

Impact:

Setting this value too low is a common cause of difficult-to-diagnose application failures. Legitimate requests from modern web applications, especially those using complex authentication (OAuth/SAML), or analytics, can easily generate headers larger than a few kilobytes. An overly restrictive value will cause these legitimate requests to fail with a generic `400 Bad Request` error, which is often mistakenly attributed to the application itself rather than the NGINX configuration.

Audit:

This is a manual check that requires context.

1. Run the following command to see if the directive is explicitly set:

```
nginx -T 2>/dev/null | grep 'large_client_header_buffers'
```

2. Evaluate the result:

- **No output:** The server is using the default value of `4 8k`. This is considered compliant and secure for general use.
- **Output shows `large_client_header_buffers 4 8k;`:** The default is explicitly set. This is compliant.
- **Output shows a different value (e.g., `2 1k;`):** This is a deviation from the default. Manually verify if this restrictive value is documented, intentional, and necessary for the specific application. If not, it represents a potential availability risk and should be flagged.

Remediation:

Unless you have a specific, documented requirement to restrict header sizes further, the recommended action is to rely on the secure default value. If your configuration has an overly restrictive value set without a clear reason, remove the `large_client_header_buffers` directive from your `http` or `server` blocks to allow NGINX to fall back to its default.

Remediation Action:

Remove any custom, overly restrictive `large_client_header_buffers` lines from your configuration files.

```
# REMOVE THIS LINE if it exists without good reason:
large_client_header_buffers 2 1k;
```

Default Value:

The default value is `large_client_header_buffers 4 8k;`, meaning NGINX will allocate up to **4x buffers** of `8 kilobytes` each per request. This generous default is designed to accommodate the vast majority of legitimate web traffic, including requests with large cookies or complex authorization headers, without problems. It is considered the secure and recommended baseline.

References:

1. https://nginx.org/en/docs/http/ngx_http_core_module.html#large_client_header_buffers

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.2.4 Ensure the number of connections per IP address is limited (Manual)

Profile Applicability:

- Level 2 - Webserver
- Level 2 - Proxy
- Level 2 - Loadbalancer

Description:

NGINX's `ngx_http_limit_conn_module` provides a mechanism to limit the number of simultaneous connections from a single client IP address. This is achieved in two steps:

- **`limit_conn_zone`:** This directive, typically defined in the `http` block, creates a shared memory zone to store the state for each client IP address.
- **`limit_conn`:** This directive, applied in a `server` or `location` block, **enforces** a specific connection limit using the previously defined zone.

When a client exceeds this limit, NGINX will reject new connections with a `503 Service Temporarily Unavailable` error.

Rationale:

The primary purpose of connection limiting is to **mitigate** resource exhaustion and certain types of Denial of Service (DoS) attacks where an attacker opens many connections and holds them open for as long as possible. It is a different tool than rate limiting (`limit_req`), which is designed to stop rapid-fire requests (like brute-force attacks). By enforcing a reasonable connection limit, the server can prevent a single malicious or misconfigured client from consuming an unfair share of worker connections.

Impact:

This is a critical consideration. A single public IP address can represent **thousands** of individual users behind a large corporate NAT, a university campus, or a Carrier-Grade NAT (CG-NAT) from a mobile provider. Setting the connection limit too low will block legitimate users in these scenarios. The appropriate limit is a delicate balance and depends entirely on the target audience. A public-facing website requires a much higher limit than a restricted admin interface.

Audit:

This is a manual check requiring context.

1. Run the following command to inspect the loaded NGINX configuration for connection limiting rules:

```
nginx -T 2>/dev/null | grep -E '^\s*(limit_conn_zone|limit_conn)'
```

2. Manually evaluate the findings:

- Is a `limit_conn_zone` defined in the `http` block?
- Is a `limit_conn` directive applied in the appropriate `server` or `location` blocks?

Critically, assess the configured limit. Is a limit of `10` appropriate for a public website accessed by customers or large companies? Or is it a sensible value for a specific `/login` location? The value must be justifiable for its context.

Remediation:

First, define a shared memory zone in the `http` block. Then, apply a carefully considered limit in the `server` or `location` context.

Understanding the zone size: The memory usage for the `$binary_remote_addr` key is fixed (`64 bytes` **on a 64-bit system**). Therefore, `1 megabyte` of zone memory can store approximately `16,384` states. A `10m` zone can store over `160,000` states.

Example Configuration:

```
http {

    # Define a 10MB zone named 'per_ip' to track connections by IP.
    # $binary_remote_addr is more memory-efficient than $remote_addr.
    limit_conn_zone $binary_remote_addr zone=per_ip:10m;

    server {

        # Apply a general limit of 20 connections per IP for this server.
        # This might be a reasonable starting point for a public site.
        limit_conn per_ip 20;

        # For a resource-intensive download area, apply a stricter limit.
        location /downloads/ {
            limit_conn per_ip 5;
        }
    }
}
```

Default Value:

By default, no connection limits are configured. NGINX will accept as many simultaneous connections from a single IP address as its worker processes can handle.

References:

1. https://nginx.org/en/docs/http/ngx_http_limit_conn_module.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.2.5 Ensure rate limits by IP address are set (Manual)

Profile Applicability:

- Level 2 - Webserver
- Level 2 - Proxy
- Level 2 - Loadbalancer

Description:

NGINX's `ngx_http_limit_req_module` provides a mechanism to limit the rate of incoming requests from a single client IP address, based on the "leaky bucket" algorithm. This is configured in two steps:

1. `limit_req_zone`: This directive, defined in the `http` block, creates a shared memory zone that defines the bucket's parameters, such as the average request rate.

2. `limit_req`: This directive, applied in a `server` or `location` block, enforces the rate limit using a specific zone and defines how "bursts" of traffic are handled.

When a client exceeds the defined rate, NGINX will reject **new** requests with a `503 Service Temporarily Unavailable` error.

Rationale:

Rate limiting is the primary defense against application-level, high-frequency attacks. Its main purpose is to prevent brute-force password guessing on login endpoints, API abuse by automated scripts, and aggressive content scraping. Unlike connection limiting (`limit_conn`), which focuses on slow, simultaneous connections, rate limiting (`limit_req`) targets clients making an excessive number of requests in a short period.

Impact:

Applying a global, aggressive rate limit is **extremely** dangerous and will almost certainly block legitimate users accessing your site from behind a shared NAT (e.g., corporate offices, datacenters, mobile networks). Rate limiting is a surgical tool that must be applied **only to specific, sensitive locations** (e.g., `/login`, `/api/v1/authenticate`) where abuse is likely. Incorrectly configured rate limits are a common source of user complaints and support tickets.

Audit:

This is a manual check requiring context.

1. Run the following command to find rate-limiting rules:

```
nginx -T 2>/dev/null | grep -E '^\s*(limit_req_zone|limit_req)'
```

2. Manually evaluate the findings:

- Is a `limit_req_zone` defined in the `http` block?
- **Important, where is `limit_req` applied?** Is it applied globally (`server` Block or `location /`)? This is a high risk. Or is it applied to specific endpoints like `/login`? Which is best practice.
- Are the `rate` and `burst` values reasonable for the protected endpoint?

Remediation:

First, define a shared memory zone in the `http` block. Then, apply a carefully tuned limit **only to the specific `location` blocks that require protection**.

Understanding the "Leaky Bucket" Parameters:

- `rate`: The average rate at which the bucket "leaks" (requests are processed). `5r/s` means 5 requests per second.
- `burst`: The size of the bucket. It's the number of "token" requests a client can make in excess of the defined rate. A `burst=10` allows a client to burst up to 10 requests instantly before the rate limit kicks in.
- `nodelay`: Without `nodelay`, excess requests in a burst are queued and processed at the defined `rate` (slowing the client down). With `nodelay`, excess requests are processed immediately as long as they fit in the burst "bucket", and only requests beyond the burst limit are rejected. For `APIs` and login pages, `nodelay` is generally preferred for a better user experience.

Example Configuration:

```
http {

    # Define a 10MB zone for login attempts.
    # Rate: 1 request per second average.
    limit_req_zone    $binary_remote_addr zone=login_limit:10m rate=1r/s;

    server {

        # NO global rate limit here!

        location / {
            # Normal traffic, no rate limit
        }

        # Apply a strict rate limit ONLY to the login endpoint
        location /login {

            # Use the 'login_limit' zone.
            # Allow a burst of 5 requests, then enforce the 1r/s limit.
            # 'nodelay' ensures a user can quickly resubmit a form without
being artificially slowed down.
            limit_req    zone=login_limit burst=5 nodelay;

            # ... other settings ...
        }
    }
}
```

Default Value:

By default, no rate limits are configured. NGINX will process requests from a single IP address as fast as the client can send them.

References:

1. https://nginx.org/en/docs/http/ngx_http_limit_req_module.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.3 Browser Security

5.3.1 Ensure X-Content-Type-Options header is configured and enabled (Manual)

Profile Applicability:

- Level 1 - Webserver

Description:

The `X-Content-Type-Options` header instructs the browser to strictly follow the MIME types declared in the `Content-Type` headers and **not to guess ("sniff") the content type based on the file's actual content**.

Rationale:

Implementing the `X-Content-Type-Options` header with the `nosniff` directive helps to prevent drive-by download attacks where a user agent is sniffing content types in responses.

This header prevents "MIME type confusion" attacks. Without this header, browsers might interpret a file declared as text (e.g., `snippet.txt`) as executable if it contains script code. Setting the `nosniff` directive **forces the browser to reject the file if the declared type doesn't match the context in which it's loaded** (e.g., loading a text file as a script).

Impact:

Low Risk: In rare cases, legacy applications or misconfigured servers might rely on the browser's ability to correct wrong Content-Type headers (e.g., serving JavaScript with a `text/plain` header). Enabling `nosniff` will break these applications because the browser will refuse to execute the script. Ensuring correct MIME types in the NGINX configuration (`mime.types`) is a prerequisite.

Audit:

Run the following command to inspect the fully loaded NGINX configuration:

```
nginx -T 2>/dev/null | grep -i 'X-Content-Type-Options'
```

Verify the output:

- Verify that the configuration contains the line: `add_header X-Content-Type-Options "nosniff" always;`
- Specifically check for the `always` parameter at the end of the directive. Without it, the header will be missing on error pages.

Remediation:

Open the NGINX configuration file that contains your `server` blocks. Add the below line into your `server` block to `add X-Content-Type-Options` header and direct your user agent to **not** sniff content types.

```
add_header X-Content-Type-Options "nosniff" always;
```

Default Value:

This header is not implemented by default.

References:

1. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Content-Type-Options
2. https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html#x-content-type-options
3. https://nginx.org/en/docs/http/ngx_http_headers_module.html

Additional Information:

The `always` parameter ensures that the header is applied to the HTTP **response** no matter what the HTTP response code is.

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

5.3.2 Ensure that Content Security Policy (CSP) is enabled and configured properly (Manual)

Profile Applicability:

- Level 2 - Webserver

Description:

Content **S**ecurity **P**olicy (CSP) is an HTTP response header that allows site administrators to declare approved sources of content that browsers are allowed to load on that page. It is a mechanism to detect and mitigate certain types of attacks, including **C**ross-**S**ite **S**cripting (XSS) and data injection attacks. Furthermore, CSP's `frame-ancestors` directive is the modern replacement for the `X-Frame-Options` header to prevent Clickjacking.

Rationale:

A robust CSP significantly reduces the attack surface of a web application. By restricting the domains from which scripts, styles, images, and other resources **can be loaded**, it effectively neutralizes many XSS vectors. Additionally, by using the `frame-ancestors` directive, it explicitly controls **which parent pages are allowed to embed the application** (e.g., via `<iframe>`), providing a more flexible protection against Clickjacking than the legacy `X-Frame-Options` header.

Impact:

Implementing a strict CSP is complex and carries **a high risk of breaking application functionality**. **If a legitimate resource** (e.g., a CDN script, a font file, or an inline script) **is not whitelisted**, the browser will block it, potentially making the site unusable. CSP implementation should always start in "Report-Only" mode (`Content-Security-Policy-Report-Only`) to gather violation reports before enforcing the policy.

Audit:

1. Run the following command to inspect the CSP configuration:

```
nginx -T 2>/dev/null | grep -i 'Content-Security-Policy'
```

2. Evaluate the policy:

- Is the header present?
- Does it include at least a restrictive `default-src` directive (e.g., '`self`' or '`none`')?
- Does it include the `frame-ancestors` directive to mitigate Clickjacking?
- Critically: Is `unsafe-inline` or `unsafe-eval` avoided in `script-src`? (Allowing these significantly weakens the protection).

Remediation:

CSP must be tailored to the specific application. There is no single "correct" policy.

Step 1: The Baseline Policy (High Security)

Start with a policy that denies everything by default and only allows resources from the same origin. It also prevents the site from being framed by anyone (Clickjacking protection).

```
add_header Content-Security-Policy "default-src 'self'; frame-ancestors
'self'; form-action 'self';" always;
```

Step 2: Adaptation (Example)

If your application loads images from a CDN and needs to be embeddable by a specific partner site:

```
add_header Content-Security-Policy "default-src 'self'; img-src 'self'
https://cdn.example.com; frame-ancestors 'self' https://partner-site.com;"
always;
```

Note: Use `Content-Security-Policy-Report-Only` during the testing phase to debug your policy without breaking the site.

Default Value:

By default, no Content Security Policy is sent. Browsers default to the standard Same-Origin Policy, which is much less restrictive.

References:

1. https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html
2. https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html
3. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy
4. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy-Report-Only
5. https://nginx.org/en/docs/http/ngx_http_headers_module.html#add_header
6. https://caniuse.com/?search=frame-ancestors

Additional Information:

OWASP HTTP Headers Cheat Sheet states:

Content Security Policy (CSP) `frame-ancestors` directive obsoletes `X-Frame-Options` for supporting browsers (source).

`X-Frame-Options` header is only useful when the HTTP response where it is included has something to interact with (e.g. links, buttons). If the HTTP response is a redirect or an API returning JSON data, `X-Frame-Options` does not provide any security.

Recommendation

Use Content Security Policy (CSP) frame-ancestors directive if possible.

Almost all modern browsers do support CSP. A comprehensive overview of compatible browsers can be found here: https://caniuse.com/?search=frame-ancestors

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	2.7 Allowlist Authorized Scripts Use technical controls, such as digital signatures and version control, to ensure that only authorized scripts, such as specific .ps1, .py, etc., files, are allowed to execute. Block unauthorized scripts from executing. Reassess bi-annually, or more frequently.			●
v7	2.9 Implement Application Whitelisting of Scripts The organization's application whitelisting software must ensure that only authorized, digitally signed scripts (such as *.ps1, *.py, macros, etc) are allowed to run on a system.			●

5.3.3 Ensure the Referrer Policy is enabled and configured properly (Manual)

Profile Applicability:

- Level 2 - Webserver

Description:

The `Referrer-Policy` HTTP header controls how much referrer information (sent via the `Referer` header) should be included with requests. It allows site administrators to restrict the data sent to upstream servers when a user clicks a link or loads a resource. This is a privacy control to prevent leaking sensitive URL parameters or internal path structures to third parties.

Rationale:

URLs often contain sensitive information such as session tokens, search queries, or **P**ersonally **I**dentifiable **I**nformation (PII) in their query parameters. Without a strict Referrer Policy, this full URL is transmitted to any third-party site the user visits from your page, potentially logging sensitive data on external servers. Configuring this header ensures that only the necessary information (e.g., just the origin domain) is shared, protecting user privacy and preventing data leakage.

Impact:

Choosing an overly restrictive policy like `no-referrer` can break functionality that relies on knowing the source of traffic, such as web analytics, affiliate tracking, or multi-site authentication flows. Conversely, a loose policy (`unsafe-url`) leaks private data. The recommended policy `strict-origin-when-cross-origin` is a balance by preserving full referrer data for internal navigation while stripping sensitive path and query data when navigating to external sites.

Audit:

Run the following command to inspect the fully loaded NGINX configuration:

```
nginx -T 2>/dev/null | grep -i 'Referrer-Policy'
```

Evaluate the findings:

- Verify that the directive `add_header Referrer-Policy "..." always;` is present.
- Check the configured value. While `no-referrer` is the most secure, `strict-origin-when-cross-origin` is the widely accepted standard for balancing security and functionality.
- Ensure the `always` parameter is present.

Remediation:

Add the following line to your `server` or `http` block. This example uses the robust default setting that protects privacy without breaking internal analytics.

```
add_header Referrer-Policy "strict-origin-when-cross-origin" always;
```

If **maximum** privacy is required and no referrer data is **needed** even for internal links:

```
add_header Referrer-Policy "no-referrer" always;
```

Default Value:

NGINX does not set this header by default. However, modern browsers (Chrome, Firefox) typically default to `strict-origin-when-cross-origin` if no header is present. Explicitly setting this header ensures consistent behavior across **all** browsers and prevents fallback to insecure legacy defaults.

References:

1. https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html#referrer-policy
2. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Referrer-Policy
3. https://nginx.org/en/docs/http/ngx_http_headers_module.html

CIS Controls:

Controls Version	Control	IG 1	IG 2	IG 3
v8	16.1 Establish and Maintain a Secure Application Development Process Establish and maintain a secure application development process. In the process, address such items as: secure application design standards, secure coding practices, developer training, vulnerability management, security of third-party code, and application security testing procedures. Review and update documentation annually, or when significant enterprise changes occur that could impact this Safeguard.		●	●
v7	18.1 Establish Secure Coding Practices Establish secure coding practices appropriate to the programming language and development environment being used.		●	●

6 Mandatory Access Control

This section is intentionally left blank / reserved.

Rationale: NGINX primarily acts as a reverse proxy and web server, relying on external **Id**entity **P**roviders (IdP) **or** the underlying Operating System for user authentication and authorization logic.

- For local user authentication controls (e.g., PAM, Shadow File permissions), refer to the CIS-Benchmark™ for your Operating System.
- For network-based access controls (IP Allow/Deny), refer to Section 5 (Network Security) of this benchmark.
- For application-level authentication (OIDC, LDAP, SAML), refer to the documentation of the specific application or Identity Provider.

Appendix: Summary Table

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
1	**Initial Setup**		
1.1	**Installation**		
1.1.1	Ensure NGINX is installed (Manual)	☐	☐
1.2	**Configure Software Updates**		
1.2.1	Ensure package manager repositories are properly configured (Manual)	☐	☐
1.2.2	Ensure the latest software package is installed (Manual)	☐	☐
2	**Basic Configuration**		
2.1	**Minimize NGINX Modules**		
2.1.1	Ensure only required dynamic modules are loaded (Manual)	☐	☐
2.2	**Account Security**		
2.2.1	Ensure that NGINX is run using a non-privileged, dedicated service account (Manual)	☐	☐
2.2.2	Ensure the NGINX service account is locked (Manual)	☐	☐
2.2.3	Ensure the NGINX service account has an invalid shell (Manual)	☐	☐
2.3	**Permissions and Ownership**		
2.3.1	Ensure NGINX directories and files are owned by root (Manual)	☐	☐
2.3.2	Ensure access to NGINX directories and files is restricted (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
2.3.3	Ensure the NGINX process ID (PID) file is secured (Manual)	☐	☐
2.4	**Network Configuration**		
2.4.1	Ensure NGINX only listens for network connections on authorized ports (Manual)	☐	☐
2.4.2	Ensure requests for unknown host names are rejected (Manual)	☐	☐
2.4.3	Ensure keepalive_timeout is 10 seconds or less, but not 0 (Manual)	☐	☐
2.4.4	Ensure send_timeout is set to 10 seconds or less, but not 0 (Manual)	☐	☐
2.5	**Information Disclosure**		
2.5.1	Ensure server_tokens directive is set to `off` (Manual)	☐	☐
2.5.2	Ensure default error and index.html pages do not reference NGINX (Manual)	☐	☐
2.5.3	Ensure hidden file serving is disabled (Manual)	☐	☐
2.5.4	Ensure the NGINX reverse proxy does not enable information disclosure (Manual)	☐	☐
3	**Logging**		
3.1	Ensure detailed logging is enabled (Manual)	☐	☐
3.2	Ensure access logging is enabled (Manual)	☐	☐
3.3	Ensure error logging is enabled and set to the info logging level (Manual)	☐	☐
3.4	Ensure proxies pass source IP information (Manual)	☐	☐
4	**Encryption**		

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
4.1	**TLS / SSL Configuration**		
4.1.1	Ensure HTTP is redirected to HTTPS (Manual)	☐	☐
4.1.2	Ensure a trusted certificate and trust chain is installed (Manual)	☐	☐
4.1.3	Ensure private key permissions are restricted (Manual)	☐	☐
4.1.4	Ensure only modern TLS protocols are used (Manual)	☐	☐
4.1.5	Disable weak ciphers (Manual)	☐	☐
4.1.6	Ensure awareness of TLS 1.3 new Diffie-Hellman parameters (Manual)	☐	☐
4.1.7	Ensure Online Certificate Status Protocol (OCSP) stapling is enabled (Manual)	☐	☐
4.1.8	Ensure HTTP Strict Transport Security (HSTS) is enabled (Manual)	☐	☐
4.1.9	Ensure upstream server traffic is authenticated with a client certificate (Manual)	☐	☐
4.1.10	Ensure the upstream traffic server certificate is trusted (Manual)	☐	☐
4.1.11	Ensure Secure Session Resumption is Enabled (Manual)	☐	☐
4.1.12	Ensure HTTP/3.0 is used (Manual)	☐	☐
5	**Request Filtering and Restrictions**		
5.1	**Access Control**		
5.1.1	Ensure allow and deny filters limit access to specific IP addresses (Manual)	☐	☐
5.1.2	Ensure only approved HTTP methods are allowed (Manual)	☐	☐

CIS Benchmark Recommendation		Set Correctly	
		Yes	No
5.2	**Request Limits**		
5.2.1	Ensure timeout values for reading the client header and body are set correctly (Manual)	☐	☐
5.2.2	Ensure the maximum request body size is set correctly (Manual)	☐	☐
5.2.3	Ensure the maximum buffer size for URIs is defined (Manual)	☐	☐
5.2.4	Ensure the number of connections per IP address is limited (Manual)	☐	☐
5.2.5	Ensure rate limits by IP address are set (Manual)	☐	☐
5.3	**Browser Security**		
5.3.1	Ensure X-Content-Type-Options header is configured and enabled (Manual)	☐	☐
5.3.2	Ensure that Content Security Policy (CSP) is enabled and configured properly (Manual)	☐	☐
5.3.3	Ensure the Referrer Policy is enabled and configured properly (Manual)	☐	☐
6	**Mandatory Access Control**		

Appendix: Change History

Date	Version	Changes for this version
Oct 13, 2017		_Default Value, Remediation Procedure, Listing Order, Section_ on **[Recommendation] 4.2 Give the NGINX User Account an Invalid Shell** were updated.
Feb 15, 2018	1.0.0	**[recommendation] 1.1.1P Installing Nginx** was created.
Nov 10, 2018	1.0.0	_Listing Order_ on **[recommendation] 2.2.2P Ensure the NGINX service account is locked** was updated.
Nov 10, 2018	1.0.0	_Listing Order, Status, references, Default Value, Impact Statement, Audit Procedure, Remediation Procedure, Rationale Statement, Description, Scoring Status, Title_ on **[recommendation] 4.1.10 Ensure a Trusted Certificate and Trust Chain is Installed**

Date	Version	Changes for this version
Nov 10, 2018	1.0.0	_Listing Order, Status, references, Default Value, Impact Statement, Audit Procedure, Remediation Procedure, Rationale Statement, Description, Scoring Status, Title_ on **[recommendation] 4.1.11 Ensure Permissions on Servers Private Key is Protected** wer
Nov 11, 2018	1.0.0	**[recommendation] 4.1.4P Ensure Permissions on Servers Private Key is Protected** was created.
Nov 13, 2018	1.0.0	**[recommendation] 4.1.4P Restrict Weak Ciphers From Being Used** was created.
Dec 10, 2018	1.0.0	_Listing Order_ on **[recommendation] 1.2.1P Ensure the Latest Software Package is Installed** was updated.
Dec 10, 2018	1.0.0	_Listing Order_ on **[recommendation] 1.1.1P Installing NGINX** was updated.
Dec 10, 2018	1.0.0	_Listing Order_ on **[recommendation] 1.2.1P Ensure Package Manager Repositories are Configured** was updated.
Dec 10, 2018	1.0.0	_Listing Order, Rationale Statement, Description_ on **[recommendation] 2.5.2P Ensure server_tokens directive is set to Off** were updated.

Date	Version	Changes for this version
Dec 10, 2018	1.0.0	_Listing Order_ on **[recommendation] 2.5.2P Ensure Default Error Pages and index.html Pages don't reference NGINX** was updated.
Dec 10, 2018	1.0.0	_Listing Order_ on **[recommendation] 2.5.2P Ensure Hidden File Serving Is Disabled** was updated.
Dec 10, 2018	1.0.0	_Listing Order_ on **[recommendation] 2.5.2P Ensure Proxy does not Enable Information Disclosure** was updated.
Jan 7, 2019	1.0.0	_Listing Order_ on **[recommendation] 1.1.1P Installing NGINX** was updated.
Sep 8, 2022	2.0.0	Update to NGINX Install instructions to show 16.1 (Ticket 9029)
Sep 8, 2022	2.0.0	No longer score this as the industry deprecates it. (Ticket 9030)
Sep 19, 2022	2.0.0	Advice for ssl_prefer_server_ciphers is outdated (Ticket 12709)
Oct 3, 2022	2.0.0	Update Artifact (Ticket 16386)
Oct 3, 2022	2.0.0	Update Artifact (Ticket 16388)
Oct 3, 2022	2.0.0	Manual or Automated (Ticket 16389)

Date	Version	Changes for this version
Oct 7, 2022	2.0.0	Verify the Username (Ticket 16390)
Oct 11, 2022	2.0.0	UPDATE - 2.3.2 Ensure access to NGINX directories and files is restricted (Ticket 16392)
Oct 12, 2022	2.0.0	Question about using Error code 404 in 2.4.2 Ensure requests for unknown host names are rejected (Ticket 15400)
Oct 12, 2022	2.0.0	Update prose (Ticket 16398)
Oct 12, 2022	2.0.0	Remove SHA from recommended hash algorithms in cipher suites. (Ticket 16405)
Oct 13, 2022	2.0.0	Review these links with community on next update (Ticket 12371)
Oct 28, 2022	2.0.0	False positive on Centos 8, zabbix appliance (Ticket 12389)
Oct 28, 2022	2.0.0	NGINX False Positives on Debian 9 (Ticket 10847)
Oct 28, 2022	2.0.0	NGINX benchmark reports false positives on Ubuntu 18.04 (Ticket 10791)
Oct 28, 2022	2.0.0	Can't forget OWASP Mod Security (Ticket 7397)
Jun 15, 2023	2.0.1	Addressed AAC bug and republished as v2.0.1

Date	Version	Changes for this version
Apr 5, 2024	2.1.0	Update Artifact - 2.5.4 Ensure the NGINX reverse proxy does not enable information disclosure (Ticket 18685)
Jun 4, 2024	2.1.0	add_header mistake Strict-Transport-Security (Ticket 19160)
Jun 4, 2024	2.1.0	1.1.1 Ensure NGINX is installed... (Ticket 18701)
Jun 4, 2024	2.1.0	2.2.1 Check NGINX user privileges checks the wrong user (Ticket 20513)
Jun 27, 2024	2.1.0	2.3.2 Ensure access to NGINX directories and files is restricted (Scored) (Ticket 21775)
Apr 1, 2025	3.0.0	Update 1.1.1 (Ticket 16385)
Aug 26, 2025	3.0.0	CIS-CAT PRO Scan: Profile does not recognize Debian based Systems, issue in BASH code (Ticket 24449)
Aug 26, 2025	3.0.0	Does not recognize Debian based systems, nor vendor repository (Ticket 24450)
Aug 26, 2025	3.0.0	Does not recognize Debian based systems and more current up-to-date vendor repository (Ticket 24451)
Oct 28, 2025	3.0.0	DROP - Ensure NGINX is installed from source (Ticket 24414)

Date	Version	Changes for this version
Oct 28, 2025	3.0.0	BUG in NGINX detection --> 1.1 Installation Fail 1.1.1 Ensure NGINX is installed (Ticket 22454)
Oct 28, 2025	3.0.0	DROP - Ensure HTTP WebDAV module is not installed (Ticket 25959)
Oct 28, 2025	3.0.0	DROP - Ensure modules with gzip functionality are disabled (Ticket 25960)
Oct 28, 2025	3.0.0	DROP -Ensure the autoindex module is disabled (Ticket 25961)
Oct 28, 2025	3.0.0	Need Artifact added (Ticket 16399)
Oct 28, 2025	3.0.0	Need clarification on what are considered as server keys (Ticket 24134)
Oct 28, 2025	3.0.0	UPDATE - 2.1.1 Ensure only required modules are loaded (Ticket 25958)
Oct 28, 2025	3.0.0	Should this be automated (Ticket 16393)
Oct 28, 2025	3.0.0	DROP - Ensure the core dump directory is secured (Ticket 26451)
Nov 4, 2025	3.0.0	UPDATE - Disable weak ciphers - Look into making this an automated check (Ticket 16739)
Nov 12, 2025	3.0.0	UPDATE - 4.1.6 Ensure custom Diffie-Hellman parameters are used (Ticket 26523)

Date	Version	Changes for this version
Nov 12, 2025	3.0.0	This recommendation should carry a condition (Ticket 24678)
Nov 12, 2025	3.0.0	UPDATE - 4.1.8 Ensure HTTP Strict Transport Security (HSTS) is enabled (Ticket 26524)
Nov 12, 2025	3.0.0	UPDATE - 4.1.9 Ensure upstream server traffic is authenticated with a client certificate (Ticket 26570)
Nov 12, 2025	3.0.0	UPDATE - 4.1.10 Ensure the upstream traffic server certificate is trusted (Ticket 26571)
Nov 12, 2025	3.0.0	DROP - 4.1.11 Ensure your domain is preloaded (Ticket 26572)
Nov 12, 2025	3.0.0	UPDATE - 4.1.11 Ensure Secure Session Resumption is Enabled (Ticket 26573)
Nov 12, 2025	3.0.0	http 3.0 (Ticket 16380)
Nov 12, 2025	3.0.0	DROP - Ensure only Perfect Forward Secrecy Ciphers are Leveraged (Ticket 26575)
Nov 20, 2025	3.0.0	UPDATE - 5.1.1 Ensure allow and deny filters limit access to specific IP addresses (Ticket 26621)
Nov 20, 2025	3.0.0	UPDATE - 5.1.2 Ensure only approved HTTP methods are allowed (Ticket 26622)

Date	Version	Changes for this version
Nov 20, 2025	3.0.0	Recommendation 5.2.1 is "Automated", but the correct timeout values are not clearly specified. (Ticket 23703)
Nov 20, 2025	3.0.0	UPDATE - 5.2.2 Ensure the maximum request body size is set correctly (Ticket 26623)
Nov 20, 2025	3.0.0	UPDATE - 5.2.3 Ensure the maximum buffer size for URIs is defined (Ticket 26624)
Nov 20, 2025	3.0.0	UPDATE - 5.2.4 Ensure the number of connections per IP address is limited (Ticket 26625)
Nov 20, 2025	3.0.0	UPDATE - 5.2.5 Ensure rate limits by IP address are set (Ticket 26626)
Nov 25, 2025	3.0.0	Additional information for http3 module in NGINX repository (Ticket 26628)
Nov 25, 2025	3.0.0	Update Artifact (Ticket 16391)
Nov 25, 2025	3.0.0	Check configuration for http3 for recommendation 2.4.1 (Ticket 26627)
Nov 25, 2025	3.0.0	UPDATE - 5.3.2 Ensure X-Content-Type-Options header is configured and enabled (Ticket 26675)

Date	Version	Changes for this version
Nov 25, 2025	3.0.0	UPDATE - 5.3.3 Ensure that Content Security Policy (CSP) is enabled and configured properly (Ticket 26676)
Nov 25, 2025	3.0.0	UPDATE - 5.3.4 Ensure the Referrer Policy is enabled and configured properly (Ticket 26677)
Nov 25, 2025	3.0.0	DROP - 5.3.1 Ensure X-Frame-Options header is configured and enabled (Ticket 26674)
Dec 2, 2025	3.0.0	DROP - 3.4 Ensure log files are rotated (Ticket 26769)
Dec 2, 2025	3.0.0	DROP - 3.5 Ensure error logs are sent to a remote SIEM server (Ticket 26770)
Dec 2, 2025	3.0.0	DROP - 3.6 Ensure access logs are sent to a remote syslog server (Ticket 26771)