HeaderNexus Documentation

HeaderNexus Tool - Detailed Documentation

Table of Contents

---

1. Introduction

1.1 What is HeaderNexus?

HeaderNexus is a powerful Python-based security analysis tool designed to audit HTTP response headers and SSL/TLS configurations of websites. Its primary function is to identify missing or misconfigured security headers and insecure cookie attributes that can expose web applications to attacks such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Session Hijacking, and other vulnerabilities.

The tool also analyzes the SSL/TLS certificates of HTTPS websites, providing valuable information on certificate validity, TLS version strength, and potential issues like expired or self-signed certificates.

1.2 Purpose and Use Cases

- **Web Security Audits:** Quickly assess security headers of web applications for compliance and risk assessment.
- **Penetration Testing:** Integrate into pentesting workflows for automated header and SSL checks.
- **DevOps and Continuous Monitoring:** Use in CI/CD pipelines or monitoring scripts to detect configuration regressions.
- **Learning and Research:** Educational tool for understanding HTTP security best practices and common misconfigurations.

1.3 Overview of File Inclusion and Insecure HTTP Headers

While the tool was initially developed with file inclusion vulnerability detection in mind, it evolved to focus on a broader range of HTTP security headers and

SSL/TLS certificate analysis. Misconfigured headers and cookies often serve as attack vectors, and securing them is critical for web application security.

2. Prerequisites and Setup

2.1 System Requirements

- Operating System: Windows, Linux, macOS
- Python 3.8 or higher (recommended Python 3.10+)
- Internet connectivity for running live tests against URLs

2.2 Installing Python

Download the latest Python version from python.org. Ensure you add Python to your system PATH during installation for easy command line access.

To verify installation:

```
python --version
pip --version
```

2.3 Required Python Libraries and Installation

The project depends on several Python standard and third-party libraries:

Library	Description	Installation Command
requests	To make HTTP/HTTPS requests	`pip install requests`
urllib.parse	URL parsing (part of Python standard library)	Included by default
ssl	SSL/TLS certificate handling (standard library)	Included by default
socket	Network connections and sockets (standard library)	Included by default
http.cookies	Parsing and analyzing HTTP cookies	Included by default
datetime	Date and time handling	Included by default
time	Timing functions (sleep, timestamps)	Included by default

Install **requests** with pip:

```
pip install requests
```

2.4 Optional Tools and Utilities

- Text Editor or IDE (e.g., VSCode, PyCharm) for running and modifying the script.
- Command Line Terminal (PowerShell, Bash, etc.)

---

3. Understanding HTTP Headers and Web Security

3.1 What are HTTP Headers?

HTTP headers are key-value pairs sent between a client and a server to provide metadata about the request or response. Security headers control aspects like content security policies, transport security, and browser behaviors.

3.2 Common Security Headers and Their Importance

- **Strict-Transport-Security (HSTS):** Forces browsers to only use HTTPS.
- **Content-Security-Policy (CSP):** Mitigates XSS and data injection attacks.
- **X-Content-Type-Options:** Prevents MIME-sniffing.
- **X-Frame-Options:** Prevents clickjacking attacks.
- **Referrer-Policy:** Controls information sent in the Referer header.
- **Permissions-Policy:** Restricts APIs and features in browsers.

3.3 Understanding Cookies and Their Security Attributes

Cookies are small data stored on the client side for sessions or tracking. Flags such as `Secure`, `HttpOnly`, and `SameSite` help prevent attacks like session hijacking, XSS, and CSRF.

3.4 Basics of SSL/TLS and HTTPS

SSL/TLS protocols encrypt data between client and server. Proper certificate management and using strong TLS versions are crucial to ensure confidentiality and trust.

---

4. Project Structure

4.1 File Organization

```
headerguard/
│
├── headerguard.py       # Main tool script
├── README.md            # Basic project info
├── requirements.txt     # List of dependencies (optional)
└── reports/             # Directory to save scan reports (created on first run)
```

4.2 Overview of Code Modules and Scripts

Currently, all logic resides in `headerguard.py`. Future versions may modularize components.

---

5. In-Depth Explanation of Libraries Used

5.1 requests

A powerful HTTP library for Python. Used to send GET requests to target URLs, retrieve HTTP responses and headers.

5.2 urllib.parse

Standard library for breaking down URLs into components (scheme, hostname, path, query). Essential for parsing target URLs for SSL checks.

5.3 ssl and socket

`ssl` module wraps network sockets to add encryption via TLS/SSL. Used here to retrieve and parse SSL certificate details from HTTPS servers.

5.4 http.cookies

Parses `Set-Cookie` headers into manageable cookie objects to analyze security flags like `Secure`, `HttpOnly`, `SameSite`.

5.5 datetime

For date/time manipulations, mainly to evaluate cookie expiry and certificate validity periods.

5.6 Others (time, sys)

- `time` for calculating response durations.
- `sys` for system-related functions (optional, e.g., graceful exits).

---

6. Code Walkthrough and Function Explanation

6.1 Main Script Flow

- Accepts a target URL as input.
- Sends an HTTP request to the URL.
- Extracts and analyzes HTTP response headers.
- Checks for missing or misconfigured security headers.
- Analyzes cookies for security flags.
- Retrieves and examines SSL certificate details if HTTPS.
- Outputs detailed findings.
- Saves the report in a `.txt` file in the `reports` directory.

6.2 URL Parsing and Request Handling

The tool uses `urllib.parse` to safely dissect the URL and `requests.get` to fetch the server response.

6.3 Header Checking Function (`check_headers`)

This function:

- Retrieves all HTTP headers.
- Matches them against a list of critical security headers.
- Flags missing headers or insecure configurations.
- Calls cookie and SSL checks.
- Collects all findings into a report list.

6.4 Cookie Analysis Function (`check_cookies`)

Parses all cookies and analyzes:

- Presence of `Secure` flag: ensures cookies sent over HTTPS only.
- Presence of `HttpOnly` : protects from client-side scripts.
- `SameSite` attribute: prevents CSRF.
- Cookie expiry and domain/path settings.
- Reports any missing or insecure configurations.

6.5 SSL/TLS Info Retrieval and Analysis (`check_ssl_tls`)

- Opens a TLS-wrapped socket to the server.
- Extracts certificate details like issuer, subject, SAN, expiry.
- Checks TLS version and warns if outdated.
- Reports self-signed vs trusted certificates.
- Flags certificates expiring soon or expired.

6.6 Vulnerability Checks and Reporting

Includes checks for known header-based vulnerabilities and outputs clear warnings or confirmations.

6.7 Saving Reports to Files

- Creates a timestamped `.txt` file under `reports/` folder.
- Writes the full scan output to the file.

- Ensures proper encoding to avoid Unicode errors.

---

7. How to Use HeaderGuard

7.1 Running the Tool

```
python headerguard.py --url https://example.com
```

Or

```
python headerguard.py
Enter the target URL: https://example.com
```

7.2 Interpreting the Output

- Each section clearly indicates the status of headers, cookies, and SSL info.
- `[✔]` means the check passed.
- `[!]` indicates a warning or vulnerability.
- Detailed explanations accompany each finding.

7.3 Customizing Target URLs

- Add batch support or input files for scanning multiple URLs (future work).
- Customize headers or add authentication if needed (advanced use).

7.4 Extending Functionality

- Add JSON output for integration with other tools.
- Include automated alerts or notifications.

---

8. Security Concepts Behind the Checks

This section explains the critical security concepts and best practices behind the checks HeaderGuard performs. Understanding these concepts is essential to appreciate why missing or misconfigured headers and cookie attributes create vulnerabilities in web applications.

8.1 Why Missing Security Headers Matter

HTTP security headers are directives sent from the web server to the client browser to instruct it on how to behave in terms of security. They provide an essential layer of defense beyond the application code by leveraging built-in browser security features.

When these headers are missing or incorrectly configured, attackers can exploit common web vulnerabilities to compromise users or the server. Let's explore the key headers that HeaderGuard inspects:

8.1.1 Strict-Transport-Security (HSTS)

- **Purpose:** Forces browsers to interact with the site only over HTTPS, preventing protocol downgrade attacks and cookie hijacking.
- **Risk if missing:** Without HSTS, attackers can perform man-in-the-middle (MITM) attacks by intercepting HTTP traffic and redirecting users to malicious sites.
- **Best practice:** Set `max-age` to at least 6 months (e.g., 31536000 seconds) and include `includeSubDomains` for comprehensive protection.

8.1.2 Content-Security-Policy (CSP)

- **Purpose:** Restricts which resources (scripts, styles, images) the browser can load, thereby mitigating Cross-Site Scripting (XSS) and data injection attacks.
- **Risk if missing:** Attackers can inject malicious scripts or resources leading to data theft or session hijacking.
- **Best practice:** Implement a strict CSP tailored to the application's needs; avoid overly permissive wildcards ().

8.1.3 X-Content-Type-Options

- **Purpose:** Prevents browsers from MIME-sniffing a response away from the declared content-type, which helps to reduce drive-by downloads and XSS.

- **Risk if missing:** Browsers may misinterpret file types leading to script execution of non-script files.
- **Best practice:** Set the header to `nosniff`.

8.1.4 X-Frame-Options

- **Purpose:** Protects against clickjacking attacks by controlling whether a site can be framed by other websites.
- **Risk if missing:** Attackers can trick users into clicking invisible UI elements by framing the site.
- **Best practice:** Use `DENY` or `SAMEORIGIN` to prevent framing or restrict it to trusted origins.

8.1.5 Referrer-Policy

- **Purpose:** Controls how much referrer information is sent along with requests to other sites.
- **Risk if missing:** Can leak sensitive URL data such as tokens or user identifiers.
- **Best practice:** Use `no-referrer` or `strict-origin-when-cross-origin` for privacy.

8.1.6 Permissions-Policy

- **Purpose:** Replaces Feature-Policy, controlling access to browser features such as geolocation, camera, microphone.
- **Risk if missing:** Unrestricted access to sensitive APIs can lead to privacy breaches.
- **Best practice:** Disable all unnecessary features explicitly.

8.2 Cookie Flags and Their Protection Roles

Cookies are critical for managing sessions and user identity but are frequently targeted by attackers. Secure cookie configurations are vital:

8.2.1 Secure Flag

- **Description:** Ensures cookies are sent only over encrypted HTTPS connections.

- **Risk if missing:** Cookies can be intercepted over unsecured HTTP, exposing session tokens.
- **Best practice:** Always set `Secure` for any sensitive cookies.

8.2.2 HttpOnly Flag

- **Description:** Prevents JavaScript access to cookies.
- **Risk if missing:** Cookies may be stolen via XSS attacks through client-side scripts.
- **Best practice:** Use `HttpOnly` on all authentication cookies.

8.2.3 SameSite Attribute

- **Description:** Controls whether cookies are sent with cross-site requests.
- **Values:**
  - `Strict`: Cookies only sent with same-site requests.
  - `Lax`: Cookies sent with top-level navigations but not with embedded requests.
  - `None`: Cookies sent in all contexts, but must be Secure.
- **Risk if missing or misconfigured:** Increased risk of Cross-Site Request Forgery (CSRF).
- **Best practice:** Use `Strict` or `Lax` depending on application needs; avoid `None` unless required.

8.3 SSL/TLS Vulnerabilities and Best Practices

SSL/TLS protocols encrypt data in transit and are essential for website security.

8.3.1 Importance of Valid Certificates

- **Certificate Authority (CA):** Certificates should be issued by trusted CAs.
- **Risk:** Self-signed or expired certificates cause browser warnings and allow MITM attacks.
- **Best practice:** Always use valid, non-expired certificates from trusted authorities.

8.3.2 TLS Version Support

- **Legacy Versions:** SSLv2, SSLv3, TLS 1.0 and 1.1 have known vulnerabilities.
- **Risk:** Using old protocols enables downgrade attacks and weak encryption.
- **Best practice:** Support TLS 1.2 and TLS 1.3 only.

8.3.3 Certificate Expiry

- **Risk:** Expired certificates cause loss of trust and possible connection refusal.
- **Best practice:** Monitor expiry dates and renew certificates timely.

---

9. Troubleshooting and Common Issues

While running HeaderGuard, users may encounter various errors or unexpected behaviors. This section covers common issues and how to resolve them.

9.1 Network and Connection Errors

9.1.1 Timeout and Connection Failures

- **Symptoms:** The tool hangs or throws timeout exceptions.
- **Causes:** Network issues, target website down, or firewall blocking requests.
- **Resolution:**
  - Check internet connectivity.
  - Verify target URL is correct and accessible.
  - Use VPN or proxies if region blocks exist.
  - Increase request timeout in code if needed.

9.1.2 SSL Handshake Failures

- **Symptoms:** Errors like `ssl.SSLError`, `CERTIFICATE_VERIFY_FAILED`.
- **Causes:** Invalid SSL certificates, expired certs, or self-signed certs.
- **Resolution:**

    - Accept self-signed certs for testing (not recommended in production).
    - Update local certificate stores.
    - Use `verify=False` in `requests` with caution.

9.2 Encoding and File Writing Issues

9.2.1 UnicodeEncodeError on Windows

- **Symptoms:** Errors when writing report files containing special characters (e.g., ✓, ×).
- **Cause:** Default Windows console encoding (`cp1252`) does not support certain Unicode characters.
- **Resolution:**
    - Open files with UTF-8 encoding explicitly:

    ```
    with open(filename, "w", encoding="utf-8") as f:
        f.write(report)
    ```

    - Run terminal supporting UTF-8 (Windows Terminal or set console code page).

9.2.2 File Permissions

- **Symptoms:** Permission denied errors when saving reports.
- **Cause:** Lack of write access in the current directory.
- **Resolution:**
    - Run script as Administrator or with sufficient privileges.
    - Change output directory to a writable location.

9.3 Handling Unexpected HTTP Responses

9.3.1 Redirects

- **Issue:** Some URLs redirect multiple times or to different domains.
- **Impact:** Headers checked may not belong to the original target.

- **Solution:**
  - Follow redirects automatically (default in `requests`).
  - Log final URL after redirects for clarity.
  - Allow option to disable redirects for advanced users.

9.3.2 HTTP Errors (4xx, 5xx)

- **Issue:** Server returns error pages instead of expected content.
- **Impact:** Headers may not reflect production configuration.
- **Solution:**
  - Check HTTP status codes before header analysis.
  - Skip or flag error responses with warnings.

9.3.3 Incomplete Headers or Proxy Interference

- **Issue:** Reverse proxies or WAFs may modify or strip headers.
- **Impact:** Results may not reflect backend server configuration.
- **Solution:**
  - Consider testing behind proxies or direct to backend servers if possible.
  - Document limitations in report.
