|  |
| --- |
| Gopher Industries – Nutrihelp |
| Vulnerability Scanner Report |
|  |

|  |
| --- |
| Lachlan Adams  04/09/2024 |

Statement of Intent

Overview

This document has been developed to assist students within Nutrihelp after their use of “VulnerabilityScannerV1.0”. This report entails all the vulnerabilities the scanner is looking for and entails how the attacks are carried out, the consequences and how to patch this vulnerability. This document has been written with the presumption that readers know how to code based on their submissions and changes to the Nutrihelp API.

Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| OS | Operating System |
| XXS | Cross-Site-Scripting |
| SQL | Structured Query Language |
| Hijack | When an attacker takes control of the session, device, webserver etc |
| Session | In the context of this report, a session is a web browser tab or active online website page where a user is interaction or accessing through an internet browser. |
| RNG | Random Number Generator |

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Dates** | **Version** | **Author** | **Comments** |
| 20/08/2024 | 1 | Lachlan Adams | 5 vulnerability patterns identified |
| 04/09/2024 | 2 | Lachlan Adams | Updated to 10 vulnerabilities (Previously 5), need to add potential issues, defences and code identification |

Table of Contents

[**SQL Injection** 5](#_Toc176374389)

[How is this attack found via the Vulnerability Scanner? 5](#_Toc176374390)

[Potential issues if Vulnerability / Attack was successful: 5](#_Toc176374391)

[How to defend against this Vulnerability / Attack: 5](#_Toc176374392)

[**Cross-Site Scripting** 5](#_Toc176374393)

[How is this attack found via the Vulnerability Scanner? 6](#_Toc176374394)

[Potential issues if Vulnerability / Attack was successful: 6](#_Toc176374395)

[How to defend against this Vulnerability / Attack: 6](#_Toc176374396)

[**Command Injection** 6](#_Toc176374397)

[How is this attack found via the Vulnerability Scanner? 6](#_Toc176374398)

[Potential issues if Vulnerability / Attack was successful: 6](#_Toc176374399)

[How to defend against this Vulnerability / Attack: 7](#_Toc176374400)

[**Insecure File Handling** 7](#_Toc176374401)

[How is this attack found via the Vulnerability Scanner? 7](#_Toc176374402)

[Potential issues if Vulnerability / Attack was successful: 7](#_Toc176374403)

[How to defend against this Vulnerability / Attack: 7](#_Toc176374404)

[**Insecure File Upload** 7](#_Toc176374405)

[How is this attack found via the Vulnerability Scanner? 8](#_Toc176374406)

[Potential issues if Vulnerability / Attack was successful: 8](#_Toc176374407)

[How to defend against this Vulnerability / Attack: 8](#_Toc176374408)

[**Code Injection** 8](#_Toc176374409)

[How is this attack found via the Vulnerability Scanner? 8](#_Toc176374410)

[Potential issues if Vulnerability / Attack was successful: 8](#_Toc176374411)

[How to defend against this Vulnerability / Attack: 8](#_Toc176374412)

[**Directory Movement – Transversal Attack** 9](#_Toc176374413)

[How is this attack found via the Vulnerability Scanner? 9](#_Toc176374414)

[Potential issues if Vulnerability / Attack was successful: 9](#_Toc176374415)

[How to defend against this Vulnerability / Attack: 9](#_Toc176374416)

[**Insecure Token Generation** 9](#_Toc176374417)

[How is this attack found via the Vulnerability Scanner? 9](#_Toc176374418)

[Potential issues if Vulnerability / Attack was successful: 9](#_Toc176374419)

[How to defend against this Vulnerability / Attack: 9](#_Toc176374420)

[**Permission Level** 10](#_Toc176374421)

[How is this attack found via the Vulnerability Scanner? 10](#_Toc176374422)

[Potential issues if Vulnerability / Attack was successful: 10](#_Toc176374423)

[How to defend against this Vulnerability / Attack: 10](#_Toc176374424)

[**Redirects** 10](#_Toc176374425)

[How is this attack found via the Vulnerability Scanner? 10](#_Toc176374426)

[Potential issues if Vulnerability / Attack was successful: 10](#_Toc176374427)

[How to defend against this Vulnerability / Attack: 10](#_Toc176374428)

[**Summary** 11](#_Toc176374429)

## **SQL Injection**

This is a web-based attack, where a user manipulates the SQL queries via entering in tehri own malicious code. This is often performed through any input field on a website. For example, if a user is prompted to enter in their username, this input field could be used to coordinate an attack. This allows the attacker to manipulate the query’s execution and lead to further damage or attacks. Dependant on which area was breached and the extent of their malicious code, the attacker may be able to manipulate the database or gain unauthorised access to the system.

### How is this attack found via the Vulnerability Scanner?

“"Sql\_Injection": re.compile(r'\.query\s\*\(.\*\+.\*\)')“ this expression is used to search through the selected file and locate any areas where a user is able to enter something into a input field. Without proper data sanitisation, a user would be able to input SQL code to the input field and begin to breach the application.

### Potential issues if Vulnerability / Attack was successful:

**Privilege Escalation:** A hacker can begin to elevate their privilege. Which could result in access to different areas of the application. This their new privileges, they could make changes to the web server and take control.

**Data Corruption:** An attacker could intentionally cause a system corruption. Where they damage files or areas of the system and cause damage to the integrity and lifespan.

### How to defend against this Vulnerability / Attack:

**Data Sanitisation:** Sanitise all data that is collected from users.

**Secure coding principles and frameworks:** Utilising online documentation and guides to code with security and data integrity first above all else.

**Pre configured parameters and allowed characters**: Allowing only a specific range of characters based on the exact needs for the input field. And example of this is within the name field, numbers and special characters will not be necessary. These should be blocked by default.

## **Cross-Site Scripting**

This is a vulnerability which can be taken advantage of through web application. This vulnerability can be exploited when a web application displays a user’s input without the correct data sanitisation and input validation. This allows for attackers to deploy their malicious code and potentially hijack the session or interfere with the website. An example of this attack is a user could enter in <script>alert(‘The session has been hijacked!!!!! Malware uploaded’)</script>. Which would cause a popup window with “The session has been hijacked!!!!! Malware uploaded” on the user’s screen.

### How is this attack found via the Vulnerability Scanner?

‘ "XSS": re.compile(r'res\.send\s\*\(.\*\+.\*\)') ‘ this line of code is used to detect XSS vulnerabilities within the JavaScript files. This identifies areas of code that an XSS attack can occur. Due to a user’s input not being properly sanitised or without the correct data validation.

### Potential issues if Vulnerability / Attack was successful:

**Session Hijacking:** The active session could be hijacked by an attacker. Which could lead to a range of issues from, unauthorised access, further attacks and damage to the integrity of the web server.

**Malware infection:** The attacker could deploy malware to the users device or the webserver. Dependant on the malware, this could corrupt files, infect other users and potentially give full access to the attacker.

### How to defend against this Vulnerability / Attack:

**Data Sanitisation:** Sanitise all data that is collected from users.

**Secure coding principles and frameworks:** Utilising online documentation and guides to code with security and data integrity first above all else.

**Develop Input Controls:** Development of controls that prevent users from inputting characters that they would not need for the given input field.

## **Command Injection**

As the name suggests this is another injection-based attack, which attackers commonly use if there is an open vulnerability to abuse. When there is little to no validation and poor coding from the developers. Applications can sometimes be configured in a way that user inputs are sent directly to the system. This means that an attacker is able to execute system level commands and pressure test a system in an attempt to cause a disruption or gain access.

### How is this attack found via the Vulnerability Scanner?

 ‘ "Command\_Injection": re.compile(r'exec\s\*\(.\*\+.\*\)') ‘ This line of code catches any attempt to submit data to an input field. If there is no sanitisation for “+” or other characteristics of malicious code. This vulnerability will be identified.

### Potential issues if Vulnerability / Attack was successful:

**System Control**: The hacker can gain control of the system via their malicious code. This could result in immediate damages, users losing access and potentially a ransomware attack.

**Data Integrity:** Users files and information may be compromised. Which could result in legal trouble for Nutrihelp. This is completely avoidable if proper sanitisation and validation is performed.

**System Corruption:** The system could be taken offline by the attacker

### How to defend against this Vulnerability / Attack:

**Data Sanitisation:** Sanitise all data that is collected from users.

**Secure coding principles and frameworks:** Utilising online documentation and guides to code with security and data integrity first above all else.

## **Insecure File Handling**

This is a vulnerability within the code of an application when files are uploaded, downloaded, changed, deleted or accessed. Without proper controls preconfigured in the code, attackers can explore and experiment freely based on their codes conditions. If there are little to no forms of validation, authentication or read only permissions set. An attacker can cause dame via deleting files, modifying files, uploading malicious files or accessing files they shouldn’t be able to see. This causes an issue within the systems integrity and can potentially lead to an attacker accessing files that contain sensitive data that was not meant for their eyes to see.

### How is this attack found via the Vulnerability Scanner?

‘ "insecure\_file\_handling": re.compile(r'fs\.unlink\s\*\(.\*\)') ‘ This line of code detects files where the Unlink function has been used. This is dangerous as it is used to delete files from the system. Which mean if the hacker was able to access this area. They would be able to delete files within specific sub directories.

### Potential issues if Vulnerability / Attack was successful:

**File Deletion:** Attackers could delete any files they would like too. This may include system files that are crucial to run the server.

**Manipulation:** The attacker may manipulate files or alter their directories.

### How to defend against this Vulnerability / Attack:

**Account Permissions:** Limit what each user can perform on the webserver. This may include only allowing users low permissions such as uploading and viewing files.

**File Paths:** Set permissions to not allow any changers to file paths and restrict access to viewing the locations of files stored on the server.

## **Insecure File Upload**

Similar to the vulnerability identified above. This vulnerability solely focuses on the file upload area. When an application or webserver allows users to upload files, there should be validation that is performed on the system to ensure those files cannot be executed on the system. Attacks may submit files that cause harm to the system via the input malicious code or malware within them. Dependant on the systems validations, an attack may be able to gain access to the systems, completely wipe it or take control.

### How is this attack found via the Vulnerability Scanner?

‘ "insecure\_file\_upload": re.compile(r'multer\s\*\(\s\*{.\*dest.\*}\s\*\)') ‘ This line of code is responsible for detecting an insecure file upload. It searches for code that does not apply security checks or validation in regards to file uploads.

### Potential issues if Vulnerability / Attack was successful:

**System Control**: The hacker can gain control of the system via their malicious code. This could result in immediate damages, users losing access and potentially a ransomware attack.

**Data Corruption:** An attacker could intentionally cause a system corruption. Where they damage files or areas of the system and cause damage to the integrity and lifespan.

**Data breach:** User’s data could be stolen by the attackers. This may include any sensitive data attached to their file.

### How to defend against this Vulnerability / Attack:

File Directories: Ensure when files are uploaded, they are stored in a quarantine zone that is isolated.

**File Types:** Only allow specific file types.

**File Scanner:** Implement a file scanning system which checks for malware within all files uploaded.

## **Code Injection**

Code injection is where an attacker enters in malicious code into an input field. This data is then executed by the webserver or application. This causes integrity issues as if this vulnerability is not properly identified. An attacker can easily access the system and deploy further attacks.

### How is this attack found via the Vulnerability Scanner?

‘ "Eval": re.compile(r'eval\s\*\(.\*\)'), ‘ this line of code is responsible for identifying code injection vulnerabilities. It searches for ‘eval’ in the selected file. If any lines of code contain this within an input section, the alert is triggered

### Potential issues if Vulnerability / Attack was successful:

### How to defend against this Vulnerability / Attack:

## **Directory Movement – Transversal Attack**

This vulnerability is within the file directory.

### How is this attack found via the Vulnerability Scanner?

‘ "Directory\_Movement": re.compile(r'fs\.readFile\s\*\(.\*\.\.\/.\*\)') ‘

### Potential issues if Vulnerability / Attack was successful:

### How to defend against this Vulnerability / Attack:

## **Insecure Token Generation**

This vulnerability refers to the generation of tokens. Which re used in session management, authentication and verifying a user is accessing the web server from their normal location. If the tokens generated are static or low include low RNG. They are weak and can be easily generated or predicted by attackers. This is also a large issue of the tokens are not hashed and contain no form of cryptographic cover.

### How is this attack found via the Vulnerability Scanner?

‘ "Insecure\_Token\_Generation": re.compile(r'Math\.random\s\*\(\)') ‘

### Potential issues if Vulnerability / Attack was successful:

### How to defend against this Vulnerability / Attack:

## **Permission Level**

This vulnerability is within the code itself. If incorrect permission settings are enabled for users, this could be damaging to the web server. As the user could accidently or intentionally cause damage to the system. Their account could also be compromised by an attacker. This would lead to much further damage and system wide corruption if the permission levels are not set correctly.

### How is this attack found via the Vulnerability Scanner?

‘ "Dangerous\_Permission\_Level": re.compile(r'fs\.chmod\s\*\(.\*\)') ‘

### Potential issues if Vulnerability / Attack was successful:

### How to defend against this Vulnerability / Attack:

## **Redirects**

This vulnerability as the name suggests, refers to redirects. An attacker can create malicious links and place them in their script. Which could redirect users to malicious URL’s. this type of attack could to a hijacked session, redirected to a webpage owed by the hacker that contains malware or further exfiltration. Without further validation and set URL’s, this could be heavily exploited by an attacker.

### How is this attack found via the Vulnerability Scanner?

‘ "Redirects": re.compile(r'res\.redirect\s\*\(.\*req\.query\..\*\)') ‘

### Potential issues if Vulnerability / Attack was successful:

### How to defend against this Vulnerability / Attack:

## **Summary**

As detailed in this report, it is crucial to ensure there are no vulnerabilities coded into applications. However, if vulnerabilities are found, they can still be patched before an attacker gets the chance to abuse them. All of the vulnerabilities discussed in this report can be updated and patched in a single line of code. Majority of the vulnerabilities are created due to improper data sanitisation, validation or file handling. When applications are developed, Integrity, accuracy and effectiveness should be the top priority to ensure success.

Please reach out to me if you have any questions in regard to patching any vulnerability my program has identified in your code.