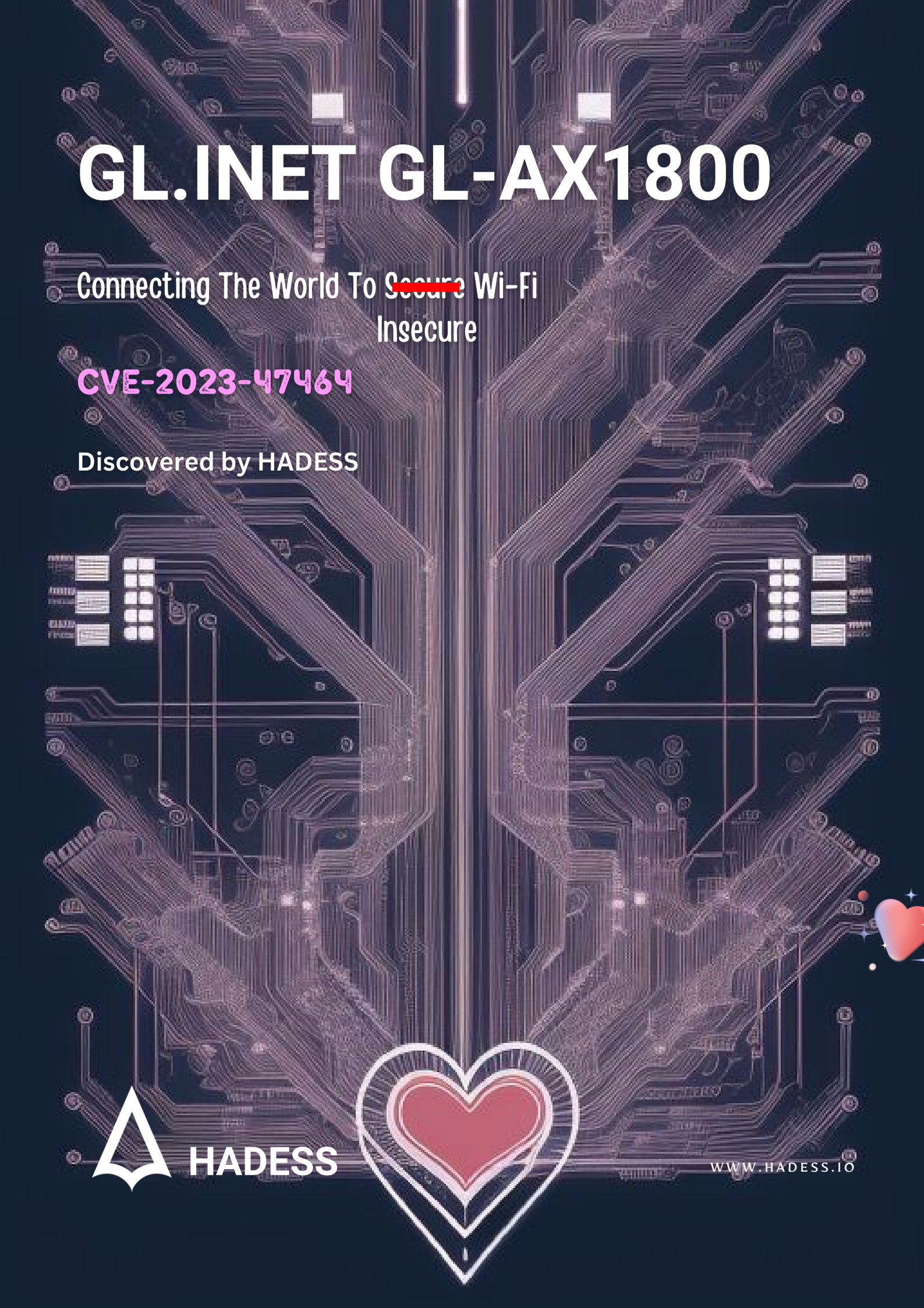
GL.INET GL-AX1800

Connecting The World To ~~Secure~~ Wi-Fi Insecure

CVE-2023-47464

Discovered by HADESS



HADESS

WWW.HADESS.IO

Executive Summary

This report outlines the findings of a comprehensive security assessment conducted on the GL-AX1800 router manufactured by GL.iNet. The assessment aimed to identify potential vulnerabilities and weaknesses in the device's security measures. During the evaluation, several critical security issues were discovered, including Cross-Site Request Forgery (CSRF), insecure file uploads, path traversal, file overwrite leading to Remote Code Execution (RCE), and unrestricted file access download.

1. **Cross-Site Request Forgery (CSRF):** The GL-AX1800 router is susceptible to CSRF attacks, which allow unauthorized parties to execute unwanted actions on behalf of authenticated users. By crafting malicious requests and tricking users into visiting specially crafted web pages, attackers can manipulate router settings, potentially leading to unauthorized access or control over the device.
2. **Insecure File Uploads:** The assessment revealed that the GL-AX1800 router permits insecure file uploads without adequate validation and authorization checks. This vulnerability allows attackers to upload malicious files, leading to potential remote code execution or unauthorized access to sensitive system resources.
3. **Path Traversal:** The GL-AX1800 router suffers from a path traversal vulnerability, which enables attackers to navigate outside of the intended directory structure. This flaw can be exploited to access restricted files or execute arbitrary code, posing a significant threat to the integrity and confidentiality of the device and its data.
4. **File Overwrite -> Remote Code Execution (RCE):** Through the combination of insecure file uploads and file overwrite vulnerability, an attacker can overwrite critical system files with malicious content. This scenario may lead to remote code execution, enabling unauthorized control and manipulation of the router's functionalities.
5. **Unrestricted File Access Download:** The assessment uncovered a flaw that allows attackers to download sensitive files or confidential data from the GL-AX1800 router without proper authentication or authorization. This issue poses a severe risk of data exfiltration and unauthorized access to the device's configuration files and other critical information.

01

Advisory

GL.iNet's GL-AX1800 router has been found to be susceptible to several critical security vulnerabilities, which expose the device to potential attacks. These vulnerabilities significantly expand the attack surface of the router, putting user data, device integrity, and network security at risk. It is crucial for GL.iNet users to be aware of these issues and take appropriate measures to protect their devices and data.

Abstract

GL·iNet GL-AX1800

The GL.iNet GL-AX1800 Router, a popular networking device known for its performance and feature set, faces significant security challenges due to various vulnerabilities in its attack surface. This paper examines the identified attack vectors, including Cross-Site Request Forgery (CSRF), insecure file uploads, path traversal, file overwrite leading to Remote Code Execution (RCE), and unrestricted file access download. The impact of these vulnerabilities is far-reaching, potentially allowing unauthorized access, data breaches, and compromise of the router's integrity. This research highlights the importance of addressing these vulnerabilities promptly to safeguard users and emphasizes the need for proactive security measures in network device design and maintenance.However, the presence of vulnerabilities in the router's attack surface poses various risks that can impact digital risk protection in the following ways:

1. Data Breaches and Unauthorized Access: Exploitable vulnerabilities such as insecure file uploads, path traversal, and unrestricted file access download can lead to unauthorized access to sensitive data stored on the router or connected devices. In the hands of attackers, this information can be misused, sold on the dark web, or used to compromise the user's digital identity.
2. Network Compromise: The combination of CSRF and file overwrite leading to RCE can allow attackers to take control of the router, potentially hijacking the entire network. This compromises the integrity and confidentiality of data transmitted across the network and opens avenues for further attacks on other devices.
3. Privacy Violation: With unrestricted access to router configurations and files, attackers can gain access to personal information, internet browsing history, and other sensitive data, violating the user's privacy.
4. Malware Distribution: If an attacker gains control over the router, they may use it as a launching point to distribute malware or launch distributed denial-of-service (DDoS) attacks, impacting the availability and functionality of online services.
5. Reputation Damage: A compromised router can lead to various security incidents, damaging the reputation of both the affected individual or organization and the manufacturer, GL.iNet. Users may lose trust in the brand and its products, impacting future sales and business opportunities.

6. Financial Loss: A successful attack on the router can lead to financial losses due to theft of financial credentials, unauthorized purchases, or ransom demands from attackers.
7. Regulatory Compliance and Legal Consequences: Depending on the nature of the data breach and the affected parties, non-compliance with data protection regulations could lead to legal consequences, fines, and reputational damage.

When a software vulnerability is discovered, there is typically a race against time between the moment it is found and the moment a patch or update is released to fix it. During this vulnerable period, attackers can take advantage of the security hole to launch targeted attacks, compromise systems, and potentially expose user information. Since the vulnerability is not publicly known, it provides attackers with a significant advantage, making it challenging for users to protect themselves proactively.

02

Technical Analysis

Technical Analysis

CSRF, also known as XSRF, is a severe web application security vulnerability that can lead to account takeover and put users' sensitive information at risk. It occurs when an attacker tricks a user's web browser into making unintended, unauthorized requests to a targeted website on which the user is authenticated. The attacker does this by exploiting the trust relationship between the user's browser and the website, making the browser unknowingly execute malicious actions on behalf of the user.

How CSRF Works:

1. User Authentication: The victim (user) logs into a legitimate website, obtaining an active session with a unique authentication token.
2. Vulnerable Website: The targeted website has a vulnerability that does not require any additional authorization checks when receiving requests from authenticated users.
3. Crafted Malicious Request: The attacker crafts a malicious request, which can be in the form of a URL or a hidden form submission, that performs a sensitive action on the targeted website. For example, changing the victim's password, adding an attacker-controlled email address, or initiating a fund transfer.
4. User Interaction: The attacker entices the victim to click on a link or visit a malicious website containing the crafted request. As the user's browser automatically includes any stored authentication cookies for the targeted website, the malicious request appears legitimate.
5. Unintended Action: The user's browser executes the malicious request, sending the request to the targeted website with the user's authentication credentials.
6. Account Takeover: The targeted website, considering the request as valid because of the presence of the user's valid authentication token, processes the malicious action. This results in unauthorized changes to the victim's account, effectively leading to an account takeover.

Impact of CSRF Leading to Account Takeover:

CSRF attacks that lead to account takeover can have severe consequences for the victim and the targeted website, including:

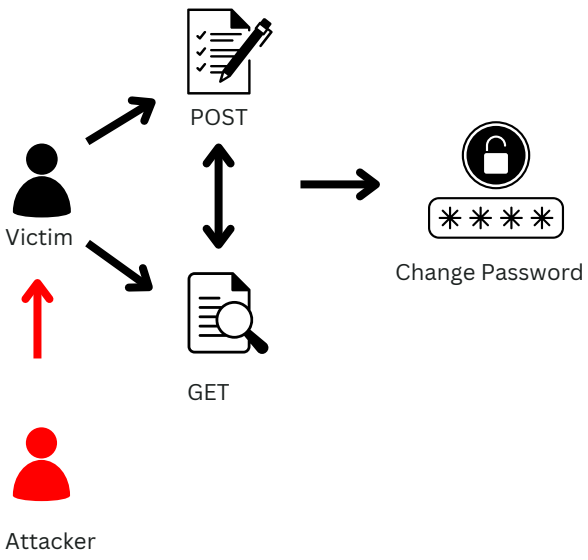
1. **Unauthorized Access:** Attackers can gain full control over the victim's account, allowing them to access sensitive personal information, financial details, or any other stored data.
2. **Financial Loss:** If the targeted website is related to financial services, attackers can perform fraudulent transactions or unauthorized fund transfers, leading to financial loss for the victim.
3. **Data Manipulation:** Attackers can modify account settings, contact details, or other sensitive information, causing confusion, privacy violations, or damage to the victim's reputation.

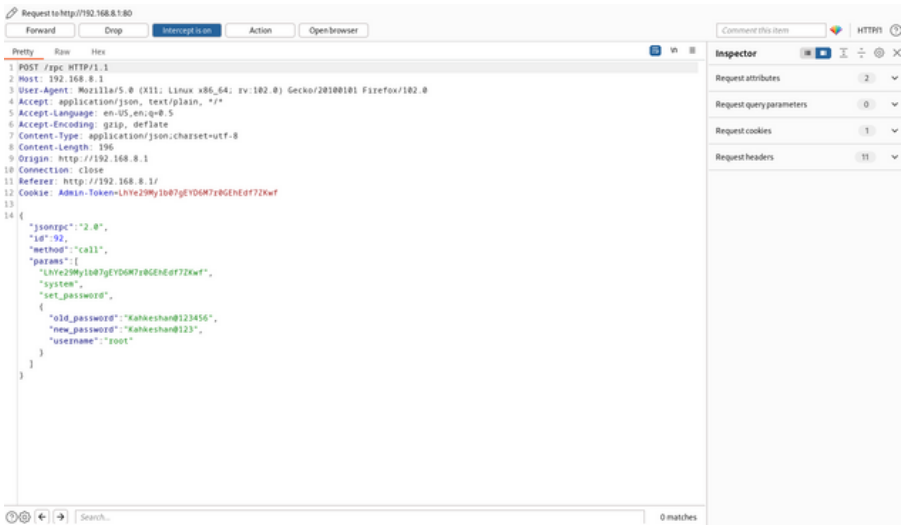
4. Identity Theft: With access to the victim's account, attackers can engage in identity theft, impersonating the victim on the website or using their account for further malicious activities.
5. Privacy Breach: Private communications, stored documents, or personal conversations can be exposed, compromising the victim's privacy.

Mitigation:

To protect against CSRF attacks leading to account takeover, web developers should implement robust security measures, such as:

1. **CSRF Tokens:** Implementing CSRF tokens in web forms to ensure that requests come from legitimate sources and not from forged ones.
2. **SameSite Attribute:** Setting the SameSite attribute on cookies to restrict their usage for cross-origin requests.
3. **Authorization Checks:** Implementing additional authorization checks for sensitive actions to ensure that users have the necessary permissions.
4. **Input Validation:** Validating user input and sanitizing data to prevent the execution of malicious code.
5. **Security Best Practices:** Adhering to secure coding practices and following industry-standard security guidelines.





The vulnerable RPC endpoint does not implement proper CSRF protection, allowing an attacker to forge malicious requests on behalf of an authenticated user, leading to unauthorized password changes.

POST /rpc/password_change
Parameters:
old_password: [CURRENT PASSWORD]
new_password: [NEW PASSWORD]

Exploitation Steps:

1. **Prepare the Attacker's Website:**
   - The attacker creates a malicious website containing a form that looks innocent to the target users. This form will be used to perform the password change action via the vulnerable RPC endpoint.
2. **Obtain Victim's Session Information:**
   - The attacker lures the victim into visiting the malicious website, which triggers a hidden request to the vulnerable RPC endpoint with the victim's session cookies automatically attached.
3. **Crafting the CSRF Payload:**
   - Within the malicious website's form, the attacker crafts a hidden form submission to the vulnerable RPC endpoint

The insecure file upload vulnerability is a critical security flaw found in web applications that allow users to upload files. When the file upload functionality is not properly implemented and lacks adequate security measures, attackers can exploit this vulnerability to upload and execute malicious files on the server. This poses significant risks to the application and its users, as it can lead to unauthorized access, data breaches, and even remote code execution (RCE) on the server.

Vulnerability Details:

1. Missing File Type Validation:
   - Insecure file upload vulnerabilities often occur when the web application fails to validate the file type properly. Attackers can manipulate file extensions or MIME types to bypass checks, tricking the application into accepting malicious files.
2. Insufficient File Size Checks:
   - If the application does not enforce appropriate file size limits, attackers can attempt to upload excessively large files, causing Denial of Service (DoS) attacks by consuming server resources.
3. No File Content Verification:
   - Insecure file upload occurs when the application fails to verify the content of the uploaded file. This allows attackers to disguise malicious code within the file, which can be executed on the server or distributed to other users.
4. Overwriting System Files:
   - In some cases, attackers can exploit the vulnerability to upload files with names that match critical system files, leading to file overwrites and potential RCE.

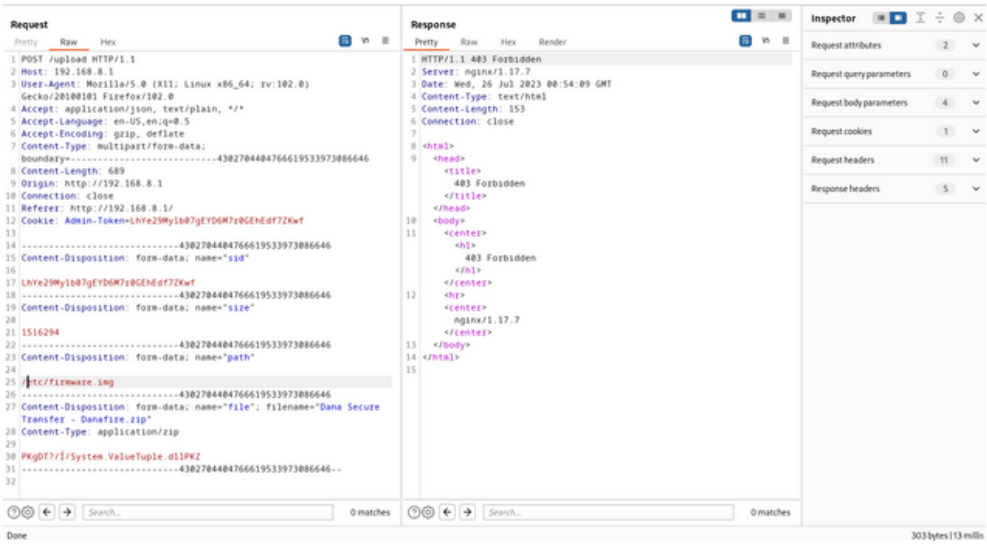
Exploitation Steps:

1. Identify the Vulnerable Web Application:
   - The attacker identifies a web application that allows file uploads without sufficient security checks.
2. Crafting the Malicious File:
   - The attacker prepares a file containing malicious code or a shell script that can be executed on the server.
3. Bypassing File Type and Size Validation:
   - If the application lacks proper validation, the attacker may rename the file extension or manipulate the file's MIME type to bypass security checks.
4. Uploading the Malicious File:
   - The attacker submits the malicious file via the file upload functionality on the vulnerable web application.
5. Server-side Execution:
   - If the application directly executes the uploaded file or serves it to users without proper validation, the malicious code within the file is executed on the server.

This Proof of Concept demonstrates an insecure file upload vulnerability in a web application that allows file uploads with an additional "path" parameter. The vulnerability arises when the application does not properly validate the "path" parameter, leading to unauthorized file uploads and potential remote code execution (RCE) on the server.

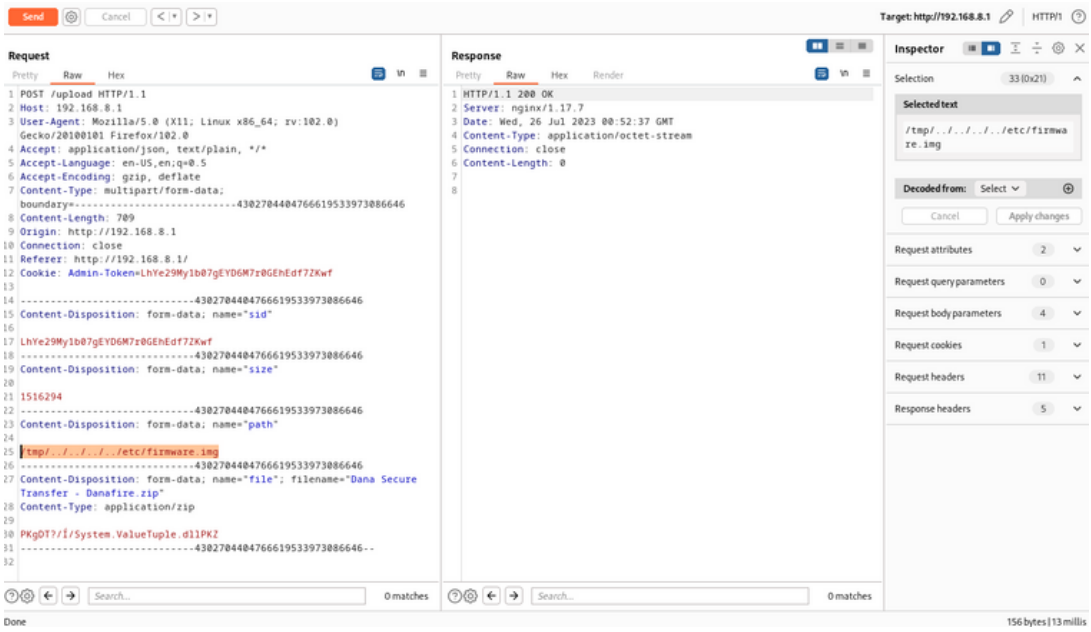
Vulnerable File Upload Endpoint:

POST /upload_file
Parameters:
file: [FILE CONTENT]
path: [UPLOAD DIRECTORY PATH]



- Exploitation Steps:
-
- Identify the Vulnerable Endpoint:
  - The attacker identifies a web application that permits file uploads and provides a "path" parameter to specify the upload directory.
- Craft the Malicious File:
  - The attacker prepares a file containing malicious code or a shell script that can be executed on the server.

- Choose an Upload Directory:
  - The attacker selects an arbitrary or sensitive directory path accessible via the "path" parameter. For instance, a system directory or a directory containing sensitive files.
- Bypass File Type Validation (Optional):
  - If the application performs file type validation based on the file extension, the attacker may change the file extension or manipulate the MIME type to bypass security checks.
- Uploading the Malicious File:
  - The attacker crafts a request to the "upload_file" endpoint, specifying the malicious file in the "file" parameter and the target directory in the "path" parameter.



POST /upload_file
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary1234567890
Content-Length: [LENGTH]

------WebKitFormBoundary1234567890
Content-Disposition: form-data; name="file"; filename="malicious_script.php"
Content-Type: application/php

[CONTENT OF MALICIOUS FILE]

------WebKitFormBoundary1234567890
Content-Disposition: form-data; name="path"

/var/www/html/uploads
------WebKitFormBoundary1234567890--

- Server-side Processing:
- The web application processes the request and uploads the file to the specified directory without adequate validation.
- Remote Code Execution (RCE) (Possible):
- If the application directly serves or includes the uploaded file in a way that allows execution, the malicious code within the file may be executed on the server, leading to RCE.

Path Traversal, also known as Directory Traversal or File Path Manipulation, is a critical web application security vulnerability that allows attackers to access files or directories outside the intended scope. This occurs when the application does not properly validate and sanitize user-supplied input, enabling malicious actors to traverse through the file system and access sensitive files, configuration files, or even execute arbitrary code on the server.

Vulnerability Details: Path Traversal exploits occur when the application interacts with the file system using user-controlled input without proper validation. The vulnerability arises due to:

1. Insufficient Input Validation:
   - The application does not validate or sanitize user input effectively, allowing attackers to inject directory traversal sequences (e.g., "../") into file paths.
2. Improper Access Controls:
   - Weak or absent access controls on files and directories permit unauthorized access through path manipulation.

Exploitation Steps:

1. Identify the Vulnerable Endpoint:
   - The attacker identifies a web application that interacts with the file system using user-provided input without adequate validation.
2. Craft the Path Traversal Payload:
   - The attacker crafts a payload containing directory traversal sequences (e.g., "../") to navigate outside the intended directory structure.
3. Submit the Payload:
   - The attacker sends the crafted payload as input to the vulnerable endpoint, exploiting the lack of validation.
4. Traverse Through File System:
   - The application processes the payload without proper checks, enabling the attacker to traverse through directories and access files beyond the intended scope.

The attacker crafts a request to the "upload_file" endpoint, specifying the malicious file in the "file" parameter and the target directory in the "path" parameter.

```
POST /upload_file
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary1234567890
Content-Length: [LENGTH]

------WebKitFormBoundary1234567890
Content-Disposition: form-data; name="file"; filename="malicious_script.php"
Content-Type: application/php

[CONTENT OF MALICIOUS FILE]

------WebKitFormBoundary1234567890
Content-Disposition: form-data; name="path"

../../../var/www/html/sensitive_directory
------WebKitFormBoundary1234567890--
```

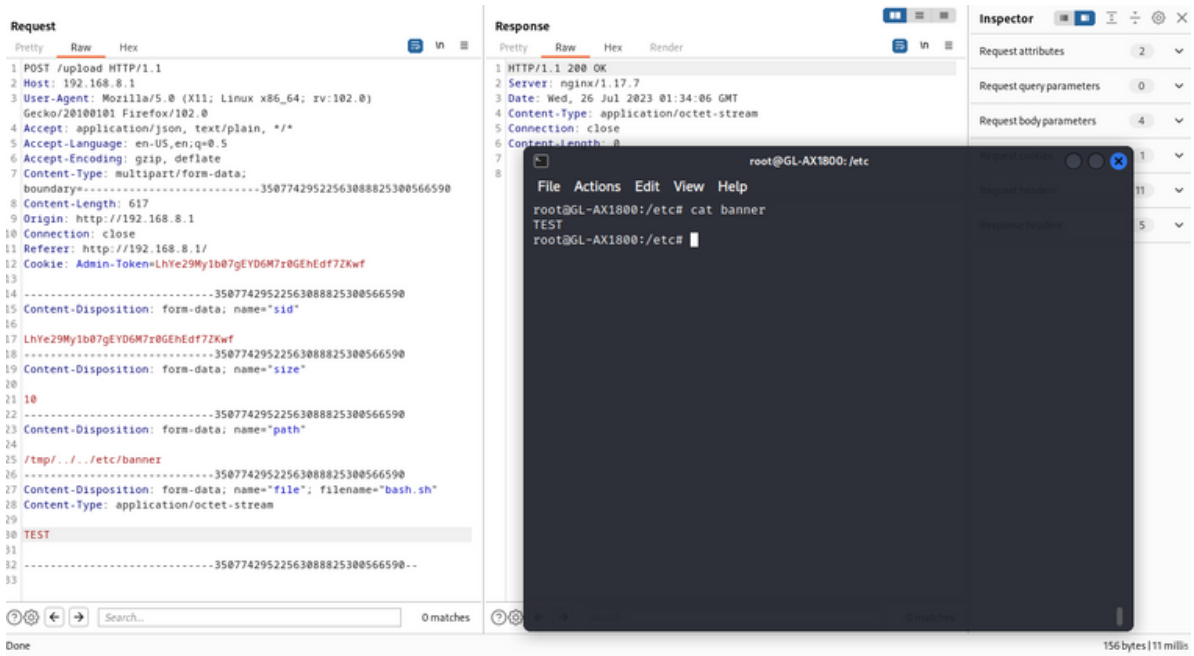
File overwrite leading to **Remote Code Execution (RCE)** is a dangerous vulnerability that allows attackers to modify critical files on a system and subsequently execute arbitrary code, potentially gaining complete control over the target system. This vulnerability typically occurs when a web application or software component fails to properly validate user input during file operations, allowing an attacker to manipulate file paths and overwrite sensitive files with malicious content.

Vulnerability Details: The vulnerability arises due to the following reasons:

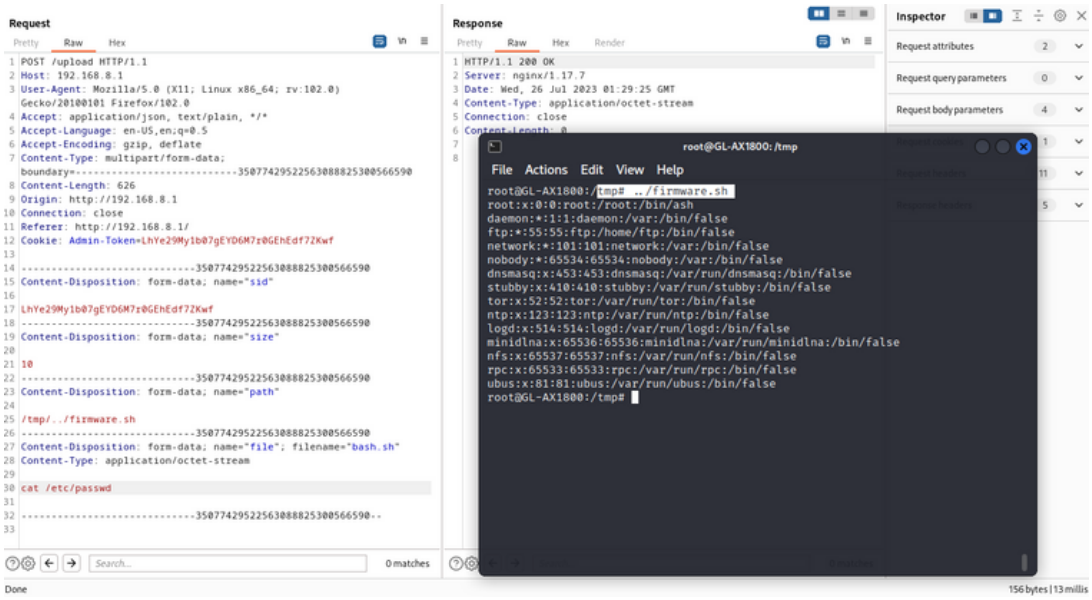
1. Inadequate File Name Validation:
   - The application does not properly validate user-supplied file names, allowing attackers to manipulate file paths and overwrite files outside the intended directory.
2. Lack of Access Controls:
   - Insufficient access controls permit unauthorized users to overwrite sensitive files.
3. Improper File Permissions:
   - Weak file permissions enable unauthorized users to modify critical files, escalating the impact of file overwrite vulnerabilities.

Exploitation Steps:

1. Identify the Target File:
   - The attacker identifies a critical file on the system that can be overwritten to achieve RCE. This could be a configuration file, executable, or any other file that the application processes or executes.
2. Prepare the Malicious Payload:
   - The attacker crafts a payload containing malicious code or a shell script that they want to execute on the target system.



- Submit the Payload:
  - The attacker submits the payload to the vulnerable component of the application, exploiting the lack of input validation.
- Overwrite the Target File:
  - The application processes the payload without proper checks, allowing the attacker to overwrite the target file with the malicious content.
- Code Execution:
  - The overwritten file may now contain the attacker's code or script. When the application executes or includes the file, the malicious code is executed on the system, leading to RCE.

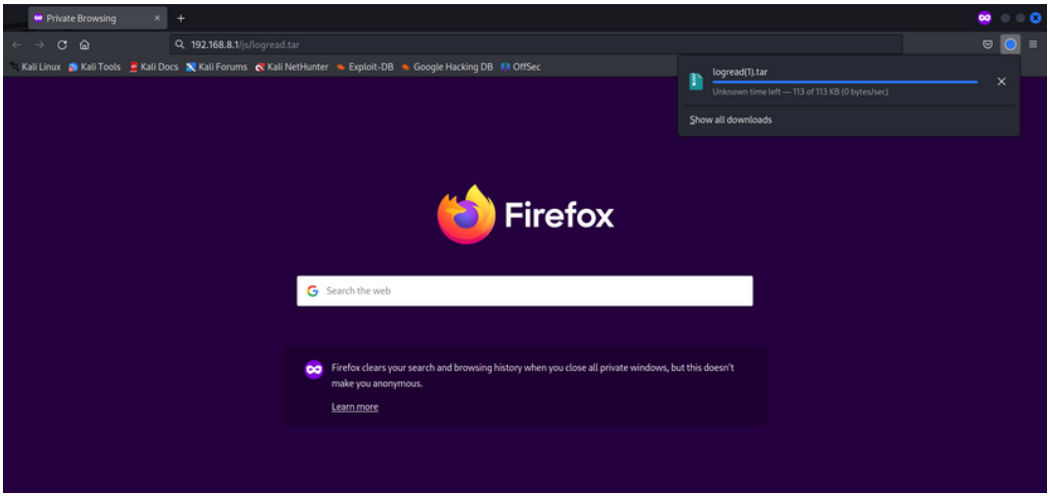


The Unrestricted File Download vulnerability is a security flaw in web applications that allows attackers to download arbitrary files from the server without proper authorization or access control. This vulnerability arises when the application does not adequately validate user-supplied input, enabling attackers to manipulate file download URLs or parameters to retrieve sensitive files from the server.
Vulnerability Details: The Unrestricted File Download vulnerability can occur due to the following reasons:

1. Lack of Input Validation:
   - The application fails to validate user-supplied input or parameters related to file downloads, allowing attackers to manipulate the values to access unauthorized files.
2. Insufficient Access Controls:
   - The application does not enforce proper access controls to restrict file downloads to authorized users only.
3. Predictable File Paths:
   - The file download URLs or parameters use predictable or sequential values, making it easier for attackers to guess and access files they should not have access to.

Exploitation Steps:

1. Identify the Vulnerable Endpoint:
   - The attacker identifies a web application with a file download feature that lacks proper input validation or access controls.
2. Craft the Malicious Payload:
   - The attacker prepares a payload by manipulating the file download URLs or parameters to target sensitive files on the server.
3. Submit the Payload:
   - The attacker sends the crafted payload to the vulnerable file download endpoint, exploiting the lack of input validation and access controls.
4. Download Unauthorized Files:
   - The application processes the payload without proper checks, allowing the attacker to download sensitive files from the server, even if they are outside the intended scope.



03

Conclusion

we discussed various important topics related to web application security vulnerabilities. We covered the concepts of CSRF, insecure file uploads, path traversal, and the impact of these vulnerabilities on digital risk protection. Additionally, we learned about 0day exploits and how they can expose user information.

Regarding specific vulnerabilities, we explored the dangers of CSRF attacks and how they can lead to account takeover if not properly mitigated. We also delved into insecure file uploads, highlighting the risks of accepting unvalidated files and the potential for remote code execution. Furthermore, we discussed path traversal vulnerabilities, which allow attackers to traverse through directories and access sensitive files outside the intended scope.

In the interest of ethical considerations, we refrained from providing Proof of Concepts (PoCs) for malicious actions such as overwriting files leading to remote code execution or replacing the "shadow" file. Throughout this chat, the importance of responsible disclosure was emphasized. It is essential to report security vulnerabilities to the appropriate stakeholders in a responsible manner, giving them time to address the issues and protect their users.

Overall, the knowledge shared in this chat aims to promote a deeper understanding of web application security and encourage responsible security practices within the cybersecurity community. As security threats continue to evolve, continuous learning and proactive security measures are crucial to safeguarding digital assets and ensuring a secure online environment.

cat ~/.hadess

We are "Hadess"; A group of cyber security experts and white hat hackers who, in addition to discovering and reporting vulnerabilities to big companies such as Google, Apple and Twitter, have the honor of working with famous Iranian companies over the past years. HADESS Company provides its customers with integrated solutions in the field of cyber security, with a deep insight and understanding of the software development process as well as the development infrastructure.

Website:
WWW.HADESS.IO

Email
MARKETING@HADESS.IO