

**Report CTF**

**CTF Empire Lupin One Penetration Testing Report**

Date: 09/05/2025

Version: 1.0

**1 Table Of Contents**

1. Document Revision History………………………………………………………….........………………………...........…2
2. Executive Summary....................................................................................................................................3
3. Scope..........................................................................................................................................................4
4. Risk Categories...........................................................................................................................................5
5. Pentest Methodology..................................................................................................................................6
6. Visual Summary..........................................................................................................................................7
7. Findings Summary......................................................................................................................................8

**1 Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Version** | **Contact Details** |
| Abhishek Joshi | 09/05/2025 | 1.0 | Email: [abhishekjoshi266@gmail.com](mailto:abhishekjoshi266@gmail.com)  Mob No: +91 -- |

# u e

**2 Executive Summary**

An analysis of a black box penetration test conducted on the Double Trouble “CTF Machine” is presented in this document. Based on a thorough security assessment performed by Internal Security Team in May of 2025.

This assessment was conducted On-Premises by the Security team. An assessment was conducted on the 09th of May to 09th of May 2025. As a comprehensive strategy for this assessment, Security Team concreted the black box penetration testing methodology and technique. To facilitate this, Company provided a walkthrough of the application and provided access to the test environment with valid different privilege accounts.

Testing was carried out by identifying vulnerabilities with the intent of accessing critical information. The objective of performing this activity was to assess the security risks associated with the developed applications and identify vulnerabilities that cybercriminals could leverage to compromise the application. The report summarizes the security findings related to the Company applications and network.

**This assessment aimed to:**

Analyze the application for technical vulnerabilities that an attacker may exploit to compromise the CTF Machine.

Provide recommendations for risk mitigation that may arise on successful exploitation of these vulnerabilities.

**3 Scope**

### Scope

The section defines the scope and boundaries of the project.

### Constraints and Limitations

The assessments, and the result(s) / finding(s) made are highly subjective to target system(s) and service(s) visibility and availability at that given point of time.

### Target Scope

Identify weaknesses that might be exploited by adversaries who have authorized or unauthorized access to Company Technical Skill Test and underlying infrastructure:

Test Perform On Harry Potter Argog CTF Environment Without Credential as Black Box Testing.

Following Machine was in the scope of the penetration test.

### Machine and Environment Details

|  |  |  |
| --- | --- | --- |
| Sr . No CTF Name Url: | | |
| 1 | Empire LupinOne | Machine Url:  <https://www.vulnhub.com/entry/empire-lupinone,750/>  Machine IP ( 192.168.120.236 ) |

### Contact Details

|  |  |
| --- | --- |
| Names Contact Details | |
| Abhishek Joshi (Penetration Tester) | Mail: [a](mailto:Abhishek.joshi@fendahl.com)bhishekjoshi266@gmail.com |

### 

**4 Risk Categories**

### Risk Categories & Rationales

Pentest use a simple risk categorisation of each vulnerability to focus the triage process at the risks which truly matter. The Common Vulnerability Scoring System (CVSS) is an industry standard formula. It generates a risk score between 0.0 and 10.0.

The table below explains the risk categories and demonstrates rule-of-thumb equivalency with CVSS scores:

|  |  |  |
| --- | --- | --- |
| **Risk Category** | **CVSS Score** | **Rationales** |
| Critical | 8.1 – 10.0 | Poses a severe risk which is easy to exploit. Begin the process of remediating immediately after the issue has been presented. |
| High | 6.1 – 8.0 | Poses a significant risk and can be exploited. Address these as soon as possible after any critical risks have been remediated. |
| Medium | 4.1 – 6.0 | Poses an important risk but may be difficult to exploit. Pentest recommends remedial work within 3 months of discovery. |
| Low | 2.1 – 4.0 | Poses a minor risk or may be exceedingly difficult to exploit. Address these over the long-term during testing cycles |
| Informational | 0.0 – 2.0 | Loss of sensitive information, or a discussion point. These are not directly exploitable but may aid an attacker. Remediate these to create a true defence-in-depth security posture, |

CVSS is not applicable to all risks. For example, it is incapable of capturing the risk of a “flat network design”. Experience has told us that this is a “high” risk in most cases.

For this reason, the reader may find vulnerabilities which have no CVSS rating in our reports.

We endeavour to provide the reason for omitting the risk score when that is the case, and to provide CVSS by default in all applicable cases.

**5 Pentest Methodology**

### Methodology

The penetration testing methodology is typically based on the NIST security methodology. The focus shifts from traditional application security, where the primary threat is from multiple sources over the Internet. The key difference is in the client-side security, file system, hardware, and network security. Traditionally for Thick Client Applications, an end user is in control of the device. Security Team used the NIST & MITRE Attack Framework testing guide for conducting penetration test of the systems and applications. The testing was done to simulate as closely as possible the viewpoint of completely external attacker, the steps involved are

1] Setup

2] Discovery

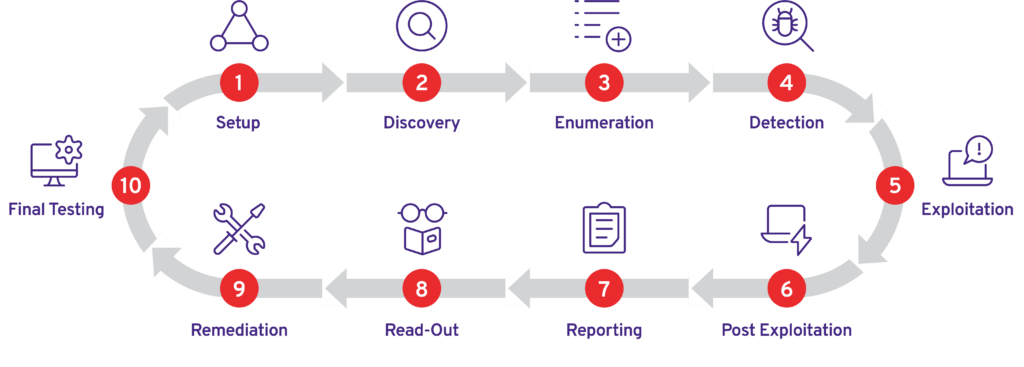
3] Enumeration

4] Detection

5] Exploitation

6] Post-Exploitation

7] Reporting



### Visual Summary

Graphical representation of Identified Vulnerabilities to Severity Risk rating

|  |  |  |
| --- | --- | --- |
| Sr. No. Severity Level Frequency | | |
| 1 | Critical | 3 |
| 2 | High | 0 |
| 3 | Medium | 0 |
| 4 | Low | 0 |

Table: Representing Severity Level

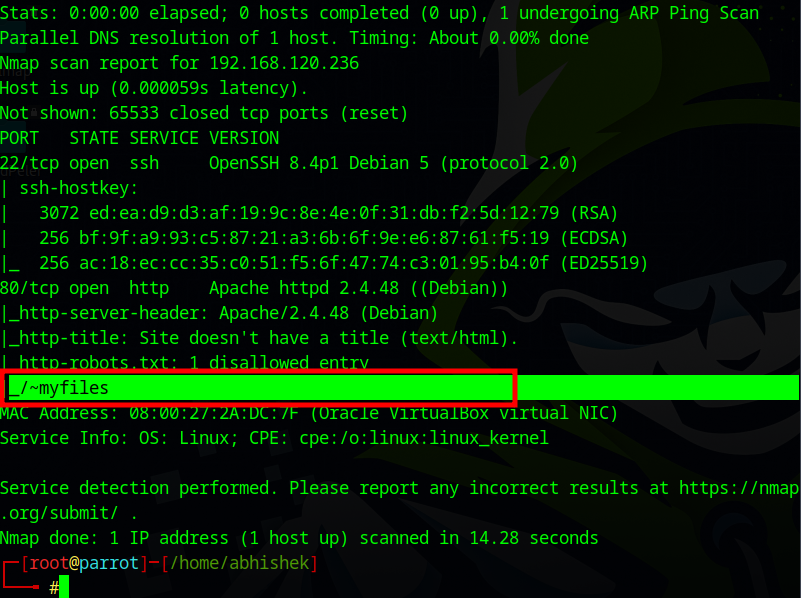
### Findings Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Observed Vulnerability** | **Risk Rating** | **Status** | **Comments** |
| 1. | Sensitive Directory And SSh Key Exposure Via Predictable Location | Critical | Not Fixed | -- |
| 2. | Insecure Cryptographic Storage of SSH Private Key (Base58 Encoded) | Critical | Not Fixed | -- |
| 3. | Privilege Escalation via Weak SSH Key Passphrase | Critical | Not Fixed | -- |

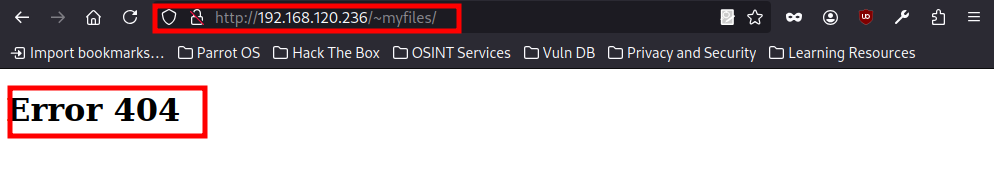
#### Sensitive Directory And SSH Key Exposure Via Predictable Location

|  |  |
| --- | --- |
| **Vulnerability** | Sensitive Directory And SSH Key Exposure Via Predictable Location |
| **Description** | During reconnaissance, the robots.txt file revealed a hidden directory (~myfiles). Subsequent directory brute-forcing (fuzzing) uncovered another sensitive path (~secret), which contained a file (secret.txt) storing an encoded SSH private key. |
| **Risk/Impact** | An attacker could decode the SSH key and gain unauthorized remote access to the server.Further compromise could lead to data theft, privilege escalation, or pivoting to internal networks. |
| **CVSS Score** | 9.0 Critical |
| **Path:** | 192.168.120.236 |
| **Remidiation / Solution** | Remidiation:  - Remove secret.txt and rotate all exposed SSH keys.  - Restrict access to ~myfiles, ~secret via server configuration (e.g., deny from all).  - Disable directory listings (e.g., Options -Indexes in Apache).  - Audit robots.txt to avoid exposing internal paths.  - Store secrets in environment variables or secure vaults (e.g., AWS Secrets Manager). |
| **Refrence Url:** | N/A |

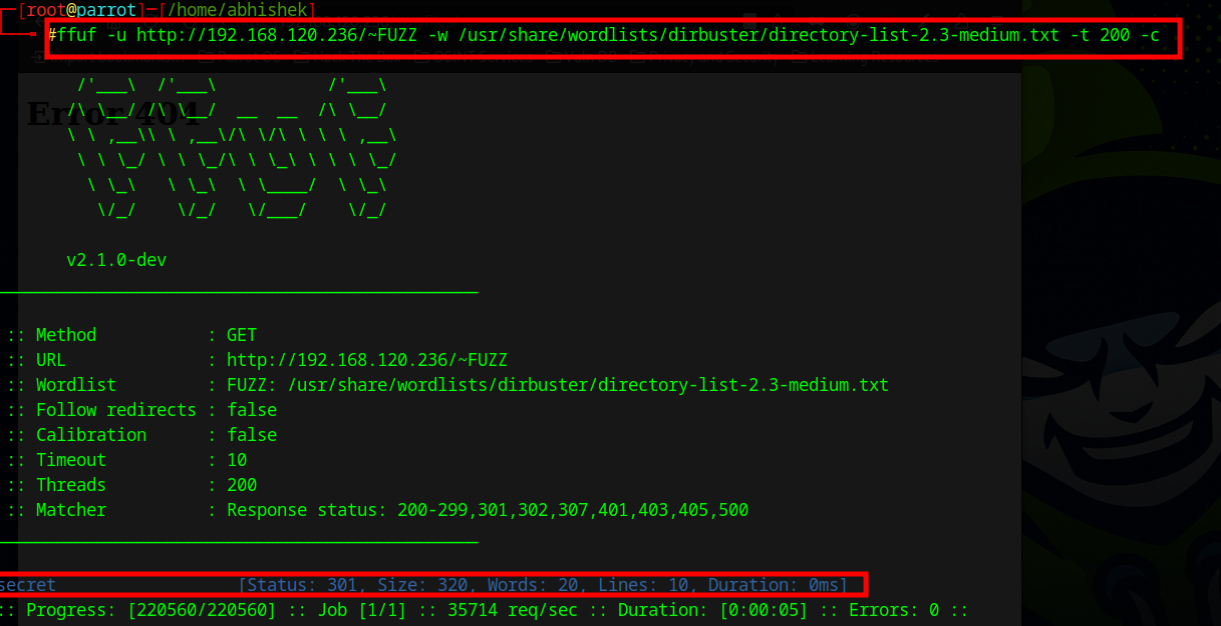
Below Screenshots shows that attacker found “~myfiles” in robots.txt.



Below screen shows that attacker not able to found anything shws 404



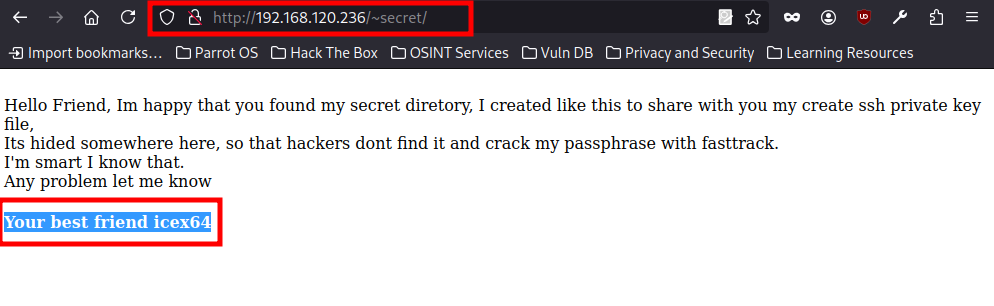
Below screen shows that attacker perform directory listing attack and foun “~secret” file location then again perform fuzzing and found “secret.txt” which contain ssh private ke in decode format

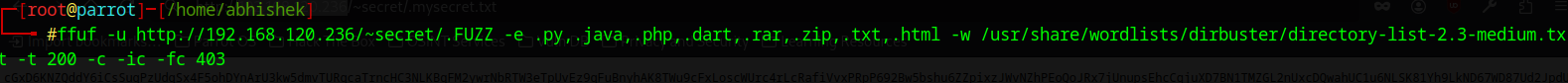
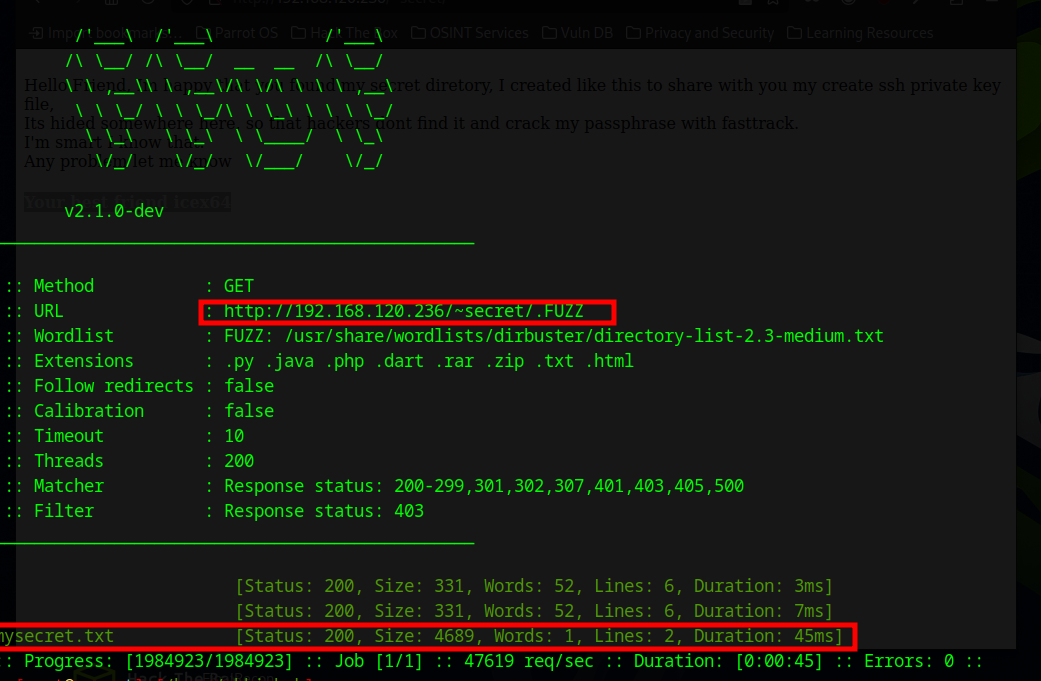


#### Insecure Cryptographic Storage of SSH Private Key (Base58 Encoded)

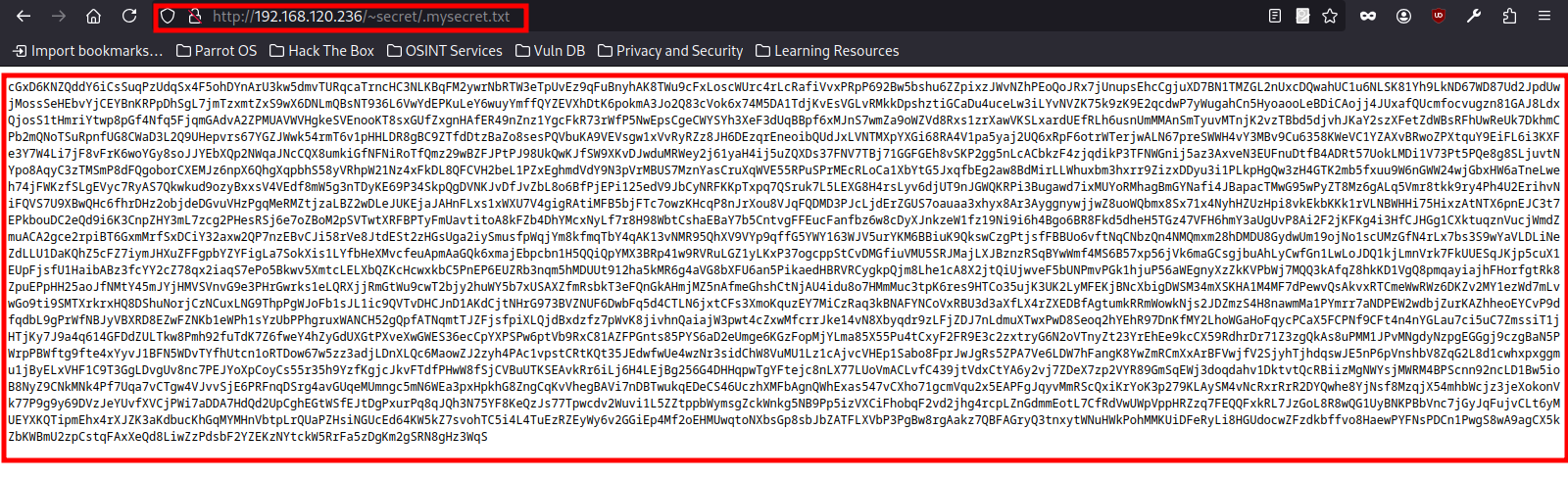
|  |  |
| --- | --- |
| **Vulnerability** | Insecure Cryptographic Storage of SSH Private Key (Base58 Encoded) |
| **Description** | The vulnerability involves the storage of an SSH private key in a weakly protected format (Base58 encoded) within a secret.txt file. While Base58 encoding provides minimal obfuscation, it does not offer real cryptographic protection. Attackers who discover this file can easily decode the key and then attempt to crack its password protection using tools like John the Ripper. |
| **Risk/Impact** | attackers can gain unauthorized access to systems where this key is authorized and Compromise of SSH keys often leads to elevated access privileges. |
| **CVSS Score** | 9.5 Critical |
| **Path:** | 192.168.120.236 |
| **Remidiation / Solution** | Remidiation:  - Remove encoded keys from insecure locations: Never store private keys in text files, even if encoded  - Use proper key encryption: Store SSH private keys with strong passphrase protection using modern algorithms (Ed25519 preferred)  - Implement secure storage: Use dedicated key management systems or hardware security modules (HSMs)  - Rotate compromised keys: Immediately revoke and replace any keys that may have been exposed  - Enforce strong passphrases: Require complex, long passphrases for all encrypted private keys  - Audit key usage: Regularly review authorized\_keys files and key usage patterns  - Consider keyless authentication: For some use cases, consider certificate-based authentication instead of static keys |
| **Refrence Url:** | N/A |

Below Screenshots shows that attacker found clue on “~secret”.

  
  
Below Screenshot shows that attacker perform fuzzing and fouund “secret.txt” file which contain encode algorithm.

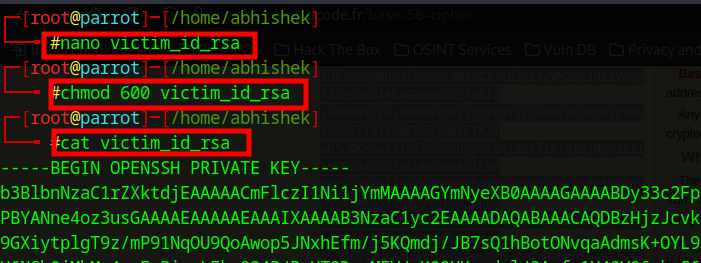
Below screenshot shows that attacker decode that file and found SSH private key.



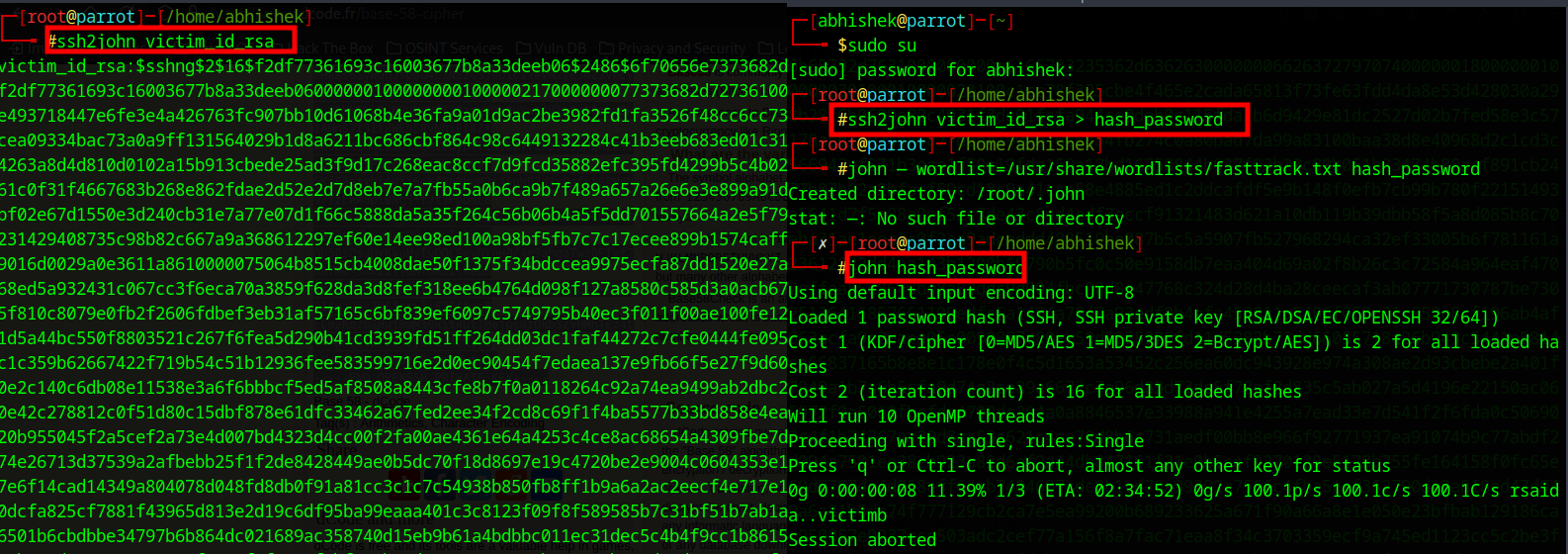


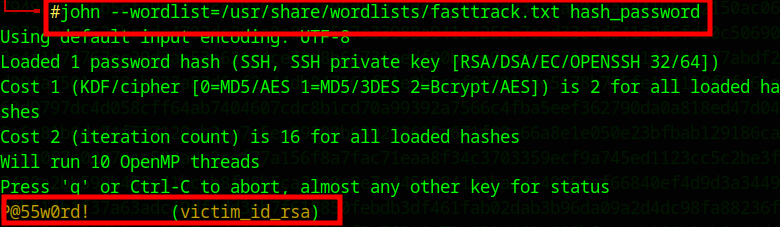
#### Privilege Escalation via Weak SSH Key Passphrase

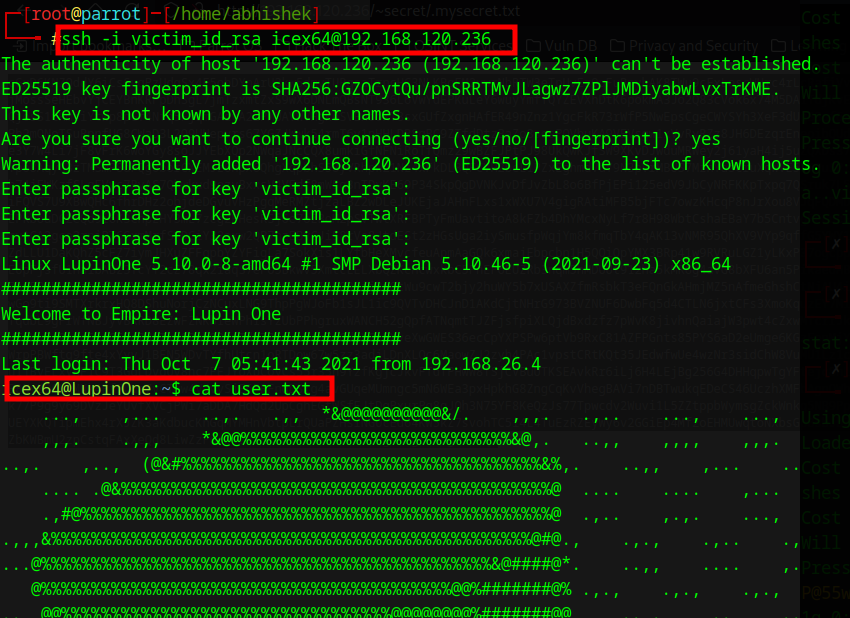
|  |  |
| --- | --- |
| **Vulnerability** | Privilege Escalation via Weak SSH Key Passphrase |
| **Description** | This vulnerability arises when an SSH private key is stored in an insecure manner—such as being encoded in Base58 within a secret.txt file—instead of being properly encrypted with a strong passphrase. Attackers who discover this key can decode it back to its original form (id\_rsa) and then extract its cryptographic hash for offline cracking. Tools like ssh2john convert the encrypted key into a crackable format, and password-cracking utilities like John the Ripper can recover weak passphrases in minutes or hours. Once the passphrase is cracked, the attacker gains unrestricted SSH access to any system where the key is authorized. |
| **Risk/Impact** | The exploitation of this vulnerability leads to full system compromise, allowing attackers to bypass authentication mechanisms entirely. Unlike password-based attacks, which might be mitigated by account lockouts or password changes, stolen SSH keys provide persistent access until manually revoked. If the key is associated with an administrative account, privilege escalation to root is trivial through sudo or su misconfigurations. Additionally, since SSH keys are often trusted for automated processes, attackers can leverage them to move undetected across networks, exfiltrate data, or deploy malware. |
| **CVSS Score** | 10.0 Critical |
| **Path:** | 192.168.120.236 |
| **Remidiation / Solution** | Remidiation:  - Never store private keys in plaintext or encoded formats in insecure locations.  - Use password managers or hardware security modules (HSMs) for key storage.  - Use long, complex passphrases (20+ chars, mix of symbols, numbers, uppercase/lowercase).  - Rotate keys periodically (e.g., every 6-12 months).  - Audit authorized\_keys files to detect unauthorized entries.  - Implement Multi-Factor Authentication (MFA)  Combine SSH keys with TOTP (Google Authenticator) or certificate-based auth for extra security.  - Monitor SSH logs for unusual login attempts |
| **Refrence Url:** | N/A |

Below Screenshots shows that attacker create a ID\_rsa file using that encode format.  


Below Screenshots how that attacker now create a HASH of that encode format and perform Brute force attack.

  
  
Below screenshots shows that attacker found password and successfully gain access.





**END OF REPORT**