Project Phoenix Report – KillChain

1. Executive Summary

This Project Phoenix engagement simulated a vulnerability-management workflow against a deliberately vulnerable asset, **Metasploitable 2** (192.168.56.101), hosted in a controlled lab environment.

Using **Greenbone Vulnerability Manager (GVM)** we performed an authenticated vulnerability scan, exported the results in XML, and fed them into a custom **Python risk engine**. The engine combined technical severity (CVSS) with business criticality to create a prioritized **risk register**.

Results show that Metasploitable 2 is critically exposed: multiple remote code-execution vulnerabilities, default credentials, outdated services, and weak cryptography. The current risk posture would be unacceptable in a production environment and is appropriate only for a training lab.

2. Environment & Asset Overview

Lab Topology

- **Scanner:** Kali Linux with GVM / OpenVAS
- **Target:** Metasploitable 2 (Ubuntu 8.04-based vulnerable VM)
- **Network:** Isolated host-only network on VMware

Asset Inventory (assets.csv)

IP Address	Asset Name	Asset Owner	Asset Criticality (1–5)
192.168.56.101	Metasploitable 2	KillChain	5 (Critical)

The group treated Metasploitable 2 as a **critical business system** (criticality = 5), meaning successful compromise would have maximum impact in a real organization.

3. Vulnerability Scanning (GVM)

3.1 Scan Configuration

- **Scanner:** Greenbone Vulnerability Manager / OpenVAS
- **Target:** 192.168.56.101

- **Scan policy:** Full & fast (default GVM policy)
- **Output:** XML report (report.xml), later consumed by the Python script

The scan identified a large number of findings across network services (FTP, SSH, Telnet, HTTP, SMB, databases, VNC, etc.) and web applications (Mutillidae, DVWA, phpMyAdmin, TWiki, Tomcat).

3.2 Raw Findings Highlights

The GVM report included dozens of vulnerabilities. Sample high-level categories:

- Remote code execution and backdoor access
- Default or weak credentials
- Cleartext authentication and data transmission
- End-of-life operating system and libraries
- Cryptographic weaknesses (weak ciphers, old protocols)
- Web-app issues (XSS, CSRF, directory traversal, information disclosure)

These raw findings were then exported in XML form and passed to the Python risk engine.

4. Python Intelligence Engine

gvm_risk_prioritizer.py performs three main tasks:

1. **Parse assets (assets.csv)**
    a. Reads each row with: ip_address, asset_name, asset_owner, asset_criticality.
    b. Stores a mapping from IP to asset metadata.
2. **Parse GVM XML (report.xml)**
    a. Iterates over each <result> in the XML.
    b. Extracts:
        i. host IP
        ii. Vulnerability name (nvt/name)
        iii. Description (nvt/description)
        iv. CVSS score (from tags like cvss_base, cvss3_base_score, etc.).
3. **Compute Risk & Build Risk Register**
    a. Maps **CVSS to Likelihood (1–5)** using this logic:
        i. CVSS ≥ 9.0 → Likelihood 5
        ii. CVSS ≥ 7.0 → Likelihood 4
        iii. CVSS ≥ 4.0 → Likelihood 3

iv. CVSS > 0 → Likelihood 2
v. CVSS ≤ 0 or missing → 1

b. **Impact** is taken from asset_criticality (Metasploitable = 5).
c. **Risk score = Impact × Likelihood**
i. For this project: risk_score = 5 × likelihood, so the highest risks are 25.
d. Writes out **risk_register.csv** with columns:

ip_address, asset_name, asset_owner, asset_criticality, vuln_name, description, cvss, likelihood, impact, risk_score

e. Sorts rows by descending risk_score so the most dangerous findings appear first.

This pipeline turns a long XML report into a prioritized, business-aware list of risks.

5. Risk Register & Key Findings

All high-priority findings for 192.168.56.101 share:

- **Asset criticality:** 5 (critical system)
- **Likelihood:** 4–5
- **Risk score:** 20–25

Below are some of the most important entries from your risk_register.

5.1 Critical Risks (Risk Score = 25)

These are CVSS ≥ 9.0 with critical impact (5) and likelihood (5).

1. **Possible Backdoor: Ingreslock**
   a. **Description:** Service responds to id; command as uid=0(root) gid=0(root).
   b. **Risk:** Evidence of a backdoor service allowing direct root-level command execution.
   c. **Impact:** Full system compromise, persistent attacker access.
2. **TWiki XSS and Command Execution Vulnerabilities**
   a. **Description:** Outdated TWiki version (01.Feb.2003) vulnerable to cross-site scripting and remote command execution.
   b. **Risk:** Attackers can execute arbitrary commands via the web app and pivot deeper into the host.
3. **rlogin Passwordless Login**
   a. **Description:** Remote rlogin access to root without a password.

    b. **Risk:** Immediate, authenticated shell as root over the network; no brute force required.
4. **Operating System End-of-Life (Ubuntu 8.04)**
    a. **Description:** Underlying OS is long past its EOL (2013-05-09); no security patches.
    b. **Risk:** All unpatched kernel and userland vulnerabilities remain exploitable; attackers can chain numerous known CVEs.
5. **Distributed Ruby (dRuby/DRb) Multiple RCE Vulnerabilities**
    a. **Description:** dRuby service allows arbitrary syscalls, returning stack traces that confirm code execution.
    b. **Risk:** Remote RCE enabling arbitrary command execution and lateral movement.
6. **rexec Service Enabled**
    a. **Description:** Legacy rexec service is running.
    b. **Risk:** Plain-text, unauthenticated/weakly-authenticated remote command execution channel.
7. **vsftpd Compromised Source Packages Backdoor**
    a. **Description:** FTP server version 2.3.4 known to contain a built-in backdoor.
    b. **Risk:** Specially crafted login opens a shell with root privileges.
8. **Apache Tomcat AJP 'Ghostcat' RCE**
    a. **Description:** AJP connector allows reading /WEB-INF/web.xml and potentially arbitrary files.
    b. **Risk:** Leads to disclosure of configuration and, in real deployments, remote code execution.
9. **MySQL / MariaDB Default Root Credentials**
    a. **Description:** Login as root with empty password is allowed.
    b. **Risk:** Full control of the database, data theft, and abuse of SQL as a pivot into OS.
10. **PHP Remote Code Execution Vulnerabilities**
    a. **Description:** Extremely outdated PHP (<5.3 / <5.6.30) with known RCE tricks, confirmed via crafted HTTP requests executing phpinfo().
    b. **Risk:** Web-level RCE, easy to exploit via browser or scripts.
11. **DistCC RCE (CVE-2004-2687)**
    a. **Description:** Remote attacker can run commands like id via the distcc service.
    b. **Risk:** Direct remote command execution pathway.
12. **VNC Brute Force Login (password password)**
    a. **Description:** VNC accepts trivial password password.
    b. **Risk:** Attacker gains full remote desktop access to the system.
13. **PostgreSQL Default Credentials**
    a. **Description:** Login as postgres with password postgres.
    b. **Risk:** Database takeover, potential OS-level control via functions.

These findings all have **risk_score = 25**, meaning **critical asset × highest likelihood**.

5.2 High Risks (Risk Score = 20)

Examples (Impact = 5, Likelihood = 4):

- rsh Unencrypted Cleartext Login
- FTP Brute Force Logins With Default Credentials
- Java RMI Server Insecure Default Configuration RCE
- Test HTTP dangerous methods (PUT/DELETE allowed)
- SSL/TLS: OpenSSL CCS Man-in-the-Middle

These still allow compromise or serious data exposure, but with slightly lower CVSS.

5.3 Medium and Low Risks (Risk Score ≤ 15)

These include:

- Deprecated SSL/TLS protocols (SSLv2, SSLv3, TLS 1.0)
- Weak SSH ciphers, MACs, and key-exchange algorithms
- Information disclosure:
    - phpinfo() pages
    - Cleartext login forms over HTTP
    - Directory browsing (/doc)

While each is less catastrophic alone, they collectively increase the attack surface and support chaining with higher-impact vulnerabilities.

5.4 "(unknown)" Asset Rows

At the bottom of the risk_register are rows where:

- ip_address is blank
- asset_name = (unknown)
- asset_criticality = 1

These are artifacts from GVM results whose IPs did not match any entry in assets.csv. The script correctly defaulted their impact to **1**, keeping them from polluting the top of the risk register. In a real environment, the team would either:

- Add those IPs to assets.csv, or

- Explicitly ignore them as out-of-scope.

6. Recommendations

1. **Do not deploy Metasploitable 2 in production.**

It is intentionally vulnerable and should remain confined to an isolated lab.

2. **Patch or replace end-of-life components.**
    a. Replace Ubuntu 8.04 with a supported Linux distribution.
    b. Upgrade PHP, Apache, Tomcat, and other server software to current versions.
3. **Disable legacy and insecure services.**
    a. Remove rlogin, rexec, rsh, Telnet, and anonymous FTP.
    b. Disable distcc, dRuby, and unnecessary remote services.
4. **Enforce strong authentication.**
    a. Remove default and empty passwords from MySQL, PostgreSQL, VNC, and FTP.
    b. Implement SSH key-based authentication and disable weak algorithms.
5. **Harden web applications.**
    a. Remove or secure test apps (Mutillidae, DVWA, sample phpinfo pages).
    b. Fix XSS, CSRF, directory traversal, and dangerous HTTP methods (PUT/DELETE).
6. **Improve crypto posture.**
    a. Disable SSLv2/SSLv3/TLS1.0 and export-grade cipher suites.
    b. Use modern TLS (1.2+/1.3) and 2048-bit+ RSA or elliptic-curve keys.
7. **Repeat scanning after remediation.**
    a. Re-run GVM scans and compare a new risk_register.csv to confirm risk reduction.

7. Conclusion

Project Phoenix successfully demonstrated an end-to-end vulnerability-management pipeline:

1. **Scan with GVM** to collect raw technical findings.
2. **Export XML** as the “intelligence dossier.”
3. **Correlate with assets via Python**, converting CVSS into likelihood, applying business impact, and generating a **prioritized risk register**.

For the KillChain environment, the analysis shows that Metasploitable 2 is **heavily compromised by design**. In a real organization, these findings would trigger emergency remediation and strong access controls.

This project shows how combining **scanning**, **asset context**, and **automation** can transform raw vulnerability data into actionable security intelligence.