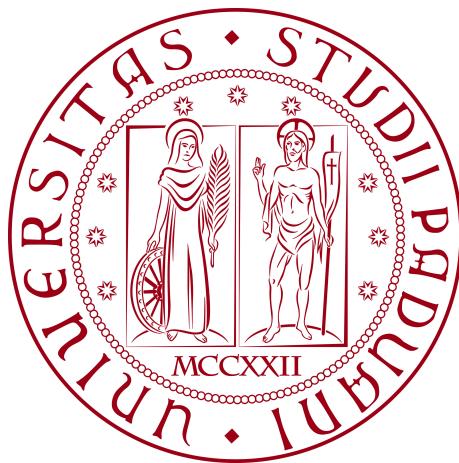


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**Un sistema Honeypot per l'analisi di attacchi
informatici in ambiente controllato**

Tesi di Laurea

Relatore

Prof. Vardanega Tullio

Laureando

Marko Peric

Matricola 2011067

© Marko Peric, Agosto 2025. Tutti i diritti riservati. Tesi di Laurea: “*Un sistema Honeypot per l’analisi di attacchi informatici in ambiente controllato*”, Università degli Studi di Padova, Dipartimento di Matematica “Tullio Levi-Civita”, Corso di Laurea in Informatica.

Ringraziamenti

Padova, Agosto 2025

Marko Peric

Sommario

Il presente elaborato descrive il lavoro svolto durante il periodo di *stage*, della durata di circa 320 ore, dal laureando Marko Peric presso l’azienda Eurosistem S.p.A. dal 18/06/2024 al 13/08/2025. L’elaborato illustra i processi, le metodologie e gli strumenti impiegati nella progettazione e nello sviluppo di un sistema *honeypot* finalizzato al monitoraggio e alla rilevazione di attività malevole in rete. Il sistema si basa sull’installazione di servizi volutamente vulnerabili su un server *Linux Debian*, con lo scopo di attirare potenziali attaccanti e analizzarne i comportamenti. I *log* generati dai servizi sono poi raccolti, elaborati e trasferiti in *InfluxDB* per la loro conservazione, e resi disponibili tramite *Grafana* per consentire un’analisi e una visualizzazione efficace delle informazioni tramite *dashboard*.

Struttura dell’elaborato

Questa sezione presenta la struttura dell’elaborato, illustrando brevemente il contenuto di ciascun capitolo:

Il primo capitolo approfondisce il contesto aziendale in cui si è svolto lo *stage*, fornendo una panoramica completa dell’azienda, dei suoi prodotti e servizi nel settore della *cybersecurity*, degli strumenti tecnologici utilizzati e dell’organizzazione del *team* di sicurezza informatica, analizzando inoltre l’approccio dell’azienda verso l’innovazione tecnologica;

Il secondo capitolo identifica e analizza il problema alla base del progetto, definendo gli obiettivi specifici dello sviluppo del sistema *honeypot*, i vincoli operativi e tecnici che hanno influenzato il lavoro, le prospettive di

sviluppo futuro e le motivazioni personali che hanno guidato la scelta di questo percorso di *stage*;

Il terzo capitolo documenta l'intero processo di sviluppo del sistema *honeypot*, dall'analisi dei requisiti e dei casi d'uso alla progettazione dell'architettura, dalla fase di codifica alle attività di verifica e validazione, presentando il prodotto finale e valutando il grado di conformità rispetto ai requisiti iniziali;

Il quarto capitolo presenta una valutazione retrospettiva dell'esperienza formativa, analizzando il raggiungimento degli obiettivi prefissati, le conoscenze tecniche acquisite nel campo della sicurezza informatica e le competenze professionali sviluppate durante il periodo di tirocinio presso l'azienda.

Criteri tipografici adottati

Riguardo la stesura del testo, relativamente al elaborato vengono adottate le seguenti convenzioni tipografiche:

- gli acronimi e le abbreviazioni sono raccolti in un'apposita sezione dedicata, denominata [Elenco degli acronimi](#) e sono segnati con la seguente nomenclatura *Esempio di acronimo (ES)_G*;
- i termini ambigui o di uso non comune, invece, sono definiti all'interno del [Glossario](#) e sono segnati con la seguente nomenclatura *esempio di termine_G*;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Indice

1	Contesto lavorativo	1
1.1	L’azienda	1
1.2	Prodotti e servizi offerti	2
1.3	Strumenti <i>hardware</i>	4
1.4	Strumenti <i>software</i>	5
1.4.1	Strumenti organizzativi	5
1.4.2	Strumenti produttivi	5
1.5	Organizzazione del <i>team</i>	7
1.5.1	<i>Chief Information Security Officer</i> (CISO)	7
1.5.2	<i>Penetration tester</i>	7
1.5.3	<i>Security analyst</i>	7
1.6	Predisposizione all’innovazione	8
2	Scopo del progetto	12
2.1	Il ruolo dei tirocini in Eurosystem S.p.A.	12
2.2	Analisi del problema	13
2.3	Obiettivi	14
2.4	Vincoli	16
2.4.1	Vincoli tecnologici	16
2.4.2	Vincoli metodologici	18
2.4.3	Vincoli temporali	18
2.5	Prospettive future	18
2.6	Motivazioni personali	19

INDICE

3 Sviluppo del progetto	22
3.1 Analisi dei requisiti	22
3.1.1 Casi d'uso	23
3.1.2 Requisiti	31
3.2 Progettazione	33
3.2.1 Architettura del sistema	33
3.2.2 Scelta dei servizi del sistema	35
3.3 Codifica	40
3.3.1 Struttura del progetto	40
3.3.2 Difficoltà incontrate	42
3.4 Verifica	43
3.4.1 Test e strumenti	46
3.4.2 Risultati quantitativi	47
3.5 Validazione	48
3.6 Visione d'insieme	51
3.6.1 Prodotto finale	51
4 Valutazione retrospettiva	i
4.1 Obiettivi raggiunti	i
4.2 Conoscenze acquisite	i
4.3 Competenze professionali acquisite	i
Sitografia	ii
Acronimi e abbreviazioni	iv
Glossario	vi

Elenco delle figure

1.1	Schema di esempio di un processo di penetration testing.	3
1.2	Schema di esempio di un processo di gestione degli endpoint.	4
1.3	Dispositivo <i>USB Rubber Ducky</i> . Fonte: https://shop.hak5.org	9
1.4	Dispositivo <i>LAN Turtle</i> . Fonte: https://shop.hak5.org	10
2.1	Esempio di utilizzo di un sistema <i>honeypot</i>	14
2.2	Schema di esempio dell'integrazione tra protocollo <i>MQTT</i> e <i>stack TIG</i>	17
3.1	Esempio di caso d'uso e sotto-caso d'uso.	23
3.2	Esempio di attore.	24
3.3	Architettura del sistema <i>ETL</i>	34
3.4	Diagramma di flusso del sistema <i>honeypot</i>	35
3.5	Diagramma delle classi del sistema <i>honeypot</i>	38
3.6	Rappresentazione dello schema a V adottato nel progetto.	45
3.7	Schema del ciclo TDD utilizzato durante lo sviluppo.	46
3.8	Risultato della copertura del codice dopo l'esecuzione dei <i>test</i>	48
3.9	Scelta del livello minimo del <i>logger</i> al momento dell'inizializzazione.	51
3.10	Esempio di <i>logger</i> in funzione in modalità <i>debug</i>	52
3.11	Interfaccia web di <i>InfluxDB</i> per la gestione del <i>database</i>	52
3.12	Esempio di <i>dashboard</i> di <i>Grafana</i> per la visualizzazione dei dati raccolti dall' <i>honeypot</i>	53
3.13	Esempio di <i>dashboard</i> di <i>Grafana</i> per la visualizzazione dei dati raccolti dall' <i>honeypot</i>	53

3.14 Esempio di <i>dashboard</i> di <i>Grafana</i> per la visualizzazione dei dati raccolti dall' <i>honeypot</i>	53
---	----

Elenco delle tabelle

2.1 Elenco degli obiettivi suddivisi per categoria.	16
2.2 Obiettivi personali del tirocinio.	21
3.1 Diagramma del caso d'uso UC.1 - Visualizzare i <i>log</i> del sistema <i>honeypot</i>	25
3.2 Diagramma del caso d'uso UC.2 - Visualizza errore interno durante l'avvio del sistema.	26
3.3 Diagramma del caso d'uso UC.3 - Impostare il livello minimo dei <i>log</i> da visualizzare.	27
3.4 Diagramma del caso d'uso UC.4 - Visualizzare i dati presenti in <i>InfluxDB</i>	28
3.5 Diagramma del caso d'uso UC.5 - Visualizzare una <i>dashboard</i> in <i>Grafana</i>	29
3.6 Diagramma del caso d'uso UC.6 - Modificare una <i>dashboard</i> in <i>Grafana</i>	30
3.7 Diagramma del caso d'uso UC.7 - Visualizza errore: <i>query</i> del pannello non valida.	31
3.8 Tabella quantitativa dell'analisi dei requisiti.	33
3.9 Tabella quantitativa delle attività di modellazione e configurazione.	39
3.10 Tabella quantitativa delle metriche di implementazione del codice.	42
3.11 Tabella quantitativa delle configurazioni e degli script ausiliari. .	42

ELENCO DELLE TABELLE

3.12 Tabella quantitativa delle attività di <i>testing</i>	47
3.13 Tabella quantitativa dei bug rilevati e del tempo di risoluzione medio.	47
3.14 Requisiti soddisfatti dal sistema sviluppato.	49
3.15 Tabella quantitativa degli scenari di attacco testati e dei log generati.	50
3.16 Tabella quantitativa degli strumenti e dei risultati dei test. . . .	50

Elenco dei codici sorgenti

Capitolo 1

Contesto lavorativo

Il primo capitolo fornisce una panoramica sull’azienda ospitante Eurosistem S.p.A., evidenziando le principali aree operative dell’azienda.

1.1 L’azienda

Eurosistem S.p.A. è un’impresa informatica con sede principale a Villorba (TV), attiva nel settore della consulenza, dei prodotti e dei servizi di *Information Technology (IT)*. L’azienda è presente sul territorio nazionale con otto sedi operative, distribuite principalmente nel Nord Italia. Tra queste, la sede di Tavagnacco (UD) riveste un ruolo centrale in quanto ospita il reparto dedicato alle attività di *cybersecurity*, ambito in cui si è svolto il tirocinio curricolare in modalità da remoto. Fondata con l’obiettivo di fornire soluzioni tecnologiche a supporto delle imprese, Eurosistem S.p.A. si rivolge prevalentemente al settore delle piccole e medie imprese, accompagnandole nei processi di digitalizzazione e nella gestione dei rischi informatici. L’azienda si caratterizza per un approccio che combina l’adozione di tecnologie consolidate con l’attenzione verso l’innovazione e la sicurezza delle infrastrutture digitali. Il reparto di *cybersecurity* si distingue per la sua funzione trasversale, in quanto collabora sia con i clienti sia con gli altri reparti aziendali, integrando misure di protezione all’interno delle soluzioni tecnologiche fornite. L’attenzione di Eurosistem S.p.A. non è rivolta unicamente alla fornitura di servizi, ma anche al mantenimento e all’aggiornamento costante delle competenze del personale. La formazione continua

rappresenta un elemento fondamentale: nei periodi di minore operatività, i *team* dedicano tempo allo studio di nuove tematiche, all’analisi di strumenti emergenti e alla sperimentazione di metodologie utili a fronteggiare l’evoluzione costante delle minacce informatiche.

1.2 Prodotti e servizi offerti

Eurosistem S.p.A. offre una gamma di prodotti e servizi informatici pensati per garantire la protezione delle infrastrutture *IT* aziendali. In particolare, il reparto di *cybersecurity* si occupa di diverse attività fondamentali per la difesa e la continuità operativa delle organizzazioni.

Un primo ambito riguarda **il monitoraggio e la risposta agli incidenti**. Il *Security Operation Center (SOC)_G* e i servizi di *Managed Detection and Response (MDR)_G* assicurano un controllo costante delle reti aziendali, con l’obiettivo di rilevare in tempo reale anomalie e potenziali minacce. Il *team* esamina ogni incidente nel dettaglio mediante la raccolta e l’analisi dei *log* e dei dati di rete, l’identificazione dei vettori di attacco e delle vulnerabilità sfruttate, l’isolamento delle minacce e il ripristino dei sistemi tramite soluzioni di *backup* e *disaster recovery*. A completamento del processo, il team redige *report* dettagliati contenenti analisi, azioni intraprese e raccomandazioni di miglioramento.

Un secondo ambito è rappresentato dalle **verifiche di sicurezza**, svolte tramite attività di *Cyber Analysis Assessment*. In questa fase rientrano il *Vulnerability Management* e i *Penetration Test*, strumenti fondamentali per individuare e valutare i punti deboli delle infrastrutture *IT*. A ciò si affiancano i servizi di *Phishing Assessment*, utili a misurare la resilienza degli utenti contro campagne fraudolente, e le attività di *Threat Intelligence*, finalizzate a raccogliere e analizzare informazioni utili a prevenire attacchi futuri.

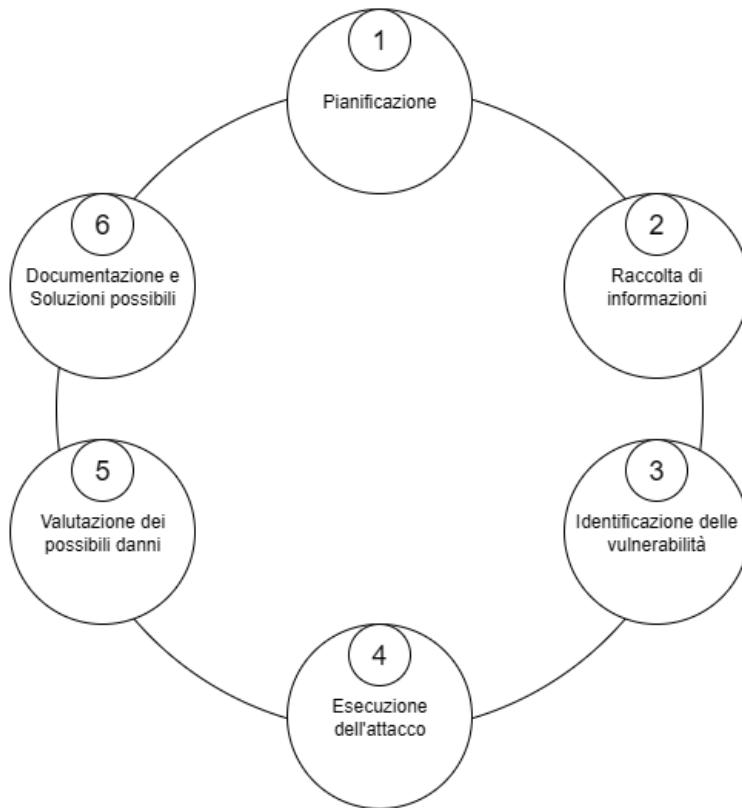


Figura 1.1: Schema di esempio di un processo di penetration testing.

L'**analisi delle reti** rappresenta un aspetto importante delle attività svolte. Questa comprende il monitoraggio del traffico di rete, il rilevamento di anomalie e la gestione degli eventi di sicurezza, con particolare attenzione alle reti industriali, al fine di contribuire alla continuità operativa dei processi produttivi.

Un ruolo centrale è svolto anche dalla **gestione degli endpoint**, ossia la protezione dei dispositivi aziendali, interni o remoti. Questa attività comprende il monitoraggio costante dello stato di sicurezza, l'applicazione degli aggiornamenti correttivi e delle politiche di protezione, il controllo delle identità e degli accessi, nonché il rilevamento e la risposta rapida alle minacce. Inoltre, sono previsti *report* e *audit* di conformità, volti a garantire trasparenza e allineamento con le normative.

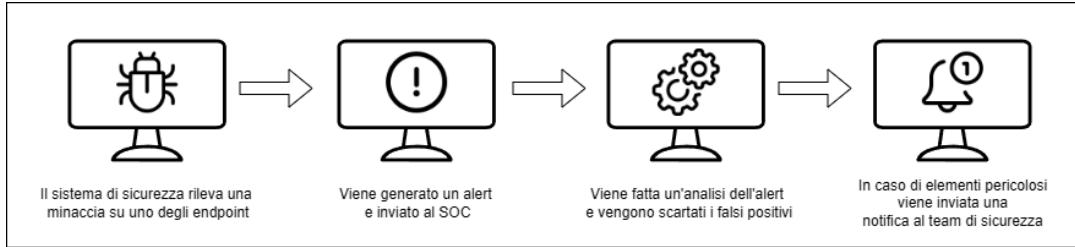


Figura 1.2: Schema di esempio di un processo di gestione degli endpoint.

Alla gestione degli endpoint si affianca la **gestione dei dati**. Eurosistem S.p.A. supporta le aziende nella protezione, nell'organizzazione e nella conformità dei dati, in linea con i requisiti normativi, assicurando così un trattamento sicuro e affidabile delle informazioni.

Un altro elemento di rilievo è la **gestione degli utenti e la sensibilizzazione**. L'azienda promuove attività formative mirate a diffondere buone pratiche di sicurezza informatica, con particolare attenzione agli accessi, ai privilegi e ai comportamenti corretti. In questo modo viene ridotto il rischio legato al fattore umano, spesso determinante nella riuscita di un attacco informatico.

Infine, Eurosistem S.p.A. offre un **supporto tempestivo in caso di criticità**. In presenza di incidenti o sospette violazioni, i clienti ricevono un'assistenza diretta e immediata, con l'obiettivo di ridurre al minimo l'impatto sull'operatività aziendale. Grazie a questo insieme di attività, l'azienda è in grado di identificare proattivamente le vulnerabilità dei sistemi informativi, predisporre misure di protezione adeguate e garantire un aggiornamento costante delle competenze necessarie per affrontare l'evoluzione delle minacce informatiche.

1.3 Strumenti *hardware*

Nel caso in cui le attività lavorative vengano svolte da remoto, l'azienda mette a disposizione del lavoratore gli strumenti necessari per garantire il corretto svolgimento delle mansioni assegnate. In particolare, l'azienda fornisce un computer portatile HP, configurato con l'account e la casella di posta elettronica

aziendale, così da assicurare l'accesso immediato a tutte le risorse interne. A supporto dell'utilizzo del dispositivo, è inoltre fornito un mouse.

1.4 Strumenti *software*

Le tecnologie e gli strumenti software a supporto delle attività lavorative si articolano principalmente in due grandi categorie: strumenti organizzativi e strumenti produttivi.

1.4.1 Strumenti organizzativi

Per quanto riguarda gli strumenti organizzativi, l'azienda fa ampio uso di **Microsoft Teams**, che rappresenta il principale canale di comunicazione interna. Questo strumento consente di coordinare le attività quotidiane, condividere documenti in modo rapido ed efficace e partecipare a riunioni o chiamate di lavoro quando necessario.

Un ruolo altrettanto centrale è ricoperto da **Microsoft Outlook**, utilizzato non solo per la gestione della posta elettronica e dell'agenda, ma anche per la pianificazione di incontri e il mantenimento di un flusso di comunicazione costante, sia con i colleghi che con i clienti. *Outlook* svolge inoltre una funzione cruciale nella ricezione di notifiche riguardanti incidenti di sicurezza provenienti da aziende esterne, garantendo così risposte tempestive in caso di necessità.

A completare questo insieme di strumenti organizzativi vi è **GitHub**, adottato come piattaforma di controllo versione. Questo strumento consente la gestione strutturata del codice sorgente, il tracciamento delle modifiche e la collaborazione efficiente tra più sviluppatori.

1.4.2 Strumenti produttivi

Tra gli strumenti produttivi, un ruolo rilevante è svolto da **LaTeX**, scelto per la redazione di documentazione tecnica e scientifica grazie alle sue eleva-

te capacità di formattazione. La possibilità di gestire con precisione formule matematiche, grafici, bibliografie e glossari permette di ottenere documenti di elevata qualità tipografica. Il modello di separazione tra contenuto e presentazione rende inoltre il codice sorgente facilmente riutilizzabile e particolarmente adatto a contesti di collaborazione accademica e professionale.

In parallelo, l’azienda fa un ampio utilizzo di ***Python***, linguaggio di programmazione versatile e diffuso, applicato soprattutto nel settore della *cybersecurity*. Le sue potenzialità vengono sfruttate per automatizzare processi, analizzare file di *log* e sviluppare strumenti di monitoraggio e rilevamento delle minacce, anche grazie alle numerose librerie specializzate che semplificano la creazione di script per il *testing* della sicurezza, l’interazione con *Application Programming Interface (API)***G** e la gestione di sistemi complessi.

A supporto dello sviluppo i dipendenti adottano frequentemente ***Visual Studio Code (VSCode)***, un ambiente di programmazione leggero, modulare e altamente personalizzabile. Questo strumento è impiegato principalmente per la scrittura, il *debugging* e la manutenzione del codice, offrendo un’ampia compatibilità con diversi linguaggi e la possibilità di integrare estensioni dedicate. Pur essendo l’*Integrated Development Environment (IDE)***G** *Integrated Development Environment***G** più diffuso tra i dipendenti, la sua adozione non è vincolante: ciascun lavoratore può infatti optare per soluzioni alternative in base alle proprie esigenze e preferenze operative.

Infine, un ulteriore strumento di grande rilevanza è ***Docker***, una tecnologia di virtualizzazione basata sui *container***G**. Grazie a questo approccio, è possibile creare ambienti isolati, scalabili e facilmente replicabili. Docker, infatti, esegue le applicazioni includendo tutte le dipendenze necessarie. La leggerezza e portabilità della tecnologia ne favoriscono l’uso in fase di sviluppo, *testing* e distribuzione, riducendo al minimo i problemi di compatibilità tra diversi sistemi.

1.5 Organizzazione del *team*

1.5.1 *Chief Information Security Officer (CISO)*

Il *Chief Information Security Officer (CISO)* è la figura responsabile della strategia di sicurezza informatica all'interno dell'azienda. Supervisiona tutte le attività del reparto di *cybersecurity* e ne garantisce la coerenza con gli obiettivi aziendali e con le normative vigenti. Durante il periodo di tirocinio, questa figura ha ricoperto anche il ruolo di tutor aziendale, fornendo supporto e guida nelle varie attività. Tra le sue principali responsabilità rientrano la definizione delle politiche e delle strategie di sicurezza, la supervisione delle attività operative del *team*, il coordinamento tra i diversi ruoli del reparto e la verifica della conformità agli standard di settore. A ciò si aggiunge un impegno costante nel supporto e nella formazione interna, volto a diffondere buone pratiche di sicurezza informatica tra i dipendenti.

1.5.2 *Penetration tester*

Il *penetration tester* ha l'obiettivo di individuare le vulnerabilità presenti nei sistemi informatici attraverso l'esecuzione di attacchi simulati in un contesto controllato. Il suo lavoro si articola in più fasi, che comprendono la pianificazione e l'esecuzione di *penetration test*, la simulazione di scenari di attacco realistici e l'analisi delle debolezze riscontrate. Al termine delle attività, il professionista redige report tecnici dettagliati nei quali vengono descritte le criticità emerse, corredati da proposte di contromisure e raccomandazioni operative, così da supportare l'azienda nell'adozione di strategie efficaci di mitigazione del rischio.

1.5.3 *Security analyst*

Il *security analyst* si occupa del monitoraggio costante dei sistemi informativi e della gestione operativa degli eventi di sicurezza, con l'obiettivo di rilevare tempestivamente potenziali incidenti e coordinare una risposta adeguata. Le sue attività comprendono il controllo delle infrastrutture tramite strumenti avanzati

di analisi, il rilevamento e la classificazione degli avvisi di sicurezza, nonché l'esame dei *log* e delle anomalie di rete. In caso di incidenti, fornisce supporto operativo nelle attività di risposta e contribuisce alla definizione di procedure correttive. Inoltre, elabora linee guida e buone pratiche da condividere con clienti e colleghi, contribuendo così a diffondere una maggiore consapevolezza in materia di sicurezza informatica.

1.6 Predisposizione all'innovazione

L'ambito della *cybersecurity* è caratterizzato da un'evoluzione continua: nuove vulnerabilità, strumenti di attacco e metodologie di compromissione emergono con frequenza sempre maggiore. Per questo motivo, l'innovazione e l'aggiornamento costante costituiscono elementi fondamentali per ogni azienda che operi in questo settore. Eurosistem S.p.A. ha sviluppato un approccio che integra formazione, ricerca e sperimentazione, con l'obiettivo di anticipare i rischi e fornire soluzioni di difesa sempre più efficaci ai propri clienti. Un aspetto centrale è la formazione continua: nei momenti di minore operatività i *team* si dedicano allo studio di nuove tematiche, alla valutazione di strumenti emergenti e all'adozione di metodologie innovative. In questo modo rimangono aggiornati sull'evoluzione delle minacce informatiche e rafforzano le proprie competenze tecniche. In parallelo, viene dato ampio spazio alla sperimentazione pratica di strumenti e tecniche di attacco. L'azienda valuta regolarmente soluzioni utilizzate dai *penetration tester* durante le verifiche di sicurezza sui sistemi dei clienti.

Tra questi strumenti rientrano dispositivi come:

- **USB Rubber Ducky**, che permette di eseguire comandi automatici sui dispositivi a cui viene collegata, simulando un attacco tramite periferica *Universal Serial Bus (USB)*_G. L'azienda impiega l'*USB Rubber Ducky* per facilitare le attività di *penetration testing* in loco presso le aziende clienti, consentendo di valutare la sicurezza delle postazioni di lavoro rispetto a questo tipo di minacce;



Figura 1.3: Dispositivo *USB Rubber Ducky*.

Fonte: <https://shop.hak5.org>

- ***LAN Turtle***, un adattatore di rete *Local Area Network (LAN)* che consente di instaurare accessi remoti nascosti e di analizzare il traffico direttamente dall'interno della rete aziendale. I *penetration tester* utilizzano il *LAN Turtle* durante le attività di *penetration testing* per verificare l'esposizione delle infrastrutture a minacce derivanti dall'inserimento fisico di dispositivi malevoli;



Figura 1.4: Dispositivo *LAN Turtle*.

Fonte: <https://shop.hak5.org>

L'adozione di tali strumenti non ha un fine puramente dimostrativo, ma rappresenta un'attività concreta di valutazione del livello di sicurezza dei clienti, in quanto permette di evidenziare vulnerabilità legate al fattore umano e alla protezione fisica delle postazioni di lavoro. Oltre agli strumenti hardware, l'azienda analizza e adotta nuove piattaforme e metodologie orientate sia all'offensiva sia alla difensiva. Tra queste si possono citare:

- l'impiego di tecniche avanzate di *Threat Intelligence*, volte a raccogliere e analizzare informazioni relative a campagne malevoli, infrastrutture di attacco e nuove vulnerabilità sfruttate in scenari reali;
- l'aggiornamento continuo dei framework di *penetration testing* e *vulnerability management*, che consente di disporre di strumenti più precisi ed efficaci per identificare punti deboli e priorità di intervento;

CAPITOLO 1. CONTESTO LAVORATIVO

- la sperimentazione di approcci innovativi al *red teaming*, in cui si simulano operazioni complesse di attacco, con l'obiettivo di valutare non soltanto le difese tecnologiche ma anche le capacità organizzative e di risposta dei clienti.

In conclusione, la predisposizione all'innovazione non è intesa come un'attività separata, ma come parte integrante delle operazioni quotidiane. L'integrazione di formazione, sperimentazione e ricerca applicata consente al reparto di *cybersecurity* di affrontare in maniera proattiva le sfide poste dalle minacce informatiche, mantenendo un equilibrio tra lo sviluppo delle competenze interne e l'offerta di servizi sempre più efficaci e aggiornati per i clienti.

Capitolo 2

Scopo del progetto

Il secondo capitolo descrive il ruolo degli *stage* nel contesto aziendale e come questo abbia portato alla definizione degli obiettivi del progetto. Inoltre vengono analizzati i vincoli e le motivazioni personali che hanno influenzato lo sviluppo del progetto stesso.

2.1 Il ruolo dei tirocini in Eurosystem S.p.A.

Eurosystem S.p.A. considera i tirocini un elemento strategico sia per la formazione dei giovani sia per l'innovazione aziendale nel settore della *cybersecurity*. Essi permettono di avvicinarsi concretamente al mondo del lavoro e di contribuire ai progetti aziendali. L'azienda non interpreta il tirocinio soltanto come un'esperienza formativa, ma anche come un'opportunità per sperimentare nuove idee di progetto e per valutare soluzioni tecnologiche che, se ritenute efficaci, possono essere integrate nei servizi offerti ai clienti con l'obiettivo di rafforzarne la sicurezza informatica. L'azienda accoglie i tirocinanti in un contesto lavorativo dinamico, caratterizzato da attività concrete e da un costante confronto con professionisti del settore. Fin dal loro inserimento, hanno la possibilità di applicare le conoscenze già acquisite e di arricchirle con competenze operative, maturate direttamente a contatto con problematiche reali della *cybersecurity*. Questa modalità consente non solo di consolidare il proprio bagaglio tecnico, ma anche di sviluppare capacità trasversali come la collaborazione, la creatività e il *problem solving*, indispensabili per affrontare scenari complessi e in continua

evoluzione. Il tirocinio assume così un ruolo bidirezionale: da un lato favorisce la crescita formativa e professionale del tirocinante, dall’altro rappresenta per Eurosystem S.p.A. un’occasione per esplorare nuove prospettive progettuali e per individuare talenti da coinvolgere attivamente nelle sfide della sicurezza informatica.

2.2 Analisi del problema

Uno dei principali problemi che le organizzazioni si trovano ad affrontare riguarda la capacità di **individuare tempestivamente attività malevole** all’interno dei propri sistemi. La sicurezza informatica rappresenta oggi una delle sfide più complesse per le aziende di qualsiasi settore. Le infrastrutture *IT* moderne sono caratterizzate da un’elevata interconnessione e da una crescente dipendenza da servizi digitali, fattori che le rendono esposte a minacce sempre più sofisticate. Gli attacchi informatici non sono più episodi sporadici, ma eventi ricorrenti che possono compromettere la continuità operativa, causare perdite economiche e danneggiare in modo significativo l’azienda.

Le tecniche di attacco evolvono rapidamente e gli strumenti difensivi tradizionali, come *firewall* e *antivirus*, non sempre riescono a garantire una protezione completa. Gli operatori malevoli, infatti, sfruttano vulnerabilità sconosciute, configurazioni errate o semplicemente il fattore umano, riuscendo così ad aggirare i controlli di sicurezza.

A ciò si aggiunge la difficoltà di comprendere a fondo il comportamento degli attaccanti. Spesso le aziende hanno una visibilità limitata su quali metodi vengano impiegati, su come gli aggressori si muovano una volta ottenuto l’accesso e su quali siano gli obiettivi finali delle intrusioni. Questa mancanza di informazioni rende complesso elaborare strategie di difesa realmente efficaci e impedisce di anticipare possibili minacce future.

In sintesi, il problema principale è legato all’**assenza di strumenti e metodologie** che consentano non solo di rilevare gli attacchi, ma anche di comprenderne le dinamiche e trarne conoscenza utile per migliorare la sicurezza complessiva dei sistemi informativi.

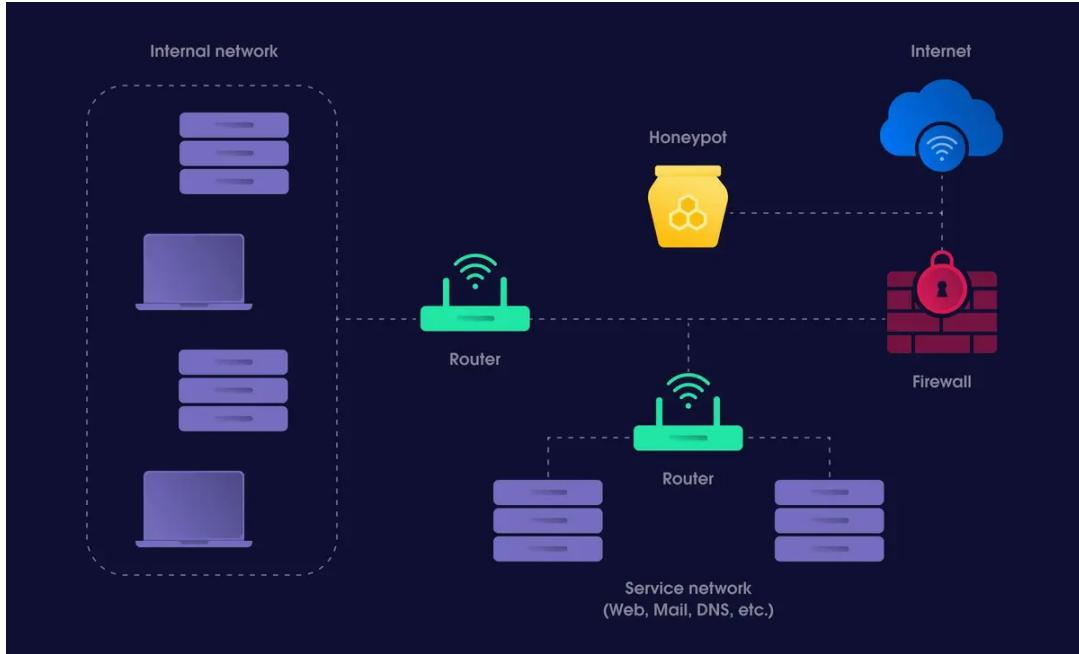


Figura 2.1: Esempio di utilizzo di un sistema *honeypot*.

Fonte: <https://oxylabs.io/blog/what-is-a-honeypot>

2.3 Obiettivi

Gli obiettivi che sono suddivisi in minimi, obbligatori, desiderabili o facoltativi e sono elencati di seguito. Utilizziamo la seguente nomenclatura per suddividere questi ultimi nelle quattro tipologie:

- **M:** sono gli obiettivi vincolanti e necessari per il completamento del progetto;
- **O:** sono gli obiettivi primari richiesti per il progetto;
- **D:** sono gli obiettivi non vincolanti o strettamente necessari, ma che portano valore aggiunto al prodotto;
- **F:** sono gli obiettivi che portano valore aggiunto al prodotto in modo non indispensabile.

Ogni obiettivo è contrassegnato con un numero e la categoria corrispondente, al fine di garantire una classificazione chiara e ordinata.

CAPITOLO 2. SCOPO DEL PROGETTO

Identificativo	Obiettivo
Obiettivi Minimi	
M01	Il codice del progetto deve essere scritto in <i>Python</i> , deve avere struttura modulare e commenti esplicativi.
M02	Il progetto deve presentare configurazioni leggibili, versionate e testate.
M03	Gli <i>script</i> devono essere riutilizzabili e gli <i>input</i> devono essere parametrizzabili.
M04	Il progetto deve avere una documentazione coerente con quanto implementato, scritta in forma tecnica, chiara e verificabile.
Obiettivi Obbligatori	
O01	Deve essere installata e configurata una macchina virtuale isolata per ambienti <i>honeypot</i> .
O02	L' <i>honeypot</i> prodotto deve essere realistico con configurazione di servizi vulnerabili (<i>Apache</i> , <i>File Transfer Protocol (FTP)</i> _G , <i>Server Message Block (SMB)</i> _G , <i>Message Queuing Telemetry Transport (MQTT)</i> _G , ecc.).
O03	Devono essere sviluppati servizi <i>dummy</i> con <i>netcat</i> o strumenti equivalenti per l'ascolto dei pacchetti.
O04	I <i>log</i> devono essere raccolti e centralizzati tramite <i>Python</i> e pipeline <i>Telegraf</i> , <i>InfluxDB</i> , <i>Grafana (TIG)</i> _G .
O05	La raccolta dei <i>log</i> deve essere automatizzata mediante <i>script Python/Bash</i> .
Obiettivi Desiderabili	
D01	Deve essere effettuata un'analisi tecnica degli attacchi ricevuti con relativa correlazione dei dati.
D02	Devono essere integrati strumenti e tecniche di <i>Threat Intelligence</i> e <i>Open Source Intelligence (OSINT)</i> _G per finalità di attribuzione.
Continua nella prossima pagina...	

Tabella 2.1 – Continuo della tabella

Identificativo	Obiettivo
D03	Devono essere simulati attacchi controllati con strumenti noti e devono essere raccolti i relativi <i>output</i> .
D04	Devono essere applicate tecniche di base di <i>data analysis</i> per il riconoscimento di <i>pattern</i> .
D05	Deve essere realizzata una <i>dashboard</i> con grafici, <i>Indicator of Compromise (IOC)G</i> e <i>timeline</i> degli eventi.
Obiettivi Facoltativi	
F01	Devono essere implementati controlli avanzati di <i>audit</i> con <i>auditd</i> e <i>logging Bash</i> persistente.
F02	Deve essere sperimentata la distribuzione dei dati verso <i>Security Information and Event Management (SIEM)G open source</i> .
F03	Devono essere introdotti strumenti di <i>Artificial Intelligence (AI)G/Machine Learning (ML)G</i> per l'identificazione automatica di anomalie.

Tabella 2.1: Elenco degli obiettivi suddivisi per categoria.

2.4 Vincoli

Durante lo sviluppo del progetto l’azienda non ha imposto particolari vincoli di tipo architetturale, ma sono emersi alcuni vincoli tecnologici, progettuali, metodologici e temporali che influenzano le scelte di implementazione.

2.4.1 Vincoli tecnologici

Dal punto di vista tecnologico, l’azienda ha imposto l’utilizzo di *Python* (v.3.12.10) come linguaggio di programmazione. Questa scelta è dettata non solo dalla sua versatilità e dalla disponibilità di numerose librerie utili allo sviluppo.

luppo, ma anche perché rappresenta uno dei linguaggi più diffusi nell’ambito della *cybersecurity*, oltre a essere quello su cui l’azienda possiede maggiore esperienza interna. Per la gestione e la visualizzazione dei dati si è optato per lo *stack TIG*, composto da *Telegraf* (v.1.24), *InfluxDB* (v.2.7) e *Grafana* (v.9.5.6), preferito al più diffuso *stack Elasticsearch, Logstash, Kibana (ELK)G* per la sua leggerezza e per la maggiore aderenza ai requisiti del progetto. Anche le modalità di comunicazione tra i diversi componenti sono soggette a vincoli: il tutor ha infatti imposto l’uso del protocollo *MQTT*, particolarmente adatto allo scambio di dati in tempo reale con un consumo ridotto di risorse. Al contrario, non erano previsti vincoli riguardo ai servizi vulnerabili da esporre all’interno del sistema *honeypot*, lasciando piena libertà di scelta in base alle esigenze di implementazione. Sul piano progettuale, il tutor aziendale ha stabilito che il sistema dovesse includere un *logger* interno. Questo vincolo ha avuto un impatto diretto sulla progettazione complessiva, in quanto ha reso necessaria l’integrazione di meccanismi di *logging* fin dalle fasi iniziali, in modo da facilitare il *debugging* durante lo sviluppo.

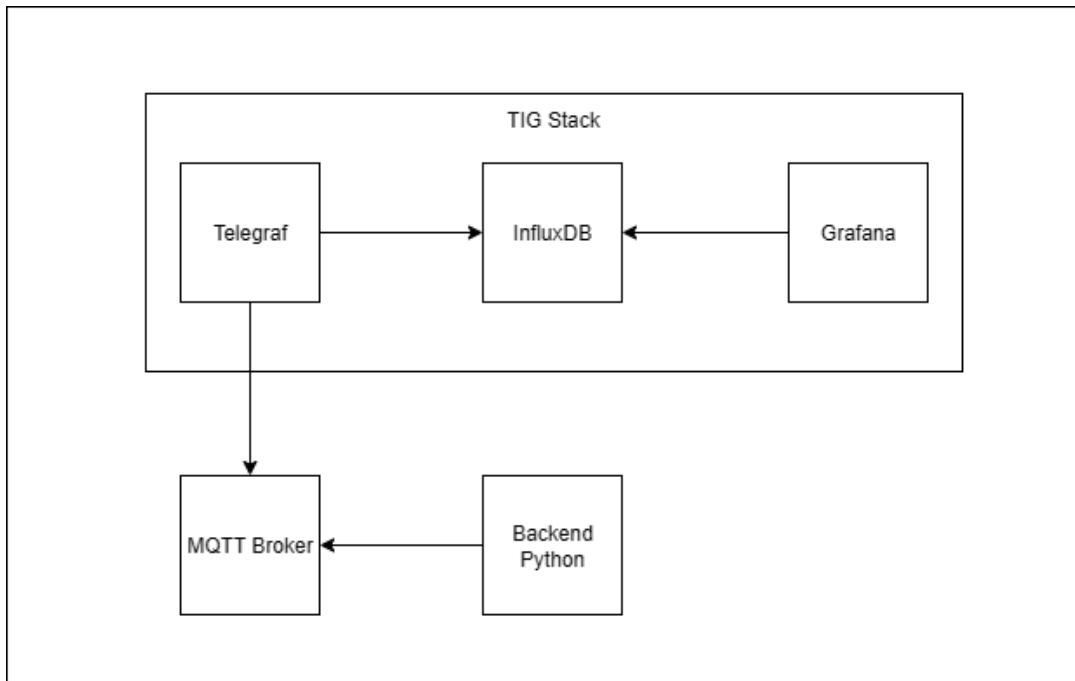


Figura 2.2: Schema di esempio dell’integrazione tra protocollo *MQTT* e *stack TIG*.

2.4.2 Vincoli metodologici

Per quanto concerne la metodologia, il progetto si è sviluppato in quattro fasi principali, che hanno guidato sia la pianificazione sia l'esecuzione:

- **Fase 1 (Setup dell'ambiente):** predisposizione di un contesto sicuro e, al tempo stesso, sufficientemente vulnerabile per ospitare *honeypot* e servizi di interesse;
- **Fase 2 (Sviluppo del *logger* e installazione della pipeline *TIG*):** costruzione di una *pipeline* di raccolta, gestione e visualizzazione dei dati, con particolare attenzione alla scalabilità;
- **Fase 3 (Setup della *data analysis*):** implementazione di strumenti e metodologie per l'analisi e la correlazione dei dati, finalizzati al miglioramento dei processi di rilevazione e risposta agli attacchi;
- **Fase 4 (Attesa o simulazione di attacchi e redazione dei *report*):** osservazione delle interazioni con il sistema e produzione di *report* operativi a supporto dell'analisi.

2.4.3 Vincoli temporali

Infine, relativamente ai vincoli temporali, l'unico vincolo significativo era rappresentato dalla durata del tirocinio, pari a due mesi. L'avanzamento delle diverse fasi è dunque dipeso in larga misura dalla rapidità con cui sono riuscito ad apprendere e applicare le competenze necessarie, rendendo il fattore tempo un elemento variabile e fortemente legato alla curva di apprendimento individuale.

2.5 Prospettive future

Sono presenti diverse possibilità di evoluzione, finalizzate a incrementarne le funzionalità e l'efficacia del progetto. Una possibile estensione riguarda l'aggiunta di ulteriori servizi vulnerabili nel sistema *honeypot*, con l'obiettivo di renderlo più realistico e in grado di raccogliere un numero maggiore di dati sulle tecniche

di attacco. Si potrebbe inoltre rendere il sistema esposto alla rete, rimuovendo alcune limitazioni imposte dai *firewall*, in modo da osservare interazioni più eterogenee con l’ambiente esterno e analizzare comportamenti degli attaccanti più complessi. Un’ulteriore evoluzione riguarda l’introduzione di meccanismi automatici per l’identificazione delle anomalie, sfruttando strumenti di *AI* e *ML*. Questo permetterebbe di automatizzare il rilevamento delle minacce e di ridurre i tempi di risposta agli attacchi. È inoltre possibile estendere il sistema di *logging* per supportare più lingue, migliorando l’usabilità e la fruibilità dei dati da parte di *team* internazionali o utenti non madrelingua. Infine, il progetto potrebbe essere reso un pacchetto unico distribuibile ai clienti, semplificando l’installazione e la configurazione del sistema e favorendone l’adozione in contesti differenti. Queste evoluzioni rappresentano opportunità concrete per aumentare il valore del sistema e garantirne una maggiore flessibilità, automatizzazione e facilità d’uso.

2.6 Motivazioni personali

La scelta di questo progetto di tirocinio è stata fortemente influenzata dal desiderio di avvicinarmi al mondo della *cybersecurity*, un ambito che negli ultimi anni ha assunto un ruolo sempre più centrale sia in ambito accademico che professionale. Durante il percorso universitario non ho avuto molte occasioni di affrontare questo tema in maniera pratica, e lo sviluppo di un sistema di *honeypot* ha rappresentato per me un’occasione per approfondire un settore che considero strategico per il futuro delle tecnologie digitali. La *cybersecurity* non riguarda soltanto la protezione dei sistemi informatici, ma ha un impatto diretto sulla vita delle persone, delle imprese e delle istituzioni. Spesso, quando si parla di informatica, ci si concentra su aspetti come lo sviluppo di nuove applicazioni o l’ottimizzazione delle prestazioni, trascurando la componente della sicurezza. Tuttavia, gli attacchi informatici sono oggi una delle principali minacce globali, e questo rende evidente l’importanza di acquisire competenze specifiche in questo settore. Un altro elemento che ha influito sulla mia scelta è il carattere applicativo del progetto, che non si limitava a un esercizio teorico ma richiede-

CAPITOLO 2. SCOPO DEL PROGETTO

va lo sviluppo di un sistema reale, con vincoli tecnologici e obiettivi concreti. Un’ulteriore motivazione è stata la possibilità di sperimentare tecnologie che non avevo avuto modo di approfondire nel corso degli studi, come lo *stack TIG* e il protocollo *MQTT*. Confrontarmi con strumenti utilizzati in contesti professionali rappresentava una sfida e al tempo stesso un’opportunità di crescita che avrebbe arricchito il mio percorso formativo. Lo svolgimento del tirocinio in modalità da remoto costituiva un’occasione importante per sviluppare autonomia e capacità di organizzazione. Gestire in modo indipendente il progetto e le attività quotidiane mi ha permesso di migliorare le mie competenze di pianificazione e di responsabilità, qualità che considero fondamentali per affrontare con successo future esperienze lavorative e accademiche. Le motivazioni che mi hanno portato alla scelta di questo progetto di tirocinio si sono tradotte in una serie di obiettivi personali.

In particolare, gli obiettivi prefissati erano i seguenti:

ID	Obiettivo personale
OP1	Progettare e realizzare un sistema di <i>honeypot</i> completo, partendo dall’analisi dei requisiti fino al collaudo finale, al fine di acquisire esperienza nello sviluppo di un progetto dall’inizio alla fine.
OP2	Sviluppare un prodotto <i>software</i> adottando i <i>design pattern</i> e i principi architetturali appresi nel corso di Ingegneria del <i>Software</i> , con particolare attenzione a modularità, scalabilità e manutenibilità del codice.
OP3	Sviluppare competenze pratiche in ambito <i>cybersecurity</i> , configurando e monitorando differenti tipologie di servizi vulnerabili da utilizzare all’interno del sistema di <i>honeypot</i> .
Continua nella prossima pagina...	

Tabella 2.2 – Continuo della tabella

ID	Obiettivo personale
OP4	Acquisire competenze pratiche nell'utilizzo di tecnologie professionali come lo <i>stack TIG</i> e il protocollo <i>MQTT</i> , comprendendone i principi di funzionamento e sperimentandone l'integrazione in una <i>pipeline</i> di raccolta e visualizzazione dati.
OP5	Approfondire le competenze di analisi dei dati raccolti, individuando pattern ricorrenti negli attacchi e distinguendo tra traffico lecito e malevolo mediante metriche descrittive e dashboard interattive.

Tabella 2.2: Obiettivi personali del tirocinio.

Questi obiettivi riflettono le motivazioni che mi hanno spinto a scegliere questo progetto di tirocinio e rappresentano le competenze che intendevo sviluppare durante questa esperienza. Raggiungerli avrebbe significato non solo completare con successo il tirocinio, ma anche arricchire il mio bagaglio di conoscenze e abilità, preparandomi al meglio per le sfide future nel campo dell'informatica e della *cybersecurity*.

Capitolo 3

Sviluppo del progetto

Il terzo capitolo illustra gli strumenti, le tecnologie e le metodologie adottate, descrivendo come siano state applicate e presentando un quadro generale dello svolgimento del progetto: dall’analisi delle richieste aziendali, fino allo sviluppo del prodotto finale.

3.1 Analisi dei requisiti

La fase di **analisi dei requisiti** consiste nello studio e nell’esame delle esigenze espresse dall’azienda, con l’obiettivo di definire in modo preciso e completo i requisiti del sistema. Questa parte del progetto si è svolta nell’arco di due settimane, durante le quali ho organizzato incontri regolari con il tutor aziendale per confrontarsi su progressi, criticità e possibili soluzioni. Le principali attività condotte includono:

- **Sessioni di *brainstorming*** con il tutor, finalizzate alla definizione degli obiettivi e delle priorità del sistema;
- **Studio della documentazione** di sistemi *honeypot* esistenti, per comprendere metodologie consolidate e funzionalità tipiche;
- **Analisi delle *best practice*** in ambito *cybersecurity* per sistemi *ETL*, a garanzia di sicurezza e affidabilità;

- **Valutazione dei vincoli tecnologici** imposti dall’ambiente aziendale, tra cui piattaforme disponibili, linguaggi di programmazione e strumenti di monitoraggio.

Dall’analisi effettuata sono emersi i [casi d’uso](#) sulla quale si è basato il progetto.

3.1.1 Casi d’uso

I **casi d’uso** descrivono una funzionalità specifica che il sistema deve fornire a uno o più attori (che possono essere utenti o altri sistemi), indicando le interazioni e i risultati ottenuti dal punto di vista degli attori. Per sviluppare i casi d’uso abbiamo utilizzato il linguaggio [Unified Modeling Language \(UML\)](#)_G, uno strumento di modellazione che mostra le interazioni tra attori e casi d’uso attraverso relazioni come «*include*», «*extend*» e generalizzazioni. La seguente convenzione di nomenclatura supporta l’identificazione dei casi d’uso e ne facilita l’organizzazione e la tracciabilità:

UC.ID.subID

- **UC**: è l’identificatore dei casi d’uso;
- **ID**: è l’identificatore numerico del caso d’uso principale;
- **subID**: è l’identificatore numerico dei sotto-casi d’uso che specifica i casi d’uso associati a un caso d’uso principale.

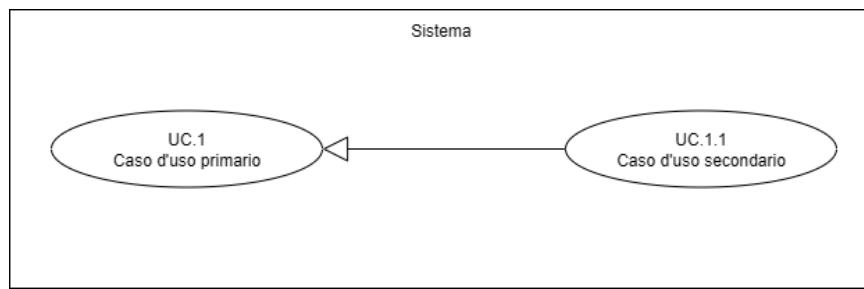


Figura 3.1: Esempio di caso d’uso e sotto-caso d’uso.

Attori dei casi d'uso

Nel contesto dei casi d'uso, gli attori del sistema si suddividono in due categorie principali: **attori primari** e **attori secondari**.

Gli **attori primari** sono coloro che avviano attivamente un caso d'uso con l'obiettivo di ottenere un risultato specifico attraverso l'interazione diretta con il sistema. Rappresentano gli utenti o le entità che traggono un beneficio concreto dall'esecuzione del caso d'uso.

All'interno del progetto l'attore principale è:

- **Utente**: cioè un utente membro del team di *cybersecurity* che ha accesso all'*honeypot*.

Al contrario, gli **attori secondari** non iniziano il caso d'uso, ma forniscono supporto o servizi necessari affinché il sistema possa completarlo correttamente. Interagiscono in modo indiretto o passivo, contribuendo al funzionamento del processo senza esserne i promotori principali. In questo caso all'interno del sistema non sono presenti attori secondari, poiché non ci sono entità che svolgono un ruolo di supporto indiretto al processo.



Figura 3.2: Esempio di attore.

Lista dei casi d'uso

Di seguito si trovano i casi d'uso principali del sistema. Non sono riportati tutti i casi d'uso individuati in fase di analisi, ma soltanto quelli ritenuti maggiormente rappresentativi e funzionali alla comprensione delle caratteristiche essenziali del sistema. La scelta di limitarne il numero risponde a due esigenze:

da un lato evitare ridondanze o descrizioni eccessivamente dettagliate di funzionalità marginali, dall'altro garantire una visione chiara e sintetica delle capacità chiave del sistema in relazione ai suoi obiettivi progettuali.

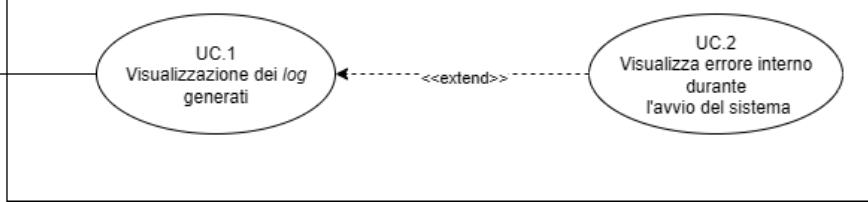
UC.1	
 Utente	Sistema 
Pre-condizioni	Il sistema è attivo e funzionante.
Post-condizioni	Vengono visualizzati i <i>log</i> generati dal sistema <i>honeypot</i> .
Attori principali	Utente
Scenario principale	<ul style="list-style-type: none"> L'utente richiede la visualizzazione dei <i>log</i> del sistema; Il sistema verifica la presenza e la conformità dei <i>log</i>; I dati del sistema vengono visualizzati in ordine cronologico di creazione.
Scenario secondario	<ul style="list-style-type: none"> Si verifica un errore interno durante l'avvio del sistema che impedisce la visualizzazione dei <i>log</i> (UC.2).

Tabella 3.1: Diagramma del caso d'uso UC.1 - Visualizzare i *log* del sistema *honeypot*.

UC.2	
Pre-condizioni	L'avvio delle diverse componenti del sistema non avviene correttamente.
Post-condizioni	L'utente visualizza gli errori che impediscono l'avvio del sistema e il loro livello di gravità.
Attori principali	Utente
Scenario principale	<ul style="list-style-type: none"> • L'utente avvia il sistema; • Il sistema verifica lo stato delle sue componenti; • Se si verifica un errore, il sistema comunica all'utente quale componente non si è avviata correttamente.
Scenario secondario	<ul style="list-style-type: none"> • N/A

Tabella 3.2: Diagramma del caso d'uso UC.2 - Visualizza errore interno durante l'avvio del sistema.

UC.3	
Utente	
Pre-condizioni	Il sistema è in fase di avvio.
Post-condizioni	Il livello minimo dei <i>log</i> da visualizzare è impostato.
Attori principali	Utente
Continua nella prossima pagina...	

Tabella 3.3 – Continuo della tabella

UC.3	
Scenario principale	<ul style="list-style-type: none"> • L'utente visualizza i possibili livelli minimi del sistema; • L'utente seleziona il livello minimo desiderato; • Il sistema applica la nuova configurazione e conferma l'avvenuta modifica.
Scenario secondario	<ul style="list-style-type: none"> • L'utente seleziona un livello non valido, oppure non seleziona nessun livello.

Tabella 3.3: Diagramma del caso d'uso UC.3 - Impostare il livello minimo dei *log* da visualizzare.

UC.4	
Utente	<pre> graph LR Actor --> UC4[UC.4 Visualizza dati in InfluxDB] UC4 --<<include>>--> UC4_1[UC.4.1 Visualizza dati di un determinato topic] </pre>
Pre-condizioni	Il sistema è attivo e funzionante.
Post-condizioni	Vengono visualizzati i dati presenti in <i>InfluxDB</i> .
Attori principali	Utente
Continua nella prossima pagina...	

Tabella 3.4 – Continuo della tabella

UC.4	
Scenario principale	<ul style="list-style-type: none"> L'utente richiede la visualizzazione dei dati presenti in <i>InfluxDB</i>; Il sistema verifica la presenza e la conformità dei dati; I dati presenti in <i>InfluxDB</i> vengono visualizzati in base ai filtri selezionati.
Scenario secondario	<ul style="list-style-type: none"> Non sono presenti dati in <i>InfluxDB</i> da visualizzare; Il sistema non mostra alcun dato all'utente.

Tabella 3.4: Diagramma del caso d'uso UC.4 - Visualizzare i dati presenti in *InfluxDB*.

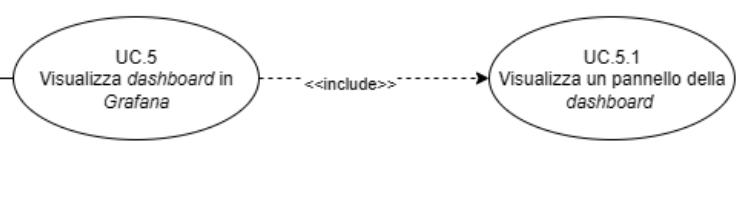
UC.5	
Utente	
Pre-condizioni	Il sistema è attivo e funzionante.
Post-condizioni	Viene visualizzata una <i>dashboard</i> in <i>Grafana</i> .
Attori principali	Utente
Continua nella prossima pagina...	

Tabella 3.5 – Continuo della tabella

UC.5	
Scenario principale	<ul style="list-style-type: none"> L'utente richiede la visualizzazione di una <i>dashboard</i> in <i>Grafana</i>; Il sistema verifica la presenza e la conformità della <i>dashboard</i>; La <i>dashboard</i> viene visualizzata con i dati aggiornati in tempo reale.
Scenario secondario	<ul style="list-style-type: none"> Non sono presenti dati in <i>InfluxDB</i> da visualizzare; Il sistema non mostra alcun dato all'utente.

Tabella 3.5: Diagramma del caso d'uso UC.5 - Visualizzare una *dashboard* in *Grafana*.

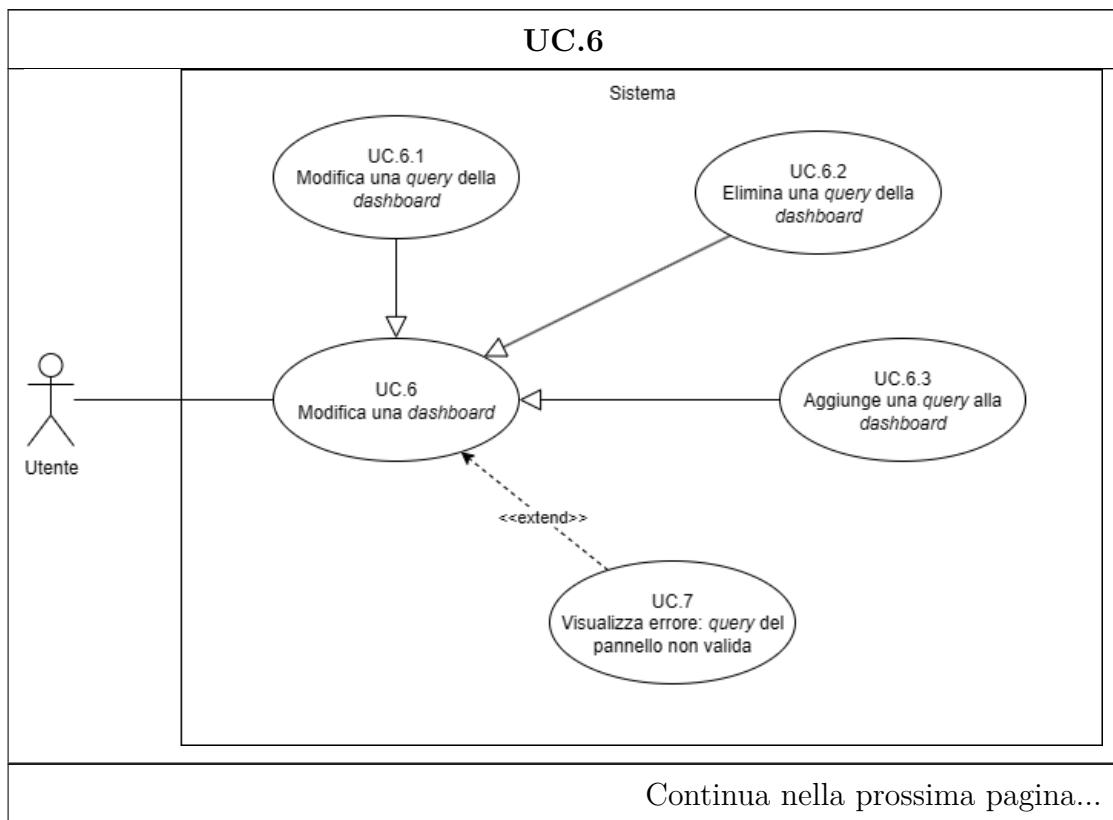


Tabella 3.6 – Continuo della tabella

UC.6	
Pre-condizioni	Il sistema è attivo e funzionante.
Post-condizioni	Viene modificata una <i>dashboard</i> in <i>Grafana</i> .
Attori principali	Utente
Scenario principale	<ul style="list-style-type: none"> • L'utente modifica una <i>dashboard</i> in <i>Grafana</i>; • Il sistema verifica la conformità della <i>query</i>; • La <i>dashboard</i> viene aggiornata con le nuove impostazioni.
Scenario secondario	<ul style="list-style-type: none"> • La modifica non è conforme agli standard di <i>Grafana</i> (UC.7).

Tabella 3.6: Diagramma del caso d'uso UC.6 - Modificare una *dashboard* in *Grafana*.

UC.7	
Pre-condizioni	La modifica effettuata dall'utente non è valida.
Post-condizioni	Viene visualizzato il pannello di errore.
Attori principali	Utente
Scenario principale	<ul style="list-style-type: none"> • L'utente modifica una <i>dashboard</i> in <i>Grafana</i>; • Il sistema verifica la conformità della modifica effettuata; • La modifica non è conforme agli standard di <i>Grafana</i>; • Il sistema visualizza il pannello di errore.
Scenario secondario	<ul style="list-style-type: none"> • N/A
Continua nella prossima pagina...	

Tabella 3.7 – Continuo della tabella

UC.7

Tabella 3.7: Diagramma del caso d'uso UC.7 - Visualizza errore: *query* del pannello non valida.

3.1.2 Requisiti

I requisiti si dividono in tre categorie principali: funzionali, non funzionali e di vincolo, che specificano rispettivamente cosa deve fare il sistema, come deve funzionare e quali condizioni deve rispettare per operare. I primi sono le azioni che il sistema deve compiere, mentre i secondi sono le qualità che deve avere (come la velocità o la sicurezza). I requisiti di vincolo, invece, riguardano le restrizioni imposte al sistema, come la piattaforma su cui deve girare o il linguaggio di programmazione da utilizzare. In seguito alla definizione dei casi d'uso, ho quindi identificato e documentato i requisiti funzionali, non funzionali e di vincolo del sistema. La determinazione di tali requisiti è avvenuta assieme al tutor aziendale e sono alla base della progettazione e dello sviluppo.

Requisiti funzionali

- RF.1: Raccogliere automaticamente i *log* generati dai servizi *honeypot*.
- RF.2: Centralizzare i *log* raccolti in un *database* (*InfluxDB*).
- RF.3: Visualizzare i *log* in tempo reale attraverso un'interfaccia o strumento di monitoraggio.
- RF.4: Consentire la configurazione del livello minimo di *log* da mostrare.
- RF.5: Reimpostare il livello minimo di *log* al valore predefinito e notificare l'utente.
- RF.6: Segnalare errori interni durante l'avvio del sistema.
- RF.7: Visualizzare i dati salvati in *InfluxDB* tramite *Grafana*.
- RF.8: Creare e modificare *dashboard* in *Grafana*.
- RF.9: Aggiungere, modificare ed eliminare *query* nelle *dashboard*.
- RF.10: Mostrare errori in caso di *query* non valida nei pannelli *Grafana*.
- RF.11: Simulare attacchi controllati e raccoglierne i *log*.

- RF.12: Fornire *dashboard* avanzate con grafici, *IOC* e *timeline* degli eventi.
- RF.13: Esportare i dati verso *SIEM open source*.

Requisiti non funzionali

- RNF.1: Modularità del codice per favorire estendibilità e manutenzione.
- RNF.2: Documentazione tecnica chiara, verificabile e coerente con il codice.
- RNF.3: Commenti esplicativi nel codice per facilitarne la comprensione.
- RNF.4: Riutilizzabilità degli *script* e parametrizzazione degli *input*.
- RNF.5: Configurazioni del codice leggibili, versionate e testate.
- RNF.6: Sicurezza nell'accesso ai *log* e alle *dashboard* tramite autenticazione.
- RNF.7: Persistenza dei dati raccolti per consentire analisi storiche.
- RNF.8: Scalabilità per aggiungere nuovi servizi *honeypot* o *pipeline* senza modifiche invasive.
- RNF.9: Affidabilità nella raccolta dei *log* (nessuna perdita di dati in caso di errore temporaneo).
- RNF.10: Chiarezza dei messaggi di errore, comprensibili anche a utenti non tecnici.

Requisiti di vincolo

- RV.1: Esecuzione in una macchina virtuale isolata per motivi di sicurezza.
- RV.2: Utilizzo del linguaggio di programmazione *Python*.
- RV.3: Utilizzo della *pipeline TIG*.
- RV.4: Presenza di servizi vulnerabili realistici da esporre nel sistema *honeypot*.
- RV.5: Uso di servizi *dummy* con *netcat* o strumenti equivalenti per simulare ascolto pacchetti.
- RV.6: Raccolta *log* automatizzata mediante *script Python* o *Bash*.
- RV.7: Gestione di configurazioni e codice tramite sistemi di versionamento (es. *Git*).

Infine, la seguente tabella riassume durata, *output* e metriche delle attività di analisi dei requisiti:

Attività	Durata	Prodotti	Metriche
Sessioni di <i>brainstorming</i> col tutor	2 settimane	11 casi d'uso definiti	8 incontri da 1 ora ciascuno
Studio documentazione honeypot esistenti	1 settimana	Relazione interna	15 sistemi analizzati
Analisi <i>best practice ETL cybersecurity</i>	3 giorni	Relazione interna	12 framework valutati
Definizione requisiti funzionali	2 giorni	13 requisiti RF	2 iterazioni di revisione
Definizione requisiti non funzionali	1 giorno	10 requisiti RNF	2 iterazioni di revisione
Definizione requisiti di vincolo	1 giorno	7 requisiti RV	2 iterazioni di revisione

Tabella 3.8: Tabella quantitativa dell'analisi dei requisiti.

3.2 Progettazione

3.2.1 Architettura del sistema

Per questo progetto, essendo fondamentalmente un sistema *Extract, Transform, Load (ETL)*, ho deciso di adottare un'architettura a più livelli, che consente di separare le diverse funzionalità del sistema in moduli distinti e indipendenti. Il primo livello è rappresentato dai servizi *honeypot*, che simulano i servizi di rete vulnerabili e raccolgono i *log* delle attività sospette. Il secondo livello è costituito dai *script* di raccolta e trasferimento dei *log*, che elaborano i dati raccolti e li inviano al database. Il terzo livello è rappresentato dal database *InfluxDB*, che memorizza i *log* in modo strutturato e consente di eseguire

query sui dati. Infine, i dati vengono visualizzati tramite *Grafana*, che offre un’interfaccia intuitiva per l’analisi e la visualizzazione delle informazioni.

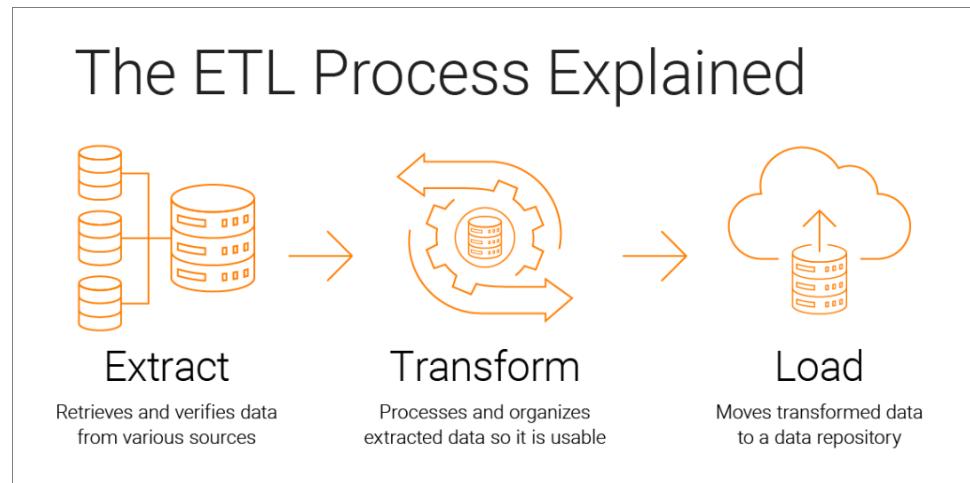


Figura 3.3: Architettura del sistema *ETL*.

Fonte:

<https://www.informatica.com/resources/articles/what-is-etl.html>

Partendo da questa architettura di base, ho progettato il sistema in maniera tale da essere modulare e scalabile, in modo da poter aggiungere facilmente nuovi servizi vulnerabili al sistema *honeypot* o modificare quelli esistenti senza influire sulle altre componenti. Ho quindi prodotto il seguente diagramma di flusso che rappresenta il funzionamento generale del prodotto, dalla raccolta dei *log* alla loro visualizzazione.

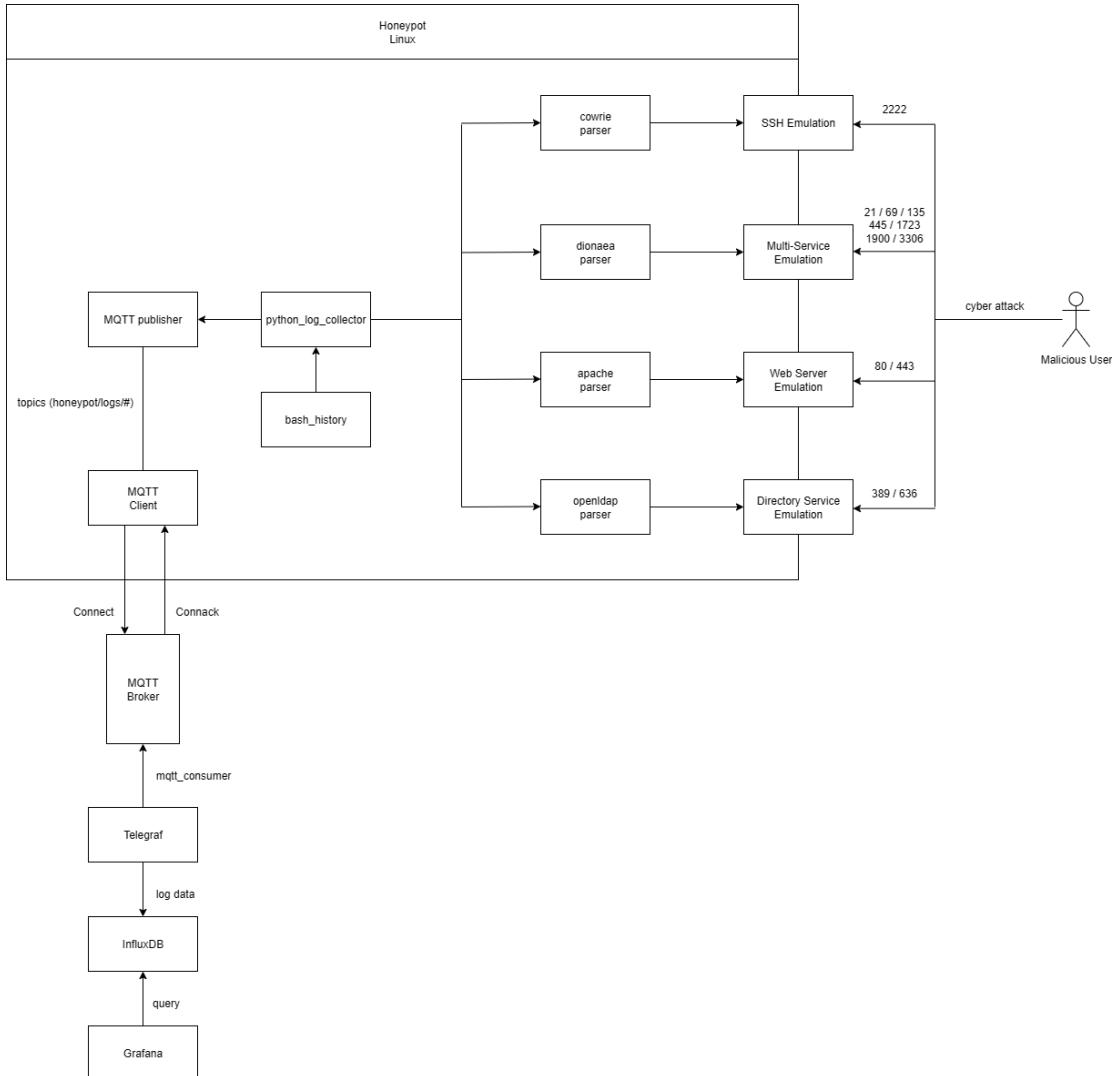


Figura 3.4: Diagramma di flusso del sistema *honeypot*.

Il diagramma mostra come i *log* vengano generati dai servizi vulnerabili in seguito a tentativi di accesso o attacchi. Gli *script* di monitoraggio raccolgono i *log*, li elaborano e li pubblicano su un *topic* tramite un *publisher* del protocollo *MQTT*. *Telegraf*, poi, preleva i dati dal *topic* e li invia al database *InfluxDB* per la memorizzazione. Infine, *Grafana* consente di visualizzare i dati memorizzati nel database tramite *dashboard* personalizzate.

3.2.2 Scelta dei servizi del sistema

Il sistema *honeypot* doveva risultare realisticamente vulnerabile, in modo da attirare l'attenzione di potenziali attaccanti e raccogliere dati utili per l'analisi.

delle minacce. Per questo motivo abbiamo selezionato servizi noti per le loro vulnerabilità e diffusione, in grado di coprire una varietà di protocolli e tecnologie. La scelta è ricaduta su *Cowrie*, *Dionaea*, *Apache* e *OpenLDAP*. Ho scelto e configurato ciascuno di questi servizi per scenari d’attacco specifici e per fornire dati di valore durante il processo di monitoraggio.

Cowrie è un servizio progettato per emulare un sistema *Linux* accessibile tramite protocollo *Secure Shell (SSH)*_G. La sua funzione principale è quella di raccogliere informazioni su tentativi di accesso non autorizzato e sulle attività post-compromissione degli attaccanti. Per questo motivo l’ho configurato per ascoltare sulla porta 2222, simulando un server *SSH* con un ambiente *Linux* realistico. Cowrie offre un *file system* virtuale, l’emulazione di numerosi comandi *bash* e il *logging* completo delle sessioni, registrando anche eventuali *upload* o *download* di file tramite *Secure Copy Protocol (SCP)*_G o *wget*. La sua capacità di simulare vulnerabilità comuni lo rende particolarmente adatto a intercettare interazioni malevoli e a raccogliere indicatori di compromissione.

Dionaea, invece, l’abbiamo scelto per la sua capacità di emulare più servizi vulnerabili contemporaneamente e di catturare *payload* malevoli. Questo lo rende particolarmente utile nell’osservare attacchi automatizzati mirati a protocolli obsoleti o insicuri. Il *container* l’ho configurato per ascoltare su diverse porte critiche: la 21 (*FTP*), la 69 (*Trivial File Transfer Protocol (TFTP)*_G), la 445 (*SMB*), la 135 (*Remote Procedure Call (RPC)*_G), la 1723 (*Point-to-Point Tunneling Protocol (PPTP)*_G) e la 3306 (*MySQL*). In questo modo *Dionaea* è in grado di registrare tentativi di *brute-force*, trasferimenti di file sospetti, comandi di *Structured Query Language (SQL)*_G malevoli e attacchi legati a vulnerabilità precedentemente sfruttate. La sua versatilità consente di coprire un ampio spettro di minacce legate a servizi di rete differenti.

Apache lo abbiamo incluso per la sua enorme diffusione come server *web* e per la frequente presenza di vulnerabilità legate al mondo delle applicazioni *web*. Configurato per rispondere sulle porte 80 (*Hypertext Transfer Protocol*

(*HTTP*)_G) e 443 (*Hypertext Transfer Protocol Secure (HTTPS)*_G), può essere utilizzato per simulare scenari di attacco tipici del *web*, come *directory traversal*, *upload* di file non autorizzati, *Cross-Site Scripting (XSS)*_G o altre iniezioni lato server. In tal modo è possibile raccogliere dati relativi a una delle superfici di attacco più comuni e sfruttate dagli attori malevoli.

Infine, l'ho integrato **OpenLDAP**, scelto per simulare un servizio di directory largamente utilizzato in ambito aziendale per l'autenticazione e la gestione centralizzata degli utenti. Il *container* l'ho configurato per esporre sia la porta 389 (*Lightweight Directory Access Protocol (LDAP)*_G in chiaro) sia la 636 (*Lightweight Directory Access Protocol over SSL/TLS (LDAPS)*_G cifrato). Questa configurazione consente di osservare tentativi di enumerazione utenti, iniezione *LDAP* e altre tecniche di compromissione legate a *directory service*, che rappresentano spesso un obiettivo strategico per gli attaccanti al fine di ottenere accessi privilegiati. La combinazione di questi servizi consente al sistema di coprire una grande varietà di scenari d'attacco: dall'accesso remoto tramite *SSH*, alle compromissioni multi-protocollo, fino agli attacchi *web* e ai tentativi di abuso di *directory service*. In questo modo l'*honeypot* risulta non solo realistico, ma anche utile per raccogliere dati diversificati e rappresentativi delle principali minacce informatiche attuali.

Diagramma delle classi e design pattern

In seguito ho sviluppato il diagramma delle classi *UML*, che rappresenta le principali classi e le loro relazioni all'interno del sistema.

CAPITOLO 3. SVILUPPO DEL PROGETTO

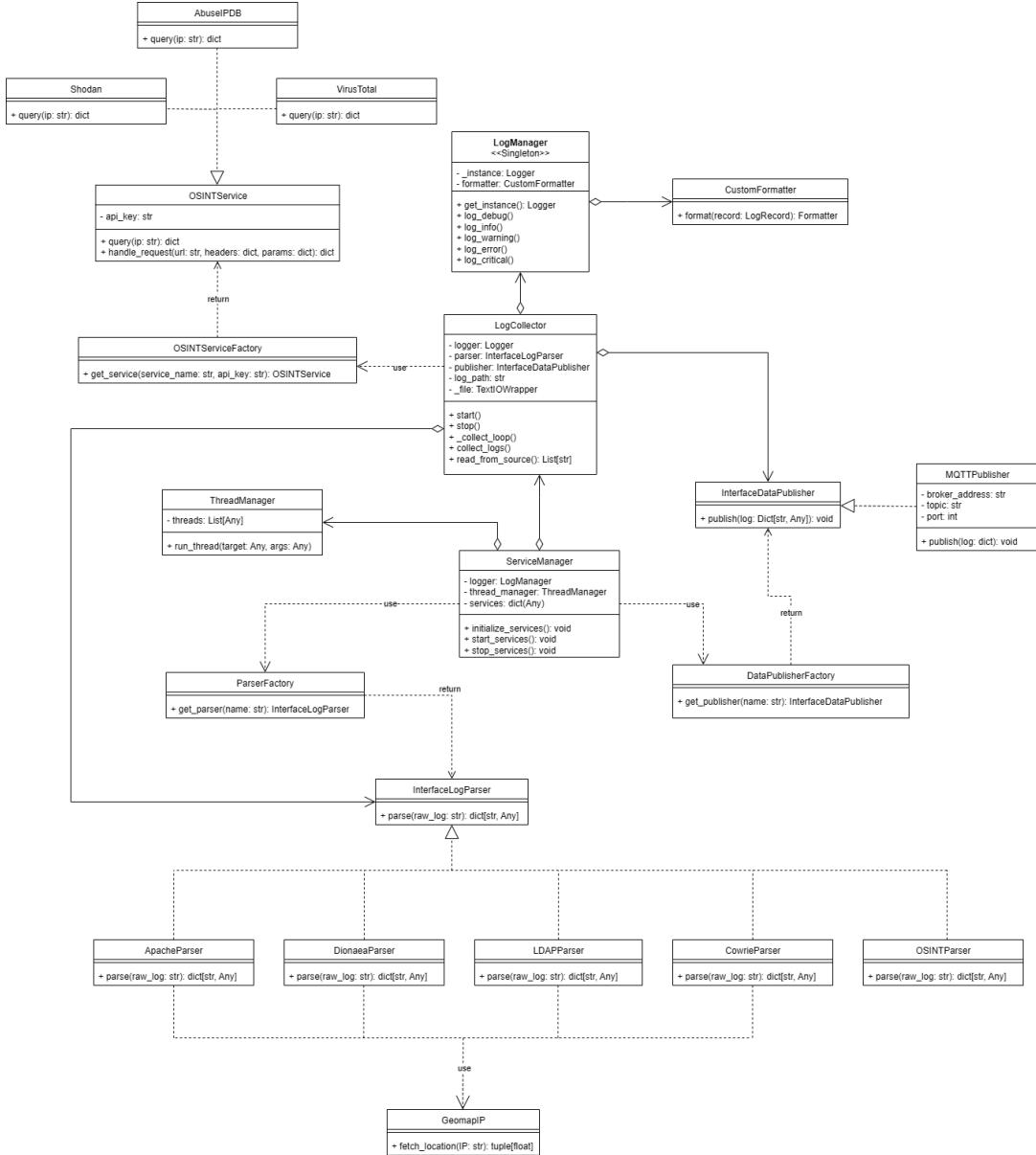


Figura 3.5: Diagramma delle classi del sistema *honeypot*.

Per la progettazione del sistema e la realizzazione del relativo diagramma ho applicato diversi *design pattern* studiati durante il corso di Ingegneria del *Software*. Un primo esempio è rappresentato dal ***Factory pattern***, utilizzato nella classe *ParserFactory*, la quale ha il compito di istanziare i diversi oggetti *parser* a seconda del servizio vulnerabile da analizzare. In questo modo la logica di creazione viene centralizzata e resa indipendente dal resto del sistema, favorendo estendibilità e manutenibilità. Un ulteriore contributo deriva dall'applicazione dell'***Adapter pattern***, che ha permesso di uniformare i *log*

provenienti da sorgenti differenti adattandoli al formato richiesto dal *publisher* MQTT; tale approccio consente di integrare componenti con interfacce distinte senza doverne modificare l'implementazione originaria. La gestione centralizzata dei *log* è invece garantita dal ***Singleton pattern***, applicato alla classe *LogManager*, che deve esistere in un'unica istanza per tutta la durata dell'esecuzione del programma, evitando così conflitti dovuti alla creazione di istanze multiple. Per quanto riguarda le operazioni di *parsing*, ho applicato lo ***Strategy pattern***, grazie al quale è possibile definire e incapsulare diverse strategie, una per ciascun servizio vulnerabile, e selezionare dinamicamente quella più appropriata durante l'esecuzione. L'***Observer pattern*** risulta applicato in maniera implicita attraverso l'uso del protocollo *MQTT*: i moduli di monitoraggio pubblicano i *log* su specifici *topic*, ai quali *Telegraf* si iscrive per ricevere in tempo reale gli aggiornamenti. Questo meccanismo riflette il principio dell'*Observer*, in cui i cambiamenti nello stato di un soggetto sono propagati automaticamente a tutti gli osservatori registrati.

Infine, la seguente tabella riassume le versioni realizzate, gli elementi modellati e il tempo dedicato alle attività di modellazione e configurazione del sistema:

Artefatto	Versioni create	Elementi modellati	Ore dedicate
Diagramma di flusso sistema	3	8 componenti principali	12
Diagramma delle classi UML	4	21 classi, 5 design pattern	28
Configurazione servizi honeypot	4 per servizio	4 servizi (Cowrie, Dionaea, Apache, LDAP)	40

Tabella 3.9: Tabella quantitativa delle attività di modellazione e configurazione.

3.3 Codifica

3.3.1 Struttura del progetto

La struttura del progetto è organizzata in diverse cartelle principali, ciascuna con una responsabilità specifica. La cartella `config` contiene i *file* per la configurazione dell'applicazione, includendo `environment_config.py` per la gestione delle variabili e dei parametri di ambiente e `__init__.py` per l'inizializzazione della cartella stessa.

All'interno della cartella `core` si trovano i moduli fondamentali per l'esecuzione del sistema. `main.py` rappresenta l'*entry point* dell'applicazione, mentre `geomap_ip.py` si occupa della geolocalizzazione degli indirizzi IP. La raccolta dei *log* viene gestita tramite `log_collector.py`, la coordinazione dei servizi è affidata a `service_manager.py` e l'esecuzione concorrente tramite *thread* è curata da `thread_manager.py`.

La cartella `logger` si dedica alla gestione centralizzata dei *log*. `log_manager.py` configura e gestisce il *logger* principale, mentre `custom_formatter.py` definisce formati personalizzati, ad esempio con codifica colore in base al livello di gravità.

Per quanto riguarda l'intelligence open source, la cartella `osint` include moduli specifici. `base_osint.py` definisce l'interfaccia comune per le integrazioni *OSINT*, `osint_factory.py` permette di istanziare dinamicamente i moduli come `abuseipdb.py`, `shodan.py` e `virustotal.py`, ciascuno dedicato a una diversa fonte di dati.

La cartella `parsers` raccoglie i parser per l'analisi e la normalizzazione dei *log*. La classe base `base_parser.py` definisce l'interfaccia comune, mentre `parser_factory.py` consente di creare i *parser* specifici per `cowrie_parser.py`, `dionaea_parser.py`, `apache_parser.py`, `osint_parser.py` e `LDAP_parser.py`. Infine, la cartella `publishers` contiene i moduli responsabili della pubblicazione dei *log* verso sistemi esterni. `base_publisher.py` definisce l'interfaccia astratta, `mqtt_publisher.py` implementa un *publisher* basato su *MQTT* e `publisher_factory.py` permette di creare dinamicamente i *publisher* in ba-

se al contesto operativo.

Per rappresentare visivamente questa struttura, è possibile utilizzare il seguente albero:

```
src/
  config/
    __init__.py
    environment_config.py
  core/
    __init__.py
    main.py
    geomap_ip.py
    log_collector.py
    service_manager.py
    thread_manager.py
  logger/
    __init__.py
    custom_formatter.py
    log_manager.py
  osint/
    __init__.py
    base_osint.py
    abuseipdb.py
    shodan.py
    virustotal.py
    osint_factory.py
  parsers/
    __init__.py
    base_parser.py
    parser_factory.py
    cowrie_parser.py
    dionaea_parser.py
    apache_parser.py
    osint_parser.py
    LDAP_parser.py
  publishers/
    __init__.py
    base_publisher.py
    mqtt_publisher.py
    publisher_factory.py
```

In conclusione, le tabelle seguenti sintetizzano la dimensione e la complessità delle componenti sviluppate, riportando linee di codice, file creati e funzioni/metodi implementati. Esse illustrano inoltre le configurazioni e gli script auxiliari realizzati per l'implementazione e l'automazione dei servizi, specificando

quantità e descrizione di ciascun tipo di file.

Componente	Linee di codice	File creati	Funzioni/Metodi
<i>Core system</i>	305	5	15
<i>Logger module</i>	79	2	4
<i>Parsers</i>	404	7	7
<i>Publishers</i>	80	3	18
<i>OSINT integrations</i>	108	5	7
<i>Configuration</i>	32	1	1
TOTALE	1.008	23	52

Tabella 3.10: Tabella quantitativa delle metriche di implementazione del codice.

Tipo	Quantità	Numero di versioni	Descrizione
<i>Docker Compose files</i>	2	5	Orchestrazione dei servizi
<i>Dockerfile</i>	3	4	<i>Container</i> personalizzati
<i>Script Bash</i>	1	3	Automazione del deployment
<i>File di configurazione Grafana</i>	2	10	<i>Dashboard</i> e pannelli
<i>File di configurazione Telegraf</i>	1	5	<i>Pipeline</i> per la raccolta dati

Tabella 3.11: Tabella quantitativa delle configurazioni e degli script ausiliari.

3.3.2 Difficoltà incontrate

Durante lo sviluppo del progetto, uno dei principali ostacoli riscontrati è rappresentato dalla complessità nel rielaborare i *log* generati dai diversi servizi.

Questi dati, provenienti dall’*honeypot* e da sistemi ausiliari, presentavano formati variabili, richiedendo l’implementazione di espressioni regolari altamente specifiche e flessibili. La difficoltà consisteva non solo nel catturare correttamente le informazioni rilevanti, ma anche nel garantire la robustezza dei *parser* di fronte a formati imprevisti o a *log* contenenti dati inconsistenti, senza interrompere il flusso complessivo di raccolta e analisi. Ho superato questo problema mediante l’utilizzo del *logger* interno del sistema per tracciare gli errori di parsing tramite messaggi di *debug*, permettendo di identificare e correggere rapidamente le espressioni regolari difettose. Un secondo aspetto critico riguardava la gestione della *containerizzazione* tramite *Docker* delle diverse componenti del sistema. La necessità di isolare i moduli, garantire l’interoperabilità tra servizi e preservare la sicurezza dell’ambiente ha reso necessario progettare con attenzione la rete dei *container*, i volumi per la persistenza dei dati e la configurazione delle dipendenze. Problemi di compatibilità tra immagini, errori nella definizione dei *Dockerfile* e difficoltà nella sincronizzazione dei *container* tra loro hanno inizialmente rallentato le fasi di *test* e *deployment*. Sono riuscito a superare queste difficoltà grazie a una pianificazione accurata della struttura dei *container*, all’uso di immagini standardizzate e alla creazione di *script* di avvio automatizzati, consentendo di ottenere un ambiente stabile, sicuro e facilmente riproducibile.

3.4 Verifica

Per verifica si intende l’insieme delle attività volte a garantire che il prodotto *software* sviluppato soddisfi i requisiti specificati e funzioni correttamente in tutte le condizioni previste. Non si tratta quindi soltanto di individuare errori, ma di adottare un approccio disciplinato che assicuri qualità, affidabilità e coerenza con le specifiche.

In questo progetto, le attività di verifica seguono metodologie consolidate dell’ingegneria del *software*, in particolare lo **schema a V** e lo sviluppo guidato dai *test* *Test-Driven Development (TDD)G*. L’adozione di tali metodologie ha

permesso di strutturare in maniera rigorosa l'intero processo, dalla fase di analisi fino al collaudo, garantendo tracciabilità e coerenza tra fasi di sviluppo e attività di verifica.

Schema a V

Abbiamo adottato lo schema a V (Figura 3.6) come riferimento metodologico per pianificare e coordinare le attività di sviluppo e verifica.

In particolare, i requisiti funzionali sono messi in relazione con i *test* di accettazione, volti a garantire che le funzionalità implementate rispondano correttamente alle esigenze espresse.

I requisiti non funzionali sono rappresentati dai *test* di sistema, che hanno permesso di validare aspetti quali prestazioni, sicurezza, usabilità e scalabilità.

I requisiti di vincolo sono associati ai *test* di conformità, necessari ad assicurare l'aderenza a *standard*, normative e specifiche tecniche.

Per quanto riguarda le fasi progettuali, la progettazione architetturale ha ricevuto la verifica tramite *test* di integrazione, mentre la progettazione di dettaglio ha trovato corrispondenza nei *test* unitari, volti al controllo puntuale di ogni singola funzione o metodo.

Questa impostazione ha garantito una verifica progressiva e multilivello, con una copertura completa dalla macro-struttura fino al singolo metodo.

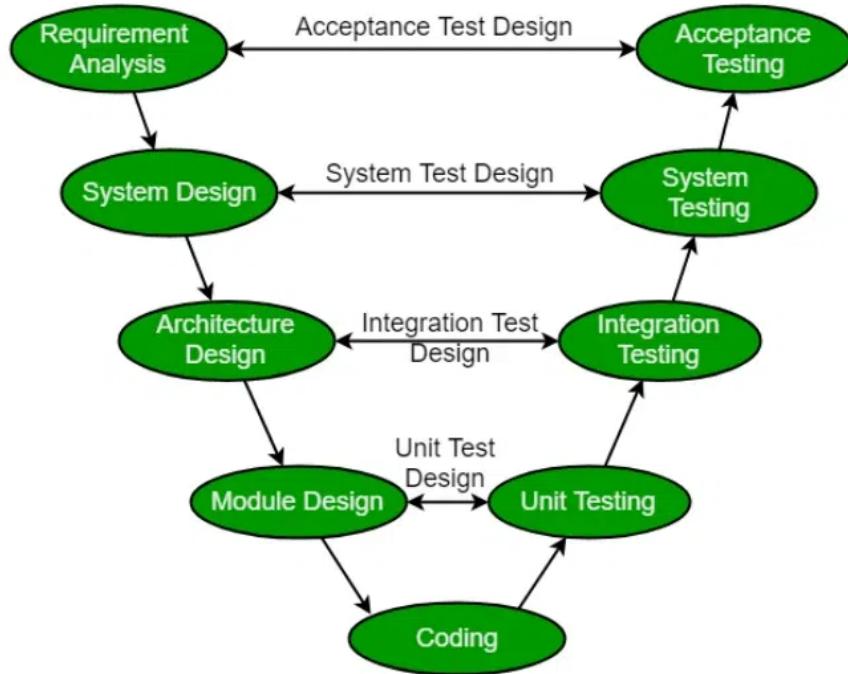


Figura 3.6: Rappresentazione dello schema a V adottato nel progetto.

Fonte: <https://www.geeksforgeeks.org/software-engineering-software-engineering-sdlc-v-model/>

Test-Driven Development (TDD)

Parallelamente, ho adottato la metodologia *TDD*, che ha guidato l'implementazione del codice attraverso cicli iterativi caratterizzati dalle tre fasi classiche:

1. ***Red***: scrittura del *test* relativo a una nuova funzionalità, inizialmente fallito;
2. ***Green***: implementazione della funzionalità per soddisfare il *test*;
3. ***Refactor***: miglioramento del codice, mantenendo il superamento dei *test*.

Questo approccio ha favorito lo sviluppo di codice modulare, riusabile e più semplice da mantenere, riducendo la probabilità di introdurre errori nelle fasi successive.

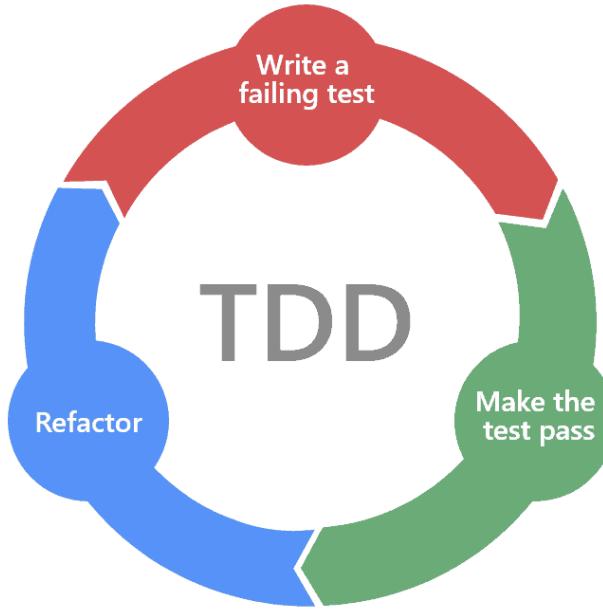


Figura 3.7: Schema del ciclo TDD utilizzato durante lo sviluppo.

Fonte: <https://marsner.com/blog/why-test-driven-development-tdd/>

3.4.1 Test e strumenti

I *test* li ho scritti utilizzando la libreria *pytest*, che consente di scrivere casi di *test* in modo semplice e leggibile. Per aumentare l'efficacia della verifica, ho utilizzato librerie complementari quali:

- *pytest-mock*, per la simulazione di oggetti e funzioni;
- *pytest-cov*, per il calcolo della copertura del codice.

Sono presenti *test* unitari per ogni funzione e metodo del progetto, con particolare attenzione a:

- casi normali, per validare il comportamento atteso;
- casi limite, per testare condizioni particolari e scenari estremi;
- casi di errore, per valutare la robustezza e la gestione delle eccezioni.

3.4.2 Risultati quantitativi

Il processo di *testing* ha prodotto risultati significativi in termini di completezza e profondità delle verifiche svolte:

- l'implementazione di 94 test unitari e di integrazione;
- la copertura del codice ha raggiunto il 98% (Figura 3.8);
- l'identificazione e correzione di 5 *bug* durante le iterazioni di *TDD*, con una riduzione progressiva degli errori riscontrati nelle fasi avanzate.

Tali risultati dimostrano l'efficacia della strategia di verifica adottata e l'intensità del lavoro svolto per garantire la qualità del prodotto finale.

Tipo di test	Quantità	Copertura	Durata esecuzione
Test unitari	78	98% componenti	14.5 secondi
Test integrazione	16	100% componenti	3.7 secondi
TOTALE	94	98% globale	18.2 secondi

Tabella 3.12: Tabella quantitativa delle attività di *testing*.

Categoria	Numero bug	Severità media	Tempo medio risoluzione
<i>Parser regex</i>	3	Media	8 ore
<i>Containerizzazione</i>	2	Alta	16 ore
TOTALE	5	Media-Alta	24 ore

Tabella 3.13: Tabella quantitativa dei bug rilevati e del tempo di risoluzione medio.

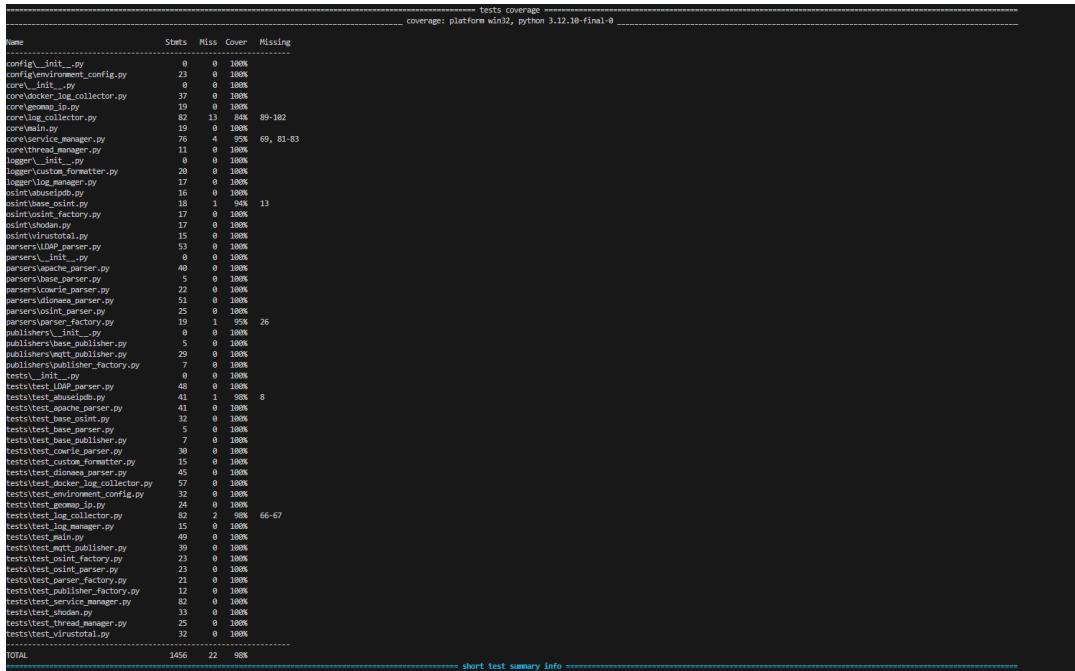


Figura 3.8: Risultato della copertura del codice dopo l'esecuzione dei *test*.

3.5 Validazione

La validazione rappresenta la fase conclusiva del processo di sviluppo, volta a garantire che il sistema *software* realizzato risponda effettivamente alle esigenze e ai requisiti definiti in fase di analisi. A differenza della verifica, che si concentra sul corretto funzionamento del prodotto rispetto alle specifiche tecniche, la validazione ha come obiettivo principale quello di accertare la coerenza del sistema rispetto agli obiettivi aziendali e operativi prefissati.

Per raggiungere tale scopo ho seguito una metodologia strutturata che ha previsto:

- la definizione di scenari di attacco realistici, basati su casi d'uso e minacce tipiche;
- l'impiego di strumenti consolidati per la simulazione, come *Nmap* per la scansione delle porte e *Hydra* per attacchi di *brute-force*;
- l'esecuzione di *test* pratici sui servizi vulnerabili dell'*honeypot*, con particolare attenzione alla generazione, raccolta e visualizzazione dei *log*.

Ho condotto le attività con un approccio incrementale: inizialmente ho testato ogni servizio singolarmente, per poi essere validarli nel contesto del sistema integrato. Questo mi ha permesso di verificare non solo la robustezza dei singoli moduli, ma anche la correttezza del flusso complessivo dei dati dal momento dell'attacco simulato fino alla loro visualizzazione nei sistemi di monitoraggio.

Le fasi di validazione si sono svolte nell'arco di una settimana, l'ultima dello stage, con revisioni e confronti quotidiani con il tutor aziendale, il quale ha seguito da vicino tutte le attività, fornendo indicazioni e approvazioni parziali. Al termine del processo, il tutor ha rilasciato la propria approvazione finale, confermando la validità del sistema sviluppato.

Conformità ai requisiti

Dei requisiti iniziali definiti nella sezione [requisiti](#), il sistema sviluppato soddisfa pienamente tutti i requisiti funzionali, non funzionali e di vincolo. Sebbene i requisiti non vadano a coprire completamente tutte le funzionalità richieste dagli obiettivi del progetto, il sistema risulta comunque essere un prodotto valido e funzionante, in grado di monitorare e analizzare efficacemente i tentativi di attacco sui servizi vulnerabili selezionati.

Categoria	Percentuale soddisfatta
Funzionali	100%
Non funzionali	100%
Vincolo	100%

Tabella 3.14: Requisiti soddisfatti dal sistema sviluppato.

Risultati quantitativi

Complessivamente, abbiamo eseguito oltre trenta scenari di simulazioni di attacco, comprendenti scansioni, *brute-force* e caricamenti di file malevoli. Tutti e quattro i servizi vulnerabili previsti hanno ricevuto la validazione ottenendo

quattro tipologie distinte di *log*, a conferma della tracciabilità completa lungo la *pipeline* di raccolta e visualizzazione. Facendo ciò abbiamo raggiunto i seguenti dati di validazione:

Servizio	Scenari testati	Tipologie attacchi	Log generati
<i>Cowrie (SSH)</i>	12	<i>Brute-force, command injection</i>	86 eventi
<i>Dionaea</i>	8	<i>Multi-protocol, malware upload</i>	77 eventi
<i>Apache</i>	7	<i>Directory traversal, XSS</i>	34 eventi
<i>OpenLDAP</i>	6	<i>LDAP injection, enumeration</i>	28 eventi
TOTALE	33	8 tipologie	225 eventi

Tabella 3.15: Tabella quantitativa degli scenari di attacco testati e dei log generati.

Strumento	Scansioni eseguite	Parametri testati	Risultati ottenuti
<i>Nmap</i>	12	Porte, servizi, <i>OS detection</i>	100% servizi rilevati
<i>Hydra</i>	8	<i>Brute-force SSH, FTP</i>	100% tentativi <i>loggati</i>
<i>LDAP-utils</i>	6	<i>LDAP queries, file upload</i>	100% eventi tracciati

Tabella 3.16: Tabella quantitativa degli strumenti e dei risultati dei test.

L’insieme di queste attività, unite all’approvazione del tutor aziendale, dimostra che il sistema soddisfa i requisiti funzionali, non funzionali e di vincolo definiti in fase di analisi, garantendo affidabilità, correttezza e coerenza operativa del prodotto sviluppato.

3.6 Visione d'insieme

In questa sezione sono presenti i risultati ottenuti durante lo sviluppo del progetto, confrontandoli con gli obiettivi iniziali e valutando il successo del progetto stesso.

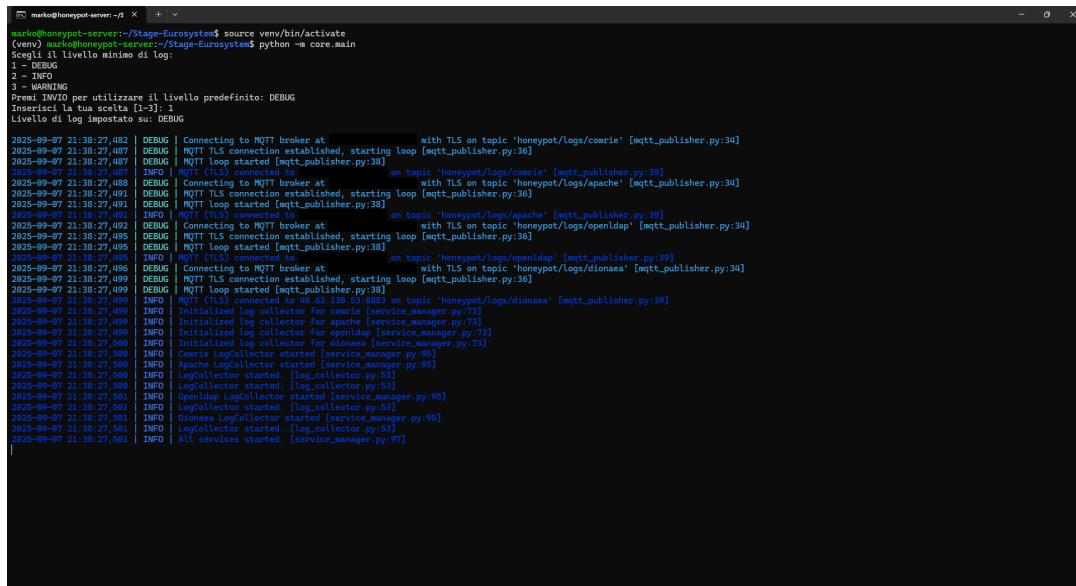
3.6.1 Prodotto finale

Il prodotto finale del progetto è un sistema *honeypot* modulare e scalabile, in grado di monitorare e analizzare tentativi di attacco su vari servizi di rete. Nelle figure seguenti sono mostrate le principali componenti del sistema, che spaziano dalla configurazione iniziale del *logger*, alla gestione del database tramite *InfluxDB*, fino alla visualizzazione dei dati attraverso le dashboard interattive fornite da *Grafana*.

```
(venv) marko@honeypot-server:~/Stage-Eurosystem$ python -m core.main
Scegli il livello minimo di log:
1 - DEBUG
2 - INFO
3 - WARNING
Premi INVIO per utilizzare il livello predefinito: DEBUG
Inserisci la tua scelta [1-3]: |
```

Figura 3.9: Scelta del livello minimo del *logger* al momento dell'inizializzazione.

CAPITOLO 3. SVILUPPO DEL PROGETTO



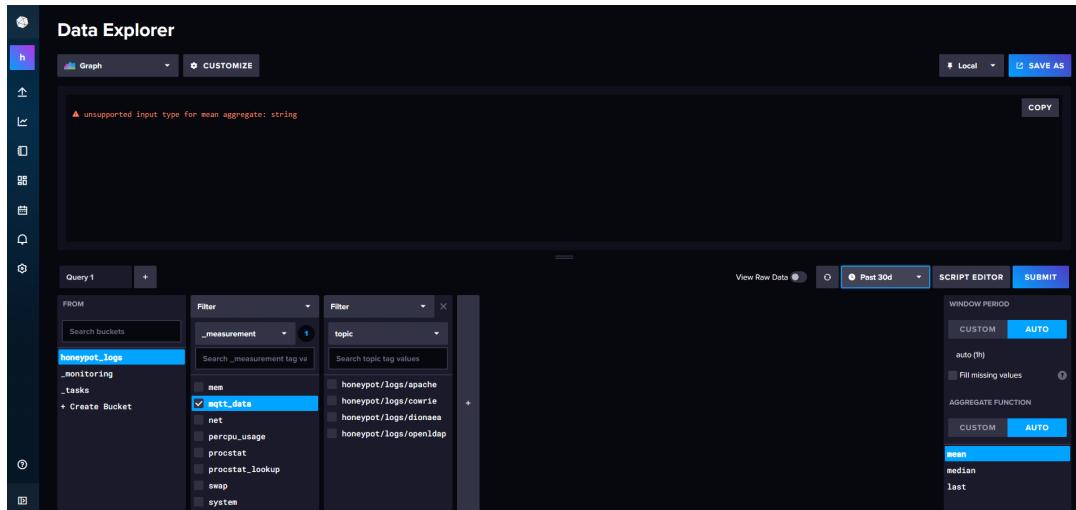
```

mario@honeypot-server:~/Stage-Eurosystems$ source venv/bin/activate
(venv) mario@honeypot-server:~/Stage-Eurosystems$ python -m core.main
Sei al livello minimo di log:
1 - DEBUG
2 - INFO
3 - WARNING
Premi INVIO per utilizzare il livello predefinito: DEBUG
Inserisci la tua scelta [1-3]: 1
Livello di log impostato su DEBUG

2025-09-07 21:38:27,482 | DEBUG | Connecting to MQTT broker at [REDACTED] with TLS on topic 'honeypot/logs/cowrie' [mqtt_publisher.py:34]
2025-09-07 21:38:27,487 | DEBUG | MQTT TLS connection established, starting loop [mqtt_publisher.py:36]
2025-09-07 21:38:27,487 | DEBUG | MQTT (TLS) started [mqtt_publisher.py:36]
2025-09-07 21:38:27,487 | INFO | MQTT (TLS) connected to [REDACTED] on topic 'honeypot/logs/cowrie' [mqtt_publisher.py:39]
2025-09-07 21:38:27,488 | DEBUG | Connecting to MQTT broker at [REDACTED] with TLS on topic 'honeypot/logs/apache' [mqtt_publisher.py:34]
2025-09-07 21:38:27,489 | DEBUG | MQTT TLS connection established, starting loop [mqtt_publisher.py:36]
2025-09-07 21:38:27,491 | DEBUG | MQTT (TLS) started [mqtt_publisher.py:36]
2025-09-07 21:38:27,491 | INFO | MQTT (TLS) connected to [REDACTED] on topic 'honeypot/logs/apache' [mqtt_publisher.py:39]
2025-09-07 21:38:27,492 | DEBUG | Connecting to MQTT broker at [REDACTED] with TLS on topic 'honeypot/logs/openldap' [mqtt_publisher.py:34]
2025-09-07 21:38:27,495 | DEBUG | MQTT TLS connection established, starting loop [mqtt_publisher.py:36]
2025-09-07 21:38:27,495 | DEBUG | MQTT (TLS) started [mqtt_publisher.py:36]
2025-09-07 21:38:27,496 | INFO | MQTT (TLS) connected to [REDACTED] on topic 'honeypot/logs/openldap' [mqtt_publisher.py:39]
2025-09-07 21:38:27,496 | DEBUG | Connecting to MQTT broker at [REDACTED] with TLS on topic 'honeypot/logs/dionaea' [mqtt_publisher.py:34]
2025-09-07 21:38:27,499 | DEBUG | MQTT TLS connection established, starting loop [mqtt_publisher.py:36]
2025-09-07 21:38:27,499 | DEBUG | MQTT (TLS) started [mqtt_publisher.py:36]
2025-09-07 21:38:27,499 | INFO | MQTT (TLS) connected to [REDACTED] on topic 'honeypot/logs/dionaea' [mqtt_publisher.py:39]
2025-09-07 21:38:27,499 | DEBUG | Initialize log collector for cowrie [service_manager.py:73]
2025-09-07 21:38:27,499 | INFO | Initialized log collector for apache [service_manager.py:73]
2025-09-07 21:38:27,499 | INFO | Initialized log collector for openldap [service_manager.py:73]
2025-09-07 21:38:27,500 | INFO | Initialized log collector for dionaea [service_manager.py:73]
2025-09-07 21:38:27,508 | INFO | Cowrie LogCollector started [service_manager.py:95]
2025-09-07 21:38:27,508 | INFO | Apache LogCollector started [service_manager.py:95]
2025-09-07 21:38:27,508 | INFO | LogCollector started [log_collector.py:53]
2025-09-07 21:38:27,508 | INFO | Openldap LogCollector started [log_collector.py:53]
2025-09-07 21:38:27,508 | INFO | LogCollector started [log_collector.py:53]
2025-09-07 21:38:27,508 | INFO | Dionaea LogCollector started [service_manager.py:95]
2025-09-07 21:38:27,508 | INFO | LogCollector started [log_collector.py:53]
2025-09-07 21:38:27,508 | INFO | All services started: [service_manager.py:97]

|
```

Figura 3.10: Esempio di *logger* in funzione in modalità *debug*.



The screenshot shows the InfluxDB Data Explorer interface. On the left, there's a sidebar with icons for Home, Data Explorer, Metrics, and Scripts. The main area has tabs for Graph and Customize. A warning message "⚠ unsupported input type for mean aggregate: string" is displayed. Below it, there's a "Query 1" section with a "FROM" clause pointing to the "honeypot_logs" bucket. The "measurement" filter is set to "net" and the "topic" filter is set to "honeypot/logs/apache". The "script editor" tab is active, showing a "WINDOW PERIOD" section with "auto (h)" selected and a "Fill missing values" checkbox. The "AGGREGATE FUNCTION" section has "mean" selected. The "SUBMIT" button is visible.

Figura 3.11: Interfaccia web di *InfluxDB* per la gestione del *database*.

CAPITOLO 3. SVILUPPO DEL PROGETTO

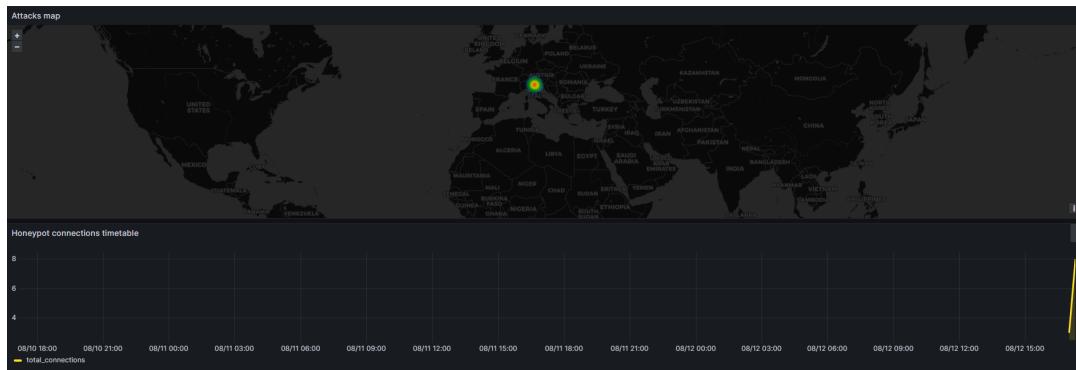


Figura 3.12: Esempio di *dashboard* di *Grafana* per la visualizzazione dei dati raccolti dall’*honeypot*.



Figura 3.13: Esempio di *dashboard* di *Grafana* per la visualizzazione dei dati raccolti dall’*honeypot*.

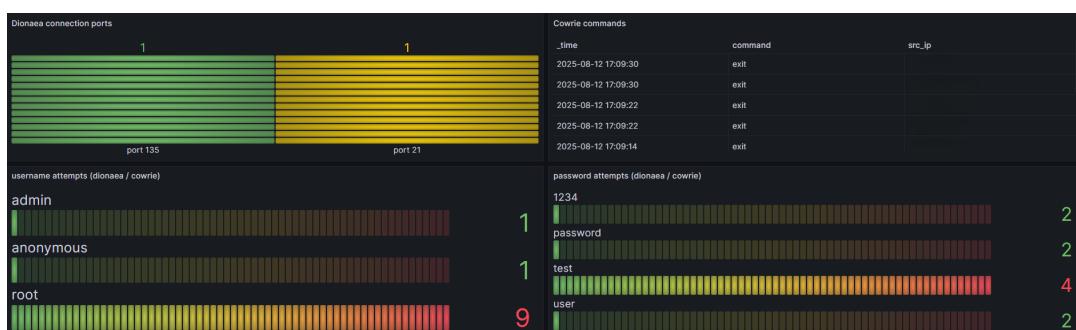


Figura 3.14: Esempio di *dashboard* di *Grafana* per la visualizzazione dei dati raccolti dall’*honeypot*.

Gli indirizzi *IP* sono opportunamente mascherati per motivi di sicurezza.

Capitolo 4

Valutazione retrospettiva

4.1 Obiettivi raggiunti

La sezione ha l'obiettivo di fornire una valutazione dettagliata dei risultati ottenuti durante lo *stage*, evidenziando quali e quanti degli obiettivi precedentemente definiti siano stati effettivamente raggiunti.

4.2 Conoscenze acquisite

Questa sezione elenca e descrive le nuove conoscenze acquisite durante lo *stage*, particolare riferimento alle tecnologie, metodologie e tecniche nel campo della cybersecurity e dello sviluppo *software*.

4.3 Competenze professionali acquisite

In questa sezione si descrivono le competenze pratiche e professionali sviluppate durante l'esperienza di *stage*, incluse competenze tecniche specifiche, capacità di *problem solving* e abilità di lavoro in *team* in ambiente professionale.

Sitografia

- Python Logging Documentation: <https://docs.python.org/3/library/logging.html>
- Logging in Python – RealPython: <https://realpython.com/python-logging/>
- Best practice per il logging in Python: https://blog.bsharp.it/wp-content/uploads/2021/01/Parte_I__Best_practice_per_il_logging_su_Python_e_come_integrarsi_con_la_dashboard_di_Kibana_tramite_AWS_Kinesis_Data_Firehose_ed_Amazon_Elasticsearch_Service-Proud2beCloud.pdf
- Log Parsing and Next-Gen SIEM – Crowdstrike: <https://www.crowdstrike.com/en-us/cybersecurity-101/next-gen-siem/log-parsing/>
- Cowrie Documentation – Output: <https://docs.cowrie.org/en/latest/OUTPUT.html>
- Serde – Data serialization for Python: <https://pypi.org/projectserde/#getting-started>
- Common Vulnerabilities and Exposures (CVE): <https://www.cve.org/>
- What is MQTT – AWS: <https://aws.amazon.com/what-is/mqtt/>
- Getting Started with MQTT – EMQX: <https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt#what-is-mqtt>
- Multithreading in Python – GeeksforGeeks: <https://www.geeksforgeeks.org/python/multithreading-python-set-1/>
- TIG Stack – DockerHub Repository: <https://github.com/alekece/tig-stack>

CAPITOLO 4. SITOGRAFIA

- Dionaea Honeypot – DockerHub: <https://hub.docker.com/r/dinotools/dionaea>
- Dionaea Honeypot – Official Documentation: <https://dionaea.readthedocs.io/en/latest/installation.html>

Acronimi e abbreviazioni

AI *Artificial Intelligence.* [16](#)

API *Application Programming Interface.* [6](#)

CISO *Chief Information Security Officer.* [7](#)

ELK *Elasticsearch, Logstash, Kibana.* [17](#)

ES Esempio di acronimo. [v](#)

ETL *Extract, Transform, Load.* [33](#)

FTP *File Transfer Protocol.* [15](#)

HTTP *Hypertext Transfer Protocol.* [36](#)

HTTPS *Hypertext Transfer Protocol Secure.* [37](#)

IDE *Integrated Development Environment.* [6](#)

IOC *Indicator of Compromise.* [16](#)

IT *Information Technology.* [1](#)

LAN *Local Area Network.* [9](#)

LDAP *Lightweight Directory Access Protocol.* [37](#)

LDAPS *Lightweight Directory Access Protocol over SSL/TLS.* [37](#)

MDR *Managed Detection and Response.* [2](#)

ML *Machine Learning.* [16](#)

MQTT *Message Queuing Telemetry Transport.* [15](#)

OSINT *Open Source Intelligence.* [15](#)

PPTP *Point-to-Point Tunneling Protocol.* [36](#)

RPC *Remote Procedure Call.* [36](#)

SCP *Secure Copy Protocol.* [36](#)

SIEM *Security Information and Event Management.* [16](#)

SMB *Server Message Block.* [15](#)

SOC *Security Operation Center.* [2](#)

SQL *Structured Query Language.* [36](#)

SSH *Secure Shell.* [36](#)

TDD *Test-Driven Development.* [43](#)

TFTP *Trivial File Transfer Protocol.* [36](#)

TIG *Telegraf, InfluxDB, Grafana.* [15](#)

UML *Unified Modeling Language.* [23](#)

USB *Universal Serial Bus.* [8](#)

XSS *Cross-Site Scripting.* [37](#)

Glossario

Application Programming Interface Insieme di regole e strumenti che consentono a software differenti di comunicare tra loro e scambiare dati o funzionalità. [6](#)

Audit Un processo sistematico di esame e valutazione delle attività, dei controlli e delle procedure di un’organizzazione, al fine di garantire la conformità alle normative e l’efficacia delle misure di sicurezza. [3](#)

Chief Information Security Officer Figura dirigenziale responsabile della strategia di sicurezza informatica e della protezione delle informazioni aziendali. [7](#)

Container Un’unità standardizzata di software che include tutto il necessario per eseguire un’applicazione, garantendo coerenza tra gli ambienti di sviluppo, test e produzione. [6](#)

Esempio di nome Esempio di descrizione. [v](#)

Extract, Transform, Load Processo per estrarre dati da diverse fonti, trasformarli in un formato coerente e caricarli in un sistema di destinazione. [33](#)

File Transfer Protocol Protocollo di rete utilizzato per il trasferimento di file tra client e server su reti TCP/IP. [15](#)

Hypertext Transfer Protocol Protocollo applicativo usato per la trasmissione di documenti ipertestuali sul web. [37](#)

Hypertext Transfer Protocol Secure Versione sicura del protocollo HTTP che utilizza SSL/TLS per cifrare le comunicazioni. [37](#)

Integrated Development Environment Ambiente software che integra strumenti di sviluppo come editor, compilatore, debugger e gestione dei progetti. [6](#)

Indicator of Compromise Elemento tecnico, come un indirizzo IP o un hash, che segnala la possibile presenza di un'attività malevola in un sistema informatico. [16](#)

Managed Detection and Response Servizio esterno che fornisce monitoraggio proattivo, rilevamento e risposta agli incidenti di sicurezza informatica. [2](#)

Open Source Intelligence Attività di raccolta e analisi di informazioni provenienti da fonti pubblicamente disponibili, come siti web e social media. [15](#)

Point-to-Point Tunneling Protocol Protocollo di rete usato per implementare connessioni VPN, oggi considerato insicuro. [36](#)

Remote Procedure Call Protocollo che consente l'esecuzione di procedure su un sistema remoto come se fossero locali. [36](#)

Secure Copy Protocol Protocollo che permette il trasferimento sicuro di file tra host utilizzando connessioni SSH. [36](#)

Security Information and Event Management Piattaforma che raccoglie, correla e analizza log di sistema per identificare potenziali incidenti di sicurezza. [16](#)

Server Message Block Protocollo di rete usato principalmente nei sistemi Windows per la condivisione di file, stampanti e risorse di rete. [15](#)

Security Operation Center Struttura organizzativa dedicata al monitoraggio e alla gestione degli incidenti di sicurezza informatica in tempo reale.

[2](#)

Structured Query Language Linguaggio standard utilizzato per interrogare, manipolare e gestire basi di dati relazionali. [36](#)

Secure Shell Protocollo di rete che consente di stabilire connessioni remote sicure e cifrate tra due sistemi. [36](#)

Unified Modeling Language Linguaggio standard per la modellazione e la rappresentazione grafica di sistemi software complessi. [23](#)

Cross-Site Scripting Vulnerabilità delle applicazioni web che permette a un attaccante di iniettare codice malevolo all'interno di pagine visualizzate da altri utenti. [37](#)