Oggetto: sicurezza all'interno di una pipeline di sviluppo in CI/CD tramite GitHub Actions.
Software SpiderFoot[1]. Relazione tecnica finale.

---

1. Introduzione: SpiderFoot

SpiderFoot è una piattaforma open-source di automazione per l'Open Source Intelligence (OSINT). Il software automatizza la raccolta di informazioni su un target specifico (asset). Interroga simultaneamente centinaia di fonti di dati pubbliche e restituisce un profilo di intelligence completo. Una particolarità: il tool è integrato anche nella distribuzione **Kali-Linux**[2], oltre che presente nella lista dei Vulnerability Scanner Tools di **OWASP**[3]. Nel novembre 2022 tale tool è stato acquisito dall'azienda statunitense di Cyber Threat Intelligence "Intel471"[4].

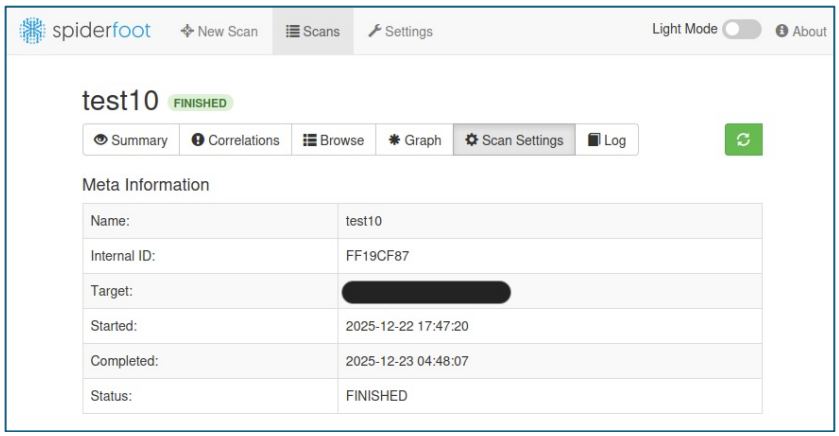
a. Scopo e funzionalità

L'obiettivo primario di SpiderFoot è la fase di Reconnaissance (o Footprinting) all'interno della Cyber Kill Chain[5]. L'applicazione accetta in input un target (che può essere un indirizzo IP, un nome di dominio, un indirizzo e-mail, un username, una subnet, ecc.) e avvia una serie di scansioni per identificare relazioni, vulnerabilità esposte, data leak o altre informazioni sensibili correlate a quell'asset.

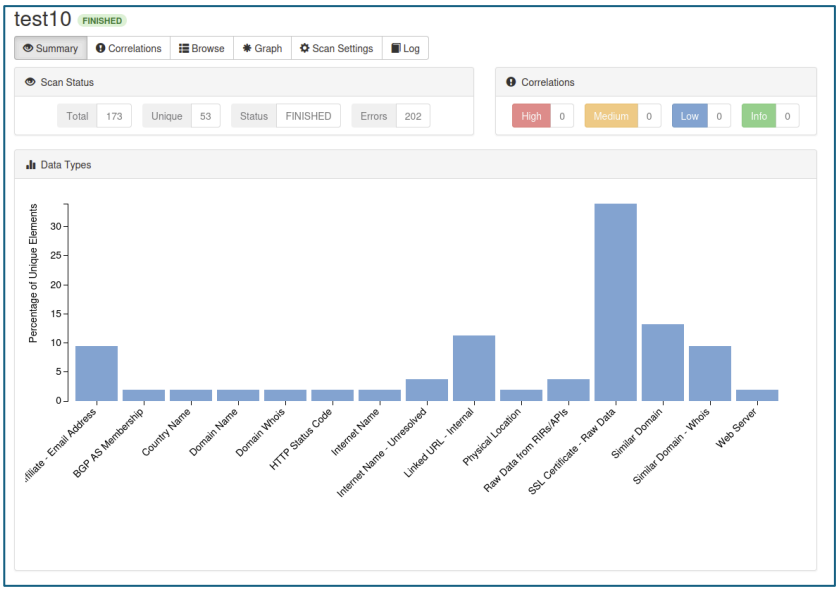
Nelle figure seguenti sono riportati alcuni screenshot del programma SpiderFoot inerenti a un test eseguito su un target di interesse investigativo.

---

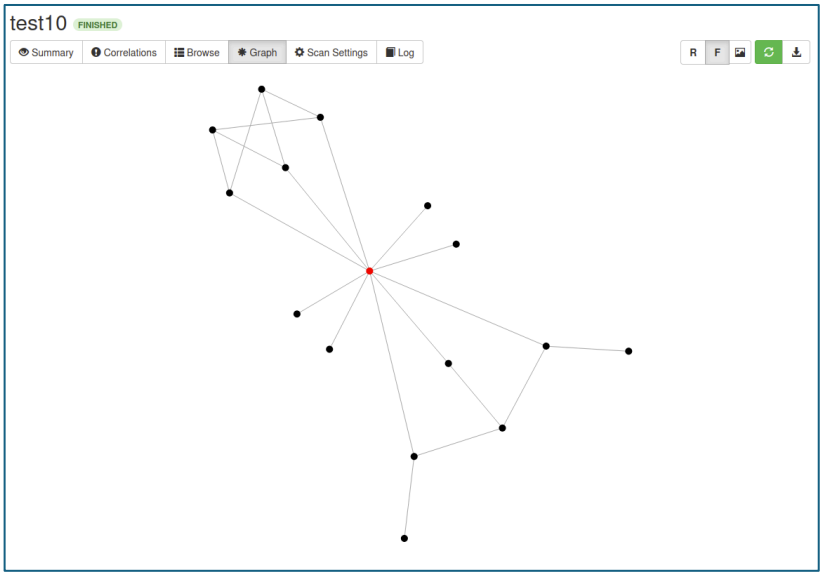
[1] Rif. progetto studente: https://github.com/numdav/spiderfoot .
[2] Rif. sito https://www.kali.org/tools/spiderfoot/.
[3] Rif. sito https://owasp.org/www-community/Vulnerability_Scanning_Tools .
[4] Rif. sito https://www.intel471.com/blog/intel-471-acquires-spiderfoot .
[5] Rif. https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.



SpiderFoot – Scan Setting



SpiderFoot – Summary



SpiderFoot – Graph

b. Architettura modulare

L'applicazione è costruita su un'architettura altamente modulare basata su Python. Il core del sistema (*sf.py* e *sfscan.py*) orchestra l'esecuzione di numerosi moduli plug-in (i file *sfp_*.py* presenti nella cartella *modules/*). Ogni modulo è responsabile dell'interazione con una specifica fonte di dati esterna (es. *Shodan, VirusTotal, HaveIBeenPwned, Whois, ecc.*). Questa struttura permette un'estensione rapida delle funzionalità ma introduce rischi di sicurezza legati alla gestione di centinaia di chiamate API verso servizi terzi.

c. Stack tecnologico

Dall'analisi dei file di configurazione (*requirements.txt*, *Dockerfile*) e del codice sorgente, lo stack tecnologico risulta composto da:

- Linguaggio: Python 3.
- Web Server: utilizza l'interfaccia web (*sfwebui.py*) per configurare scansioni e visualizzare grafici.
- Interfaccia CLI: dispone di una Command Line Interface (*sfcli.py*) per l'integrazione in script.
- Persistenza dati: utilizza SQLite (*db.py*) come database locale per archiviare i risultati delle scansioni e le correlazioni tra gli eventi.
- Containerizzazione: è distribuito tramite immagini Docker (basate su Alpine Linux).

d. Criticità nel contesto DevSecOps

SpiderFoot richiede elevati standard di sicurezza poiché gestisce credenziali sensibili e analizza dati esterni potenzialmente malevoli. Trattandosi di un tool critico per operazioni sia difensive che offensive, l'affidabilità è fondamentale. Le ricerche a lungo termine (oltre 12 ore) devono completarsi senza blocchi o perdite di dati.
Disponibilità e **Integrità** dei dati rappresentano requisiti critici per un'azienda di Cyber Threat Intelligence.

2. La Pipeline di Sicurezza *SE4SCS*[6]

Per garantire la sicurezza lungo tutto il ciclo di vita del software (SDLC), è stata implementata una pipeline CI/CD **DevSecOps** ("Fail-Safe"[7]) sul repository GitHub. La pipeline orchestra quattro motori di analisi complementari in parallelo:

a. **GitGuardian:** per rilevare segreti hardcodati (API Key, password).
b. **Snyk:** utilizzato in tre modalità per una copertura totale:
   - *Snyk Open Source (SCA):* analizza le dipendenze in *requirements.txt*.
   - *Snyk Code (SAST):* analizza il codice sorgente per vulnerabilità applicative.
   - *Snyk Container:* analizza l'immagine Docker base per vulnerabilità di sistema.
c. **Semgrep:** per l'analisi statica del codice (SAST).
d. **SonarCloud:** per la qualità del codice (Quality Gate) identificando *Code Smells* e debito tecnico.

La pipeline genera report JSON delle risultanze e crea a run-time una dashboard riepilogativa per una visione immediata. I report vengono inoltre inviati ai relativi servizi online di GitGuardian, Snyk, Semgrep e SonarQube Cloud, consentendo una consultazione centralizzata e il monitoraggio storico delle metriche di sicurezza.

Per una descrizione più dettagliata della pipeline in argomento si rimanda al file *README_SE4SCS.md*[8].

3. Analisi dei report di sicurezza (Discovery)

L'analisi dei 6 report JSON della pipeline SE4SCS ha evidenziato una postura di sicurezza compromessa. La causa principale è il mancato aggiornamento del progetto dal 2022 (ultima release v4.0 del 7 aprile 2022), che ha esposto il software a vulnerabilità critiche applicative e infrastrutturali. In particolare:

a. **GitGuardian** *(ggshield.json)*: il tool non ha rilevato occorrenze.[9]

[6] Rif. https://github.com/numdav/spiderfoot/blob/fix-vuln-numda/.github/workflows/se4scs.yml.
[7] Il job, in questo caso essendo a scopo didattico, non deve fermarsi, permettendo di scansionare tutto il progetto. Deve tuttavia rilevare gli eventuali errori di esecuzione della pipeline (es. "continue-on-error: true").
[8] Rif. https://github.com/numdav/spiderfoot/blob/fix-vuln-numda/.github/workflows/README_SE4SCS.md.
[9] La versione web (Deep Scan sulla .git history) mostra alcuni **Falsi Positivi** su *commit* storici.

b. **Semgrep** *(semgrep.json)*: oltre 20 occorrenze rilevate. Molte di esse sono vulnerabilità di configurazione o uso di legacy code (es. protocollo SSL deprecato). Solamente due vulnerabilità sono state segnalate come CRITICAL:
   . SQL Injection (CWE-89) nel file *sfwebui.py*
   . Hardcoded API Key nel modulo *sfp_citadel.py*. Si tratta di una public API key che, se condivisa, espone il sistema a interruzioni per *rate limiting* di Leak-Lookup, compromettendo la **Disponibilità** dello strumento.

c. **Snyk**
   – **Snyk Open Source** *(snyk_deps.json):* tra le decine di vulnerabilità rilevate:
      . il file *requirements.txt* richiede versioni obsolete di librerie critiche come cryptography (< 39.x) e pypdf2 (<2), vulnerabili a *Denial of Service* (CVE-2023-0286 e CWE-400).

   – **Snyk Code** *(snyk_code.json):* ha rilevato le seguenti vulnerabilità principali:
      . Hardcoded Secret: una API Key in chiaro nel file *sfp_citadel.py*. Trattasi tuttavia di una public API key.
      . Path Traversal: gestione insicura dei path in *sfcli.py* (CWE-23). L'input utente non viene sanitizzato prima del passaggio alla funzione.

   – **SnykContainer** *(snyk_container.json)*: il tool ha rilevato che nel file:
      . *Dockerfile*, l'immagine "base alpine:3.13" è *End-of-Life (EOL)*, esponendo il container a decine di vulnerabilità di sistema critiche. Durante l'analisi preliminare, è emerso che il report di Snyk Container indicava zero vulnerabilità per l'immagine base Alpine 3.13, nonostante questa sia in End-of-Life. Lo stesso repository analizzato via webapp Snyk forniva vulnerabilità note nella versione Alpine 3.13.
      L'analisi manuale del codice (cd. *code review*) ha rivelato che il Dockerfile originale esegue *"rm -rf /lib/apk/db"* per ridurre le dimensioni dell'immagine. Questa pratica rimuove il manifesto dei pacchetti installati, rendendo l'immagine opaca agli strumenti di scansione. Di conseguenza, si genera un **Falso Negativo** che crea un falso senso di sicurezza.

d. **SonarCloud:** ha segnalato oltre 4000 "Code Smells", indicando un elevato debito tecnico e complessità cognitiva. All'interno del codice, infatti, sono presenti molti *if/else* annidati, che aumentano proporzionalmente la probabilità di nascondere bug logici e rendono difficile l'auditing manuale.

4. Metodologia di Prioritizzazione (SSVC)

Con oltre 4.000 *issues* rilevati (SonarCloud, Snyk), una metrica basata esclusivamente su CVSS risulta insufficiente per una gestione operativa efficiente. È stato adottato il framework SSVC (Stakeholder-Specific Vulnerability Categorization)[10] seguendo le linee guida CISA (Cybersecurity and Infrastructure Security Agency).
Il decision tree è stato adattato al contesto OSINT del tool, che richiede elevate garanzie di **Disponibilità** e **Integrità**. Ogni vulnerabilità è stata valutata secondo tre vettori:

- **Exploitation:** stato dello sfruttamento (None, PoC, Active).
- **Technical Impact**: impatto tecnico sulla missione (Partial, Total).
- **Mission & Well-being**: conseguenze sul business (Low, Medium, High)

Applicando l'albero decisionale SSVC, le vulnerabilità sono state categorizzate in tre azioni operative:

- **ACT**: Vulnerabilità critiche che richiedono un fix immediato (Patch o Configurazione).
- **TRACK**: Vulnerabilità da monitorare o pianificare per sprint futuri.
- **DEFER**: Rischi accettati (Risk Acceptance) o **Falsi Positivi**.

Di seguito l'analisi dettagliata per i file esaminati nel progetto.

a. Livello ACT (Priorità Massima)

Vulnerabilità che richiedono intervento immediato (Bloccanti).
Criterio CISA: Exploitation = Active/PoC AND (Technical Impact = Total OR Mission Impact = High)

[10] Rif. https://www.cisa.gov/stakeholder-specific-vulnerability-categorization-ssvc.

Vulnerabilità (CWE)	Fonte (Report)	File	Decisione SSVC	Remediation
Vulnerable Dependencies (DoS – CVE-2023-0286 – CWE-835)	Snyk Deps [SNYK-PYTHON-CRYPTOGRAPHY-*, SNYK-PYTHON-PYPDF2-*]	requirements.txt	**Expl:** PoC Available **TechImp:** Partial (DoS) **Miss:** High (Availability)	**Dependency Update:** Aggiornamento librerie critiche: cryptography>=42.0.0 pyOpenSSL>=24.0.0 PyPDF2>=3.0.0.
OS End-of-Life (Container Vulns – CWE 1104)	Snyk Container [alpine:3.13.0]	Dockerfile	**Expl:** Active (KEV) **TechImp:** Total (System) **Miss**: High (compromission tool – Availab.)	**Infrastructure Upgrade:** Passaggio ad alpine:3.19 e rimozione comando di pulizia *(rm -rf /lib/apk/db)* per il **Falso Negativo**.
Path Traversal (CWE-23)	Snyk Code [python/PT]	sfcli.py	**Expl:** PoC Available **TechImp:** Total (File Write) **Miss:** High (System Integrity)	**Input Validation:** Implementazione funzione *is_safe_path* basata su *os.path.commonpath* per garantire che il file di output resti nella directory prevista.
Hardcoded Secret (CWE-798)	- Snyk Code [HardcodedNon CryptoSecret] - Semgrep [detected-generic-api-key]	sfp_citadel.py	**Expl:** Active (Public) **TechImp:** Partial (Access Denied) **Miss:** Medium/ High (Service Ban for rate limiting – Availability)	**Secret Removal:** Rimozione fisica della stringa dal codice sorgente e implementazione del recupero dinamico *(self.opts.get)* che forza l'utente ad inserire la propria chiave.

SQL Injection (aiosqlite via cherrypy – CWE-89)	Semgrep [sqli.cherrypy-aiosqlite-sqli]	sfwebui.py	**Expl:** PoC (pattern vuln) **TechImp:** Total (db access/ modified) **Miss:** High (Data Integrity)	**Secure by design:** rimozione della funzionalità. Il db non viene istanziato, bloccando preventivamente qualsiasi richiesta di accesso.

b. Livello TRACK (Monitoraggio)

Vulnerabilità serie ma non bloccanti o mitigate dal contesto.
Criterio CISA: Exploitation = None/PoC AND Technical Impact = Partial

Vulnerabilità (CWE)	Fonte (Report)	File	Decisione SSVC	Remediation
SSRF Server-Side Request Forgery (CWE-918)	Snyk Code [python/Ssrf]	sfcli.py	**Expl:** PoC (Richiede accesso locale) **TechImp:** Partial (Local Access) **Miss:** Medium	La connessione a URL arbitrari è una funzionalità core del client CLI. Mitigata dall'accesso locale. Si monitora per evitare future esposizioni via API.
Privilege Escalation (Misconfiguration)	Semgrep [docker-compose.security.no-new-privileges]	docker-compose.yml	**Expl:** None **TechImp:** Partial **Miss:** Medium	Manca *"no-new-privileges:true"*. Rischio mitigato dall'uso di utente non-root nel Dockerfile.

Deprecated SSL Method (CWE-326)	- Snyk Code [python/ssl~wrap_ socket~without~pro tocol] - Semgrep [ssl.wrap_socket]	sflib.py	**Expl:** None **TechImp:** Partial **Miss:** Medium (compatibilità servizi OSINT)	Uso di *ssl.wrap_ socket*. Necessario per compatibilità con target legacy durante scansioni OSINT. La rimozione immediata comprometterebbe la funzionalità core del programma verso target obsoleti.

c. Livello DEFER (Debito Tecnico – Risk Acceptance)

Problematiche a basso rischio o Falsi Positivi accettati.
Criterio CISA: Technical Impact = Low OR False Positive.

Vulnerabilità (CWE)	Fonte (Report)	File	Decisione SSVC	Note
Weak Hashing (MD5) (CWE-916)	Snyk Code [InsecureHash]	db.py sfp_punkspider.py sfp_gravatar.py	**Expl:** None **TechImp:** Low (Non-Crypto) **Miss:** None	MD5 usato per ID univoci interni o per query esterne, non per password. Rischio collisione irrilevante.
Cognitive Complexity (Code Smells)	SonarCloud [python:S3776, python:S6546…]	*Multipli*	**Expl:** None **TechImp:** None **Miss:** Low	Oltre 4000 code smells differiti a futuri refactoring.

L'adozione del framework CISA SSVC è stata determinante per trasformare un elenco ingestibile di vulnerabilità in un piano d'azione concreto. Focalizzandosi sui vettori Mission & Well-being, è stato possibile distinguere tra problemi tecnici (es. complessità del codice) e minacce reali alla stabilità dello strumento (es. Alpine

EOL, Path Traversal). Questo ha garantito l'allocazione delle risorse esclusivamente sui fix ACT che proteggono direttamente **Disponibilità** e **Integrità** dello strumento OSINT.

5. Baseline Testing (Verifica Pre-Remediation)

Prima di procedere con l'applicazione delle misure di remediation, è stata istituita ed eseguita una pipeline di Baseline Testing *(test-baseline.yml)* mirata al branch master[11]. Sono state eseguite 5 verifiche sulla versione vulnerabile (branch master):

a. Verifica Web Server

Questo step ha eseguito l'applicazione nella sua versione legacy (vulnerabile), lanciando il servizio *sf.py* in background e verificando la risposta HTTP e la connettività di base tramite *curl*.

b. Verifica Operatività CLI

È stata verificata l'operatività del client da riga di comando (*sfcli.py*), con l'esecuzione di comandi di help e tentativo di connessione con il web server.

c. Verifica SQL Injection

È stato predisposto un probe automatico verso l'endpoint */query*, con l'invio di un payload SQL arbitrario *("SELECT 1")*. Il server ha risposto con codice *"HTTP 200"* ed eseguito la query, confermando che l'input non è sanificato e che il database è esposto a manipolazioni dirette.

d. Verifica Hardcoded Secrets

È stato implementato uno script di Analisi Dinamica (*test_citadel_vuln.py*) per analizzare il comportamento del modulo *sfp_citadel.py*, con mocking delle chiamate di rete per intercettare i dati in uscita. Lo script ha intercettato la trasmissione in chiaro della chiave API hardcodata *(3edfb560...)*, confermando la violazione delle policy di Secret Management.

[11] Il file di integration test originale in /test non era eseguibile. È stata creata una pipeline baseline dedicata per le funzionalità fondamentali.

e. Verifica Build Docker Legacy

La pipeline ha verificato la costruibilità dell'immagine Docker originale. La build su base alpine:3.13 (versione End-of-Life) è stata completata con successo, dimostrando che l'ambiente di partenza, seppur obsoleto e insicuro, è tecnicamente riproducibile.

Il test ha confermato la funzionalità "as-is" dell'applicazione, fornendo una baseline di riferimento per le fasi successive. Questo approccio garantisce che eventuali fallimenti futuri nel *regression testing* siano imputabili alle modifiche di sicurezza e non a difetti funzionali preesistenti.

Per una descrizione più dettagliata della pipeline in argomento si rimanda al file *README_test-baseline.md.*[12]

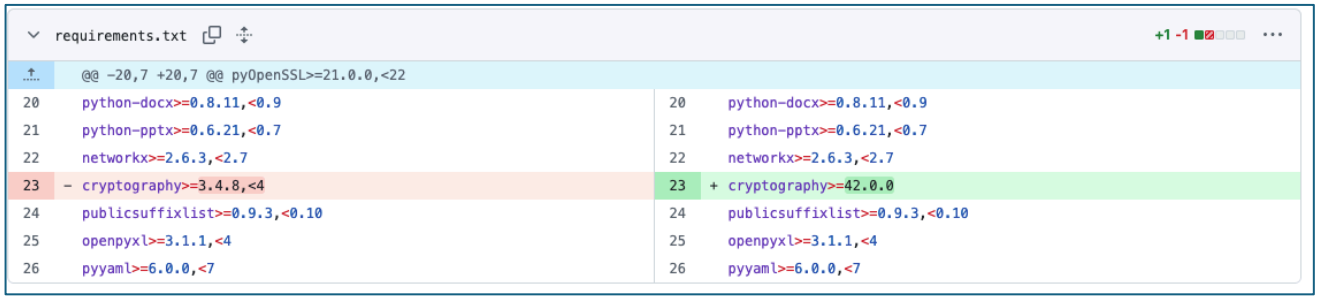
6. Remediation: interventi di livello ACT

Di seguito il dettaglio tecnico degli interventi eseguiti per risolvere le vulnerabilità critiche (ACT).

a. Dependency Management *(requirements.txt)*

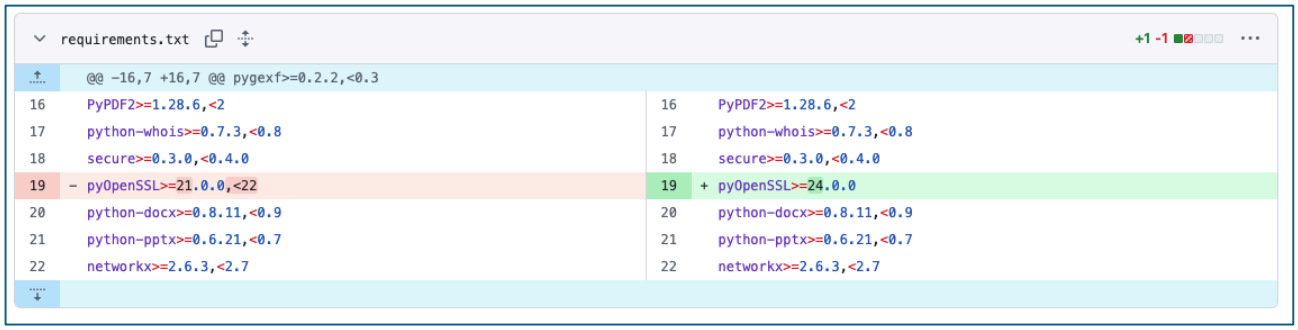
Problema: Dipendenze critiche obsolete. La libreria cryptography (<4) è affetta da CVE-2023-0286 (DoS e Memory Corruption), sfruttabile tramite certificati malformati. PyPDF2 (<2) è vulnerabile a DoS (loop infinito).

Soluzione: Aggiornamento delle librerie alle versioni *patchate*, reso compatibile dall'upgrade ad Alpine 3.19 (supporto compilatori Rust per Cryptography). L'aggiornamento di cryptography ha richiesto anche l'upgrade di pyOpenSSL.



requirements.txt

[12] Rif. https://github.com/numdav/spiderfoot/blob/fix-vuln-numda/.github/workflows/README_test-baseline.md.



requirements.txt

requirements.txt

b. Infrastructure Hardening *(Dockerfile)*

Problema: L'immagine Alpine 3.13 è End-of-Life (gennaio 2021) e contiene vulnerabilità di sistema critiche. Il comando *"rm -rf /lib/apk/db"* impedisce agli scanner di sicurezza di inventariare i pacchetti installati, generando un **Falso Negativo**.[13]

Soluzione: Aggiornamento ad Alpine 3.19 (supporto toolchain Rust per le nuove librerie crittografiche) e rimozione del comando di cancellazione del DB.

[13] **Snyk Web** (analisi statica del Dockerfile) ha rilevato correttamente Alpine 3.13 come obsoleta. **Snyk CLI** (analisi filesystem compilato) ha generato un **Falso Negativo** a causa della rimozione di */lib/apk/db*.

Dockerfile +2 -3

```
@@ -34,7 +34,7 @@
34  #   sudo docker build -t spiderfoot-test --build-arg REQUIREMENTS=test/requirements.txt .
35  #   sudo docker run --rm spiderfoot-test -m pytest --flake8 .
36
37 - FROM alpine:3.12.4 AS build
38  ARG REQUIREMENTS=requirements.txt
39  RUN apk add --no-cache gcc git curl python3 python3-dev py3-pip swig tinyxml-dev \
40   python3-dev musl-dev openssl-dev libffi-dev libxslt-dev libxml2-dev jpeg-dev \
@@ -49,7 +49,7 @@ RUN pip3 install -r "$REQUIREMENTS"
49
50
51
52 - FROM alpine:3.13.0
53  WORKDIR /home/spiderfoot
54
55  # Place database and logs outside installation directory
@@ -63,7 +63,6 @@ RUN apk --update --no-cache add python3 musl openssl libxslt tinyxml libxml2 jpe
63      && adduser -G spiderfoot -h /home/spiderfoot -s /sbin/nologin \
64              -g "SpiderFoot User" -D spiderfoot \
65      && rm -rf /var/cache/apk/* \
66 -    && rm -rf /lib/apk/db \
67      && rm -rf /root/.cache \
68      && mkdir -p $SPIDERFOOT_DATA || true \
69      && mkdir -p $SPIDERFOOT_LOGS || true \

34  #   sudo docker build -t spiderfoot-test --build-arg REQUIREMENTS=test/requirements.txt .
35  #   sudo docker run --rm spiderfoot-test -m pytest --flake8 .
36
37 + FROM alpine:3.19 AS build
38  ARG REQUIREMENTS=requirements.txt
39  RUN apk add --no-cache gcc git curl python3 python3-dev py3-pip swig tinyxml-dev \
40   python3-dev musl-dev openssl-dev libffi-dev libxslt-dev libxml2-dev jpeg-dev \
49
50
51
52 + FROM alpine:3.19
53  WORKDIR /home/spiderfoot
54
55  # Place database and logs outside installation directory
63      && adduser -G spiderfoot -h /home/spiderfoot -s /sbin/nologin \
64              -g "SpiderFoot User" -D spiderfoot \
65      && rm -rf /var/cache/apk/* \
66      && rm -rf /root/.cache \
67      && mkdir -p $SPIDERFOOT_DATA || true \
68      && mkdir -p $SPIDERFOOT_LOGS || true \
```

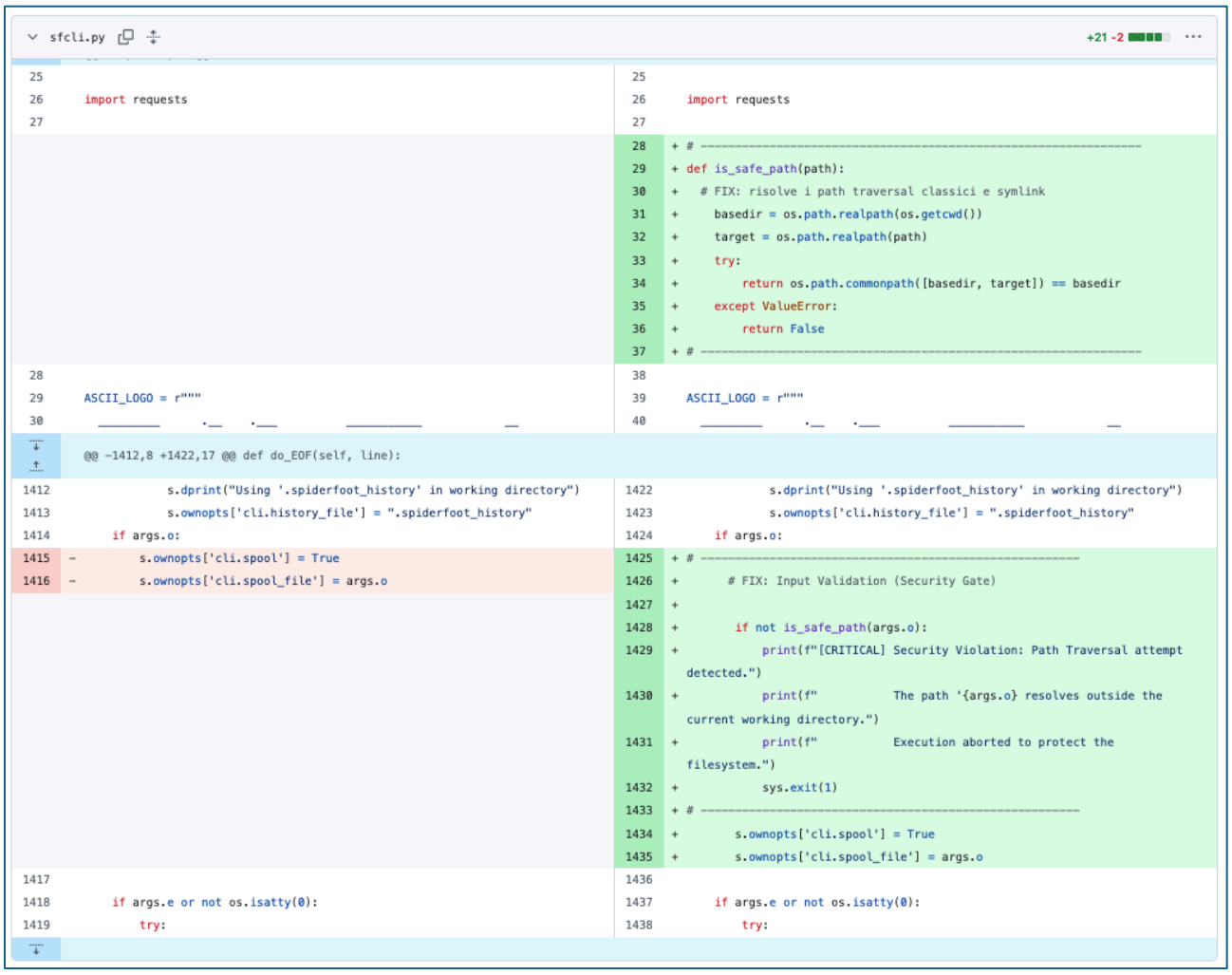
Dockerfile

c. Code Security (Path Traversal) *(sfcli.py)*

Problema: Path Traversal (CWE-22). L'argomento -o (output file) accetta percorsi arbitrari dall'utente senza validazione[14], permettendo la scrittura fuori dalla directory di lavoro (es. *../../etc/passwd*) o sovrascrittura di script di sistema.

Soluzione: Implementazione della funzione di validazione *is_safe_path()* basata su *os.path.commonpath()*. La funzione verifica che il file risieda nella directory consentita *(basedir)*. In caso di tentativo di path traversal, il programma termina con errore "CRITICAL: Security Violation - Path Traversal attempt detected."

[14] *"s.ownopts["cli.spool_file"] = args.o"*: codice vulnerabile. L'input viene passato alla funzione *open ()* senza sanitizzazione.



sfcli.py

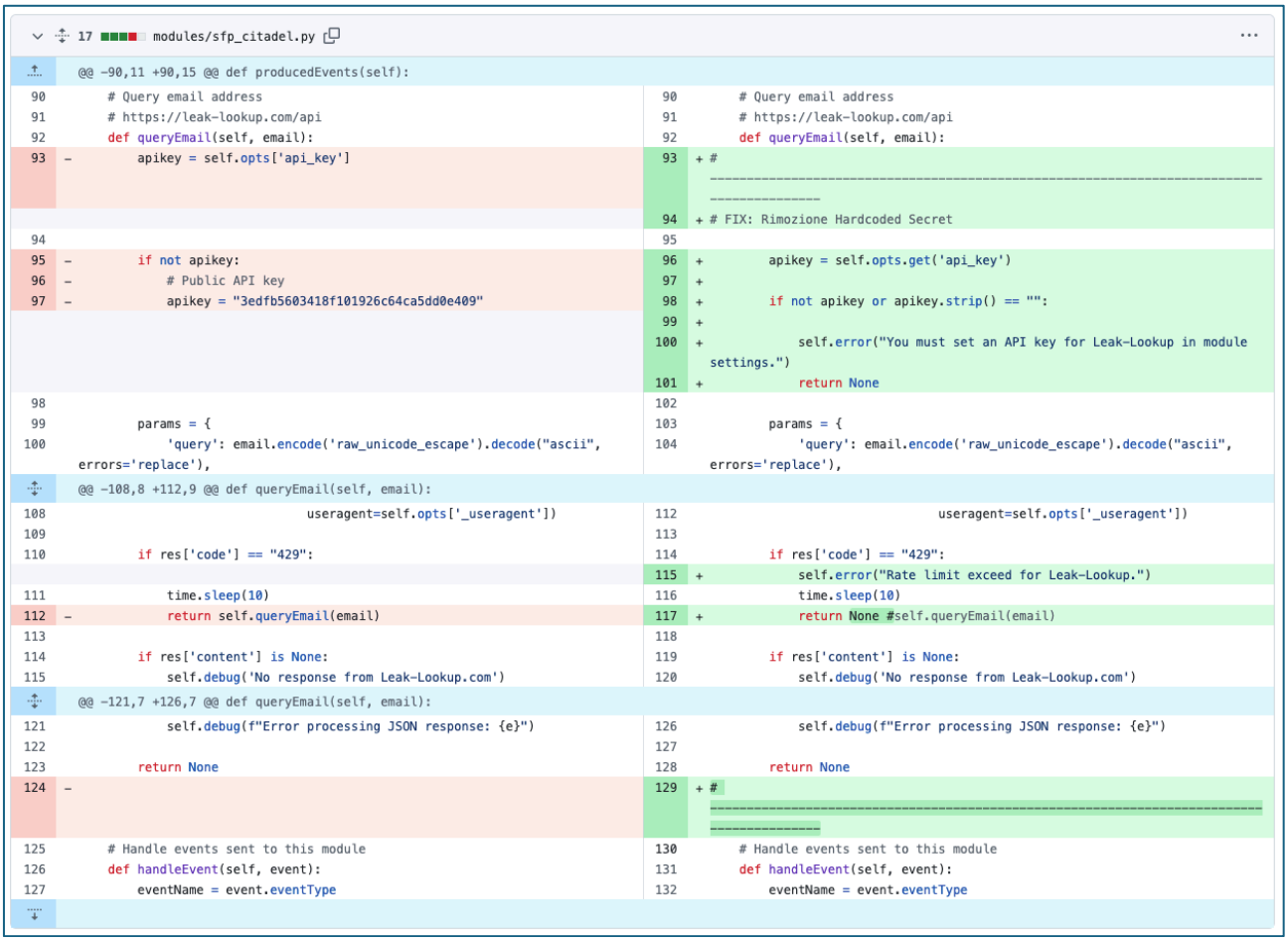
d. Secret Management *(modules/sfp_citadel.py)*

Problema: Hardcoded Secret (CWE-798). Una API Key[15] valida per Leak-Lookup è scritta in chiaro nel codice sorgente, esposta a chiunque abbia accesso al repository. L'uso massivo della chiave esposta può causare mancanza di **Disponibilità** del servizio (DoS per *rate limiting* o *ban*).

Soluzione: Rimozione della stringa *hardcodata* e implementazione di logica per il recupero dinamico tramite *self.opts*. Il modulo viene bloccato se la chiave è

[15] apikey = "3edfb5603418f101926c64ca5dd0e409".

assente. In caso di errore *HTTP 429*[16], il logging mostra “Rate limit exceed for Leak-Lookup.” evitando loop infiniti di richieste.



sfp_citadel.py

e. SQL Injection *(sfwebui.py)*

Problema: Il metodo *query()* esposto all’endpoint */query* accetta una stringa SQL grezza dall’utente e la passa direttamente al motore del database *(dbh.dbh.execute(query))*. Nonostante un controllo superficiale *(startswith(“select”))*, la funzione trasforma l’applicazione in una “SQL Injection as-a-Service”. Un attaccante autenticato (o in caso di bypass dell’auth) può esfiltrare l’intero database tramite una semplice richiesta HTTP.

Soluzione: Data la natura della funzione (verosimilmente per integrazione CLI-Web) e la criticità della vulnerabilità per **Disponibilità** e **Integrità**, si è

[16] HTTP Code 429 = Too many requests.

disabilitata l'istanziazione del database, attuando un approccio "**Secure by Design**".

sfwebui.py +7 -5

```
@@ -1349,12 +1349,14 @@ def ping(self: 'SpiderFootWebUi') -> list:
1349      def query(self: 'SpiderFootWebUi', query: str) -> str:
1350          """For the CLI to run queries against the database.
1351
1352 -        Args:
1353 -            query (str): SQL query
1354
1355 -        Returns:
1356 -            str: query results as JSON
1357          """
1352 +        SECURITY PATCH (ACT): Arbitrary SQL execution via Web interface is disabled.
1353
1354
1355 +        # FIX: il DB non viene istanziato.
1356 +        if query:
1357 +            self.log.error(f"SECURITY: Attempted arbitrary SQL execution blocked: {query}")
1358 +        return elf.jsonify_error('403', "Access Denied: Arbitrary SQL execution is disabled for security reasons.")
1359 + """
1358          dbh = SpiderFootDb(self.config)
1359
1360          if not query:
@@ -1370,7 +1372,7 @@ def query(self: 'SpiderFootWebUi', query: str) -> str:
1370              return [dict(zip(columnNames, row)) for row in data]
1371          except Exception as e:
1372              return self.jsonify_error('500', str(e))
1373 -
1375 + """
1374      @cherrypy.expose
1375      def startscan(self: 'SpiderFootWebUi', scanname: str, scantarget: str, modulelist: str, typelist: str, usecase: str) -> str:
1376          """Initiate a scan.
```

sfwebui.py

7. Funcional Testing (Verifica Post-Remediation)

A conclusione delle attività di hardening, è stata rieseguita la pipeline di Functional Testing *(test-functional.yml)* sul branch *fix-vuln-numda*. L'obiettivo è duplice: validare l'efficacia delle patch (Security Verification) e garantire l'assenza di regressioni funzionali (Regression Testing).
Sono state pertanto effettuate le seguenti verifiche:

a. Verifica Web Server

L'applicazione è stata rieseguita con cryptography v42 e pyOpenSSL v24. Il servizio *sf.py* si è avviato correttamente e la verifica HTTP *(curl)* ha restituito codice *"200"*, confermando la stabilità del core.

b. Verifica Operatività CLI

È stata verificata l'operatività di *sfcli.py* e dell'endpoint */scanlist*. L'API restituisce JSON valido, confermando che il database SQLite è accessibile e integro dopo i fix di Input Validation.

c. Verifica SQL Injection (mitigata)

Il probe verso */query* è stato ripetuto con payload *"SELECT 1"*. Il server ha risposto con *"HTTP 403 Forbidden"*, confermando che l'esecuzione SQL arbitraria è ora inibita a livello di codice, neutralizzando il rischio di esfiltrazione dati.

d. Verifica Hardcoded Secrets (risolti)

Lo script *test_citadel_vuln.py* è stato rieseguito monitorando il traffico dal modulo *sfp_citadel.py*. Non è stata intercettata alcuna chiave API, confermando che il modulo ora fallisce in modo sicuro *(Fail-Safe)* senza credenziali utente, eliminando il rischio di Rate Limiting globale.

e. Verifica Build Docker Aggiornato

La ricostruzione dell'immagine Docker su Alpine 3.19 è stata completata con successo. L'aggiornamento ha fornito la toolchain Rust/Cargo necessaria per le nuove librerie di sicurezza, risolvendo l'incompatibilità bloccante di Alpine 3.13.

La pipeline *test-functional.yml* si è conclusa con successo in tutti i job.
L'applicazione mantiene le funzionalità operative originali (no *regressions*) con una postura di sicurezza radicalmente migliorata, avendo azzerato tutte le vulnerabilità classificate come ACT.
Per ulteriori dettagli si rimanda al file *README_test-funcional.md.*[17]

[17] Rif. https://github.com/numdav/spiderfoot/blob/fix-vuln-numda/.github/workflows/README_test-funcional.md .

8. Riesecuzione della pipeline di sicurezza SE4SCS – Analisi post-remediation

La riesecuzione della pipeline SE4SCS sul branch *fix-vuln-numda* ha prodotto 6 nuovi report JSON. L'analisi comparativa certifica l'efficacia delle remediation (fase ACT), evidenziando la necessità di *triage* manuale per i **Falsi Positivi** residui. In particolare:

a. **GitGuardian** *(ggshield.json)*: Nessuna occorrenza rilevata, confermando l'assenza di segreti nei nuovi *commit*.

b. **Semgrep** *(semgrep.json)*: Le due vulnerabilità CRITICAL (SQL Injection in *sfwebui.py* e Hardcoded key in *sfp_citadel.py*) sono state risolte.

c. **Snyk**

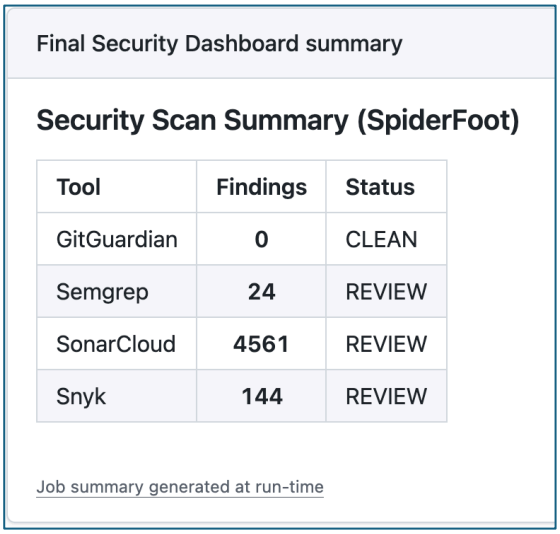
- **Snyk Open Source** *(snyk_deps.json):* L'aggiornamento di *requirements.txt* ha mitigato le criticità sulle dipendenze Python. Le librerie cryptography e pypdf2 sono state aggiornate a versioni sicure, risolvendo le CVE *Denial of Service* precedentemente bloccanti.
- **Snyk Code** *(snyk_code.json):* l'analisi statica ha confermato la rimozione della chiave API hardcoded.
  Il report segnala ancora Path Traversal in *sfcli.py* (CWE-23). La *code review* ha confermato che la vulnerabilità è stata mitigata tramite la funzione *is_safe_path()* (validazione con *os.path.realpath()* e *os.path.commonpath()*). La persistenza della segnalazione è probabilmente dovuta al ruleset di Snyk che non riconosce la mitigazione. Si tratta quindi di un **Falso Positivo**, con il controllo di sicurezza implementato e validato dalla pipeline *test-funcional*.

- **Snyk Container** *(snyk_container.json)*: Migrazione dell'immagine base e ripristino della trasparenza completati con successo.
  . Base Image Aggiornata: Il passaggio ad Alpine 3.19 ha eliminato le vulnerabilità sistemiche critiche della versione EOL 3.13.
  . Ripristino dell'Osservabilità: La rimozione del comando *"rm -rf /lib/apk/db"* ha ripristinato la scansione corretta dei pacchetti OS. Le vulnerabilità ora visibili (es. CVE su BusyBox) sono di severità Low/Medium e correttamente inventariate, eliminando il falso senso di sicurezza (**Falso Negativo**) del report precedente.

d. **SonarCloud:** Oltre 4000 *Code Smells* confermati. Le remediation sono state focalizzate sulla sicurezza e non sulla struttura del codice, prediligendo stabilità rispetto a refactoring stilistico.

La pipeline certifica il passaggio da rischio CRITICO a stato GESTITO. Le vulnerabilità ACT sono state risolte; le segnalazioni residue sono classificate come non bloccanti o **Falsi Positivi**, permettendo il rilascio sicuro dell'applicazione.

Le dashboard seguenti mostrano i risultati PRIMA (*branch master*) e DOPO (*branch fix-vuln-numda*).



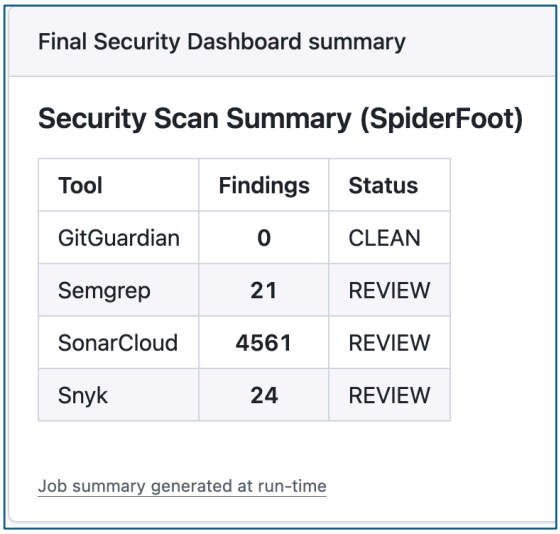
Final Security Dashboard summary

Security Scan Summary (SpiderFoot)

Tool	Findings	Status
GitGuardian	0	CLEAN
Semgrep	24	REVIEW
SonarCloud	4561	REVIEW
Snyk	144	REVIEW

Job summary generated at run-time

Scan Summary – Pre-Remediation



Final Security Dashboard summary

Security Scan Summary (SpiderFoot)

Tool	Findings	Status
GitGuardian	0	CLEAN
Semgrep	21	REVIEW
SonarCloud	4561	REVIEW
Snyk	24	REVIEW

Job summary generated at run-time

Scan Summary – Post-Remediation

9. Conclusioni e Riflessioni

Il progetto ha evidenziato come la sicurezza non sia un modulo aggiuntivo, ma un requisito architetturale integrato – e continuo – nel ciclo di vita del software. Il presente lavoro ha fatto emergere lezioni fondamentali e criticità operative che tendono a rafforzare la necessarietà dell'analisi umana sulla sempre più inevitabile e crescente automazione degli strumenti. In particolare, il lavoro ha evidenziato:

- **Il paradosso dell'automazione:** Il **Falso Negativo** su Snyk Container ha dimostrato che l'automazione è efficace solo se l'ambiente è osservabile. La rimozione di */lib/apk/db* durante la build dell'immagine ha reso il container opaco agli scanner, creando falsa sicurezza. Solo la *code review* ha identificato questa criticità.

- **Limiti della SAST**: La persistenza della segnalazione Path Traversal in *sfcli.py* (Snyk Code) dopo la remediation ha evidenziato i limiti dei motori di *Taint Analysis*. Nonostante l'implementazione di *is_safe_path()*, il tool non ha riconosciuto il contesto del fix. L'output automatico richiede quindi validazione umana per distinguere vulnerabilità critiche da **Falsi Positivi**;
- **Interdipendenza dello Stack Tecnologico**: Il patching non è un'operazione atomica. L'aggiornamento di cryptography in Python ha richiesto prima la migrazione dell'infrastruttura (Alpine 3.13 → 3.19) e poi l'upgrade di pyOpenSSL per evitare crash. Un debito tecnico sull'OS blocca la messa in sicurezza del livello applicativo, confermando che sicurezza infrastrutturale e applicativa sono indissolubilmente legate;
- **Il Valore Strategico di SSVC**: Con oltre 4.000 *Code Smells* e decine di alert infrastrutturali, SSVC è stato essenziale per evitare la paralisi da analisi. Discriminare tra interventi immediati (ACT: Path Traversal, SQL Injection, Hardcoded Secrets) e gestibili nel tempo (TRACK/DEFER: protocolli deprecati, *code smells*) ha permesso di rilasciare un prodotto sicuro senza disperdere risorse in refactoring stilistici;
- **Verifica Funzionale della Sicurezza**: I test custom (es. script in *test-functional.yml*) hanno sancito il passaggio da Detection a Prevention. La pipeline ha verificato attivamente tramite mocking l'eradicazione delle vulnerabilità, garantendo assenza di regressioni funzionali.

In conclusione, il progetto ha trasformato una pipeline CI/CD standard in un gateway di qualità e sicurezza (Quality Gate), dove strumenti automatici e validazione umana collaborano per ridurre la superficie d'attacco in modo sostenibile e scalabile.