Università degli Studi di Milano
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

CORSO DI LAUREA TRIENNALE IN
SICUREZZA DEI SISTEMI E DELLE RETI INFORMATICHE

VERIFICHE DI COMPLIANCE IN AMBIENTI CLOUD

Relatore: Prof. Marco Anisetti
Correlatore: Dott. Antongiacomo Polimeno

Elaborato Finale di:
Niccoló Volontè
Matr. Nr. 20642A

Anno Accademico 2024-2025

Ringraziamenti

Not yet

Indice

Capitolo 1

Introduzione

Negli ultimi anni, il cloud computing ha assunto un ruolo sempre più centrale nel panorama informatico. In particolare, Amazon Web Services (AWS) si è affermato come uno dei principali fornitori globali di servizi cloud, grazie alla sua gamma di strumenti scalabili e altamente configurabili. Tuttavia, con l'aumento della complessità delle architetture distribuite, è diventato essenziale garantire oltre alle proprietà non funzionali come la disponibilità e le prestazioni, anche la sicurezza e l'affidabilità dei sistemi.

La configurazione errata di una risorsa cloud può rappresentare la causa di incidenti di sicurezza. Per questo motivo, negli ambienti AWS, risulta fondamentale implementare strumenti capaci di verificare automaticamente la corretta configurazione delle risorse, in base a standard di sicurezza dedicati.

All'interno di questo elaborato si propone lo sviluppo di una suite di *sonde di assurance*, ovvero strumenti capaci di eseguire controlli automatizzabili sulla configurazione dei servizi AWS, valutandone la compliance rispetto a benchmark riconosciuti, come il *CIS AWS Foundations Benchmark*, il *CIS Amazon EKS Benchmark* e le raccomandazioni *NIST SP 800-53*. Le sonde sono state integrate in MoonCloud, un framework per la gestione e l'analisi dell'assurance della sicurezza in ambienti ICT.

Il lavoro si articola in più fasi, ciascuna descritta nei capitoli successivi.

Nel **Capitolo 2**, viene presentato lo *stato dell'arte*, con una descrizione dei termini *assurance* e *compliance*, una panoramica dei principali standard di sicurezza, delle best practice per la protezione in ambienti cloud e dei benchmark utilizzati come riferimento per la verifica della conformità.

Nel **Capitolo 3**, si descrivono le *tecnologie utilizzate*, tra cui Python, Docker, GitLab e MoonCloud, analizzando le motivazioni delle scelte tecniche e l'architettura del framework su cui si basano le sonde.

Il **Capitolo 4** è dedicato al *caso di studio*: viene illustrata in dettaglio la progettazione e l'implementazione delle sonde sviluppate, riportando ogni controllo implementato e le relative funzionalità. Inoltre, si tratta la loro integrazione nella piattaforma MoonCloud.

Infine, nel **Capitolo 5**, vengono tratte le *conclusioni*, accompagnate da una riflessione sugli sviluppi futuri: tra questi, l'estensione della compliance a servizi aggiuntivi, l'estensione a nuovi benchmark, l'integrazione con ulteriori cloud provider.

Capitolo 2

Stato dell'arte

Questo capitolo è dedicato ad una panoramica dei servizi AWS e delle best practice di sicurezza associate. Si analizza anche il significato di assurance nel contesto del cloud computing; in aggiunta un focus sul Center for Internet Security (CIS) e sul suo benchmark per AWS.

2.1 Assurance e compliance

Nel contesto del cloud computing è sempre più importante assicurarsi che infrastrutture e applicazioni si comportino in modo corretto, anche in presenza di guasti o attacchi. Questo principio viene chiamato **assurance** e indica la capacità di garantire, in modo continuo, che i sistemi rispettino determinati requisiti di sicurezza. Non si tratta solo di definire la sicurezza a livello teorico, ma anche di mettere in pratica strumenti e metodi che permettano di raccogliere e verificare prove concrete dell'efficacia delle misure adottate.

Il cloud è un ambiente dinamico: i servizi vengono aggiornati frequentemente, le macchine virtuali possono essere spostate, e le configurazioni cambiano nel tempo. Tutto questo rende instabile la base su cui si costruiscono le misure di sicurezza. Per questo motivo, l'assurance deve essere continua e flessibile. Non è più sufficiente effettuare controlli una sola volta: è necessario monitorare costantemente il comportamento dei sistemi.

Un altro aspetto fondamentale per rafforzare la fiducia degli utenti è la **trasparenza**: devono poter accedere a informazioni e prove che dimostrino che il servizio è effettivamente sicuro. Per ottenere questo risultato, si utilizzano tecniche come il monitoraggio, il testing e la certificazione, che migliorano la visibilità e il controllo, sia per il provider che per il cliente [1].

Infine, l'assurance è anche la base per la *compliance*, cioè la conformità a normative, leggi e regolamenti. La compliance implica l'applicazione concreta delle regole definite dalle regolamentazioni, che spesso mirano a garantire attributi come la riservatezza, l'integrità, la disponibilità e la responsabilità dei dati. A causa della natura stessa del cloud, la

compliance è una responsabilità condivisa tra provider, clienti e auditor, cioè soggetti incaricati di verificare la conformità ai requisiti normativi. Tuttavia, molte regolamentazioni sono lunghe, ridondanti e difficili da interpretare, rendendo complicata la loro applicazione nei sistemi cloud. Inoltre, la mancanza di architetture di riferimento standard rende difficile per le aziende valutare il livello reale di conformità dei provider. Per questo motivo, è importante che essa venga considerata già a livello architetturale, così da facilitare la valutazione e la selezione di servizi cloud conformi da parte dei clienti [2].

2.2 Amazon Web Services

Amazon Web Services (AWS) è una piattaforma di cloud computing che offre una vasta gamma di servizi, tra cui calcolo, archiviazione, database e networking. I sistemi AWS sono progettati per essere altamente scalabili e flessibili, consentendo alle aziende di adattare le risorse in base alle proprie esigenze. Ad oggi AWS è il fornitore di servizi cloud più diffuso al mondo e copre circa il 31% del mercato globale. Questo dato risulta stabile nel tempo, a testimonianza della solidità dell'infrastruttura AWS e della fiducia del mercato verso l'azienda. Insieme ad Azure e Google Cloud, AWS contribuisce a controllare oltre i due terzi del mercato globale, testimoniando la predominanza di questi pochi fornitori nel mercato cloud [3].

Questo cloud provider offre una varietà di servizi, ed ognuno ha le proprie caratteristiche e funzionalità specifiche, ma tutti condividono un'architettura comune che consente agli utenti di accedere e gestire le risorse in modo semplice ed efficiente.

AWS offre i 3 modelli di servizio tipici del cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) e Software as a Service (SaaS).

IaaS consente agli utenti di noleggiare risorse hardware virtuali, come server e storage, senza dover gestire fisicamente l'infrastruttura: è particolarmente utile per le aziende che desiderano ridurre i costi e semplificare la gestione dell'infrastruttura IT. Con IaaS, gli utenti possono scalare rapidamente le risorse in base alle esigenze, senza dover investire in hardware costoso. PaaS è ideale per gli sviluppatori che desiderano concentrarsi sulla creazione di applicazioni senza doversi preoccupare della gestione dell'infrastruttura sottostante, mettendo a disposizione un ambiente di sviluppo completo. SaaS è perfetto per le aziende che desiderano accedere a software e applicazioni pronte all'uso via Internet, senza dover installare e gestire applicazioni in locale [4].

Un esempio di servizio IaaS è Amazon EC2, che consente agli utenti di noleggiare server virtuali per eseguire applicazioni su macchine virtuali. I servizi PaaS includono AWS RDS, un database relazionale per l'archiviazione di grandi moli di dati. Infine, i servizi SaaS di AWS comprendono AWS Lambda Functions, che consentono di eseguire codice senza dover gestire l'infrastruttura sottostante.

Visti gli innumerevoli servizi offerti da AWS, è fondamentale comprendere le best practice di sicurezza associate a ciascun servizio. AWS Security Hub è una risorsa offerta da

AWS che consente agli utenti di monitorare e gestire la sicurezza delle proprie risorse. Security Hub aggrega i dati di sicurezza provenienti da diversi servizi AWS e fornisce una panoramica della sicurezza dell'account. Questo servizio si basa su una serie di standard di sicurezza, tra cui Payment Card Industry Data Security Standard (PCI DSS), National Institute of Standards and Technology (NIST) e Center for Internet Security (CIS) Foundations Benchmark.

2.3 Center for Internet Security

Il *Center for Internet Security* (CIS) è un'organizzazione no-profit riconosciuta a livello mondiale per la sua mission di migliorare la sicurezza informatica. Il CIS sviluppa e pubblica benchmark di sicurezza per vari sistemi e applicazioni, tra cui AWS. La verifica di conformità tramite questi benchmark garantisce conformità alle best practice stilate da ricercatori ed esperti del settore, adattandosi ad un panorama, quello della sicurezza informatica, in cui le minacce cambiano frequentemente. Inoltre l'automatizzazione di queste verifiche di conformità permette di ridurre al minimo la possibilità di errore umano nell'ambito di configurazioni errate, segnalando in tempo reale quali istanze siano potenzialmente a rischio [5].

Il CIS AWS Foundations Benchmark è un insieme di **best practice** progettate per aiutare le organizzazioni a configurare e gestire in modo sicuro le proprie risorse AWS. Questo benchmark fornisce linee guida dettagliate su come proteggere le istanze computazionali, i servizi di archiviazione, la gestione degli accessi e altri aspetti critici della sicurezza cloud.

Il documento è suddiviso in diverse **sezioni**, ognuna delle quali affronta un aspetto specifico della sicurezza AWS: Identity and Access Management, Storage, Logging, Monitoring e Networking. In ognuna di queste sezioni, il benchmark fornisce raccomandazioni dettagliate su ogni particolare servizio AWS ritenuto importante da rendere compliant e una serie di controlli di sicurezza da implementare descritti in maniera verbosa, con relative modalità di verifica e implementazione. In particolare, la struttura di ogni controllo include una descrizione del controllo, eventuali note sulla gestione delle eccezioni, il razionale del controllo, la modalità di esecuzione del controllo (sia da command-line che da console) e da ultimo le azioni da intraprendere in caso di non conformità.

AWS Security Hub supporta il **CIS AWS Foundations Benchmark v3**, che include 37 controlli di sicurezza suddivisi in 12 categorie. Nulla documentazione AWS ogni controllo è descritto in dettaglio, con indicazioni discorsive su come implementarlo e verificarne la conformità, con le spiegazioni necessarie per comprendere il motivo per cui è importante rispettare quel controllo, la severità e l'intervallo per la programmazione temporale del controllo. Se necessari, sono descritti anche parametri da configurare in input per la verifica di conformità. Inoltre, il benchmark suggerisce anche azioni da compiere in caso di

non conformità, come ad esempio la modifica delle configurazioni o l'implementazione di misure di sicurezza aggiuntive [6].

Di seguito si riporta un esempio di controllo tratto dal documento CIS AWS Foundations Benchmark v3:

1.4 Ensure MFA is enabled for the 'root' user account (Automated)

- ***Profile Applicability:*** *Level 1*
- ***Description****: The 'root' user account is the most privileged user in an AWS account. Multi-factor Authentication (MFA) adds an extra layer of protection on top of a username and password. With MFA enabled, when a user signs in to an AWS website, they will be prompted for their username and password as well as for an authentication code from their AWS MFA device.*
- ***Note****: When virtual MFA is used for 'root' accounts, it is recommended that the device used is NOT a personal device, but rather a dedicated mobile device (tablet or phone) that is kept charged and secured, independent of any individual personal devices ("non-personal virtual MFA"). This lessens the risks of losing access to the MFA due to device loss, device trade-in, or if the individual owning the device is no longer employed at the company.*

  Where an AWS Organization is using centralized root access, root credentials can be removed from member accounts. In that case it is neither possible nor necessary to configure root MFA in the member account.
- ***Rationale****: Enabling MFA provides increased security for console access as it requires the authenticating principal to possess a device that emits a time-sensitive key and have knowledge of a credential.*
- ***Audit****: Perform the following to determine if the 'root' user account is enabled and has MFA setup:*

  From Console:

  1. *Login to the AWS Management Console*
  2. *Click* `Services`
  3. *Click* `IAM`
  4. *Click on* `Credential Report`
  5. *This will download a .csv file which contains credential usage for all IAM users within an AWS Account - open this file*

6. *For the* `<root_account>` *user, ensure the* `mfa_active` *field is set to* `TRUE` *or the* `password_enabled` *field is set to* `FALSE`

From Command Line:

1. *Run the following command:*

```
aws iam get-account-summary | grep "AccountMFAEnabled"
aws iam get-account-summary | grep "AccountPasswordPresent"
```

2. *Ensure the* `AccountMFAEnabled` *property is set to 1 or the* `AccountPasswordPresent` *property is set to 0.*

- ***Remediation****: To manage MFA devices for the 'root' AWS account, you must use your 'root' account credentials to sign in to AWS. You cannot manage MFA devices for the 'root' account using other credentials. Perform the following to establish MFA for the 'root' user account:*

...

[1]

Il controllo sopra riportato è un esempio rappresentativo di come il benchmark *CIS AWS Foundations Benchmark v3* definisca i controlli di sicurezza in modo dettagliato e sistematico. Ogni controllo è concepito per essere comprensibile anche da team tecnici non specializzati in sicurezza e, allo stesso tempo, sufficientemente rigoroso da rispondere a requisiti di conformità industriali o normativi.

I controlli sono classificati in due livelli di severità (**Level 1** e **Level 2**), che rappresentano rispettivamente un compromesso tra facilità di implementazione e livello di protezione offerto. I controlli di **Level 1** forniscono una baseline di sicurezza consigliata per la maggior parte degli ambienti, mentre quelli di **Level 2** sono più restrittivi e adatti a contesti ad alto rischio o soggetti a regolamentazioni specifiche.

Il benchmark è regolarmente aggiornato dal CIS per rispecchiare l'evoluzione del cloud computing e delle minacce emergenti, con una comunità attiva che include esperti di sicurezza, provider cloud (inclusa AWS stessa) e organizzazioni del settore pubblico e privato. La versione 3, supportata da *AWS Security Hub*, è attualmente una delle più diffuse per la valutazione della sicurezza nel cloud AWS.

Oltre al benchmark generale per AWS, il CIS fornisce anche benchmark specifici per servizi AWS come EC2, RDS, S3, EKS. Questi si specializzano su aspetti più vasti e dettagliati della sicurezza di ogni tipo di servizio, permettendo alle organizzazioni di applicare controlli di sicurezza specifici per le risorse AWS che utilizzano.

[1] Contenuto tratto da CIS Amazon Web Services Foundations Benchmark v3, sezione 1.4, disponibile su `https://www.cisecurity.org/benchmark/amazon_web_services/`.

L'approccio strutturato e trasparente del CIS permette alle organizzazioni di adottare un framework solido per la gestione della sicurezza nel cloud, migliorando non solo la postura difensiva, ma anche l'aderenza a standard e normative riconosciuti a livello internazionale.

Nei capitoli successivi si esploreranno in dettaglio come implementare questi controlli di sicurezza in ambienti AWS, utilizzando strumenti che ne facilitano l'applicazione e la verifica.

2.4 National Institute of Standards and Technology

Il *National Institute of Standards and Technology* (NIST) è un'agenzia governativa degli Stati Uniti che si occupa dello sviluppo di standard, linee guida e best practice in diversi settori strategici tra cui le comunicazioni, la chimica, l'energetica e, in particolare, la sicurezza informatica.

Nel contesto della cybersecurity, il NIST fornisce un contributo fondamentale attraverso la definizione di modelli, controlli e raccomandazioni che coprono un ampio spettro di ambiti: dalla crittografia alla gestione del rischio, dalla sicurezza delle reti alla protezione dei dati sensibili e delle identità digitali.

L'approccio adottato è basato su standard aperti e best practice consolidate, con l'obiettivo di supportare le organizzazioni pubbliche e private nel garantire l'affidabilità, la resilienza e la sicurezza dei propri sistemi informativi.

Tra i principali contributi del NIST in ambito di sicurezza informatica si distinguono il **NIST Cybersecurity Framework (CSF)** e la serie delle **Special Publications (SP)**, in particolare la collana **SP 800**, che offre linee guida dettagliate per l'implementazione di controlli di sicurezza in diversi contesti organizzativi e tecnologici.

In particolare la serie **Special Publication 800 (SP 800)** costituisce una raccolta tecnica molto ampia di documenti che approfondiscono vari aspetti della sicurezza informatica e della gestione del rischio. Le SP 800 sono destinate a organizzazioni federali ma adottate ampiamente anche dal settore privato. Ogni pubblicazione affronta un tema specifico e fornisce indicazioni operative, architetturali e normative.

Tra i documenti più rilevanti della collana SP 800 si possono citare:

- **SP 800-53**: fornisce un catalogo completo di controlli di sicurezza e privacy per i sistemi informativi, organizzati per famiglie (es. Access Control, Audit, Identification and Authentication).

- **SP 800-171**: specifica requisiti di sicurezza per la protezione di informazioni controllate non classificate (*Controlled Unclassified Information, CUI*) nei sistemi e nelle organizzazioni non federali.

- **SP 800-30**: tratta la gestione del rischio attraverso metodologie per la valutazione dei rischi di sicurezza dei sistemi informatici.
- **SP 800-61**: fornisce linee guida per la gestione degli incidenti di sicurezza informatica.
- **SP 800-207**: introduce il modello di sicurezza *Zero Trust Architecture*, in cui nessun attore è automaticamente considerato affidabile, nemmeno all'interno del perimetro aziendale.

[2]

Questi documenti, insieme al CSF, costituiscono un riferimento per la progettazione, l'implementazione e la valutazione di misure di sicurezza nei moderni sistemi informatici, sia in ambito pubblico che privato.

2.4.1 SP 800-53 Rev. 5

Tra i documenti più rilevanti della serie SP 800, la **SP 800-53 Rev. 5** propone un framework dettagliato di controlli di sicurezza e privacy per i sistemi informativi, suddivisi in famiglie tematiche. Una delle famiglie centrali è quella relativa al **Controllo degli Accessi (Access Control, AC)**, che definisce misure per la gestione, la limitazione e l'enforcement degli accessi a dati e risorse [7].

Questa famiglia è particolarmente importante nei contesti cloud e distribuiti, dove l'accesso alle risorse deve essere regolato con precisione attraverso politiche automatizzate, monitorate e basate sul principio del minimo privilegio.

I principali controlli della famiglia `AC` includono:

- **AC-2 - Account Management**: riguarda la gestione formale degli account utente, compresa la creazione, la modifica, la disattivazione e la revoca degli account. Promuove un controllo accurato sui soggetti che accedono al sistema.
- **AC-3 - Access Enforcement**: definisce l'obbligo di applicare i meccanismi di controllo degli accessi definiti dalle politiche, in modo coerente e automatizzato, per limitare l'accesso solo a soggetti autorizzati.
- **AC-5 - Separation of Duties**: prescrive la separazione dei compiti tra più soggetti per prevenire conflitti di interesse e ridurre i rischi legati all'abuso di privilegi.
- **AC-6 - Least Privilege**: afferma che ciascun utente o processo deve disporre solo dei privilegi necessari per svolgere le proprie funzioni, evitando l'attribuzione indiscriminata di permessi elevati.

[2] Contenuto tratto da `https://www.nist.gov/cyberframework` e `https://csrc.nist.gov/publications/sp800`.

[3]

Questi controlli rappresentano un fondamento per l'implementazione di politiche di sicurezza efficaci nei sistemi informativi moderni, in particolare in ambienti ad alta scalabilità e automazione. Il loro obiettivo è quello di garantire che l'accesso alle risorse sia sempre autorizzato, monitorato e proporzionato, in conformità ai principi di sicurezza "zero trust" e gestione del rischio.

Essi saranno ripresi nei capitoli successivi per mostrare come possano essere implementati concretamente in ambienti reali, attraverso strumenti che garantiscano l'applicazione corretta per la compliance delle risorse AWS.

[3] Contenuto tratto da `https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final`.

Capitolo 3

Tecnologie utilizzate

Questo capitolo è dedicato alla descrizione delle tecnologie e degli strumenti utilizzati per svolgere il progetto di tesi, in particolare per lo sviluppo delle sonde di assurance: delle applicazioni che tramite il framework MoonCloud verificano la conformità di servizi e infrastrutture ICT in base a benchmark di sicurezza e best practice. Si analizzano i linguaggi di programmazione, le librerie e i framework adottati, nonché le metodologie di sviluppo seguite.

3.1 Python

Python è un linguaggio di programmazione ad alto livello, interpretato e orientato agli oggetti. È ampiamente utilizzato per lo sviluppo di applicazioni web, automazione, analisi dei dati e machine learning. Durante lo sviluppo delle sonde, Python è stato scelto per la sua versatilità e compatibilità con molteplici librerie, tra cui quella per interfacciarsi con AWS.

3.1.1 Boto3

Boto3 è la libreria ufficiale di AWS per Python, che consente di interagire con i servizi AWS in modo semplice ed efficiente. Boto3 fornisce un'interfaccia intuitiva per accedere alle API di AWS e gestire le risorse cloud. Permette di creare un client per ciascun servizio AWS e di eseguire operazioni come la creazione, la modifica e l'eliminazione di risorse, nonostante per la verifica di conformità ai benchmark CIS siano sufficienti permessi di sola lettura. Per accedere a un servizio AWS, è necessario fornire le credenziali di accesso (*Access Key* e *Secret Key*) e la regione in cui si desidera operare. Un esempio di utilizzo di Boto3 è la creazione di un client per Amazon S3 ed elencare tutti i buckets presenti nella regione specificata sul determinato account.

```
1 import boto3
```

```
2
3  s3 = boto3.client (
4  's3',
5  aws_access_key_id="YOUR_ACCESS_KEY",
6  aws_secret_access_key="YOUR_SECRET_KEY",
7  region_name="YOUR_REGION"
8  )
9
10 s3.list_buckets()
```

Listing 3.1: Esempio di utilizzo di Boto3 per elencare i bucket S3

3.2 Docker

Docker è una piattaforma di containerizzazione che consente di creare, distribuire ed eseguire applicazioni in ambienti isolati chiamati container. Permette inoltre di generare **immagini** di applicazioni e servizi, eseguibili su qualsiasi sistema che supporti Docker, garantendo coerenza tra gli ambienti di sviluppo, test e produzione. Le immagini contengono tutto il necessario per far funzionare un'applicazione: codice, librerie, dipendenze e configurazioni. Durante lo sviluppo delle sonde di assurance, Docker è stato utilizzato per creare le immagini delle applicazioni, che vengono poi eseguite su MoonCloud. Ogni immagine Docker viene costruita a partire da un file chiamato **Dockerfile**, che descrive in modo dettagliato tutti i passaggi necessari a configurare l'ambiente di esecuzione. In questo modo, una volta scritta una sonda e generata l'immagine corrispondente, è possibile eseguirla su MoonCloud direttamente, senza ulteriori modifiche.

3.3 GitLab

GitLab è una piattaforma di gestione del codice sorgente che consente di ospitare repository Git, gestire progetti e ospitare pipeline di **Continuous Integration/Continuous Deployment** (CI/CD). Il framework MoonCloud ospita un proprio deployment di GitLab, che consente di gestire il codice sorgente delle sonde di assurance e di automatizzare il processo di build dell'immagine Docker. Le pipeline CI/CD sono configurate per eseguire automaticamente i test e costruire l'immagine Docker della sonda, rendendola disponibile per l'esecuzione su MoonCloud. In questo modo, ogni volta che viene apportata una modifica al codice sorgente della sonda, la pipeline si occupa di ricostruire l'immagine e renderla disponibile per l'esecuzione.

Ogni sonda prodotta viene inserita in un repository GitLab dedicato, che contiene il codice sorgente, i file necessari per la configurazione della sonda e le pipeline CI/CD, il README con la documentazione della sonda.

3.4 MoonCloud

MoonCloud [8] è un framework per la valutazione di conformità e assurance di applicazioni o infrastrutture ICT che permette di avere una verifica completa dei servizi durante il loro ciclo di vita. Si concentra prevalentemente su security, AI e machine learning assurance.

Le sonde di assurance sono sviluppate per essere eseguite su MoonCloud, che fornisce un **driver** che defisce struttura e funzionalità delle sonde. Si tratta di una macchina a stati finiti con due principali flow di esecuzione: **forward** e **rollback**. Il flow forward è il flusso principale di esecuzione della sonda, in cui vengono eseguiti i controlli di conformità e le azioni correttive. Il flow rollback viene eseguito in caso di errore o fallimento del flow forward, e consente di ripristinare lo stato precedente della sonda in caso il target sia stato modificato. La sonda prende in input dei parametri relativi al target di esecuzione e restituisce in output un risultato predefinito definito secondo un preciso schema arricchito poi dal programmatore.

Il ciclo di vita di una sonda su MoonCloud inizia con la scrittura del codice sorgente: esso viene poi inserito in un repository Git che si occupa, tramite una pipeline CI/CD, di costruire l'immagine Docker della sonda e di renderla disponibile per l'esecuzione. MoonCloud dispone di un **backend** che gestisce l'interazione tra la sonda e la dashboard, ovvero l'interfaccia utente che consente di configurare, gestire, eseguire le sonde. In questo modo, è possibile creare un **form** tramite JSON schema per la configurazione di ciò che la sonda dovrebbe ricevere in input quando eseguita sulla dashboard. Si differenziano **Control** e **Abstract Evaluation Rule**; il primo si occupa di definire:

- **Nome** della sonda;
- **Driver** della sonda, ovvero il link GitLab che identifica l'immagine Docker della sonda;
- **Metadata**, ovvero il form che definisce i parametri di input della sonda secondo un JSON schema;
- **Category** della sonda;
- **Supported Interval**, ovvero la frequenza con cui la sonda può essere eseguita;
- **Cred Type**: diverse sonde possono richiedere credenziali di accesso differenti, dalla più semplice coppia *username-password* a chiavi SSH o Git login;

In questo modo si definisce la sonda e le sue caratteristiche, per renderla disponibile per l'esecuzione. L'Abstract Evaluation Rule, invece, si occupa di presentare la sonda all'interno della dashboard, definendo:

- **Nome** della sonda;
- **Immagine** della sonda, ovvero il logo che appare nella dashboard;
- **Descrizione**
- **Category** della sonda;
- **Formula**, ovvero il tag numerico progressivo del controllo che identifica la sonda;
- **Related Controls**, ovvero i controlli prima definiti che sono correlati a questa sonda;
- **Owner** della sonda;
- **isPublic**, ovvero se la sonda è pubblica o privata;

Una volta che la sonda è stata definita nel backend e l'immagine Docker è stata costruita, è possibile eseguirla sulla dashboard di MoonCloud. La dashboard consente di creare zone e target di esecuzione, ovvero le istanze o i servizi su cui si desidera eseguire delle sonde; definire credenziali di accesso per i target e da ultimo aggiungere ed eseguire **evaluations**, ovvero le sonde di assurance. I risultati sono mostrati sia tramite sommario con grafici e percentuali di conformità, sia tramite un log dettagliato che mostra l'esito di ogni singola sonda.

Capitolo 4

Caso di studio

Questo capitolo è dedicato alla descrizione del caso di studio scelto per lo sviluppo delle sonde di assurance. Si analizzano le modalità di codifica e implementazione delle 13 sonde secondo i 51 controlli del benchmark CIS AWS Foundations Benchmark v3, di NIST SP 800-53 e di CIS EKS Benchmark. Inoltre, si descrivono le funzionalità offerte da ciascuna sonda.

4.1 Descrizione

Lo sviluppo delle sonde di assurance per AWS è stato organizzato in più fasi. L'obiettivo principale era quello di creare sonde in grado di verificare automaticamente se risorse e servizi AWS rispettando i 51 controlli di sicurezza definiti nei benchmark in questione.

Le sonde sono state progettate per essere eseguite all'interno di *MoonCloud*, un framework che si occupa di gestire l'intero ciclo di vita delle sonde: dalla configurazione all'esecuzione, fino alla visualizzazione dei risultati.

Analisi e progettazione Per ogni sonda è stato necessario prima di tutto comprendere i controlli di sicurezza descritti nei benchmark. Essendo documenti di tipo descrittivo, è stato fondamentale tradurre ogni controllo in logica eseguibile, individuando:

- il servizio AWS coinvolto (ad esempio IAM, EC2, S3, ecc.),
- il tipo di verifica da effettuare (ad esempio presenza di MFA, assenza di accesso pubblico, rotazione delle chiavi),
- le API di AWS necessarie, tramite la libreria `boto3` in Python.

Una volta chiarita la logica, si è passati alla progettazione vera e propria della sonda. Ogni sonda ha una struttura comune: il file `probe.py` contiene la logica di esecuzione,

mentre file come `schema.json` e `test.json` servono per la gestione dell'input. La sonda viene poi containerizzata tramite Docker e integrata con MoonCloud.

Listing 4.1: Struttura di ogni sonda per MoonCloud

```
aws_sonda/
|-- probe/
|   |-- probe.py
|   |-- test.json
|   |-- schema.json
|-- .dockerignore
|-- .gitlab-ci.yml
|-- .gitignore
|-- requirements.txt
|-- Dockerfile
|-- README.md
```

Testing e verifica Durante la fase di sviluppo, le sonde sono state testate in locale usando delle credenziali AWS di test, salvate in un file `credential.json` (escluso dal repository tramite `.gitignore`).

L'obiettivo dei test era verificare che:

- i controlli restituissero i risultati attesi,
- le eventuali eccezioni venissero gestite in modo chiaro,
- la struttura dell'output fosse compatibile con quella richiesta da MoonCloud.

Integrazione su MoonCloud Una volta completata e testata, ogni sonda viene integrata nel framework MoonCloud. Il codice sorgente viene salvato su GitLab, e una pipeline CI/CD si occupa di creare automaticamente l'immagine Docker della sonda.

MoonCloud permette poi di configurare ed eseguire la sonda tramite una dashboard web. Per ogni sonda vengono definite:

- un *Control*, che specifica il driver, i metadati di input, il tipo di credenziali richieste e altre informazioni tecniche;
- una *Abstract Evaluation Rule*, che definisce il nome, la descrizione, l'immagine visibile in dashboard, e i controlli a cui la sonda è collegata.

In questo modo è possibile selezionare facilmente la sonda dalla dashboard, eseguirla su un account AWS target e visualizzare i risultati in forma aggregata oppure dettagliata.

4.2 Implementazione

Dapprima è necessario scrivere tutti i file necessari alla pipeline CI/CD per costruire l'immagine della sonda: questi file sono comuni e identici per ogni sonda AWS codificata.

.dockerignore e .gitignore Questi file sono identici e servono a escludere file e cartelle specifiche dal repository Git e dalla build dell'immagine Docker. In questo caso, è stato escluso il virtual environment creato per lo sviluppo della sonda, i file temporanei come `__pycache__` e i file di configurazione dell'IDE o relativi all'OS in uso come `.DS_Store` o `.idea`.

Particolare attenzione è l'inclusione di `credential.json` in `.gitignore`, in modo da evitare di caricare le credenziali di accesso al repository Git.

.gitlab-ci.yml Questo file definisce la pipeline CI/CD per creare l'immagine Docker della sonda quando si effettua un push sul repository Git. Questo file standard è fornito dallo sviluppatore del driver ed è necessario per il corretto funzionamento della pipeline. Viene incluso in quanto parte integrante del processo di integrazione continua previsto dal progetto. Di seguito è riportato il contenuto del file `.gitlab-ci.yml`:

```
1  stages:
2  - build
3
4  build_probe:
5  stage: build
6  variables:
7      CONTAINER_RELEASE_IMAGE: $CI_REGISTRY_IMAGE:latest
8
9  image: quay.io/podman/stable:latest
10
11 before_script:
12     - podman login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
13 script:
14     - podman build -t $CONTAINER_RELEASE_IMAGE --build-arg GITLAB_TOKEN_USER=
   gitlab-ci-token --build-arg GITLAB_TOKEN=$CI_JOB_TOKEN .
15     - podman push $CONTAINER_RELEASE_IMAGE
16 interruptible: true
17 artifacts:
18     expire_in: 30 days
```

Listing 4.2: File `.gitlab-ci.yml` per la definizione della pipeline CI/CD

Il file definisce una pipeline automatizzata che viene attivata ad ogni push sul repository. La struttura del file si articola in diverse sezioni principali. La sezione **stages** dichiara

le fasi della pipeline, in questo caso una singola fase di build dedicata alla costruzione dell'immagine Docker. Il job **build_probe** viene associato a questa fase in cui si associa un'immagine come ambiente di esecuzione. Le variabili d'ambiente vengono configurate attraverso la sezione **variables**, dove `CONTAINER_RELEASE_IMAGE` definisce il nome e il tag dell'immagine finale della sonda. La sezione **before_script** esegue l'autenticazione al registry di GitLab utilizzando le credenziali del job corrente. La parte principale della pipeline risiede nella sezione **script**, che esegue in sequenza due operazioni: la costruzione dell'immagine tramite Docker passando i token di autenticazione come argomenti, e successivamente il push dell'immagine nel registry. La configurazione **interruptible** con valore `true` permette l'interruzione del job se necessario, mentre la sezione **artifacts** definisce un periodo di conservazione di 30 giorni per eventuali file generati durante l'esecuzione.

Dockerfile Questo file definisce come viene creata l'immagine Docker di ogni sonda. Anche in questo caso il file è standard e fornito dallo sviluppatore del driver, pertanto scelte di design e implementazione sono state fatte in fase di sviluppo del driver stesso. Di seguito è riportato il contenuto del file `Dockerfile`:

```
FROM python:3.10

ARG GITLAB_TOKEN_USER
ARG GITLAB_TOKEN

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app
ADD requirements.txt /usr/src/app

RUN sed -i -e 's/git+https:\/\/repository.v2.moon-cloud.eu
\/dev\/driver.git/git+https:\/\/${GITLAB_TOKEN_USER}:
${GITLAB_TOKEN}@repository.v2.moon-cloud.eu
\/dev\/driver.git/' requirements.txt


RUN pip install --no-cache-dir -r requirements.txt
ADD probe/schema.json /etc/probe/schema.json
ADD . /usr/src/app
RUN chmod +x /usr/src/app/probe/probe.py

ENTRYPOINT [ "/bin/bash", "-c" ]
CMD ["/usr/src/app/probe/probe.py"]
```

Listing 4.3: `Dockerfile` per la creazione dell'immagine della sonda

Il Dockerfile si struttura in diverse fasi che definiscono il processo di creazione dell'immagine. La fase iniziale stabilisce l'**ambiente** di base utilizzando l'immagine `python:3.10` e definisce gli argomenti `GITLAB_TOKEN_USER` e `GITLAB_TOKEN` che vengono passati alla pipeline di CI/CD per l'autenticazione. Creando la **directory** `/usr/src/app` si imposta come directory di lavoro. Il file `requirements.txt` viene copiato per primo nella directory per installare le dipendenze necessarie per la sonda. Successivamente si modifica il file `requirements.txt` inserendo le credenziali di **autenticazione** per il driver MoonCloud, per poi installare le **dipendenze** elencate nello stesso. La fase finale del build prevede la **copia** dei file necessari all'esecuzione: tutti i file sorgente vengono destinati alla directory di lavoro. Da ultimo il file `probe.py` viene reso **eseguibile** e viene configurato come entry point del container, garantendo che la sonda si avvii automaticamente all'esecuzione del container.

requirements.txt Questo file elenca le dipendenze Python necessarie per eseguire la sonda. Le uniche necessarie per le sonde AWS sono `boto3`, `requests` e il driver di MoonCloud. Di seguito è riportato il contenuto del file `requirements.txt`:

Listing 4.4: File `requirements.txt` per le dipendenze della sonda

```
git+https://repository.v2.moon-cloud.eu/dev/driver.git
requests==2.32.3
boto3==1.37.14
```

README.md Questo file contiene informazioni generali sulla sonda, schema di input e output, form per la validazione su MoonCloud e azioni AWS necessarie per l'esecuzione della sonda. Di seguito è riportato un esempio del contenuto del file `README.md`, che varia a seconda della sonda:

Listing 4.5: File `README.md` per la sonda AWS SQS

```
# AWS SQS

This probe is used to verify the compliance of AWS ...
with CIS Foundations Benchmark v3 for AWS. It uses [Boto3]
(https://boto3.amazonaws.com/v1/documentation/api/latest/
index.html) to interact with AWS ...

## Required AWS actions
- ‘instance:action1‘
- ‘instance:action2‘
```

````
## Input

```json5
{
    "config": {
        "aws_access_key_id": "your_access_key",
        "aws_secret_access_key": "your_secret_key",
        "region": "your_region",
        ...
    }
}
```

## Output

```json5
{
{
    "integer_result": 1,
    "pretty_result": "The probe executed successfully!",
    "extra_data": {
        "refined": {
            "cve": []
        },
        "raw": {
            "Summary": "2/3 succeeded",
            "CIS ..."{

            }
            // and so on
        }

}
}
```

## MoonCloud Form

```json5
{
"target":{
"section":"config",
````
````

```
"type":"host"
},
"description":"AWS ... Parameters",
"schema":{
"config":{
    "properties":{
    "region":{
        "title":"AWS Region",
        "x-form":{
        "form_type":"input-fix-label"
        },
        "x-schema-form":{
        "type":"text"
        }
    }
    }
}
}
}
‘‘‘
```

test.json Questo file contiene un esempio di input per la sonda, che viene utilizzato per testare la sonda durante la creazione dell'immagine. Il file contiene le credenziali di accesso e la regione in cui si desidera operare. In fase di testing esiste invece un file con questa stessa struttura, `credential.json`, che contiene le credenziali effettive di accesso all'account AWS. Come detto prima, questo file viene escluso dal repository essendo incluso in `.gitignore`. Di seguito è riportato un esempio del contenuto del file `test.json`:

```
{
    "config": {
        "aws_access_key_id": "your_access_key",
        "aws_secret_access_key": "your_secret_key",
        "region": "your_region"
    }
}
```

schema.json Questo file definisce lo schema del file `test.json` per validarne la correttezza. Il file contiene i tipi di dati e le proprietà richieste per ciascun campo. Di seguito è riportato un esempio del contenuto del file `schema.json`:

```
{
"type": "object",
```

```
"properties": {
    "config" : {
        "type": "object",
        "properties": {
            "aws_access_key_id": {
                "type": "string",
                "description": "AWS access key ID"
            },
            "aws_secret_access_key": {
                "type": "string",
                "description": "AWS secret access key"
            },
            "region": {
                "type": "string",
                "description": "AWS region"
            }
            // other input parameters
        }
    }
}
}
```

probe.py Questo file implementa la logica della sonda e tutti i controlli di conformità per un dato target. Esistono delle caratteristiche strutturali comuni a tutte le sonde AWS: di seguito si analizzano le principali funzioni e classi implementate.

Ogni sonda inizia con lo shebang `#!/usr/bin/env python3` per indicare che il file deve essere eseguito con Python 3. La sonda importa le librerie necessarie e quelle comuni a tutte le sonde AWS sono `boto3` e `typing`. Inoltre è fondamentale importare le componenti del driver di MoonCloud: `abstract_probe, atom, result, entrypoint`.

La sonda è una sottoclasse di `AbstractProbe`, che è una classe base fornita dal driver di MoonCloud: per questo tutto il codice è definito all'interno della classe `Probe`.

Come già detto ogni sonda necessita di credenziali di accesso AWS: la funzione `requires_credentials`, quando la sonda viene eseguita, indica la necessità di caricare le credenziali di accesso AWS durante il deployment; durante fase di testing invece vengono caricate dal file `credentials.json`. Ogni funzione prende come parametri `self` e `inputs=none`, per poi ritornare un valore `bool`.

La funzione `init` contiene la logica di inizializzazione della sonda: in particolare usa `boto3` per creare un client per il servizio AWS specifico, carica le credenziali di accesso AWS, controlla che non siano mancanti, definisce le strutture `self.control_result` e `self.collected_data` usate per immagazzinare i risultati dei controlli, ed infine esegue una semplice chiamata al servizio per verificare che le credenziali siano valide.

Le funzioni riguardanti i controlli specifici CIS prendono il nome **instance_check_n** ed ognuna di essa esegue chiamate API per poi salvare salvare i dati rilevati in `self.collected_data` e il risultato del controllo in `self.control_result`, entrambi con un formato comune: `"CIS *servizio*.*numero-controllo* - *descrizione*` per i dati rilevati, `"CIS *servizio*.*numero-controllo*"` per i risultati del controllo.

Dopo che vengono eseguiti tutti i controlli la funzione `execute_all_controls` si occupa di stabilire il risultato finale della sonda in base al numero di controlli passati, oppure in base alla presenza credenziali errate. Crea inoltre la struttura di ogni blocco relativo ad un controllo aggiungendo i dati rilevati in maniera annidata. Di seguito è riportata la struttura della funzione:

```
def execute_all_controls(self, inputs=None) -> bool:
passed_controls = sum(1 for c in self.control_results
.values() if c["Passed"])
total_controls = len(self.control_results)
self.result.put_raw_extra_data("Summary",
f"passed_controls}/{total_controls} succeeded")

for name, ctrl in self.control_results.items():
    block = {
        "Passed": ctrl["Passed"],
        "Result": ctrl["Result"],
    }
    prefix = f"{name} - "
    for k, v in self.collected_data.items():
        if k.startswith(prefix):
            field = k[len(prefix):]
            block[field] = v
    self.result.put_raw_extra_data(name, block)

if passed_controls == 0:
    if total_controls == 0:
        self.result.integer_result = 2
        self.result.pretty_result = "Credential error: AWS credentials are
missing or invalid"
else:
    self.result.integer_result = 0 if passed_controls == total_controls else 1
    self.result.pretty_result = "The probe executed successfully!"

return True
```

Listing 4.6: Funzione `execute_all_controls` per la gestione dei risultati della sonda

L'ultima funzione è atoms, che racchiude la logica della catena di esecuzione della sonda definendo i vari stati che rappresentano una singola funzione, oltre che specificare il comportamento del programma in caso si verifichino eccezioni. In particolare, si noti come per la funzione init l'azione in caso di eccezione è STOP, in modo che si fermi subito la catena di esecuzione e sia possibile gestire il risultato della sonda in modo diretto. Tutte le altre funzioni sono configurate come GO_ON in quanto è l'ultima funziona presentata in precedenza a gestire il risultato finale. Di seguito si riporta un esempio della funzione:

```
def atoms(self) -> typing.Sequence[atom.AtomPairWithException]:
return [
    atom.AtomPairWithException(
        forward=self.init,
        forward_captured_exceptions=[
            atom.PunctualExceptionInformationForward(
                exception_class=Exception,
                action=atom.OnExceptionActionForward.STOP,
                result_producer=lambda e: result.Result(
                    integer_result=2,
                    pretty_result="Credential error: AWS credentials missing or invalid",
                    base_extra_data=result.Extradata(raw={"ExceptionRecovered": str(e)})
                )
            )
        ]
    ),
    atom.AtomPairWithException(
        forward=self.sqs_check_1,
        forward_captured_exceptions=[
            atom.PunctualExceptionInformationForward(
                exception_class=Exception,
                action=atom.OnExceptionActionForward.GO_ON,
                result_producer=lambda e: result.Result(base_extra_data=result.Extradata(raw={'ExceptionRecovered': str(e)}))
            )
        ]
    ),
    // and so on
    atom.AtomPairWithException(
        forward=self.execute_all_controls,
        forward_captured_exceptions=[
            atom.PunctualExceptionInformationForward(
                exception_class=Exception,
```

```
33                 action=atom.OnExceptionActionForward.GO_ON,
34                 result_producer=lambda e: result.Result(base_extra_data=result.
Extradata(raw={'ExceptionRecovered': str(e)}))
35             )
36         ]
37     ),
38 ]
```

Listing 4.7: Funzione `atoms` per la definizione della catena di esecuzione della sonda

4.2.1 aws_sqs

La sonda `aws_sqs` è dedicata al servizio Amazon Simple Queue Service (SQS), un servizio di messaggistica che consente di inviare, ricevere e memorizzare messaggi tra componenti software. La sonda verifica la conformità di SQS rispetto a 3 controlli definiti da Security Hub: si è partiti da questo servizio in quanto era stata implementata in precedenza una sonda, infatti CIS AWS Foundations Benchmark v3 non comprende controlli per SQS.

I tre controlli sono:

- *SQS.1 - Amazon SQS queues should be encrypted at rest.*
  - *This control checks whether an Amazon SQS queue is encrypted at rest.*
  - *The control fails if the queue isn't encrypted with an SQS-managed key (SSE-SQS) or an AWS Key Management Service (AWS KMS) key (SSE-KMS).*
  - *Encrypting data at rest reduces the risk of an unauthorized user accessing data stored on disk. Server-side encryption (SSE) protects the contents of messages in SQS queues using SQS-managed encryption keys (SSE-SQS) or AWS KMS keys (SSE-KMS).*
- *SQS.2 - SQS queues should be tagged.*
  - *Parameter:* `requiredKeyTags`
    - *Description: List of non-system tag keys that the evaluated resource must contain. Tag keys are case sensitive.*
    - *Type: StringList (maximum of 6 items)*
    - *Allowed custom values: 1-6 tag keys that meet AWS requirements.*
    - *Security Hub default value: No default value*
  - *This control checks whether an Amazon SQS queue has tags with the specific keys defined in the parameter requiredTagKeys. The control fails if the*

queue doesn't have any tag keys or if it doesn't have all the keys specified in the parameter requiredTagKeys. If the parameter requiredTagKeys isn't provided, the control only checks for the existence of a tag key and fails if the queue isn't tagged with any key. System tags, which are automatically applied and begin with aws:, are ignored.

  - *A tag is a label that you assign to an AWS resource, and it consists of a key and an optional value. You can create tags to categorize resources by purpose, owner, environment, or other criteria. Tags can help you identify, organize, search for, and filter resources. Tagging also helps you track accountable resource owners for actions and notifications. When you use tagging, you can implement attribute-based access control (ABAC) as an authorization strategy, which defines permissions based on tags. You can attach tags to IAM entities (users or roles) and to AWS resources. You can create a single ABAC policy or a separate set of policies for your IAM principals. You can design these ABAC policies to allow operations when the principal's tag matches the resource tag.*

- *SQS.3 - SQS queue access policies should not allow public access.*

  - *This controls checks whether an Amazon SQS access policy allows public access to an SQS queue. The control fails if an SQS access policy allows public access to the queue.*
  - *An Amazon SQS access policy can allow public access to an SQS queue, which might allow an anonymous user or any authenticated AWS IAM identity to access the queue. SQS access policies typically provide this access by specifying the wildcard character (*) in the Principal element of the policy, not using proper conditions to restrict access to the queue, or both. If an SQS access policy allows public access, third parties might be able to perform tasks such as receive messages from the queue, send messages to the queue, or modify the access policy for the queue. This could result in events such as data exfiltration, a denial of service, or injection of messages into the queue by a threat actor.*

[1]

Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda. Questa sonda è stata usata come prova per l'implementazione multiregione: ogni controllo itera tra gli *n* client SQS, eseguendo le chiamate API necessarie e salvando i risultati in una struttura annidata suddivisa per regione.

[1] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/sqs-controls.html` .

Il controllo **SQS.1** verifica se le code SQS siano crittografate quando sono inattive. In un ciclo che itera tra le regioni specificate, la sonda esegue una chiamata API a `list_queues`, inserisce le code in una lista e per ognuna di esse controlla se il parametro `SqsManagedSseEnabled` è `True`. Se non lo è, la coda viene aggiunta alla lista `unencrypted_queues`, `encrypted_queues` altrimenti. Il controllo viene marcato come passato se tutte le code in tutte le regioni sono crittografate, altrimenti fallisce.

Il controllo **SQS.2** richiede un parametro di input, `requiredTagKeys`, che è un array di stringhe contenente le chiavi dei tag richiesti. Per questo motivo in maniera analoga a quanto fatto per le regioni, la stringa in input alla sonda viene convertita in un array di stringhe. La sonda esegue una chiamata a `list_queues` e per ognuna di esse esegue una chiamata a `list_queue_tags`. Se la coda non ha tag, viene aggiunta alla lista `untagged_queues`, altrimenti si controlla se contiene i tag richiesti. Se non li contiene, viene aggiunta alla lista `missing_tags`. Il controllo viene marcato come passato se tutte le code in tutte le regioni hanno i tag richiesti, oppure, se non ci sono tag specificati nel parametro in ingresso, se tutte le code hanno almeno un tag.

Il controllo **SQS.3** verifica se le politiche di accesso delle code SQS non consentono l'accesso pubblico. La sonda esegue una chiamata API a `list_queues` e per ognuna di esse esegue una chiamata a `get_queue_attributes` per ottenere gli attributi della coda. Se l'attributo `Policy` esiste, si itera per `statement` e si controlla se tra quelli con valore `Effect: Allow`, ne esista uno con `Principal: {*}` oppure `Principal: {'AWS'}`. Il controllo viene marcato come passato se tutte le code in tutte le regioni non hanno politiche di accesso pubbliche.

La gestione della funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2, e la funzione `atoms` crea una catena di **forward** di 5 atomi: `init`, `sqs_check_1`, `sqs_check_2`, `sqs_check_3` e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `sqs:ListQueues`
- `sqs:GetQueueAttributes`
- `sqs:ListQueueTags`

Il form di MoonCloud è stato configurato per accettare una sola stringa, che viene poi convertita in un array di stringhe nella funzione `init`. Successivamente, si istanziano n client SQS, uno per ogni regione specificata. La chiamata API per la verifica della correttezza delle credenziali è fatta solo per la prima regione, in quanto non è necessario verificarle per ogni regione. Di seguito è riportato il codice che gestisce queste funzionalità:

```
1 # split on commas, semicolons or whitespace into list
```

```
if isinstance(raw_regions, str):
    raw_regions = raw_regions.strip()
    specific_regions = re.split(r'[,;\s]+', raw_regions) if raw_regions else []
elif isinstance(raw_regions, list):
    specific_regions = raw_regions
else:
    specific_regions = []

#max 6 regions
if len(specific_regions) > 6:
    specific_regions = specific_regions[:6]

self.clients = {}
# ensure at least one region
if not specific_regions:
    specific_regions = ['eu-central-1']

for idx, region in enumerate(specific_regions, start=1):
    self.clients[f'client_{idx}'] = boto3.client(
        'sqs',
        aws_access_key_id=access_key_id,
        aws_secret_access_key=secret_access_key,
        region_name=region
    )

self.clients['client_1'].list_queues(MaxResults=1)
return True
```

Listing 4.8: Parte della funzione `init` per la sonda `aws_sqs`

Il codice completo della sonda `aws_sqs` è disponibile nell'appendice A.

4.2.2 aws_inspector

La sonda `aws_inspector` è dedicata al servizio Amazon Inspector, un servizio di sicurezza che consente di eseguire valutazioni di sicurezza automatizzate sulle risorse di un account AWS. La sonda verifica la conformità di Amazon Inspector rispetto a 4 controlli definiti all'interno di AWS Security Hub. Questi controlli si basano sulle raccomandazioni del CIS AWS Foundations Benchmark, ma nella versione 3 del benchmark non sono ancora inclusi controlli specifici per Amazon Inspector. Tuttavia, è stato deciso di implementare comunque questa sonda, anche per continuità con una versione precedente già esistente. I controlli attualmente considerati riguardano principalmente la corretta abilitazione del servizio per diversi tipi di risorse AWS.

- *Inspector.1 - Amazon Inspector EC2 scanning should be enabled.*
  - *This control checks whether Amazon Inspector EC2 scanning is enabled. For a standalone account, the control fails if Amazon Inspector EC2 scanning is disabled in the account. In a multi-account environment, the control fails if the delegated Amazon Inspector administrator account and all member accounts don't have EC2 scanning enabled.*
  - *In a multi-account environment, the control generates findings in only the delegated Amazon Inspector administrator account. Only the delegated administrator can enable or disable the EC2 scanning feature for the member accounts in the organization. Amazon Inspector member accounts can't modify this configuration from their accounts. This control generates FAILED findings if the delegated administrator has a suspended member account that doesn't have Amazon Inspector EC2 scanning enabled. To receive a PASSED finding, the delegated administrator must disassociate these suspended accounts in Amazon Inspector.*
  - *Amazon Inspector EC2 scanning extracts metadata from your Amazon Elastic Compute Cloud (Amazon EC2) instance, and then compares this metadata against rules collected from security advisories to produce findings. Amazon Inspector scans instances for package vulnerabilities and network reachability issues.*
- *Inspector.2 - Amazon Inspector ECR scanning should be enabled.*
- *Inspector.3 - Amazon Inspector Lambda code scanning should be enabled.*
- *Inspector.4 - Amazon Inspector Lambda standard scanning should be enabled.*

[2]

Non sono riportati i dettagli di ogni controllo secondo Security Hub in quanto sono molto simili a quelli di Inspector.1, cambia solo il tipo di risorsa da controllare. Tutte le sonde da ora in poi sono state implementate in modo da essere eseguite solo in una specifica regione, ricevuta come parametro di input. Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda.

Nella funzione `init` viene istanziato il client `inspector2` tramite Boto3 e viene eseguita una chiamata a `list_members` per verificare che le credenziali siano valide. La sonda salva i risultati dei controlli in `self.control_results` e i dati raccolti in `self.collected_data`, come descritto nella sezione 4.2.

[2] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/inspector-controls.html` .

Il controllo **Inspector.1** verifica che Amazon Inspector sia abilitato per scansionare le risorse EC2. La sonda esegue una chiamata a `batch_get_account_status` per ottenere lo stato dell'account e verifica che `resourceState` sia `ENABLED` per l'account amministratore. Successivamente procede a verificare che tutti i membri dell'account abbiano lo stato `ENABLED` per le risorse EC2: la chiamata `list_members` crea un lista di membri e se vuota marca il controllo come passato se sia abilitato per l'account amministratore; se invece esiste almeno un membro, la sonda verifica che essi siano `ENABLED` o `SUSPENDED` per le risorse EC2. Se tutti i membri hanno lo stato `ENABLED`, il controllo viene marcato come passato, altrimenti fallisce.

I controlli **Inspector.2**, **Inspector.3**, **Inspector.4** sono implementati con la stessa logica, differenziandosi solo per il tipo di risorsa sottoposta a verifica.

La gestione della funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2: la funzione `atoms` crea una catena di **forward** di 5 atomi: `init`, `inspector_check_1`, `...`, `inspector_check_4`, e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `inspector2:ListMembers`
- `inspector2:BatchGetAccountStatus`

Il codice completo della sonda `aws_inspector` è disponibile nell'appendice B.

4.2.3 aws_iam

La sonda `aws_iam` è dedicata al servizio AWS Identity and Access Management (IAM), che consente di gestire l'accesso alle risorse AWS tramite la creazione e la gestione di utenti, gruppi e permessi. La sonda verifica la conformità di IAM rispetto a 13 controlli di sicurezza definiti dal CIS AWS Foundations Benchmark v3, integrati in AWS Security Hub. I controlli selezionati riguardano aspetti critici della gestione di identità e accessi, come la configurazione di utenti, gruppi, policy e credenziali.

- *IAM.2 - IAM users should not have IAM policies attached.*
  - *This control checks whether your IAM users have policies attached. The control fails if your IAM users have policies attached. Instead, IAM users must inherit permissions from IAM groups or assume a role.*
  - *By default, IAM users, groups, and roles have no access to AWS resources. IAM policies grant privileges to users, groups, or roles. We recommend that you apply IAM policies directly to groups and roles but not to users. Assigning*

privileges at the group or role level reduces the complexity of access management as the number of users grows. Reducing access management complexity might in turn reduce the opportunity for a principal to inadvertently receive or retain excessive privileges.

- *IAM.3 - IAM users' access keys should be rotated every 90 days or less.*
  - *This control checks whether the active access keys are rotated within 90 days.*
  - *If you already have an access key, Security Hub recommends that you rotate the access keys every 90 days. Rotating access keys reduces the chance that an access key that is associated with a compromised or terminated account is used. It also ensures that data cannot be accessed with an old key that might have been lost, cracked, or stolen. Always update your applications after you rotate access keys.*
- *IAM.4 - IAM root user access key should not exist.*
  - *This control checks whether the root user access key is present.*
  - *The root user is the most privileged user in an AWS account. AWS access keys provide programmatic access to a given account. Security Hub recommends that you remove all access keys that are associated with the root user. This limits that vectors that can be used to compromise your account. It also encourages the creation and use of role-based accounts that are least privileged.*
- *IAM.5 - MFA should be enabled for all IAM users that have a console password.*
  - *This control checks whether AWS multi-factor authentication (MFA) is enabled for all IAM users that use a console password.*
  - *Multi-factor authentication (MFA) adds an extra layer of protection on top of a user name and password. With MFA enabled, when a user signs in to an AWS website, they are prompted for their user name and password. In addition, they are prompted for an authentication code from their AWS MFA device.*
  - *We recommend that you enable MFA for all accounts that have a console password. MFA is designed to provide increased security for console access. The authenticating principal must possess a device that emits a time-sensitive key and must have knowledge of a credential.*
- *IAM.6 - Hardware MFA should be enabled for the root user.*

  - *This control checks whether your AWS account is enabled to use a hardware multi-factor authentication (MFA) device to sign in with root user credentials. The control fails if hardware MFA isn't enabled or virtual MFA devices are permitted for signing in with root user credentials.*
  - *Virtual MFA might not provide the same level of security as hardware MFA devices. We recommend that you use a virtual MFA device only while you wait for hardware purchase approval or for your hardware to arrive. To learn more, see Assign a virtual MFA device (console) in the IAM User Guide.*

- *IAM.9 - MFA should be enabled for the root user.*
  - *This control checks whether multi-factor authentication (MFA) is enabled for the IAM root user of an AWS account to sign in to the AWS Management Console. The control fails if MFA isn't enabled for the root user of the account.*
  - *The IAM root user of an AWS account has complete access to all the services and resources in the account. If MFA is enabled, the user must enter a username, a password, and an authentication code from their AWS MFA device in order to sign in to the AWS Management Console. MFA adds an extra layer of protection on top of a username and password.*

- *IAM.15 - Ensure IAM password policy requires minimum password length of 14 or greater.*
  - *Password policies, in part, enforce password complexity requirements. Use IAM password policies to ensure that passwords are at least a given length.*
  - *CIS recommends that the password policy require a minimum password length of 14 characters. Setting a password complexity policy increases account resiliency against brute force login attempts.*

- *IAM.16 - Ensure IAM password policy prevents password reuse.*
  - *This control checks whether the number of passwords to remember is set to 24. The control fails if the value is not 24.*
  - *IAM password policies can prevent the reuse of a given password by the same user. CIS recommends that the password policy prevent the reuse of passwords. Preventing password reuse increases account resiliency against brute force login attempts.*

- *IAM 18 - Ensure a support role has been created to manage incidents with AWS Support.*

  - *AWS provides a support center that can be used for incident notification and response, as well as technical support and customer services.*
  - *Create an IAM role to allow authorized users to manage incidents with AWS Support. By implementing least privilege for access control, an IAM role will require an appropriate IAM policy to allow support center access in order to manage incidents with Support.*

- *IAM.22 - IAM user credentials unused for 45 days should be removed.*
  - *This control checks whether your IAM users have passwords or active access keys that have not been used for 45 days or more. To do so, it checks whether the* `maxCredentialUsageAge` *parameter of the AWS Config rule is equal to 45 or more.*
  - *Users can access AWS resources using different types of credentials, such as passwords or access keys. CIS recommends that you remove or deactivate all credentials that have been unused for 45 days or more. Disabling or removing unnecessary credentials reduces the window of opportunity for credentials associated with a compromised or abandoned account to be used.*

- *IAM.26 - Expired SSL/TLS certificates managed in IAM should be removed.*
  - *This controls checks whether an active SSL/TLS server certificate that is managed in IAM has expired. The control fails if the expired SSL/TLS server certificate isn't removed.*
  - *To enable HTTPS connections to your website or application in AWS, you need an SSL/TLS server certificate. You can use IAM or AWS Certificate Manager (ACM) to store and deploy server certificates. Use IAM as a certificate manager only when you must support HTTPS connections in an AWS Region that isn't supported by ACM. IAM securely encrypts your private keys and stores the encrypted version in IAM SSL certificate storage. IAM supports deploying server certificates in all Regions, but you must obtain your certificate from an external provider for use with AWS.*

- *IAM.27 - IAM identities should not have the* `AWSCloudShellFullAccess` *policy attached.*
  - *This control checks whether an IAM identity (user, role, or group) has the AWS managed policy* `AWSCloudShellFullAccess` *attached. The control fails if an IAM identity has the* `AWSCloudShellFullAccess` *policy attached.*

  - *The AWS managed policy AWSCloudShellFullAccess provides full access to CloudShell, which allows file upload and download capability between a user's local system and the CloudShell environment. Within the CloudShell environment, a user has sudo permissions, and can access the internet. As a result, attaching this managed policy to an IAM identity gives them the ability to install file transfer software and move data from CloudShell to external internet servers. We recommend following the principle of least privilege and attaching narrower permissions to your IAM identities.*

- *IAM.28 - IAM Access Analyzer external access analyzer should be enabled.*

  - *This control checks whether an AWS account has an IAM Access Analyzer external access analyzer enabled. The control fails if the account doesn't have an external access analyzer enabled in your currently selected AWS Region.*
  - *IAM Access Analyzer external access analyzers help identify resources, such as Amazon Simple Storage Service (Amazon S3) buckets or IAM roles, that are shared with an external entity. This helps you avoid unintended access to your resources and data. IAM Access Analyzer is Regional and must be enabled in each Region. To identify resources that are shared with external principals, an access analyzer uses logic-based reasoning to analyze resource-based policies in your AWS environment. When you create an external access analyzer, you can create and enable it for your entire organization or individual accounts.*

[3]

Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda.

Nella funzione `init` viene istanziato il client `iam` tramite Boto3 e viene eseguita una chiamata a `list_users` con il parametro `MaxItems` impostato a 1 per verificare che le credenziali siano valide riducendo al minimo il numero di chiamate API. La sonda salva i risultati dei controlli in `self.control_results` e i dati raccolti in `self.collected_data`, come descritto nella sezione 4.2.

Il controllo **IAM.2** controlla che gli utenti IAM non abbiano policy attaccate. La sonda esegue una chiamata a `list_users` e per ognuno di essi esegue una chiamata a `list_attached_user_policies` e `list_inline_user_policies`. Se l'utente ha almeno una policy attaccata, viene aggiunto alla lista `non_compliant` e il controllo viene marcato come fallito. Se non ci sono utenti o nessuno di essi ha policy attaccate, il controllo viene marcato come passato.

[3] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/iam-controls.html` .

Il controllo **IAM.3** verifica che le chiavi di accesso degli utenti IAM siano ruotate ogni 90 giorni o meno. La sonda esegue una chiamata a `list_users` e si serve di `datetime` per ottenere la data corrente. Per ogni utente, esegue una chiamata a `list_access_keys` ed estrae il valore `AccessKeyMetadata` per ogni chiave di accesso. Se la chiave è attiva, si calcola l'età della chiave e se è maggiore di 90 giorni, l'utente viene aggiunto alla lista `non_compliant_users`. Se non ci sono utenti o nessuno di essi ha chiavi di accesso attive, il controllo viene marcato come passato, altrimenti fallisce.

Il controllo **IAM.4** verifica che non esista una chiave di accesso per l'utente root. La sonda esegue una chiamata a `get_account_summary` e controlla il valore `AccountAccessKeysPresent`. Se è diverso da 0, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **IAM.5** verifica che l'autenticazione a più fattori (MFA) sia abilitata per tutti gli utenti IAM che hanno una password di accesso alla console. Anche in questo caso la sonda esegue una chiamata a `list_users` e per ognuno di essi estrae il valore `PasswordLastUsed` per verificare se l'utente ha una password di accesso alla console. In caso affermativo, esegue una chiamata a `list_mfa_devices` passando come parametro l'username estratto dalla lista degli utenti in precedenza. Se il valore `MFADevices` è vuoto, l'utente viene aggiunto alla lista `non_compliant_users` e il controllo viene marcato come fallito. Se non ci sono utenti o nessuno di essi ha una password di accesso alla console, il controllo viene marcato come passato.

Il controllo **IAM.6** verifica che l'utente root abbia abilitato un dispositivo MFA hardware. La sonda esegue una chiamata a `get_account_summary` e controlla il valore `AccountMFAEnabled`. Se è `False`, il controllo viene marcato come fallito. Altrimenti, si esegue una chiamata a `list_virtual_mfa_devices` per verificare che non esista un dispositivo MFA virtuale associato all'utente root. Se non esiste nessun dispositivo MFA virtuale, questo significa che l'utente root ha un dispositivo MFA hardware abilitato e il controllo viene marcato come passato, altrimenti fallisce.

Il controllo **IAM.9** verifica che l'utente root abbia abilitato MFA. La sonda esegue una chiamata a `get_account_summary` e controlla il valore `AccountMFAEnabled`. Se è `False`, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **IAM.15** verifica che la policy di password IAM richieda una lunghezza minima di 14 caratteri. La sonda esegue una chiamata a `get_account_password_policy` in un blocco `try-except` per gestire l'eccezione `NoSuchEntityException` nel caso in cui non esista una policy di password. Se la policy esiste, si controlla il valore `MinimumPasswordLength`. Se è minore di 14, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **IAM.16** verifica che la policy di password IAM impedisca il riutilizzo delle password. Anche in questo caso la sonda crea un blocco `try-except` per gestire l'eccezione `NoSuchEntityException`. Se la policy esiste, si controlla il valore `PasswordReusePrevention`. Se è minore di 24, il controllo viene marcato come fallito,

altrimenti passato.

Il controllo **IAM.18** verifica che sia stato creato un ruolo di supporto per gestire gli incidenti con il supporto AWS. La sonda esegue una chiamata a `get_role` per il ruolo `AWSServiceRoleForSupport` ottenendo `Role` e `Arn`. Questo avviene in un blocco `try-except` per gestire l'eccezione `NoSuchEntity` nel caso in cui il ruolo non esista. Se il ruolo esiste, il controllo viene marcato come passato, altrimenti fallisce.

Il controllo **IAM.22** verifica che le credenziali degli utenti IAM non utilizzate per 45 giorni vengano rimosse. Non potendo accedere alla regola di configurazione `maxCredentialUsageAge`, la sonda esegue una chiamata a `list_users` e per ogni utente esegue una chiamata a `list_access_keys`, per ottenere il valore `AccessKeyMetadata`. Per ogni utente che ha delle chiavi di accesso, si calcola l'età della chiave usando `datetime` e il valore `LastUsedDate` (o `CreateDate` se non esiste `LastUsedDate`). Se l'età della chiave è maggiore di 45 giorni, l'utente viene aggiunto alla lista `non_compliant_users` e il controllo viene marcato come fallito. Se non ci sono utenti o nessuno di essi ha chiavi di accesso non utilizzate per 45 giorni, il controllo viene marcato come passato.

Il controllo **IAM.26** verifica che i certificati SSL/TLS scaduti gestiti in IAM vengano rimossi. La sonda esegue una chiamata a `list_server_certificates` estraendo il valore `ServerCertificateMetadataList`. Per ogni certificato, sempre tramite `datetime`, si stabilisce se il certificato è scaduto confrontando la data di scadenza con la data corrente. Se il certificato è scaduto, ed esiste in quanto trovato nella lista, viene aggiunto alla lista `non_compliant_certificates` e il controllo viene marcato come fallito. Se non ci sono certificati o nessuno di essi è scaduto, il controllo viene marcato come passato.

Il controllo **IAM.27** verifica che le identità IAM non abbiano la policy `AWSCloudShellFullAccess` attaccata. La sonda esegue una chiamata a `list_users` e per ogni utente esegue una chiamata a `list_attached_user_policies`. Se l'utente ha la policy `AWSCloudShellFullAccess` attaccata, viene aggiunto alla lista `non_compliant_users` e il controllo viene marcato come fallito. Se non ci sono utenti o nessuno di essi ha la policy attaccata, il controllo viene marcato come passato.

Il controllo **IAM.28** verifica che l'accesso esterno di Access Analyzer IAM sia abilitato. La sonda instanzia il client `accessanalyzer` e in un blocco `try-except` esegue una chiamata a `list_analyzers`. Se la lista è vuota, significa che non esiste Access Analyzer IAM abilitato e il controllo viene marcato come fallito. Se ne esiste almeno uno, si controlla che il tipo di analizzatore sia `ORGANIZATION` e che il suo stato sia `ACTIVE`. Se entrambi i controlli sono soddisfatti, il controllo viene marcato come passato, altrimenti fallisce.

La gestione della funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2: la funzione `atoms` crea una catena di **forward** di 15 atomi: `init`, `iam_check_2`, ..., `iam_check_28`, e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `iam:ListUsers`
- `iam:ListUserPolicies`
- `iam:ListAttachedUserPolicies`
- `iam:ListAccessKeys`
- `iam:GetAccountSummary`
- `iam:ListMFADevices`
- `iam:ListVirtualMFADevices`
- `iam:GetAccountPasswordPolicy`
- `iam:GetRole`
- `iam:ListServerCertificates`
- `accessanalyzer:ListAnalyzers`

4.2.4 aws_ec2

La sonda `aws_ec2` è dedicata al servizio Amazon Elastic Compute 2 (EC2), che consente di eseguire istanze di calcolo scalabili su AWS. La sonda verifica la conformità di EC2 rispetto a 7 controlli definiti dal CIS AWS Foundations Benchmark v3, integrati in AWS Security Hub. I controlli selezionati riguardano sicurezza di rete, logging, crittografia.

- *EC2.2 - VPC default security groups should not allow inbound or outbound traffic.*
  - *This control checks whether the default security group of a VPC allows inbound or outbound traffic. The control fails if the security group allows inbound or outbound traffic.*
  - *The rules for the default security group allow all outbound and inbound traffic from network interfaces (and their associated instances) that are assigned to the same security group. We recommend that you don't use the default security group. Because the default security group cannot be deleted, you should change the default security group rules setting to restrict inbound and outbound traffic. This prevents unintended traffic if the default security group is accidentally configured for resources such as EC2 instances.*
- *EC2.6 - VPC flow logging should be enabled in all VPCs.*

  - *This control checks whether Amazon VPC Flow Logs are found and enabled for VPCs. The traffic type is set to* `Reject`*. The control fails if VPC Flow Logs aren't enabled for VPCs in your account.*
  - *Security Hub recommends that you enable flow logging for packet rejects for VPCs. Flow logs provide visibility into network traffic that traverses the VPC and can detect anomalous traffic or provide insight during security workflows.*

- *EC2.7 - EBS default encryption should be enabled.*
  - *This control checks whether account-level encryption is enabled by default for Amazon Elastic Block Store (Amazon EBS) volumes. The control fails if the account level encryption isn't enabled for EBS volumes.*
  - *When encryption is enabled for your account, Amazon EBS volumes and snapshot copies are encrypted at rest. This adds an additional layer of protection for your data.*

- *EC2.8 - EC2 instances should use Instance Metadata Service Version 2 (IMDSv2).*
  - *This control checks whether your EC2 instance metadata version is configured with Instance Metadata Service Version 2 (IMDSv2). The control passes if HttpTokens is set to required for IMDSv2. The control fails if* `HttpTokens` *is set to optional.*
  - *You use instance metadata to configure or manage the running instance. The IMDS provides access to temporary, frequently rotated credentials. These credentials remove the need to hard code or distribute sensitive credentials to instances manually or programmatically. The IMDS is attached locally to every EC2 instance. It runs on a special "link local" IP address of 169.254.169.254. This IP address is only accessible by software that runs on the instance. Security Hub recommends that you configure your EC2 instances with IMDSv2.*

- *EC2.21 - Network ACLs should not allow ingress from 0.0.0.0/0 to port 22 or port 3389.*
  - *This control checks whether a network access control list (network ACL) allows unrestricted access to the default TCP ports for SSH/RDP ingress traffic. The control fails if the network ACL inbound entry allows a source CIDR block of '0.0.0.0/0' or '::/0' for TCP ports 22 or 3389. The control doesn't generate findings for a default network ACL.*

  - *Access to remote server administration ports, such as port 22 (SSH) and port 3389 (RDP), should not be publicly accessible, as this may allow unintended access to resources within your VPC.*

- *EC2.53 - EC2 security groups should not allow ingress from 0.0.0.0/0 to remote server administration ports.*

  - *This control checks whether an Amazon EC2 security group allows ingress from 0.0.0.0/0 to remote server administration ports (ports 22 and 3389). The control fails if the security group allows ingress from 0.0.0.0/0 to port 22 or 3389.*
  - *Security groups provide stateful filtering of ingress and egress network traffic to AWS resources. We recommend that no security group allow unrestricted ingress access to remote server administration ports, such as SSH to port 22 and RDP to port 3389, using either the TDP (6), UDP (17), or ALL (-1) protocols. Permitting public access to these ports increases resource attack surface and the risk of resource compromise.*

- *EC2.54 - EC2 security groups should not allow ingress from ::/0 to remote server administration ports.*

  - *...*

[4]

Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda. Nella funzione `init` viene istanziato il client `ec2` tramite Boto3 e viene eseguita una chiamata a `describe_regions` per verificare che le credenziali siano valide.

Il controllo **EC2.2** verifica che i gruppi di sicurezza predefiniti delle VPC non consentano traffico in entrata o in uscita. La sonda esegue una chiamata a `describe_security_groups` e accedendo all'attributo `SecurityGroups`, se il `GroupName` è `default` e il `IpPermissions` o `IpPermissionsEgress` non contengano regole, il controllo viene marcato come passato. Se invece esiste almeno una regola, il controllo viene marcato come fallito.

Il controllo **EC2.6** verifica che il logging del flusso VPC sia abilitato in tutte le VPC. La sonda esegue una chiamata a `describe_vpcs` e per ogni VPC esegue una chiamata a `describe_flow_logs` passando come parametro l'ID della VPC. Se tutti i flussi di log hanno `FlowLogStatus` uguale a `ACTIVE`, il controllo viene marcato come passato, altrimenti fallisce. Se non esistono flussi di log, il controllo viene marcato come fallito.

[4] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/ec2-controls.html` .

Il controllo **EC2.7** verifica che la crittografia predefinita EBS sia abilitata. La sonda esegue una chiamata a `get_ebs_encryption_by_default` e se il valore `EbsEncryptionByDefault` è `True`, il controllo viene marcato come passato, altrimenti fallisce.

Il controllo **EC2.8** verifica che le istanze EC2 utilizzino la versione 2 del servizio di metadati delle istanze (IMDSv2). La sonda esegue una chiamata a `describe_instances` e filtra per `Reservations` e `Instances`. Se il valore `MetadataOptions.HttpTokens` è `required` per ogni istanza, il controllo viene marcato come passato, altrimenti fallisce. Se non esistono istanze, il controllo viene marcato come passato.

Il controllo **EC2.21** verifica che le ACL di rete non consentano ingressi da `0.0.0.0/0` sulle porte 22 o 3389. La sonda esegue una chiamata a `describe_network_acls` e per ogni elemento di `NetworkAcls.Entries` che non sia `Egress` o non sia abilitata, controlla se il valore `CidrBlock` è `0.0.0.0/0` o `::/0` e se il valore `PortRange.From` è 22 o 3389. Se esiste almeno un elemento che soddisfa queste condizioni, il controllo viene marcato come fallito, altrimenti passato. Se non esistono ACL di rete, il controllo viene marcato come passato.

Il controllo **EC2.53** verifica che i gruppi di sicurezza EC2 non consentano ingressi da `0.0.0.0/0` sulle porte di amministrazione dei server remoti. La sonda esegue una chiamata a `describe_security_groups` e per ogni elemento in `SecurityGroups.IpPermissions` si estrae `IpProtocol`, `FromPort` e `ToPort`. Se il valore `IpProtocol` è `-1`, questo vuol dire che la regola consente tutto il traffico, quindi si controlla se il valore `CidrIp` è `0.0.0.0/0` e il gruppo di sicurezza viene marcato come non conforme. Altrimenti si controllano solo le porte 22 e 3389: se in `IpPermissions` esiste almeno un elemento con `CidrIp` uguale a `0.0.0.0/0` e con `FromPort` o `ToPort` uguale a 22 o 3389, il gruppo di sicurezza viene marcato come non conforme e il controllo viene marcato come fallito. É importante in questo controllo rimuovere i duplicati, in quanto un gruppo di sicurezza può avere più regole che soddisfano le condizioni. Se non esistono gruppi di sicurezza il controllo viene marcato come passato.

Il controllo **EC2.54** verifica che i gruppi di sicurezza EC2 non consentano ingressi da `::/0` sulle porte di amministrazione dei server remoti. La logica di controllo è identica a quella del controllo **EC2.53**, ma il controllo viene fatto sul valore `CidrIpv6` invece di `CidrIp`.

La gestione della funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2: la funzione `atoms` crea una catena di **forward** di 8 atomi: `init`, `ec2_check_2`, ..., `ec2_check_54`, e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `ec2:DescribeRegions`

- `ec2:DescribeSecurityGroups`
- `ec2:DescribeVpcs`
- `ec2:DescribeFlowLogs`
- `ec2:GetEbsEncryptionByDefault`
- `ec2:DescribeInstances`
- `ec2:DescribeNetworkAcls`

4.2.5 aws_s3

La sonda `aws_s3` è dedicata al servizio Amazon Simple Storage Service (S3), che consente di archiviare e recuperare dati in modo scalabile e sicuro. La sonda verifica la conformità di S3 rispetto a 6 controlli definiti dal CIS AWS Foundations Benchmark v3, integrati in AWS Security Hub. I controlli selezionati riguardano l'accesso pubblico, il logging, la sicurezza delle operazioni sui bucket.

- *S3.1 - S3 general purpose buckets should have block public access settings enabled.*
  - *This control checks whether the preceding Amazon S3 block public access settings are configured at the account level for an S3 general purpose bucket. The control fails if one or more of the block public access settings are set to false. The control fails if any of the settings are set to false, or if any of the settings are not configured.*
  - *Amazon S3 public access block is designed to provide controls across an entire AWS account or at the individual S3 bucket level to ensure that objects never have public access. Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, or both. Unless you intend to have your S3 buckets be publicly accessible, you should configure the account level Amazon S3 Block Public Access feature.*
- *S3.5 - S3 general purpose buckets should require requests to use SSL.*
  - *This control checks whether an Amazon S3 general purpose bucket has a policy that requires requests to use SSL. The control fails if the bucket policy doesn't require requests to use SSL.*
  - *S3 buckets should have policies that require all requests (Action: S3:*) to only accept transmission of data over HTTPS in the S3 resource policy, indicated by the condition key aws:SecureTransport.*

- *S3.8 - S3 general purpose buckets should block public access.*
  - *This control checks whether an Amazon S3 general purpose bucket blocks public access at the bucket level. The control fails if any of the following settings are set to false:*
    - *ignorePublicAcls*
    - *blockPublicPolicy*
    - *blockPublicAcls*
    - *restrictPublicBuckets*
  - *Block Public Access at the S3 bucket level provides controls to ensure that objects never have public access. Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, or both. Unless you intend to have your S3 buckets publicly accessible, you should configure the bucket level Amazon S3 Block Public Access feature*
- *S3.20 - S3 general purpose buckets should have MFA delete enabled.*
  - *This control checks whether multi-factor authentication (MFA) delete is enabled for an Amazon S3 general purpose bucket. The control fails if MFA delete is not enabled for the bucket. The control doesn't produce findings for buckets that have a lifecycle configuration.*
  - *If you enable versioning for an S3 general purpose bucket, you can optionally add another layer of security by configuring MFA delete for the bucket. If you do this, the bucket owner must include two forms of authentication in any request to delete a version of an object in the bucket or change the versioning state of the bucket. MFA delete provides added security if, for example, the bucket owner's security credentials are compromised. MFA delete can also help prevent accidental bucket deletions by requiring the user who initiates the delete action to prove physical possession of an MFA device with an MFA code, which adds an extra layer of friction and security to the delete action.*
- *S3.22 - S3 general purpose buckets should log object-level write events.*
  - *This control checks whether an AWS account has at least one AWS CloudTrail multi-Region trail that logs all write data events for Amazon S3 buckets. The control fails if the account doesn't have a multi-Region trail that logs write data events for S3 buckets.*
  - *S3 object-level operations, such as* `GetObject`, `DeleteObject`, *and* `PutObject`, *are called data events. By default, CloudTrail doesn't log data*

> *events, but you can configure trails to log data events for S3 buckets. When you enable object-level logging for write data events, you can log each individual object (file) access within an S3 bucket. Enabling object-level logging can help you meet data compliance requirements, perform comprehensive security analysis, monitor specific patterns of user behavior in your AWS account, and take action on object-level API activity within your S3 buckets by using Amazon CloudWatch Events. This control produces a* `PASSED` *finding if you configure a multi-Region trail that logs write-only or all types of data events for all S3 buckets.*
>
> - *S3.23 - S3 general purpose buckets should log object-level read events*
>
> – *...*

[5]

Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda.

Nella funzione `init` vengono istanziati tramite Boto3 i client `s3`, `sts`, `s3control` e `cloudtrail`, necessari per eseguire i controlli. Successivamente si esegue una chiamata a `list_buckets` con parametro `MaxBuckets` impostato a 1 per verificare che le credenziali siano valide riducendo al minimo il numero di chiamate API.

Il controllo **S3.1** verifica che i bucket S3 general purpose abbiano abilitata la funzionalità di blocco dell'accesso pubblico. La sonda esegue una chiamata tramite il client `sts` a `get_caller_identity` per ottenere l'account ID e successivamente, in un blocco `try-except`, esegue una chiamata tramite il client `s3control` a `get_public_access_block` passando come parametro l'account ID per ottenere le informazioni sul blocco dell'accesso pubblico, che come specificato nella documentazione ufficiale, sono:

- `BlockPublicAcls`
- `IgnorePublicAcls`
- `BlockPublicPolicy`
- `RestrictPublicBuckets`

Se almeno uno di questi valori è `False`, il controllo viene marcato come fallito, altrimenti passato. Se non esiste un blocco dell'accesso pubblico, il controllo viene marcato come fallito.

[5] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/s3-controls.html` .

Il controllo **S3.5** verifica che i bucket S3 general purpose richiedano l'uso di SSL per le richieste. La sonda esegue una chiamata tramite il client `s3` a `list_buckets` e crea una lista di buckets. Per ogni bucket, esegue una chiamata a `get_bucket_policy` in un blocco `try-except` per gestire l'eccezione `NoSuchBucketPolicy`, passando come parametro il nome del bucket. Se la policy esiste si controlla ogni `Statement` e se `Effect` è `Deny`, se `Action` include `S3:*`; si esegue un controllo su `Resource` per verificare che, creando ad hoc gli `arn` dei bucket, essi siano presenti (siano un sottoinsieme). Da ultimo, si controlla se `Condition.Bool.aws:SecureTransport` neghi quando non si usa SSL: in questo caso il controllo viene marcato come passato, altrimenti fallisce. Se non esiste una policy per il bucket, il controllo viene marcato come fallito. 4.9 mostra il codice del controllo **S3.5**.

Il controllo **S3.8** verifica che i bucket S3 general purpose blocchino l'accesso pubblico. La sonda esegue una chiamata a `list_buckets` e per ogni bucket, in un blocco `try-except`, esegue una chiamata a `get_public_access_block` passando come parametro il nome del bucket. Ora si estrae il valore `PublicAccessBlockConfiguration` e se esistono tutti i valori `IgnorePublicAcls`, `BlockPublicPolicy`, `BlockPublicAcls` e `RestrictPublicBuckets` e sono tutti `True`, il controllo viene marcato come passato, altrimenti fallisce. Se non esiste un blocco dell'accesso pubblico, il controllo viene marcato come fallito.

Il controllo **S3.20** verifica che i bucket S3 general purpose abbiano abilitato MFA per l'operazione di eliminazione. Anche in questo caso si ottiene una lista di bucket con la chiamata a `list_buckets` e per ognuno di essi, in un blocco `try-except`, si esegue una chiamata a `get_bucket_versioning` passando come parametro il nome del bucket. Se il valore `Status` è `Enabled` e `MfaDelete` è `Enabled`, il controllo viene marcato come passato, altrimenti fallisce. Se non esiste una versione del bucket, il controllo viene marcato come fallito.

Il controllo **S3.22** verifica che i bucket S3 general purpose abbiano abilitato il logging degli eventi di scrittura a livello di oggetto. La sonda esegue una chiamata tramite il client `cloudtrail` a `describe_trails` per ottenere la lista di code. Per ognuna di esse, si controlla se `IsMultiRegionTrail` e si estraggono gli `EventSelectors`. Se esiste almeno un evento di `ReadWriteType` uguale a `WriteOnly` o `All`, si controlla se `DataResources` contiene un oggetto con `Type` uguale a `AWS::S3::Object` e `Type` contenga almeno un valore `arn:aws:s3:::*`. Se tutto questo è soddisfatto, il controllo viene marcato come passato, altrimenti fallisce. Se non esistono code, il controllo viene marcato come fallito.

Il controllo **S3.23** verifica che i bucket S3 general purpose abbiano abilitato il logging degli eventi di lettura a livello di oggetto. La logica di controllo è identica a quella del controllo **S3.22**, ma si controlla se esiste almeno un evento di `ReadWriteType` uguale a `ReadOnly` o `All`.

La gestione della funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2: la funzione `atoms` crea una catena di **forward** di 8 atomi: `init`, `s3_check_1`, ..., `s3_check_23`, e `execute_all_controls`. Anche la gestione delle eccezioni è identica

a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `s3:ListBuckets`
- `sts:GetCallerIdentity`
- `s3control:GetPublicAccessBlock`
- `s3:GetBucketPolicy`
- `s3:GetBucketVersioning`
- `cloudtrail:DescribeTrails`
- `cloudtrail:GetEventSelectors`

Di seguito si riporta il controllo **S3.5** per comprendere meglio come è implementata la logica di controllo.

```
def s3_check_5(self, inputs=None) -> bool:
    """
    S3.5 - S3 general purpose buckets should require requests to use SSL
    """

    compliant, non_compliant = [], []

    buckets = self.client.list_buckets().get('Buckets', [])
    for b in buckets:
        bucket = b['Name']
        try:
            resp = self.client.get_bucket_policy(Bucket=bucket)
            policy = json.loads(resp['Policy'])
            requires_ssl = False
            for stmt in policy.get("Statement", []):
                #deny statements
                if stmt.get("Effect") != "Deny":
                    continue

                #action include s3:*
                actions = stmt.get("Action", [])
                if isinstance(actions, str):
                    actions = [actions]
                if not any(a.lower() == "s3:*" for a in actions):
```

```
                    continue

                resources = stmt.get("Resource", [])
                if isinstance(resources, str):
                    resources = [resources]
                bucket_arn       = f"arn:aws:s3:::{bucket}"
                objects_arn_wild = f"arn:aws:s3:::{bucket}/*"
                if not ({bucket_arn, objects_arn_wild} <= set(resources)):
                    continue

                #check connd
                cond = stmt.get("Condition", {}).get("Bool", {})
                if cond.get("aws:SecureTransport") in ("false", False):
                    requires_ssl = True
                    break

            if requires_ssl:
                compliant.append(bucket)
            else:
                non_compliant.append(bucket)

        except ClientError as e:
            code = e.response.get('Error', {}).get('Code')
            if code == 'NoSuchBucketPolicy':
                non_compliant.append(bucket)
            else:
                non_compliant.append(bucket)

    total = len(buckets)
    count_non = len(non_compliant)
    if count_non == 0:
        pretty = "All buckets require SSL-only requests"
    else:
        pretty = f"{count_non} out of {total} buckets do not require SSL-only
requests"

    self.collected_data["CIS S3.5 - Compliant Buckets"] = compliant
    self.collected_data["CIS S3.5 - Non-Compliant Buckets"] = non_compliant

    passed = len(non_compliant) == 0
    self.control_results["CIS S3.5"] = {
        "Passed": passed,
        "Result": pretty
```

```
67      }
68      return True
```

Listing 4.9: Funzione `s3_check_5` della sonda `aws_s3`

4.2.6 aws_vulnerability

La sonda `aws_vulnerability` è dedicata alla presentazione delle vulnerabilità trovate tramite il servizio Amazon Inspector, che consente di eseguire scansioni di sicurezza su risorse AWS. La sonda ha una struttura diversa dalle precedenti, in quanto non segue controlli specifici del CIS AWS Foundations Benchmark, ma si concentra su raccolta, organizzazione, presentazione delle vulnerabilità trovate.

La sonda necessita anch'essa di credenziali AWS valide e di una regione in cui Amazon Inspector sia abilitato. In aggiunta, si richiede in input una soglia che indichi il livello minimo di gravità delle vulnerabilità da considerare, stabilitio in base al **Common Vulnerability Scoring System (CVSS)**, un sistema di classificazione delle vulnerabilità informatiche da 0 a 10, dove 0 indica nessuna vulnerabilità e 10 indica una vulnerabilità critica. La soglia è quindi un intero compreso tra 0 e 10, e se vengono trovate solo vulnerabilità con un punteggio inferiore alla soglia, la sonda risulta compliant.

Da ultimo, in input si richiede anche il tipo di risorse da analizzare in quanto Amazon Inspector consente di eseguire scansioni su risorse EC2, Lambda Functions e container ECR. Il form di MoonCloud consente di selezionare uno o più tipi di risorse, oppure tutte le risorse.

Viene definita per prima la funzione `_serialize_datetimes` che si occupa di serializzare le date in formato ISO 8601, uno standard comune per rappresentare le date in formato stringa. Questa funzione è utilizzata per normalizzare le date dei findings recuperati da Amazon Inspector, in modo da poterle presentare correttamente nella sonda.

Anche in questa sonda si definisce la funzione `init` che effettua il parsing dei dati in input convertendo la soglia in un float, limitandola a 10 in caso di valori superiori; viene effettuato anche il parsing del tipo di risorse, convertendolo in una lista di stringhe contenente i tipi di risorse selezionati, oppure in una lista contenente la stringa `ALL` se sono state selezionate tutte le risorse. Successivamente si istanzia il client `inspector2` tramite Boto3 e si esegue una chiamata a `list_members` con parametro `MaxResults` impostato a 1 per verificare che le credenziali siano valide riducendo al minimo il numero di chiamate API. Da ultimo, si controlla che il servizio Amazon Inspector sia abilitato per ogni tipo di risorsa selezionato, chiamando `batch_get_account_status` e verificando che il valore `resourceState` nel primo account sia `ENABLED` per ogni tipo di risorsa. Se il servizio non è abilitato, la sonda termina e viene marcata come non conforme, specificando in `extradata` i tipi di risorse non abilitati.

La funzione `cve_scan` è il cuore della sonda e si occupa di recuperare ed elaborare le vulnerabilità trovate da Amazon Inspector. Se la variabile `self.skip` è stata impostata a `True` durante l'inizializzazione (quando Inspector non è abilitato), la funzione termina immediatamente restituendo `False`. Il primo passo consiste nel creare un oggetto paginator tramite `get_paginator("list_findings")`, necessario per gestire efficacemente risultati potenzialmente molto numerosi. Si definiscono poi i parametri di paginazione con un limite massimo di 1000 elementi totali e una dimensione di pagina di 50 elementi per chiamata API, in modo da bilanciare performance e numero di richieste.

La costruzione dei filtri avviene in base al contenuto di `self.target`: se contiene `"ALL"`, vengono inclusi tutti e tre i tipi di risorse supportate (EC2, ECR e Lambda), altrimenti si costruisce il filtro solo per i tipi selezionati. Ogni filtro utilizza l'operatore di confronto `"EQUALS"` per limitare i risultati al tipo di risorsa specificato. Si definisce anche un criterio di ordinamento per `INSPECTOR_SCORE` in ordine decrescente in modo da ottenere prima le vulnerabilità più gravi.

Durante l'iterazione delle pagine, ogni finding viene serializzato in JSON tramite la funzione `_serialize_datetimes` e aggiunto alla lista `clean_findings`. Se non vengono trovate vulnerabilità, la sonda viene marcata come conforme con `integer_result` pari a 0 e un messaggio appropriato.

La parte successiva della funzione si occupa di processare ogni vulnerabilità trovata. Per evitare duplicati, si mantiene un set `seen_cve_ids` che tiene traccia degli ID CVE già processati. Per ogni finding unico, si estrae:

- L'ID CVE dalla struttura `packageVulnerabilityDetails` o dal campo `cve`,
- Il titolo e la descrizione della vulnerabilità,
- La severità,
- Il punteggio CVSS dal primo elemento dell'array `cvss`,
- Le eventuali CWE (Common Weakness Enumeration) correlate,

Questi dati vengono utilizzati per creare un oggetto `ExtradataCVE` che viene aggiunto ai risultati `refined` della sonda. Questo tipo di oggetto è definito dal driver di MoonCloud ed è specifico per la presentazione delle vulnerabilità in sonde per MoonCloud; particolare attenzione viene data alla gestione dei risultati: si utilizza il campo `refined` invece del solito campo `raw` del driver, in modo da presentare le vulnerabilità in un formato strutturato e facilmente interpretabile dalla dashboard di MoonCloud. Durante questo processo, si tiene traccia del numero totale di vulnerabilità (`tot_ctr`) e di quante superano la soglia CVSS impostata (`above_ctr`). Infine, la sonda determina il suo stato di conformità: se almeno una vulnerabilità supera la soglia CVSS, `integer_result` viene impostato a 1 (non conforme), altrimenti rimane 0 (conforme). Il risultato della sonda riporta sempre

il numero di vulnerabilità trovate e quante di esse superano la soglia, fornendo così un quadro completo della situazione delle vulnerabilità delle risorse AWS analizzate.

La funzione `atoms` in questo caso crea una catena di **forward** di 2 atomi: `init` e `cve_scan`. Il codice completo della sonda è riportato nell'appendice C.

4.2.7 aws_account

La sonda `aws_account` è dedicata alla verifica di un solo controllo del CIS AWS Foundations Benchmark v3: esso riguarda il contatto di sicurezza dell'account AWS, che dovrebbe essere configurato correttamente per l'account AWS.

> *Account.1 - Security contact information should be provided for an AWS account*
>
> - *This control checks if an Amazon Web Services (AWS) account has security contact information. The control fails if security contact information is not provided for the account.*
> - *Alternate security contacts allow AWS to contact another person about issues with your account in case you're unavailable. Notifications can be from Support, or other AWS service teams about security-related topics associated with your AWS account usage.*

[6]

Da questa descrizione si è derivata la logica di controllo, che è stata implementata nella sonda. Nella funzione `init` viene istanziato il client `account` tramite Boto3 e viene eseguita una chiamata a `list_regions` per verificare che le credenziali siano valide.

Il controllo **Account.1** verifica che le informazioni di contatto di sicurezza siano fornite per l'account AWS. La funzione, in un blocco `try-except`, esegue una chiamata a `get_alternate_contact` passando come parametro il tipo di contatto (`AlternateContactType`) impostato a `SECURITY`. Se la chiamata ha successo, si controlla se i campi `Name`, `EmailAddress` e `PhoneNumber` sono valorizzati. Se tutti e tre i campi sono valorizzati, il controllo viene marcato come passato, altrimenti fallisce. Se la chiamata fallisce con l'eccezione `ResourceNotFoundException`, il controllo viene marcato come fallito. La funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2. Si una catena di **forward** di 3 atomi: `init` e `account_check_1`, e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `account:GetAlternateContact`

[6] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/account-controls.html` .

- `account:ListRegions`

4.2.8 aws_config

La sonda `aws_config` è dedicata al servizio AWS Config, che consente di monitorare e registrare le configurazioni delle risorse AWS. La sonda verifica la conformità di AWS Config rispetto a un controllo definito dal CIS AWS Foundations Benchmark v3, integrato in AWS Security Hub. Il controllo selezionato riguarda la registrazione delle configurazioni delle risorse.

Config.1 - AWS Config should be enabled and use the service-linked role for resource recording

- *This control checks whether AWS Config is enabled in your account in the current AWS Region, records all resources that correspond to controls that are enabled in the current Region, and uses the service-linked AWS Config role. The name of the service-linked role is AWSServiceRoleForConfig. If you don't use the service-linked role and don't set the includeConfigServiceLinkedRoleCheck parameter to false, the control fails because other roles might not have the necessary permissions for AWS Config to accurately record your resources*
- *The AWS Config service performs configuration management of supported AWS resources in your account and delivers log files to you. The recorded information includes the configuration item (AWS resource), relationships between configuration items, and any configuration changes within resources. Global resources are resources that are available in any Region.*
- *The control is evaluated as follows:*
  - *If the current Region is set as your aggregation Region, the control produces PASSED findings only if AWS Identity and Access Management (IAM) global resources are recorded (if you have enabled controls that require them).*
  - *If the current Region is set as a linked Region, the control doesn't evaluate whether IAM global resources are recorded.*
  - *If the current Region isn't in your aggregator, or if cross-Region aggregation isn't set up in your account, the control produces PASSED findings only if IAM global resources are recorded (if you have enabled controls that require them).*
- *Control results aren't impacted by whether you choose daily or continuous recording of changes in resource state in AWS Config. However, the results of this control*

> *can change when new controls are released if you have configured automatic enablement of new controls or have a central configuration policy that automatically enables new controls. In these cases, if you don't record all resources, you must configure recording for resources that are associated with new controls in order to receive a PASSED finding.*

[7]

Da questa descrizione si è derivata la logica di controllo, che è stata implementata nella sonda. Nella funzione `init` viene istanziato il client `config` tramite Boto3 e viene eseguita una chiamata a `describe_configuration_recorders` per verificare che le credenziali siano valide.

Il controllo **Config.1** verifica che AWS Config sia abilitato e utilizzi il ruolo collegato al servizio per la registrazione delle risorse. Il controllo inizia recuperando i configuration recorder presenti nell'account tramite la chiamata `describe_configuration_recorders`. Se non viene trovato alcun recorder, il controllo fallisce immediatamente con un messaggio appropriato. Nel caso in cui sia presente almeno un recorder, viene selezionato il primo dalla lista e si procede all'estrazione delle informazioni per la verifica:

- Il `roleARN` associato al recorder, che deve contenere `"AWSServiceRoleForConfig"` per essere considerato valido;
- Il valore `includeGlobalResourceTypes` dal recording group, che indica se le risorse globali vengono registrate;
- Lo stato di registrazione attivo, recuperato tramite `describe_configuration_recorder_status`

Un aspetto particolare di questo controllo riguarda la gestione delle regioni collegate attraverso gli aggregatori di configurazione. La funzione recupera tutti gli aggregatori tramite `describe_configuration_aggregators` e verifica se la regione corrente è configurata come regione collegata in qualche aggregatore. Questo viene fatto iterando attraverso le sorgenti di aggregazione di ogni aggregatore e controllando se:

- Il flag `AllAwsRegions` è `False` (indicando che non tutte le regioni sono incluse automaticamente);
- La regione corrente è presente nella lista `AwsRegions` delle regioni specificamente selezionate.

[7] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/config-controls.html` .

La logica per la registrazione delle risorse globali tiene conto di questa configurazione: se la regione corrente è una regione collegata, il requisito di registrare le risorse globali non viene applicato, in quanto sarebbe ridondante. Questo evita duplicazioni quando si utilizzano aggregatori multi-regione. La valutazione finale del controllo si basa su tre condizioni che devono essere soddisfatte:

- Il ruolo ARN deve contenere `"AWSServiceRoleForConfig"`;
- Il recorder deve essere attivo (`is_recording = True`);
- Le risorse globali devono essere registrate, a meno che non si tratti di una regione collegata.

La funzione costruisce un messaggio di output che elenca specificamente quali condizioni non sono state soddisfatte. Altrimenti il controllo viene marcato come passato.

La funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2. Si crea una catena di **forward** di 3 atomi: `init` e `config_check_1` e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `config:DescribeConfigurationRecorders`
- `config:DescribeConfigurationRecorderStatus`
- `config:DescribeConfigurationAggregators`

4.2.9 aws_cloudtrail

La sonda `aws_cloudtrail` è dedicata al servizio AWS CloudTrail che consente di monitorare e registrare le attività degli utenti e delle risorse creando un registro di log delle azioni eseguite nell'account AWS. La sonda verifica la conformità di CloudTrail rispetto a 4 controlli definiti dal CIS AWS Foundations Benchmark v3, integrati in AWS Security Hub. I controlli selezionati riguardano code multi-regione, crittografia, log sui bucket S3.

- *CloudTrail.1 - CloudTrail should be enabled and configured with at least one multi-Region trail that includes read and write management events.*
  - *This control checks whether there is at least one multi-Region AWS CloudTrail trail that captures read and write management events. The control fails if CloudTrail is disabled or if there isn't at least one CloudTrail trail that captures read and write management events. AWS CloudTrail records AWS API calls for your account and delivers log files to you. The recorded information includes the following information:*

    - *Identity of the API caller*
    - *Time of the API call*
    - *Source IP address of the API caller*
    - *Request parameters*
    - *Response elements returned by the AWS service*
  - *CloudTrail provides a history of AWS API calls for an account, including API calls made from the AWS Management Console, AWS SDKs, command line tools. The history also includes API calls from higher-level AWS services such as AWS CloudFormation.*
  - *The AWS API call history produced by CloudTrail enables security analysis, resource change tracking, and compliance auditing. Multi-Region trails also provide the following benefits.*
    - *A multi-Region trail helps to detect unexpected activity occurring in otherwise unused Regions.*
    - *A multi-Region trail ensures that global service event logging is enabled for a trail by default. Global service event logging records events generated by AWS global services.*
    - *For a multi-Region trail, management events for all read and write operations ensure that CloudTrail records management operations on all resources in an AWS account.*

- *CloudTrail.2 - CloudTrail should have encryption at-rest enabled.*
  - *This control checks whether CloudTrail is configured to use the server-side encryption (SSE) AWS KMS key encryption. The control fails if the* `KmsKeyId` *isn't defined.*
  - *For an added layer of security for your sensitive CloudTrail log files, you should use server-side encryption with AWS KMS keys (SSE-KMS) for your CloudTrail log files for encryption at rest. Note that by default, the log files delivered by CloudTrail to your buckets are encrypted by Amazon server-side encryption with Amazon S3-managed encryption keys (SSE-S3).*

- *CloudTrail.4. - CloudTrail log file validation should be enabled.*
  - *This control checks whether log file integrity validation is enabled on a CloudTrail trail.*
  - *CloudTrail log file validation creates a digitally signed digest file that contains a hash of each log that CloudTrail writes to Amazon S3. You can use*

these digest files to determine whether a log file was changed, deleted, or unchanged after CloudTrail delivered the log.

- *CloudTrail.7 - Ensure S3 bucket access logging is enabled on the CloudTrail S3 bucket.*
  - *S3 bucket access logging generates a log that contains access records for each request made to your S3 bucket. An access log record contains details about the request, such as the request type, the resources specified in the request worked, and the time and date the request was processed.*
  - *CIS recommends that you enable bucket access logging on the CloudTrail S3 bucket. By enabling S3 bucket logging on target S3 buckets, you can capture all events that might affect objects in a target bucket. Configuring logs to be placed in a separate bucket enables access to log information, which can be useful in security and incident response workflows.*

[8]

Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda.

Nella funzione `init` vengono istanziati i client `cloudtrail` e `s3` tramite Boto3 e viene eseguita una chiamata a `describe_trails` per verificare che le credenziali siano valide.

Il controllo **CloudTrail.1** verifica che CloudTrail sia abilitato e configurato con almeno una code multi-regione che includa eventi di gestione di lettura e scrittura. La funzione esegue una chiamata a `describe_trails` per ottenere una lista di code. Se non esiste alcuna coda, il controllo viene marcato come fallito in quanto CloudTrail non è abilitato. Se esiste almeno una coda, si itera tra di esse e si controlla se `IsMultiRegionTrail` è `True`, e se `IncludeGlobalServiceEvents` è `True`. Se entrambe le condizioni sono soddisfatte, si esegue una chiamata a `get_event_selectors` passando come parametro il nome della coda e si controlla se esiste almeno un evento di `ReadWriteType` uguale a `All` e `IncludeManagementEvents` è `True`. Se tutte queste condizioni sono soddisfatte, la coda viene aggiunta alla lista `compliant`, altrimenti alla lista `non_compliant`. Se non esiste alcuna coda che soddisfa le condizioni, il controllo viene marcato come fallito. Se esiste almeno una coda che soddisfa le condizioni, il controllo viene marcato come passato.

Il controllo **CloudTrail.2** verifica che CloudTrail abbia abilitata la crittografia dei dati a riposo. La funzione esegue una chiamata a `describe_trails` per ottenere una lista di code. Se non esiste alcuna coda, il controllo viene marcato come fallito in quanto CloudTrail non è abilitato. Se esiste almeno una coda, si itera tra di esse e si controlla se `KmsKeyId` abbia un valore: in questo caso l'ARN della coda viene aggiunto alla lista

[8] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/cloudtrail-controls.html` .

compliant, altrimenti alla lista non_compliant. Se esiste almeno una coda che non ha un KmsKeyId definito, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **CloudTrail.4** verifica che la validazione dei file di log di CloudTrail sia abilitata. La funzione esegue una chiamata a describe_trails per ottenere una lista di code. Se non esiste alcuna coda, il controllo viene marcato come fallito in quanto CloudTrail non è abilitato. Se esiste almeno una coda, si itera tra di esse e si controlla se LogFileValidationEnabled è True: in questo caso l'ARN della coda viene aggiunto alla lista compliant, altrimenti alla lista non_compliant. Se esiste almeno una coda che non ha la validazione dei file di log abilitata, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **CloudTrail.7** verifica che il logging degli accessi al bucket S3 di CloudTrail sia abilitato. La funzione esegue una chiamata a describe_trails per ottenere una lista di code. Se non esiste alcuna coda, il controllo viene marcato come fallito in quanto CloudTrail non è abilitato. Se esiste almeno una coda, si itera tra di esse e si controlla se S3BucketName è valorizzato: se non esistono bucket S3, il controllo viene marcato come fallito. Se esiste almeno un bucket S3, si esegue una chiamata a tramite il client s3 a get_bucket_logging passando come parametro il nome del bucket. Se la risposta contiene un oggetto LoggingEnabled e TargetBucket punta ad un bucket diverso da quello target, il controllo viene marcato come passato, altrimenti fallisce. Se non esiste un bucket S3, il controllo viene marcato come fallito.

La funzione execute_all_controls è identica a quella descritta nella sezione 4.2. Si crea una catena di **forward** di 6 atomi: init, cloudtrail_check_1, cloudtrail_check_2, cloudtrail_check_4, cloudtrail_check_7 e execute_all_controls. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- cloudtrail:DescribeTrails
- cloudtrail:GetEventSelectors
- s3:GetBucketLogging

4.2.10 aws_efs

La sonda aws_efs è dedicata al servizio Amazon Elastic File System (EFS) che consente di creare file system scalabili e condivisi per le risorse AWS. La sonda verifica la conformità di EFS rispetto ad un controllo definito dal CIS AWS Foundations Benchmark v3, integrato in AWS Security Hub. Esso riguarda la crittografia a riposo tramite KMS.

EFS.1 - Elastic File System should be configured to encrypt file data at-rest using AWS KMS

- *This control checks whether Amazon Elastic File System is configured to encrypt the file data using AWS KMS. The check fails in the following cases.*
  - `Encrypted` *is set to* `false` *in the* `DescribeFileSystems` *response.*
  - *The* `KmsKeyId` *key in the* `DescribeFileSystems` *response does not match the* `KmsKeyId` *parameter for* `efs-encrypted-check`.
- *Note that this control does not use the* `KmsKeyId` *parameter for* `efs-encrypted-check`. *It only checks the value of* `Encrypted`.
- *For an added layer of security for your sensitive data in Amazon EFS, you should create encrypted file systems. Amazon EFS supports encryption for file systems at-rest. You can enable encryption of data at rest when you create an Amazon EFS file system.*

[9]

Da questa descrizione si è derivata la logica di controllo, che è stata implementata nella sonda. Nella funzione `init` viene istanziato il client `efs` tramite Boto3 e viene eseguita una chiamata a `describe_file_systems` per verificare che le credenziali siano valide.

Il controllo **EFS.1** verifica che Amazon EFS sia configurato per cifrare i dati a riposo utilizzando AWS KMS. La funzione esegue una chiamata a `describe_file_systems` per ottenere una lista di file system. Se non esiste alcun file system, il controllo viene marcato come passato in quanto non sono presenti dati da cifrare. Se esiste almeno un file system, si itera tra di essi e si controlla se il campo `Encrypted` è `True`. Se lo è, il file system viene aggiunto alla lista `compliant`, altrimenti alla lista `non_compliant`. Se esiste almeno un file system che non ha la cifratura abilitata, il controllo viene marcato come fallito, altrimenti passato.

La funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2. Si crea una catena di **forward** di 3 atomi: `init`, `efs_check_1` e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

L'**azione AWS** necessaria per eseguire la sonda è:

- `efs:DescribeFileSystems`

[9] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/efs-controls.html` .

4.2.11 aws_kms

La sonda `aws_kms` è dedicata al servizio AWS Key Management Service (KMS) che consente di creare e gestire chiavi di crittografia per proteggere i dati dell'account AWS. La sonda verifica la conformità di KMS rispetto ad un controllo definito dal CIS AWS Foundations Benchmark v3, integrato in AWS Security Hub. Esso riguarda la rotazione delle chiavi di crittografia.

> *KMS.4 - AWS KMS key rotation should be enabled*
>
> - *AWS KMS enables customers to rotate the backing key, which is key material stored in AWS KMS and is tied to the key ID of the KMS key. It's the backing key that is used to perform cryptographic operations such as encryption and decryption. Automated key rotation currently retains all previous backing keys so that decryption of encrypted data can take place transparently.*
> - *CIS recommends that you enable KMS key rotation. Rotating encryption keys helps reduce the potential impact of a compromised key because data encrypted with a new key can't be accessed with a previous key that might have been exposed.*

[10]

Da questa descrizione si è derivata la logica di controllo, che è stata implementata nella sonda. Nella funzione `init` viene istanziato il client `kms` tramite Boto3 e viene eseguita una chiamata a `list_keys` per verificare che le credenziali siano valide.

Il controllo **KMS.4** verifica che AWS KMS abbia abilitata la rotazione delle chiavi. La funzione esegue una chiamata a `list_keys` per ottenere una lista di chiavi. Per ognuna di essa si estrae il `KeyId` per poi eseguire una chiamata a `get_key_rotation_status` (passando come parametro il `KeyId`) per verificare se la rotazione è abilitata. Se `KeyRotationEnabled` esiste nella risposta, la chiave viene aggiunta alla lista `compliant`, altrimenti alla lista `non_compliant`. Se esiste almeno una chiave che non ha la rotazione abilitata, il controllo viene marcato come fallito, altrimenti passato.

La funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2. Si crea una catena di **forward** di 3 atomi: `init`, `kms_check_4` e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `kms:ListKeys`
- `kms:GetKeyRotationStatus`

[10] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/kms-controls.html` .

4.2.12 aws_rds

La sonda `aws_rds` è dedicata al servizio Amazon Relational Database Service (RDS) che consente di creare e gestire database relazionali scalabili. La sonda verifica la conformità di RDS rispetto a 3 controlli definiti dal CIS AWS Foundations Benchmark v3, integrati in AWS Security Hub. I controlli selezionati riguardano l'accesso pubblico, crittografia a riposo e aggiornamenti automatici.

- *RDS.2 - RDS DB Instances should prohibit public access, as determined by the PubliclyAccessible configuration*
  - *This control checks whether Amazon RDS instances are publicly accessible by evaluating the* `PubliclyAccessible` *field in the instance configuration item.*
  - *The* `PubliclyAccessible` *value in the RDS instance configuration indicates whether the DB instance is publicly accessible. When the DB instance is configured with* `PubliclyAccessible`*, it is an Internet-facing instance with a publicly resolvable DNS name, which resolves to a public IP address. When the DB instance isn't publicly accessible, it is an internal instance with a DNS name that resolves to a private IP address.*
  - *Unless you intend for your RDS instance to be publicly accessible, the RDS instance should not be configured with* `PubliclyAccessible` *value. Doing so might allow unnecessary traffic to your database instance.*
- *RDS.3 - RDS DB instances should have encryption at-rest enabled*
  - *This control checks whether storage encryption is enabled for your Amazon RDS DB instances.*
  - *For an added layer of security for your sensitive data in RDS DB instances, you should configure your RDS DB instances to be encrypted at rest. To encrypt your RDS DB instances and snapshots at rest, enable the encryption option for your RDS DB instances. Data that is encrypted at rest includes the underlying storage for DB instances, its automated backups, read replicas, and snapshots.*
  - *RDS encrypted DB instances use the open standard AES-256 encryption algorithm to encrypt your data on the server that hosts your RDS DB instances. After your data is encrypted, Amazon RDS handles authentication of access and decryption of your data transparently with a minimal impact on performance. You do not need to modify your database client applications to use encryption.*

- *RDS.13 - RDS automatic minor version upgrades should be enabled*
  - *This control checks whether automatic minor version upgrades are enabled for the RDS database instance.*
  - *Automatic minor version upgrades periodically update a database to recent database engine versions. However, the upgrade might not always include the latest database engine version. If you need to keep your databases on specific versions at particular times, we recommend that you manually upgrade to the database versions that you need according to your required schedule. In cases of critical security issues or when a version reaches its end-of-support date, Amazon RDS might apply a minor version upgrade even if you haven't enabled the* ***Auto minor version upgrade*** *option.*

[11]

Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda. Nella funzione `init` viene istanziato il client `rds` tramite Boto3 e viene eseguita una chiamata a `describe_db_instances` con il parametro `MaxRecords` impostato a 20 per verificare che le credenziali siano valide.

Il controllo **RDS.2** verifica che le istanze RDS non siano accessibili pubblicamente. La funzione esegue una chiamata a `describe_db_instances` per ottenere una lista di istanze di DB. Se non esiste alcuna istanza, il controllo viene marcato come passato in quanto non sono presenti istanze da verificare. Se esiste almeno un'istanza, si itera tra di esse e si estrae il valore `PubliclyAccessible`. Se il valore è esiste l'istanza viene aggiunta alla lista `non_compliant`, altrimenti alla lista `compliant`. Se esiste almeno un'istanza che è accessibile pubblicamente, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **RDS.3** verifica che le istanze RDS abbiano la crittografia a riposo abilitata. La funzione esegue una chiamata a `describe_db_instances` per ottenere una lista di istanze di DB. Se non esiste alcuna istanza, il controllo viene marcato come passato in quanto non sono presenti istanze da verificare. Se esiste almeno un'istanza, si itera tra di esse e si estrae il valore `StorageEncrypted`. Se non esiste, si aggiunge alla lista `unencrypted_instances` l'istanza (`DBInstanceIdentifier`), altrimenti alla lista `encrypted_instances`. Se esiste almeno un'istanza che non ha la crittografia a riposo abilitata, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **RDS.13** verifica che gli aggiornamenti automatici delle versioni minori siano abilitati per le istanze RDS. La funzione esegue una chiamata a

[11] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/rds-controls.html` .

`describe_db_instances` per ottenere una lista di istanze di DB. Se non esiste alcuna istanza, il controllo viene marcato come passato in quanto non sono presenti istanze da verificare. Se esiste almeno un'istanza, si itera tra di esse e si estrae il valore `AutoMinorVersionUpgrade`. Se il valore esiste, l'istanza viene aggiunta alla lista `auto_minor`, altrimenti alla lista `no_auto_minor`. Se esiste almeno un'istanza che non ha gli aggiornamenti automatici delle versioni minori abilitati, il controllo viene marcato come fallito, altrimenti passato.

La funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2. Si crea una catena di **forward** di 5 atomi: `init`, `rds_check_2`, `rds_check_3`, `rds_check_13` e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

L'**azione AWS** necessaria per eseguire la sonda è:

- `rds:DescribeDBInstances`

4.2.13 aws_eks

La sonda `aws_eks` è dedicata al servizio Amazon Elastic Kubernetes Service (EKS) che consente di creare e gestire cluster Kubernetes scalabili. La sonda verifica la conformità di EKS rispetto a 1 controllo definito da CIS Amazon EKS Benchmark v1.6.0 e 7 controlli definiti da NIST SP 800-53 Rev.5 integrati in AWS Security Hub. I controlli selezionati riguardano l'uso di versioni supportate, crittografia, tagging, logging.

- *EKS.1 - EKS cluster endpoints should not be publicly accessible*
  - *This control checks whether an Amazon EKS cluster endpoint is publicly accessible. The control fails if an EKS cluster has an endpoint that is publicly accessible.*
  - *When you create a new cluster, Amazon EKS creates an endpoint for the managed Kubernetes API server that you use to communicate with your cluster. By default, this API server endpoint is publicly available to the internet. Access to the API server is secured using a combination of AWS Identity and Access Management (IAM) and native Kubernetes Role Based Access Control (RBAC). By removing public access to the endpoint, you can avoid unintentional exposure and access to your cluster.*
- *EKS.2 - EKS clusters should run on a supported Kubernetes version*
  - *Parameter:* `oldestVersionSupported : 1.30`

  - *This control checks whether an Amazon Elastic Kubernetes Service (Amazon EKS) cluster runs on a supported Kubernetes version. The control fails if the EKS cluster runs on an unsupported version.*
  - *If your application doesn't require a specific version of Kubernetes, we recommend that you use the latest available Kubernetes version that's supported by EKS for your clusters.*

- *EKS.3 - EKS clusters should use encrypted Kubernetes secrets*
  - *This control checks whether an Amazon EKS cluster uses encrypted Kubernetes secrets. The control fails if the cluster's Kubernetes secrets aren't encrypted.*
  - *When you encrypt secrets, you can use AWS Key Management Service (AWS KMS) keys to provide envelope encryption of Kubernetes secrets stored in etcd for your cluster. This encryption is in addition to the EBS volume encryption that is enabled by default for all data (including secrets) that is stored in etcd as part of an EKS cluster. Using secrets encryption for your EKS cluster allows you to deploy a defense in depth strategy for Kubernetes applications by encrypting Kubernetes secrets with a KMS key that you define and manage.*

- *EKS.6 - EKS queues should be tagged.*
  - *Parameter:* `requiredKeyTags`
    - *Description: List of non-system tag keys that the evaluated resource must contain. Tag keys are case sensitive.*
    - *Type: StringList (maximum of 6 items)*
    - *Allowed custom values: 1-6 tag keys that meet AWS requirements.*
    - *Security Hub default value: No default value*
  - *This control checks whether an Amazon EKS cluster has tags with the specific keys defined in the parameter* `requiredTagKeys`*. The control fails if the queue doesn't have any tag keys or if it doesn't have all the keys specified in the parameter* `requiredTagKeys`*. If the parameter* `requiredTagKeys` *isn't provided, the control only checks for the existence of a tag key and fails if the queue isn't tagged with any key. System tags, which are automatically applied and begin with* `aws:`*, are ignored.*
  - *A tag is a label that you assign to an AWS resource, and it consists of a key and an optional value. You can create tags to categorize resources by purpose, owner, environment, or other criteria. Tags can help you identify, organize,*

search for, and filter resources. Tagging also helps you track accountable resource owners for actions and notifications. When you use tagging, you can implement attribute-based access control (ABAC) as an authorization strategy, which defines permissions based on tags. You can attach tags to IAM entities (users or roles) and to AWS resources. You can create a single ABAC policy or a separate set of policies for your IAM principals. You can design these ABAC policies to allow operations when the principal's tag matches the resource tag.

- *EKS.7 - EKS identity provider configurations should be tagged*
  - *Parameter:* `requiredKeyTags`
    - *Description: List of non-system tag keys that the evaluated resource must contain. Tag keys are case sensitive.*
    - *Type: StringList (maximum of 6 items)*
    - *Allowed custom values: 1-6 tag keys that meet AWS requirements.*
    - *Security Hub default value: No default value*
  - *This control checks whether an Amazon EKS identity provider configuration has tags with the specific keys defined in the parameter* `requiredTagKeys`*. The control fails if the queue doesn't have any tag keys or if it doesn't have all the keys specified in the parameter* `requiredTagKeys`*. If the parameter* `requiredTagKeys` *isn't provided, the control only checks for the existence of a tag key and fails if the queue isn't tagged with any key. System tags, which are automatically applied and begin with* `aws:`*, are ignored.*
  - *A tag is a label that you assign to an AWS resource, and it consists of a key and an optional value. You can create tags to categorize resources by purpose, owner, environment, or other criteria. Tags can help you identify, organize, search for, and filter resources. Tagging also helps you track accountable resource owners for actions and notifications. When you use tagging, you can implement attribute-based access control (ABAC) as an authorization strategy, which defines permissions based on tags. You can attach tags to IAM entities (users or roles) and to AWS resources. You can create a single ABAC policy or a separate set of policies for your IAM principals. You can design these ABAC policies to allow operations when the principal's tag matches the resource tag.*
- *EKS.8 - EKS clusters should have audit logging enabled*
  - *Parameter:* `logTypes : audit`

- *This control checks whether an Amazon EKS cluster has audit logging enabled. The control fails if audit logging isn't enabled for the cluster.*
- *EKS control plane logging provides audit and diagnostic logs directly from the EKS control plane to Amazon CloudWatch Logs in your account. You can select the log types you need, and logs are sent as log streams to a group for each EKS cluster in CloudWatch. Logging provides visibility into the access and performance of EKS clusters. By sending EKS control plane logs for your EKS clusters to CloudWatch Logs, you can record operations for audit and diagnostic purposes in a central location.*

[12]

CIS 2.1.1 - Ensure API-server audit logs are enabled on the EKS cluster

- ***Description**: Control plane logs provide visibility into operation of the EKS Control plane component systems. The API server audit logs record all accepted and rejected requests in the cluster. When enabled via EKS configuration the control plane logs for a cluster are exported to a CloudWatch Log Group for persistence.*
- ***Rationale**: Audit logs enable visibility into all API server requests from authentic and anonymous sources. Stored log data can be analyzed manually or with tools to identify and understand anomalous or negative activity and lead to intelligent remediations.*
- ***Impact**: Enabling control plane logs, including API server audit logs for Amazon EKS clusters, significantly strengthens our security posture by providing detailed visibility into all API requests, thereby reducing our attack surface. By exporting these logs to a CloudWatch Log Group, we ensure persistent storage and facilitate both manual and automated analysis to quickly identify and remediate anomalous activities. While this configuration might slightly impact usability or performance due to the overhead of logging, the enhanced security and compliance benefits far outweigh these drawbacks, making it a critical component of our security strategy.*

[13]

Da queste descrizioni si è derivata la logica di ogni controllo, che è stata implementata nella sonda. Nella funzione `init` viene istanziato il client `eks` tramite Boto3 e viene eseguita una chiamata a `list_clusters` per verificare che le credenziali siano valide.

[12] Contenuto tratto dalla documentazione ufficiale AWS Security Hub: `https://docs.aws.amazon.com/securityhub/latest/userguide/eks-controls.html` .

[13] Contenuto tratto dalla documentazione ufficiale CIS Amazon EKS Benchmark v1.6.0: `https://www.cisecurity.org/benchmark/amazon_web_services` .

Il controllo **EKS.1** verifica che gli endpoint dei cluster EKS non siano accessibili pubblicamente. La funzione esegue una chiamata a `list_clusters` per ottenere una lista di cluster. Se non esiste alcun cluster, il controllo viene marcato come passato in quanto non sono presenti cluster da verificare. Se esiste almeno un cluster, si esegue una chiamata a `describe_cluster` passando come parametro il nome del cluster. Si estrae il valore `resourcesVpcConfig` e si controlla se `EndpointPublicAccess` esiste: in questo caso se esiste almeno un cluster con l'endpoint pubblico accessibile, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **EKS.2** verifica che i cluster EKS siano eseguiti su una versione di Kubernetes supportata. La funzione esegue una chiamata a `list_clusters` per ottenere una lista di cluster. Se non esiste alcun cluster, il controllo viene marcato come passato in quanto non sono presenti cluster da verificare. Se esiste almeno un cluster, si itera tra di essi e si esegue una chiamata a `describe_cluster` passando come parametro il nome del cluster. Si estrae il valore `version` e, come da parametro `oldestVersionSupported`, si controlla se la versione è maggiore o uguale a 1.30 eseguendo una suddivisione tra major e minor version tramite funzione `lstrip` e `split`. Se esiste almeno un cluster che non è eseguito su una versione supportata, il controllo viene marcato come fallito, altrimenti passato. Di seguito l'implementazione chiave del controllo per il controllo della versione:

Listing 4.10: Controllo della versione di Kubernetes supportata in EKS

```
oldest_major, oldest_minor = 1, 30
...
# 4) parse and compare
parts = version.lstrip('v').split('.')
try:
    major, minor = int(parts[0]), int(parts[1])
    supported = (major > oldest_major) or (major == oldest_major and minor >=
oldest_minor)
except (IndexError, ValueError):
    supported = False
```

Il controllo **EKS.3** verifica che i cluster EKS utilizzino segreti Kubernetes crittografati. La funzione esegue una chiamata a `list_clusters` per ottenere una lista di cluster. Se non esiste alcun cluster, il controllo viene marcato come passato in quanto non sono presenti cluster da verificare. Se esiste almeno un cluster, si itera tra di essi e si esegue una chiamata a `describe_cluster` passando come parametro il nome del cluster. Si estrae il valore `encryptionConfig` e si controlla se esiste almeno un oggetto con `resources` uguale a `secrets`. Se esiste almeno un cluster che non utilizza segreti Kubernetes crittografati, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **EKS.6** verifica che i cluster EKS siano taggati. La funzione è del tutto analoga a quella descritta per la sonda `aws_sqs` nella sezione 4.2.1. In sostanza si analizzano i tag passati come parametro in input `clusterRequiredTagKeys` e si verifica che ogni

cluster abbia almeno uno di essi. Se esiste almeno un cluster che non ha i tag richiesti, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **EKS.7** verifica che le configurazioni del provider di identità EKS siano taggate. Anche in questo caso la funzione è analoga a quella della sonda `aws_sqs` nella sezione 4.2.1. Si analizzano i tag passati come parametro in input `idProvConfRequiredTagKeys` e si verifica che ogni configurazione del provider di identità abbia almeno uno di essi. Se esiste almeno una configurazione del provider di identità che non ha i tag richiesti, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **EKS.8** verifica che il logging di audit sia abilitato per i cluster EKS. La funzione esegue una chiamata a `list_clusters` per ottenere una lista di cluster. Se non esiste alcun cluster, il controllo viene marcato come passato in quanto non sono presenti cluster da verificare. Se esiste almeno un cluster, si itera tra di essi e si esegue una chiamata a `describe_cluster` passando come parametro il nome del cluster. Si estrae il valore `logging` e si controlla se `clusterLogging` contiene un oggetto con `types` e si controlla se esso contiene il valore `audit`. Se esiste almeno un cluster che non ha il logging di audit abilitato, il controllo viene marcato come fallito, altrimenti passato.

Il controllo **EKS 2.1.1** verifica che i log di audit dell'API server siano abilitati sui cluster EKS. La funazione è analoga a quella del controllo **EKS.8**, ma si controlla se il valore `api` è presente in `types` di `clusterLogging`. Se esiste almeno un cluster che non ha il logging di audit dell'API server abilitato, il controllo viene marcato come fallito, altrimenti passato.

La funzione `execute_all_controls` è identica a quella descritta nella sezione 4.2. Si crea una catena di **forward** di 9 atomi: `init`, `eks_check_1`, `eks_check_2`, `eks_check_3`, `eks_check_6`, `eks_check_7`, `eks_check_8`, `eks_check_2_1_1` e `execute_all_controls`. Anche la gestione delle eccezioni è identica a quella di default.

Le **azioni AWS** necessarie per eseguire la sonda sono:

- `eks:ListClusters`
- `eks:DescribeCluster`
- `eks:ListTagForResource`
- `eks:ListIdentityProviderConfigs`

4.3 Integrazione in MoonCloud

Le sonde descritte nelle sezioni precedenti sono progettate per essere eseguite e integrate in MoonCloud, come descritto nella sezione 3.4. Questo passaggio è fondamentale per

rendere eseguibili nel pratico le sonde e per permettere l'esecuzione periodica e automatizzata delle stesse, in modo da garantire un monitoraggio continuo della compliance delle risorse AWS.

4.3.1 Backend

Per ogni sonda si crea dapprima un **Control** nel backend di MoonCloud, una pagina di configurazione accessibile via Web in cui è possibile specificare i parametri di configurazione di ogni sonda eseguibile su MoonCloud. I parametri comprendono **Controls Abstract Evaluation Rules**.

Particolare attenzione è data a *Control*: qui si definisce il nome della sonda, il driver recuperato dal repository GitLab, e i **Metadata**, ovvero un form che definisce i parametri di input della sonda secondo un JSON schema creato secondo il framework *Angular Formly*. I parametri di input sono gli stessi che ogni sonda si aspetta di ricevere in input quando eseguita su MoonCloud. Un esempio di form di configurazione per la sonda `aws_sqs` il seguente:

```
{
  "target":{
    "section":"config",
    "type":"host"
  },
  "description":"AWS SQS Parameters",
  "schema":{
    "config":{
      "properties":{
        "region":{
          "title":"AWS Region",
          "x-form":{
            "form_type":"input-fix-label"
          },
          "x-schema-form":{
            "type":"text"
          }
        },
        "requiredTagKeys":{
          "title":"Interested Queue Tags",
          "default":"",
          "x-form":{
            "form_type":"input-fix-label"
          },
          "x-schema-form":{
```

```
            "type":"string"
          }
        }
      }
    }
  }
}
```

Listing 4.11: Form di configurazione per la sonda aws_sqs

In questo caso per l'implementazione multiregione il campo region è una stringa, che viene poi convertita in una lista di regioni direttamente all'interno della sonda. Invece, per la maggior parte delle sonde il campo region è una selezione tra le regioni AWS disponibili, come ad esempio per la sonda aws_cloudtrail:

```
{
    "target": {
        "section": "config",
        "type": "host"
    },
    "description": "AWS Cloudtrail Parameters",
    "schema": {
        "config": {
            "properties": {
                "region": {
                    "title": "AWS Region",
                    "options": {
                        "us-east-1": {
                            "title": "us-east-1"
                        },
                        "us-east-2": {
                            "title": "us-east-2"
                        },
                        "us-west-1": {
                            "title": "us-west-1"
                        },
                        "us-west-2": {
                            "title": "us-west-2"
                        },

                        ...

                        "cn-north-1": {
                            "title": "cn-north-1"
```

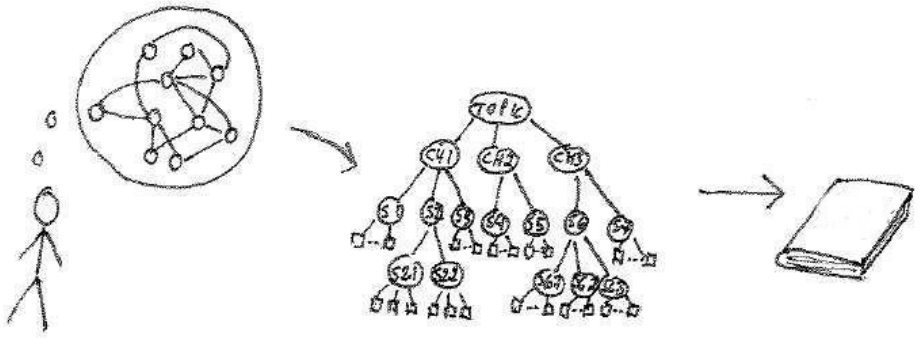

Figura 1: Frontend di configurazione della sonda `aws_cloudtrail` su MoonCloud.

```
                        },
                        "cn-northwest-1": {
                            "title": "cn-northwest-1"
                        }
                    },
                    "x-form": {
                        "form_type": "select-fix-label"
                    },
                    "x-schema-form": {
                        "type": "text"
                    }
                }
            }
        }
    }
}
```

Listing 4.12: Form di configurazione per la sonda `aws_cloudtrail`

In questo modo, quando si crea un **Control** per ogni sonda, sulla dashboard di MoonCloud viene visualizzato in frontend il form definito in JSON, che consente di configurare la sonda con i parametri richiesti, come mostrato in Figura 1.

L'*Abstract Evaluation Rule* invece è necessaria per presentare la sonda sulla dashboard di MoonCloud, in modo che l'utente possa vederla se marcata come pubblica, identificarla anche grazie ad un'immagine di copertina, filtrarla per categoria.

4.3.2 Dashboard

La dashboard di MoonCloud è accessibile via Web e consente di creare il proprio spazio operativo in cui definire *Zone* da analizzare, quali sonde eseguire e su quali *Target*, con quali credenziali eseguire le sonde e con quali parametri di configurazione.

Un account può creare più zone per suddividere le risorse da analizzare, ad esempio in base all'infrastruttura cloud pubblica, come nel caso di AWS, o in base all'infrastruttura privata come in caso di repository GitLab privati, oppure piattaforme di container come Kubernetes o Docker.

In ogni zona è possibile creare dei *Target*, ovvero le risorse da analizzare definite da un URL e a cui è possibile associare diversi tipi di credenziali, da specificare nel backend durante la creazione del controllo: tutte le sonde si basano sulla coppia `Username-Password` che per il caso specifico di AWS è *Access Key ID* e *Secret Access Key*; altre tipologie di credenziali sono `Git Login`, `SSH Key`.

Una volta definito il target, nella sezione *Evaluations* è possibile creare un *Evaluation*, ovvero eseguire una sonda che che produce un risultato, una valutazione, che può essere positiva o negativa. Durante la creazione si seleziona la sonda di interesse, si associa ad un Target, si specificano i parametri di configurazione definiti nel form specifico della sonda ed essa viene eseguita. Inoltre, è possibile specificare se si tratta di un esecuzione *One Shot* o *Scheduled*, ovvero se si desidera eseguire la sonda una sola volta o se si desidera pianificarne l'esecuzione periodica ogni ora o ogni giorno.

Quando viene eseguita il risultato JSON mostrato nella sezione 4.2 viene mostrato in frontend tramite un menù a tendina che consente di espandere i risultati `raw` e visualizzare le informazioni di dettaglio.

Per quanto riguarda i risultati `refined` invece, essi oltre alla visualizzazione mostrata sopra hanno una sezione dedicata nel menù della dashboard, in cui è possibile ogni CVE e visualizzare informazioni come il nome, la descrizione, il punteggio CVSS per cui è possibile anche l'ordinamento, e il nome del controllo che l'ha generata:

Da ultimo, pagina principale mostra dei grafici a torta che mostrano la percentuale di compliance dell'account e di ogni zona, facendo il rapporto tra i controlli passati e quelli falliti, riportando anche la percentuale numerica. In aggiunta sono disposte al di sotto una parte delle sonde eseguite e il loro stato dopo l'ultima esecuzione.

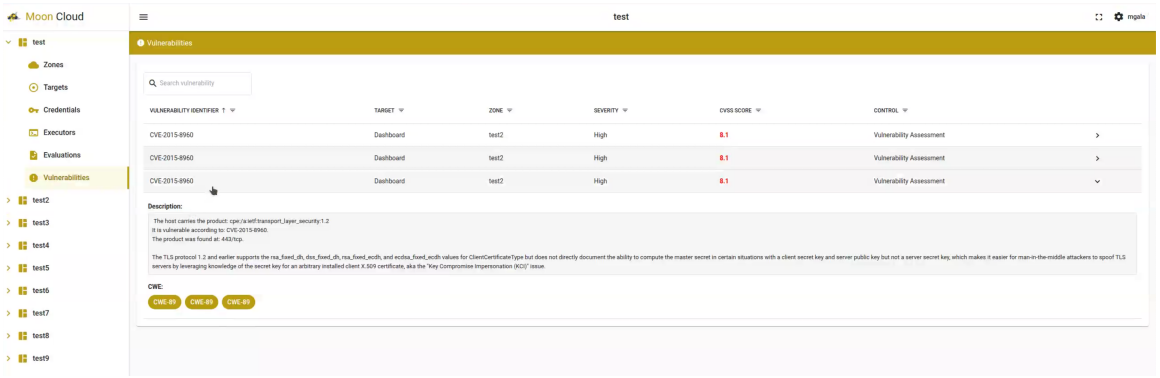

Figura 2: Esempio di visualizzazione delle CVE su MoonCloud.

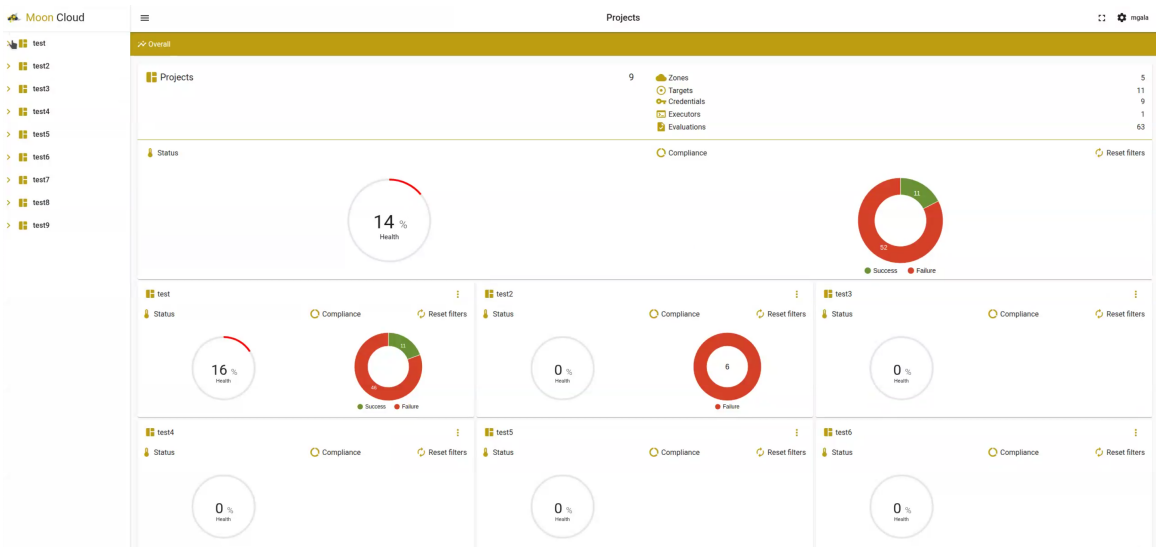

Figura 3: Dashboard di MoonCloud con i grafici di compliance.

La dashboard di MoonCloud è l'interfaccia principale del sistema, che consente di gestire le sonde, i target e le valutazioni, e di visualizzare i risultati delle sonde eseguite. Grazie all'interfaccia intuitiva e alle funzionalità di filtraggio e ordinamento, gli utenti possono facilmente navigare tra le sonde, i target e le valutazioni, e ottenere una visione chiara dello stato di compliance delle proprie risorse cloud.

4.4 Demonstration

Per dimostrare l'esecuzione delle sonde e la loro integrazione in MoonCloud, esse sono state eseguite su un account AWS di test, in cui sono presenti 3 Zone:

- **IaaS**: zona dedicata alle infrastrutture e alle risorse non gestite come IAM, CloudTrail
- **PaaS**: zona dedicata alle piattaforme come EC2, RDS, EKS, SQS
- **SaaS**: zona dedicata ai servizi come S3, Lambda functions

Le sonde sono state catalogate in base alla loro tipologia e sono state eseguite su ogni zona, in modo da verificare la loro funzionalità e l'integrazione con MoonCloud.

Questo esempio dimostra come le sonde siano in grado di analizzare le risorse AWS in un contesto reale, assimilabile ad un ambiente di produzione aziendale, e come MoonCloud sia in grado di presentare i risultati delle sonde eseguite, consentendo agli utenti di monitorare la compliance delle proprie risorse cloud in modo efficace.

Capitolo 5

Conclusioni

L'obiettivo di questo lavoro è stato quello di sviluppare delle soluzioni per la verifica della compliance in ambienti Cloud, in particolare AWS. La prima parte del progetto ha previsto l'analisi delle direttive CIS e NIST. Queste erano disponibili direttamente nella documentazione web di AWS Security Hub, da cui è stato semplice estrarre le richieste di ogni controllo. Per quanto riguarda invece documenti più specifici, come il CIS Amazon Elastic Kubernetes Service Benchmark, è stata necessaria la consultazione dell'intero documento per individuare i controlli pertinenti.

La scelta dei controlli da implementare è stata fatta in modo da aderire completamente alle direttive del CIS AWS Foundations Benchmark, includendo anche direttive NIST SP 800-53 per sonde riguardanti servizi come SQS, Inspector ed EKS, non trattate nel CIS.

Una volta compresi i requisiti, si è proceduto con lo studio approfondito del funzionamento della piattaforma MoonCloud e del suo driver per la realizzazione delle sonde. Questa fase ha rappresentato una delle principali sfide del progetto, poiché la comprensione della struttura delle sonde e dei parametri richiesti per l'esecuzione dei controlli è risultata fondamentale per garantirne l'integrazione corretta nella piattaforma. La programmazione delle sonde è stata effettuata in Python, utilizzando la libreria Boto3 per interagire con i servizi AWS. Anche in questo caso, la consultazione approfondita della documentazione dei client Boto3 si è rivelata cruciale per comprendere quali funzioni restituissero i parametri utili ai fini del controllo.

Successivamente si è proceduto con l'integrazione delle sonde nella piattaforma MoonCloud, a partire dalla creazione nel backend di *Control* e *Abstract Evaluation Rule*. Questi sono necessari per creare una sonda eseguibile, associarvi l'immagine Docker costruita dalla pipeline CI/CD, creare un form per i parametri di input e catalogare la sonda nella dashboard di MoonCloud.

L'ultimo passaggio ha previsto l'esecuzione delle sonde e la verifica dei risultati ottenuti sull'account di demo. I risultati sono stati soddisfacenti, dimostrando che le sonde

sono in grado di verificare la compliance dei servizi AWS rispetto ai controlli implementati. Le sonde hanno prodotto output coerenti con le aspettative e le istanze di AWS sono state correttamente valutate in base ai criteri stabiliti.

Le principali sfide affrontate sono state dunque la varietà delle API tra i diversi servizi AWS e la necessità di standardizzare input e output delle sonde per consentirne l'integrazione nella dashboard di MoonCloud. Le soluzioni implementate hanno portato allo sviluppo di un sistema modulare, flessibile e capace di essere eseguito in modo automatizzato su diversi target AWS.

5.1 Sviluppi futuri

Tra i possibili sviluppi futuri del progetto si individuano due principali direzioni:

- **Estensione della copertura dei controlli**: attualmente le sonde sviluppate coprono integralmente il *CIS AWS Foundations Benchmark* e alcuni controlli tratti dalle raccomandazioni *NIST SP 800-53*. Per servizi specifici come SQS e Inspector, sono stati inclusi anche controlli suggeriti dalla documentazione di *AWS Security Hub*. Inoltre, sono stati implementati controlli per EKS, seguendo il *CIS Amazon EKS Benchmark*. Tuttavia, esistono altri benchmark CIS per ulteriori servizi AWS che non sono ancora stati coperti e che rappresentano una possibile evoluzione futura. Anche *AWS Security Hub* propone controlli basati su altri standard di sicurezza, come *PCI-DSS* o ulteriori direttive *NIST*. L'obiettivo è quello di estendere la copertura delle sonde, per aumentare la superficie di compliance e garantire una verifica più completa delle risorse AWS.

- **Estensione dell'implementazione multiregione**: attualmente solo la sonda `aws_sqs` supporta la verifica in più regioni AWS. L'obiettivo è estendere questa funzionalità a tutte le sonde, in modo da garantire una copertura completa e centralizzata delle verifiche di compliance in scenari distribuiti su più regioni.

- **Adattamento ad altri cloud provider**: la struttura modulare delle sonde e l'architettura della piattaforma MoonCloud costituiscono una buona base per estendere le verifiche di compliance anche ad altri ambienti cloud, come Microsoft Azure o Google Cloud Platform. Naturalmente, ciò richiederà modifiche alle sonde per adattarle alle API e ai servizi specifici di ciascun provider. Questo permetterebbe di ampliare l'utilizzo della piattaforma MoonCloud e di offrire un sistema di compliance più ampio.

In conclusione, il progetto ha raggiunto gli obiettivi prefissati, dimostrando la fattibilità di implementare un sistema di verifica della compliance in ambienti cloud mediante l'utilizzo di MoonCloud. I risultati ottenuti offrono una base solida per ulteriori sviluppi,

contribuendo a rendere la sicurezza e la compliance nel cloud più accessibili e gestibili, anche in contesti aziendali dove si desidera una supervisione continua e automatizzata della sicurezza in questo ambito.

Appendice A

aws_sqs

```
#! /usr/bin/env python3

import copy
import re
import boto3 # type: ignore
import os
import requests # type: ignore
import json
import typing
from mooncloud_driver import abstract_probe, atom, result, entrypoint # type:
ignore

class Probe(abstract_probe.AbstractProbe):

    def requires_credential(self):
        return True

    def init(self, inputs=None) -> bool:

        self.control_results = {}
        self.collected_data = {}

        config = self.config.input.get('config')
        # # Local load method
        # access_key_id = config.get('username')
        # secret_access_key = config.get('password')
        access_key_id = self.config.credential.get('username')
        secret_access_key = self.config.credential.get('password')

```

```
        raw_regions = ''
        try:
            raw_regions = config.get('region', '')
        except Exception:
            raw_regions = ''

        # split on commas, semicolons or whitespace into list
        if isinstance(raw_regions, str):
            raw_regions = raw_regions.strip()
            specific_regions = re.split(r'[,;\s]+', raw_regions) if raw_regions
 else []
        elif isinstance(raw_regions, list):
            specific_regions = raw_regions
        else:
            specific_regions = []

        #max 6 regions
        if len(specific_regions) > 6:
            specific_regions = specific_regions[:6]

        assert access_key_id is not None, "AWS Access Key ID is missing"
        assert secret_access_key is not None, "AWS Secret Access Key is missing
"

        self.clients = {}
        # ensure at least one region
        if not specific_regions:
            specific_regions = ['eu-central-1']
        for idx, region in enumerate(specific_regions, start=1):
            self.clients[f'client_{idx}'] = boto3.client(
            'sqs',
            aws_access_key_id=access_key_id,
            aws_secret_access_key=secret_access_key,
            region_name=region
            )

        self.clients['client_1'].list_queues(MaxResults=1)

        return True

    def sqs_control_1(self, inputs=None) -> bool:
        """
        SQS Control 1 - Check if SQS queues are encrypted at rest
```

```
        """
        region_results = {}

        for client_name, client in self.clients.items():
            region = client.meta.region_name
            response = client.list_queues()

            encrypted_queues = []
            unencrypted_queues = []

            if 'QueueUrls' in response:
                for queue_url in response['QueueUrls']:
                    queue_name = queue_url.split('/')[-1]
                    attr_response = client.get_queue_attributes(
                        AttributeNames=['SqsManagedSseEnabled'],
                        QueueUrl=queue_url
                    )
                    if (attr_response.get('Attributes', {})
                            .get('SqsManagedSseEnabled') == 'true'):
                        encrypted_queues.append({'name': queue_name, 'url': queue_url})
                    else:
                        unencrypted_queues.append({'name': queue_name, 'url': queue_url})
            # store per-region collected_data
            self.collected_data[f"CIS SQS.1 - {region} - Encrypted Queues"] = encrypted_queues
            self.collected_data[f"CIS SQS.1 - {region} - Unencrypted Queues"] = unencrypted_queues

            # compute pass/fail & message per region
            passed = (len(unencrypted_queues) == 0)
            if 'QueueUrls' not in response:
                pretty = "No SQS queues found in the account"
            elif unencrypted_queues:
                pretty = (
                    f"Found {len(unencrypted_queues)} unencrypted queues out of "
                    f"{len(encrypted_queues)+len(unencrypted_queues)} total queues"
                )
            else:
```

```
                pretty = f"All {len(encrypted_queues)} queues are properly encrypted at rest"

            region_results[region] = {
                "Passed": passed,
                "Result": pretty
            }

        # multi-region control
        self.control_results['CIS SQS.1'] = region_results
        return True

    def sqs_control_2(self, inputs=None) -> bool:
        """
        SQS Control 2 - Check if SQS queues are tagged
        """
        region_results = {}

        # Normalize specific_tags once
        raw_tags = ''
        try:
            cfg = self.config.input.get('config', {})
            raw_tags = cfg.get('requiredTagKeys', '')
        except Exception:
            raw_tags = ''
        if isinstance(raw_tags, str):
            raw_tags = raw_tags.strip()
            specific_tags = re.split(r'[,;\s]+', raw_tags) if raw_tags else []
        elif isinstance(raw_tags, list):
            specific_tags = raw_tags
        else:
            specific_tags = []

        # limit to 6
        specific_tags = specific_tags[:6]

        for client_name, client in self.clients.items():
            region = client.meta.region_name
            response = client.list_queues()

            tagged = []
            untagged = []
            missing = []
```

```
148
149             if 'QueueUrls' in response:
150                 for url in response['QueueUrls']:
151                     name = url.split('/')[-1]
152                     tags = client.list_queue_tags(QueueUrl=url).get('Tags', {})
153                     user = {k: v for k, v in tags.items() if not k.startswith('
aws:')}
154                     if specific_tags:
155                         miss = [t for t in specific_tags if t not in user]
156                         if miss:
157                             missing.append({'name': name, 'url': url, 'current_
tags': user, 'missing_tags': miss})
158                         else:
159                             tagged.append({'name': name, 'url': url, 'tags':
user})
160                     else:
161                         (tagged if user else untagged).append({'name': name, '
url': url, **({'tags': user} if user else {})})
162                 total = len(tagged) + (len(missing) if specific_tags else len(
untagged))
163                 # save per-region details
164                 self.collected_data[f"CIS SQS.2 - {region} - Tagged Queues"] =
tagged
165                 self.collected_data[f"CIS SQS.2 - {region} - Untagged Queues"]
= untagged
166                 self.collected_data[f"CIS SQS.2 - {region} - Queues Missing
Required Tags"] = missing
167                 self.collected_data[f"CIS SQS.2 - {region} - Required Tags"] =
specific_tags
168
169                 if not response['QueueUrls']:
170                     ok, msg = True, "No SQS queues found in the account"
171                 elif specific_tags:
172                     ok = not missing
173                     msg = (f"All {len(tagged)} queues have the required tags"
174                             if ok else
175                             f"Found {len(missing)} queues missing required tags
 out of {total} total queues")
176                 else:
177                     ok = not untagged
178                     msg = (f"All {len(tagged)} queues have at least one user-
defined tag"
179                             if ok else
```

```
                            f"Found {len(untagged)} queues with no user-defined
 tags out of {total} total queues")
            else:
                ok, msg = True, "No SQS queues found in the account"
                # still record empty lists
                for kind in ("Tagged Queues", "Untagged Queues", "Queues
Missing Required Tags"):
                    self.collected_data[f"CIS SQS.2 - {region} - {kind}"] = []
                self.collected_data[f"CIS SQS.2 - {region} - Required Tags"] =
specific_tags

            region_results[region] = {"Passed": ok, "Result": msg}

        self.control_results['CIS SQS.2'] = region_results
        return True

    def sqs_control_3(self, inputs=None) -> bool:
        """
        SQS Control 3 - Check if SQS queues are publicly accessible
        """
        region_results = {}

        for client_name, client in self.clients.items():
            region = client.meta.region_name
            response = client.list_queues()

            public_queues = []
            private_queues = []

            if 'QueueUrls' in response:
                for queue_url in response['QueueUrls']:
                    queue_name = queue_url.split('/')[-1]
                    policy_resp = client.get_queue_attributes(
                        AttributeNames=['Policy'],
                        QueueUrl=queue_url
                    )
                    policy_str = policy_resp.get('Attributes', {}).get('Policy'
)
                    is_public = False

                    if policy_str:
                        policy = json.loads(policy_str)
                        for stmt in policy.get('Statement', []):
```

```
                            if stmt.get('Effect') == 'Allow':
                                princ = stmt.get('Principal', {})
                                if princ == "*" or princ.get('AWS') == "*":
                                    is_public = True
                                    break

                    if is_public:
                        public_queues.append({
                            'name': queue_name,
                            'url': queue_url,
                            'policy': policy_str
                        })
                    else:
                        private_queues.append({
                            'name': queue_name,
                            'url': queue_url,
                            'policy': policy_str
                        })

                self.collected_data[f"CIS SQS.3 - {region} - Public Queues"] = public_queues
                self.collected_data[f"CIS SQS.3 - {region} - Private Queues"] = private_queues

                if public_queues:
                    passed = False
                    pretty = (
                        f"Found {len(public_queues)} queues with public access "
                        f"out of {len(public_queues)+len(private_queues)} total queues"
                    )
                else:
                    passed = True
                    pretty = f"All {len(private_queues)} queues have private access policies"
            else:
                # no queues in this region
                self.collected_data[f"CIS SQS.3 - {region} - Public Queues"] = []
                self.collected_data[f"CIS SQS.3 - {region} - Private Queues"] = []
                passed = True
```

```
255                 pretty = "No SQS queues found in the account"
256
257             region_results[region] = {"Passed": passed, "Result": pretty}
258
259         self.control_results['CIS SQS.3'] = region_results
260         return True
261
262     def execute_all_controls(self, inputs=None) -> bool:
263         passed_controls = sum(
264             1 for c in self.control_results.values()
265             if (
266             isinstance(c, dict)
267             and all(isinstance(v, dict) for v in c.values())
268             and all(v.get("Passed") for v in c.values())
269             ) or (c.get("Passed") is True)
270         )
271         total_controls = len(self.control_results)
272
273         self.result.put_raw_extra_data("Summary", f"{passed_controls}/{total_
controls} succeeded")
274
275         for name, ctrl in self.control_results.items():
276             # dump multi-region control and merge in per-region collected_data
277             if isinstance(ctrl, dict) and all(isinstance(v, dict) for v in ctrl
.values()):
278                 merged = {}
279                 for region, info in ctrl.items():
280                     block = {
281                         "Passed": info["Passed"],
282                         "Result": info["Result"]
283                     }
284                     prefix = f"{name} - {region} - "
285                     for k, v in self.collected_data.items():
286                         if k.startswith(prefix):
287                             field = k[len(prefix):]
288                             block[field] = v
289                     merged[region] = block
290                 self.result.put_raw_extra_data(name, merged)
291                 continue
292
293             # single-block controls
294             block = {
295                 "Passed": ctrl["Passed"],
```

```
                "Result": ctrl["Result"],
            }
            prefix = f"{name} - "
            for k, v in self.collected_data.items():
                if k.startswith(prefix):
                    field = k[len(prefix):]
                    block[field] = v
            self.result.put_raw_extra_data(name, block)

        if total_controls == 0:
            self.result.integer_result = 2
            self.result.pretty_result = "Credential error: AWS credentials are
missing or invalid"
        else:
            self.result.integer_result = 0 if passed_controls == total_controls
 else 1
            self.result.pretty_result = "The probe executed successfully!"

        return True

    def atoms(self) -> typing.Sequence[atom.AtomPairWithException]:
        return [
            atom.AtomPairWithException(
                forward=self.init,
                forward_captured_exceptions=[
                    atom.PunctualExceptionInformationForward(
                        exception_class=Exception,
                        action=atom.OnExceptionActionForward.STOP,
                        result_producer=lambda e: result.Result(
                            integer_result=2,
                            pretty_result="Credential error: AWS credentials
missing or invalid",
                            base_extra_data=result.Extradata(raw={"
ExceptionRecovered": str(e)})
                        )
                    )
                ]
            ),
            atom.AtomPairWithException(
                forward=self.sqs_control_1,
                forward_captured_exceptions=[
                    atom.PunctualExceptionInformationForward(
                        exception_class=Exception,
```

```
335                         action=atom.OnExceptionActionForward.GO_ON,
336                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
337                     )
338                 ]
339             ),
340             atom.AtomPairWithException(
341                 forward=self.sqs_control_2,
342                 forward_captured_exceptions=[
343                     atom.PunctualExceptionInformationForward(
344                         exception_class=Exception,
345                         action=atom.OnExceptionActionForward.GO_ON,
346                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
347                     )
348                 ]
349             ),
350             atom.AtomPairWithException(
351                 forward=self.sqs_control_3,
352                 forward_captured_exceptions=[
353                     atom.PunctualExceptionInformationForward(
354                         exception_class=Exception,
355                         action=atom.OnExceptionActionForward.GO_ON,
356                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
357                     )
358                 ]
359             ),
360             atom.AtomPairWithException(
361                 forward=self.execute_all_controls,
362                 forward_captured_exceptions=[
363                     atom.PunctualExceptionInformationForward(
364                         exception_class=Exception,
365                         action=atom.OnExceptionActionForward.GO_ON,
366                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
367                     )
368                 ]
369             )
370         ]
371
372 if __name__ == "__main__":
373     entrypoint.start_execution(Probe)
```

Appendice B

aws_inspector

```
#! /usr/bin/env python3

import boto3 # type: ignore
import typing
import datetime
from mooncloud_driver import abstract_probe, atom, result, entrypoint # type:
ignore

class Probe(abstract_probe.AbstractProbe):

    def requires_credential(self):
        return True

    def init(self, inputs=None) -> bool:

        self.control_results = {}
        self.collected_data = {}

        config = self.config.input.get('config')
        # # Local load method
        # access_key_id = config.get('username')
        # secret_access_key = config.get('password')
        access_key_id = self.config.credential.get('username')
        secret_access_key = self.config.credential.get('password')
        region = config.get('region')

        assert access_key_id is not None, "AWS Access Key ID is missing"
        assert secret_access_key is not None, "AWS Secret Access Key is missing
"
```

```
28
29          self.client = boto3.client(
30              'inspector2',
31              aws_access_key_id = access_key_id,
32              aws_secret_access_key = secret_access_key,
33              region_name = region or 'eu-central-1'
34          )
35
36          self.client.list_members(maxResults=1)
37
38          return True
39
40      def inspector_check_1(self, inputs=None) -> bool:
41          """
42          Inspector.1 - Amazon Inspector EC2 scanning should be enabled
43          """
44          response = self.client.batch_get_account_status()
45          current_account = response.get('accounts', [{}])[0]
46          ec2_status = current_account.get('resourceState', {}).get('ec2', {}).
get('status')
47          admin_enabled = ec2_status == 'ENABLED'
48
49          members = self.client.list_members().get('members', [])
50          compliant, non_compliant = [], []
51
52          if not members:
53              acct = current_account.get('accountId', 'current')
54              (compliant if admin_enabled else non_compliant).append(acct)
55              pretty = (
56                  "Amazon Inspector EC2 scanning is properly enabled"
57                  if admin_enabled else
58                  "Amazon Inspector EC2 scanning is not enabled"
59              )
60          else:
61              # delegated admin + members
62              root_id = current_account.get('accountId', 'admin')
63              (compliant if admin_enabled else non_compliant).append(root_id)
64
65              for m in members:
66                  sid = m.get('accountId')
67                  st = m.get('resourceState', {}).get('ec2', {}).get('status')
68                  rel = m.get('relationshipStatus')
69                  if st == 'ENABLED' or rel == 'SUSPENDED':
```

```
                    compliant.append(sid)
                else:
                    non_compliant.append(sid)

            if non_compliant:
                pretty = (
                    f"Found {len(non_compliant)} accounts without Inspector EC2 scanning "
                    f"enabled out of {len(compliant)+len(non_compliant)} total accounts"
                )
            else:
                pretty = f"Amazon Inspector EC2 scanning is enabled for all {len(compliant)} accounts"

        passed = len(non_compliant) == 0
        self.control_results["CIS Inspector.1"] = {
            "Passed": passed,
            "Result": pretty
        }
        return True

    def inspector_check_2(self, inputs=None) -> bool:
        """
        Inspector.2 - Amazon Inspector ECR scanning should be enabled
        """
        response = self.client.batch_get_account_status()
        current_account = response.get('accounts', [{}])[0]
        ecr_status = current_account.get('resourceState', {}).get('ecr', {}).get('status')
        admin_enabled = ecr_status == 'ENABLED'

        members = self.client.list_members().get('members', [])
        compliant, non_compliant = [], []

        if not members:
            acct = current_account.get('accountId', 'current')
            (compliant if admin_enabled else non_compliant).append(acct)
            pretty = (
                "Amazon Inspector ECR scanning is properly enabled"
                if admin_enabled else
                "Amazon Inspector ECR scanning is not enabled"
            )
```

```
        else:
            # delegated admin + members
            root_id = current_account.get('accountId', 'admin')
            (compliant if admin_enabled else non_compliant).append(root_id)

            for m in members:
                sid = m.get('accountId')
                st = m.get('resourceState', {}).get('ecr', {}).get('status')
                rel = m.get('relationshipStatus')
                if st == 'ENABLED' or rel == 'SUSPENDED':
                    compliant.append(sid)
                else:
                    non_compliant.append(sid)

            if non_compliant:
                pretty = (
                    f"Found {len(non_compliant)} accounts without Inspector ECR scanning "
                    f"enabled out of {len(compliant)+len(non_compliant)} total accounts"
                )
            else:
                pretty = f"Amazon Inspector ECR scanning is enabled for all {len(compliant)} accounts"

        passed = len(non_compliant) == 0
        self.control_results["CIS Inspector.2"] = {
            "Passed": passed,
            "Result": pretty
        }
        return True

    def inspector_check_3(self, inputs=None) -> bool:
        """
        Inspector.3 - Amazon Inspector Lambda code scanning should be enabled
        """
        response = self.client.batch_get_account_status()
        current_account = response.get('accounts', [{}])[0]
        lambda_code_status = current_account.get('resourceState', {}).get('lambdaCode', {}).get('status')
        admin_enabled = lambda_code_status == 'ENABLED'

        members = self.client.list_members().get('members', [])
```

```
148         compliant, non_compliant = [], []
149 
150         if not members:
151             acct = current_account.get('accountId', 'current')
152             (compliant if admin_enabled else non_compliant).append(acct)
153             pretty = (
154                 "Amazon Inspector Lambda code scanning is properly enabled"
155                 if admin_enabled else
156                 "Amazon Inspector Lambda code scanning is not enabled"
157             )
158         else:
159             # delegated admin + members
160             root_id = current_account.get('accountId', 'admin')
161             (compliant if admin_enabled else non_compliant).append(root_id)
162 
163             for m in members:
164                 sid = m.get('accountId')
165                 st = m.get('resourceState', {}).get('lambdaCode', {}).get('
status')
166                 rel = m.get('relationshipStatus')
167                 if st == 'ENABLED' or rel == 'SUSPENDED':
168                     compliant.append(sid)
169                 else:
170                     non_compliant.append(sid)
171 
172             if non_compliant:
173                 pretty = (
174                     f"Found {len(non_compliant)} accounts without Inspector
Lambda code scanning "
175                     f"enabled out of {len(compliant)+len(non_compliant)} total
accounts"
176                 )
177             else:
178                 pretty = f"Amazon Inspector Lambda code scanning is enabled for
 all {len(compliant)} accounts"
179 
180         passed = len(non_compliant) == 0
181         self.control_results["CIS Inspector.3"] = {
182             "Passed": passed,
183             "Result": pretty
184         }
185         return True
186 
```

```
    def inspector_check_4(self, inputs=None) -> bool:
        """
        Inspector.4 - Amazon Inspector Lambda standard scanning should be enabled
        """
        response = self.client.batch_get_account_status()
        current_account = response.get('accounts', [{}])[0]
        lambda_status = current_account.get('resourceState', {}).get('lambda', {}).get('status')
        admin_enabled = lambda_status == 'ENABLED'

        members = self.client.list_members().get('members', [])
        compliant, non_compliant = [], []

        if not members:
            acct = current_account.get('accountId', 'current')
            (compliant if admin_enabled else non_compliant).append(acct)
            pretty = (
                "Amazon Inspector Lambda standard scanning is properly enabled"
                if admin_enabled else
                "Amazon Inspector Lambda standard scanning is not enabled"
            )
        else:
            # delegated admin + members
            root_id = current_account.get('accountId', 'admin')
            (compliant if admin_enabled else non_compliant).append(root_id)

            for m in members:
                sid = m.get('accountId')
                st = m.get('resourceState', {}).get('lambda', {}).get('status')
                rel = m.get('relationshipStatus')
                if st == 'ENABLED' or rel == 'SUSPENDED':
                    compliant.append(sid)
                else:
                    non_compliant.append(sid)

            if non_compliant:
                pretty = (
                    f"Found {len(non_compliant)} accounts without Inspector Lambda standard scanning "
                    f"enabled out of {len(compliant)+len(non_compliant)} total accounts"
                )
```

```
226             else:
227                 pretty = f"Amazon Inspector Lambda standard scanning is enabled
 for all {len(compliant)} accounts"
228 
229         passed = len(non_compliant) == 0
230         self.control_results["CIS Inspector.4"] = {
231             "Passed": passed,
232             "Result": pretty
233         }
234         return True
235 
236     def execute_all_controls(self, inputs=None) -> bool:
237         passed_controls = sum(1 for c in self.control_results.values() if c["
Passed"])
238         total_controls = len(self.control_results)
239         self.result.put_raw_extra_data("Summary", f"{passed_controls}/{total_
controls} succeeded")
240 
241         for name, ctrl in self.control_results.items():
242             if name == "CVE Scan":
243                 continue
244             block = {
245                 "Passed": ctrl["Passed"],
246                 "Result": ctrl["Result"],
247             }
248             prefix = f"{name} - "
249             for k, v in self.collected_data.items():
250                 if k.startswith(prefix):
251                     field = k[len(prefix):]
252                     block[field] = v
253             self.result.put_raw_extra_data(name, block)
254 
255         if passed_controls == 0:
256             if total_controls == 0:
257                 self.result.integer_result = 2
258                 self.result.pretty_result = "Credential error: AWS credentials
are missing or invalid"
259         else:
260             self.result.integer_result = 0 if passed_controls == total_controls
 else 1
261             self.result.pretty_result = "The probe executed successfully!"
262 
263         return True
```

```
264
265     def atoms(self) -> typing.Sequence[atom.AtomPairWithException]:
266         return [
267             atom.AtomPairWithException(
268                 forward=self.init,
269                 forward_captured_exceptions=[
270                     atom.PunctualExceptionInformationForward(
271                         exception_class=Exception,
272                         action=atom.OnExceptionActionForward.STOP,
273                         result_producer=lambda e: result.Result(
274                             integer_result=2,
275                             pretty_result="Credential error: AWS credentials
missing or invalid",
276                             base_extra_data=result.Extradata(raw={"
ExceptionRecovered": str(e)})
277                         )
278                     )
279                 ]
280             ),
281             atom.AtomPairWithException(
282                 forward=self.inspector_check_1,
283                 forward_captured_exceptions=[
284                     atom.PunctualExceptionInformationForward(
285                         exception_class=Exception,
286                         action=atom.OnExceptionActionForward.GO_ON,
287                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
288                     )
289                 ]
290             ),
291             atom.AtomPairWithException(
292                 forward=self.inspector_check_2,
293                 forward_captured_exceptions=[
294                     atom.PunctualExceptionInformationForward(
295                         exception_class=Exception,
296                         action=atom.OnExceptionActionForward.GO_ON,
297                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
298                     )
299                 ]
300             ),
301             atom.AtomPairWithException(
302                 forward=self.inspector_check_3,
```

```
303                 forward_captured_exceptions=[
304                     atom.PunctualExceptionInformationForward(
305                         exception_class=Exception,
306                         action=atom.OnExceptionActionForward.GO_ON,
307                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
308                     )
309                 ]
310             ),
311             atom.AtomPairWithException(
312                 forward=self.inspector_check_4,
313                 forward_captured_exceptions=[
314                     atom.PunctualExceptionInformationForward(
315                         exception_class=Exception,
316                         action=atom.OnExceptionActionForward.GO_ON,
317                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
318                     )
319                 ]
320             ),
321             atom.AtomPairWithException(
322                 forward=self.execute_all_controls,
323                 forward_captured_exceptions=[
324                     atom.PunctualExceptionInformationForward(
325                         exception_class=Exception,
326                         action=atom.OnExceptionActionForward.GO_ON,
327                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
328                     )
329                 ]
330             )
331         ]
332
333 if __name__ == "__main__":
334     entrypoint.start_execution(Probe)
```

Appendice C

aws_vulnerability

```
#! /usr/bin/env python3

import boto3 #type: ignore
import json
import typing
import datetime
from mooncloud_driver import abstract_probe, atom, result, entrypoint # type:
ignore

class Probe(abstract_probe.AbstractProbe):

    def _serialize_datetimes(self, obj):
        if isinstance(obj, datetime.datetime):
            return obj.isoformat()
        raise TypeError(f"Type {type(obj)} not serializable")

    def requires_credential(self):
       return True

    def init(self, inputs=None) -> bool:

        config = self.config.input.get('config')
        # # Local load method
        # access_key_id = config.get('username')
        # secret_access_key = config.get('password')
        access_key_id = self.config.credential.get('username')
        secret_access_key = self.config.credential.get('password')
        region = config.get('region')

```

```
        cvss = config.get('cvssthreshold')
        # parse and clamp CVSS threshold
        try:
            cvss = float(cvss)
        except (TypeError, ValueError):
            cvss = 0.0
        if cvss >= 10:
            cvss = 10.0
        self.cvss_th = cvss
        self.skip = False
        raw = config.get('targetinstances', ['ALL'])
        if isinstance(raw, str):
            raw = [raw]
        elif not isinstance(raw, list):
            raw = ['ALL']
        allowed = ["AWS_LAMBDA_FUNCTION", "AWS_EC2_INSTANCE", "AWS_ECR_
CONTAINER_IMAGE"]
        self.target = [t for t in raw if t in allowed]
        if not self.target:
            self.target = ["ALL"]

        assert access_key_id is not None, "AWS Access Key ID is missing"
        assert secret_access_key is not None, "AWS Secret Access Key is missing
"

        self.client = boto3.client(
            'inspector2',
            aws_access_key_id = access_key_id,
            aws_secret_access_key = secret_access_key,
            region_name = region or 'eu-central-1'
        )

        self.client.list_members(maxResults=1)

        not_enabled = []
        # verify Inspector2 is enabled for each requested resource type
        types_to_check = (
            ["AWS_ECR_CONTAINER_IMAGE", "AWS_EC2_INSTANCE", "AWS_LAMBDA_
FUNCTION"]
            if "ALL" in self.target else self.target
        )
        raw = self.client.batch_get_account_status()
        res = json.loads(json.dumps(raw, default=self._serialize_datetimes))
```

```
        accounts = res.get("accounts", [])
        resource_state = accounts[0].get("resourceState", {})

        # map resource types
        key_map = {
            "AWS_EC2_INSTANCE": "ec2",
            "AWS_ECR_CONTAINER_IMAGE": "ecr",
            "AWS_LAMBDA_FUNCTION": "lambda"
        }
        for rt in types_to_check:
            key = key_map.get(rt)
            status = resource_state.get(key, {}).get("status")
            if status != "ENABLED":
                not_enabled.append(rt)
        if not_enabled:
            self.skip = True
            self.result.integer_result = 1
            self.result.pretty_result = "Amazon Inspector2 not enabled for
requested resource types"
            self.result.put_raw_extra_data(
                "not_enabled",
                not_enabled
            )
        return True

    def cve_scan(self, inputs=None) -> bool:
        """
        CVE Scan - Amazon Inspector findings should be emitted
        """
        if self.skip:
            return False

        paginator = self.client.get_paginator("list_findings")
        clean_findings = []
        limit = 1000
        pagconf = {
            "MaxItems": limit,
            "PageSize": 50
        }

        # create filters variable basing in the array self.target in input
        if "ALL" in self.target:
            resource_types = [
```

```
                "AWS_ECR_CONTAINER_IMAGE",
                "AWS_EC2_INSTANCE",
                "AWS_LAMBDA_FUNCTION"
            ]
            filters = {
                "resourceType": [
                    {"comparison": "EQUALS", "value": rt}
                    for rt in resource_types
                ]
            }
        else:
            filters = {"resourceType": []}
            for rt in self.target:
                filters["resourceType"].append({
                    "comparison": "EQUALS",
                    "value": rt
                })

        sort = {
            "field": "INSPECTOR_SCORE",
            "sortOrder": "DESC"
        }
        above_th = False
        tot_ctr = 0
        above_ctr = 0
        # collect all findings, normalize dates
        for page in paginator.paginate(PaginationConfig=pagconf, sortCriteria=
sort, filterCriteria=filters):
            for f in page.get("findings", []):
                clean = json.loads(json.dumps(f, default=self._serialize_
datetimes))
                clean_findings.append(clean)

        if not clean_findings:
            self.result.integer_result = 0
            self.result.pretty_result = "CVE Scan - Amazon Inspector findings
not found: no vulnerabilities detected"
            return True

        # one ExtradataCVE per finding
        seen_cve_ids = set()
        for finding in clean_findings:
            # compute CVE ID and skip duplicates
```

```
151             cve_id = (
152                 finding.get('packageVulnerabilityDetails', {})
153                        .get('vulnerabilityId')
154                 or finding.get('cve', {}).get('cveId')
155             )
156             if not cve_id or cve_id in seen_cve_ids:
157                 continue
158             seen_cve_ids.add(cve_id)
159 
160             vuln        = finding.get('packageVulnerabilityDetails', {})
161             cve_id      = vuln.get('vulnerabilityId') or finding.get('cve', {})
    .get('cveId')
162             title       = finding.get('title', '')
163             description = finding.get('description', '')
164             severity    = finding.get('severity', '')
165             cvss_entries= vuln.get('cvss', [])
166             cvss_score  = str(cvss_entries[0].get('baseScore')) if cvss_entries
     else ""
167             cwe_objs = [
168                 result.ExtradataCWE(id=c.get('cweId',''), name=c.get('cweName',
    ''))
169                 for c in vuln.get('relatedVulnerabilities', [])
170                 if isinstance(c, dict)
171             ]
172 
173             cve_obj = result.ExtradataCVE(
174                 id=cve_id,
175                 name=title,
176                 cwe=cwe_objs,
177                 description=description,
178                 cvss=cvss_score,
179                 severity=severity
180             )
181             self.result.put_refined_extra_data_cve(cve_obj)
182             tot_ctr += 1
183             try:
184                 if float(cvss_score) >= self.cvss_th:
185                     above_ctr += 1
186                     above_th = True
187             except ValueError:
188                 pass
189 
190         if above_th:
```

```
191             self.result.integer_result = 1
192             self.result.pretty_result = f"Found {above_ctr} out of {tot_ctr}
vulnerabilities with CVSS higher than {self.cvss_th}"
193         else:
194             self.result.integer_result = 0
195             self.result.pretty_result = f"All {tot_ctr} vulnerabilities below
CVSS threshold: {self.cvss_th}"
196         return True
197 
198     def atoms(self) -> typing.Sequence[atom.AtomPairWithException]:
199         return [
200             atom.AtomPairWithException(
201                 forward=self.init,
202                 forward_captured_exceptions=[
203                     atom.PunctualExceptionInformationForward(
204                         exception_class=Exception,
205                         action=atom.OnExceptionActionForward.STOP,
206                         result_producer=lambda e: result.Result(
207                             integer_result=2,
208                             pretty_result="Credential error: AWS credentials
missing or invalid",
209                             base_extra_data=result.Extradata(raw={"
ExceptionRecovered": str(e)})
210                         )
211                     )
212                 ]
213             ),
214             atom.AtomPairWithException(
215                 forward=self.cve_scan,
216                 forward_captured_exceptions=[
217                     atom.PunctualExceptionInformationForward(
218                         exception_class=Exception,
219                         action=atom.OnExceptionActionForward.GO_ON,
220                         result_producer=lambda e: result.Result(base_extra_data
=result.Extradata(raw={'ExceptionRecovered': str(e)}))
221                     )
222                 ]
223             )
224         ]
225 
226 if __name__ == "__main__":
227     entrypoint.start_execution(Probe)
```

Bibliografia

[1] Claudio A Ardagna, Rasool Asal, Ernesto Damiani, and Quang Hieu Vu. From security to assurance in the cloud: A survey. *ACM Computing Surveys (CSUR)*, 48(1):1–50, 2015.

[2] Dereje Yimam and Eduardo B Fernandez. A survey of compliance issues in cloud computing. *Journal of Internet Services and Applications*, 7:1–12, 2016.

[3] Statista. Amazon maintains cloud lead as microsoft edges closer, October 2024. Accessed: 2025-05-09.

[4] Rehmana Younis, Mansoor Iqbal, Khalid Munir, Muhammad Aaqib Javed, Muhammad Haris, and Saad Alahmari. A comprehensive analysis of cloud service models: Iaas, paas, and saas in the context of emerging technologies and trend. In *2024 International Conference on Electrical, Communication and Computer Engineering (ICECCE)*, pages 1–6, 2024.

[5] Ambika P. H and G. Sujatha. System hardening using cis benchmarks. In *2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, pages 1–6, 2024.

[6] Center for Internet Security. Cis amazon web services foundations benchmark v3.0.0, May 2024. Accessed: 2025-06-09.

[7] Muhammad Imran Tariq, Shahzadi Tayyaba, Muhammad Waseem Ashraf, Haroon Rasheed, and Fariha Khan. Risk based nist effectiveness analysis for cloud security. *Bahria University Journal of Information & Communication Technology*, 10:23–31, 2017.

[8] Marco Anisetti, Claudio A. Ardagna, Filippo Gaudenzi, Ernesto Damiani, Nicla Diomede, and Patrizio Tufarolo. Moon cloud: A cloud platform for ict security governance. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, 2018.

Progetto sviluppato presso il SEcure Service-oriented Architectures Research (SESAR) Lab
https://sesar.di.unimi.it