



UNIVERSITÀ DEGLI STUDI DI BARI “ALDO MORO”

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica

Tesi di Laurea in Modelli e metodi per la sicurezza delle applicazioni

Sicurezza delle applicazioni tramite DevSecOps

Relatore:

Prof. Donato Impedovo

Co-relatore:

Dott. Stefano Galantucci

Laureando/a:

Vincenzo Quagliarella

ANNO ACCADEMICO 2021/22

Abstract

Questo lavoro di tesi si focalizza su DevSecOps, una metodologia innovativa che enfatizza il tema della sicurezza, aspetto che viene spesso trascurato dalle aziende che producono software. Questa metodologia è un'estensione di DevOps ed integra la sicurezza in tutte le fasi del ciclo di vita del software. Nell'elaborato viene introdotta e descritta la metodologia, dopodiché viene effettuato un lavoro di raccolta e analisi delle difficoltà e delle relative soluzioni, inoltre vengono descritti gli aspetti migliorabili della metodologia che richiedono ulteriori ricerche. Il fine è quello di motivare ed incentivare la sua adozione al giorno d'oggi.

Ringraziamenti

Desidero esprimere la mia gratitudine al Professore Donato Impedovo, relatore di questa tesi, per la sua disponibilità e per gli stimoli che mi ha fornito durante questo percorso. Inoltre, ringrazio il Dottor Stefano Galantucci, per i suoi preziosi suggerimenti e la sua guida.

Infine, vorrei esprimere un particolare ringraziamento ai miei genitori, per avermi dato l'opportunità di intraprendere questo percorso accademico.

Indice

Sommario

Capitolo 1 – Introduzione	9
1.1 Contesto della tesi	9
1.2 Obiettivo della tesi	10
1.3 Organizzazione del lavoro	10
Capitolo 2 – Stato dell’arte	11
2.1 DevOps	11
2.2 DevSecOps	12
2.2.1 Cultura orientata alla sicurezza	13
2.2.2 Shift-Left	15
2.2.3 CI/CD	15
2.2.4 Pipeline DevSecOps	18
2.2.5 Elementi di DevSecOps	20
Capitolo 3 – Analisi della metodologia	24
3.1 – Difficoltà e soluzioni relative agli strumenti di sicurezza	25
3.1.1 – D1: difficoltà nella scelta degli strumenti	26
3.1.2 – D2: Problemi di sicurezza legati alla complessità degli strumenti e problemi di integrazione ..	27
3.1.3 – D3: problemi di gestione della configurazione degli strumenti	29
3.1.4 – D4: limiti degli strumenti di analisi statica che colpiscono i cicli di distribuzione rapida	30
3.1.5 – D5: limitato utilizzo degli strumenti DAST in DevSecOps	32
3.1.6 – D6: limiti di sicurezza o vulnerabilità che colpiscono l’ecosistema dei Container	33
3.1.7 – D7: vulnerabilità che riguardano i sistemi di <i>Continuous Integration</i>	35
3.1.8 – D8: Limitazioni degli script e strumenti IaC o CaC	35
3.1.9 – D9: limiti e vulnerabilità di sicurezza che colpiscono la pipeline CD	36
3.2 – Difficoltà e soluzioni relative alle pratiche	38
3.2.1 – D10: difficoltà ad automatizzare le tradizionali pratiche di sicurezza manuali	38
3.2.2 – D11: inabilità ad effettuare una rapida valutazione dei requisiti di sicurezza	40
3.2.3 – D12: difficoltà relative alle pratiche di misura di sicurezza in DevOps	41
3.2.4 – D13: difficoltà relative all’analisi della sicurezza continua	43
3.2.5 – D14: incompatibilità tra sicurezza e pratiche DevOps	44

3.3 – Difficoltà e soluzioni relative alle infrastrutture.....	46
3.3.1 – D15: difficoltà nell’adottare DevSecOps in ambienti altamente regolamentati.....	47
3.3.2 – D16: difficoltà nell’adottare DevSecOps in ambienti con risorse limitate	49
3.3.3 – D17: difficoltà nell’adottare DevSecOps in complessi ambienti Cloud	50
3.4 – Difficoltà e soluzioni relative al personale e alla cultura organizzativa.....	52
3.4.1 – D18: problemi di collaborazione tra i gruppi.....	52
3.4.2 – D19: differenze di conoscenza in sicurezza	54
3.4.3 – D20: difficoltà nella cultura organizzativa	55
3.4.4 – D21: Minacce interne	56
3.5 – Risultati ottenuti.....	57
3.6 – Ricerca e miglioramenti.....	62
Capitolo 4 – Conclusione	64
Riferimenti.....	65

Capitolo 1 – Introduzione

1.1 Contesto della tesi

L'utilizzo di metodologie obsolete e inefficienti, che rallentano la produzione del software è molto frequente; tuttavia, i clienti sono sempre più esigenti e il mercato è sempre più competitivo. A tal scopo, urge il bisogno di adottare delle nuove metodologie, che consentano di produrre e rilasciare il software più rapidamente, al fine di minimizzare il più possibile gli errori manuali, mediante l'ausilio dell'automazione.

In questo scenario nasce DevOps [1], una metodologia di sviluppo software nata dalla combinazione di Development e Operations. Grazie alla collaborazione e alla comunicazione tra le parti, nonché all'ausilio dell'automazione, questa metodologia consente di accelerare le fasi del ciclo di vita del software, consentendo rilasci più rapidi senza compromettere la qualità, l'integrazione continua e il rilascio automatizzato.

Sebbene DevOps porti miglioramenti rispetto alle metodologie obsolete, purtroppo spesso trascura l'aspetto della sicurezza. Le aziende, spinte dalla necessità di rilasciare il software in modo rapido, tendono a sottovalutare l'importanza della sicurezza e a non investire sufficienti risorse in questo ambito. Tuttavia, questa scelta può comportare l'origine di falle di sicurezza e problemi con i clienti. A tale scopo, la metodologia DevOps si è evoluta in DevSecOps, che combina Development, Security e Operations. Questa metodologia è un approccio alla cultura, all'automazione e alla progettazione software, che integra la sicurezza come responsabilità condivisa lungo l'intero ciclo di vita IT. Un aspetto fondamentale di questa metodologia è che ciascun membro del gruppo è responsabile della sicurezza e deve occuparsi di questo aspetto. Anche gli sviluppatori devono essere in grado di trattare la sicurezza del codice, senza che questa ricada esclusivamente sugli esperti di sicurezza.

DevSecOps [2] consente di integrare la sicurezza sin dalle prime fasi del ciclo di vita del software, garantendo l'individuazione tempestiva delle vulnerabilità e prevenendo così, che i problemi di sicurezza si propaghino alle fasi successive. Il vantaggio principale di questa metodologia è che permette di conseguire un elevato livello di sicurezza attraverso l'automazione, senza aumentare i tempi di produzione.

1.2 Obiettivo della tesi

Il presente lavoro di tesi si colloca nel contesto dell'adozione della metodologia DevSecOps, con l'obiettivo di promuoverne e giustificare l'adozione. In tal senso, viene fornita una descrizione dettagliata della metodologia, dei suoi elementi e della cultura della sicurezza che la caratterizza. Successivamente, viene condotta un'analisi finalizzata all'identificazione delle difficoltà connesse all'adozione della metodologia, attribuendo a ciascuna di esse un livello di criticità e proponendo soluzioni adeguate.

Al fine di agevolare l'attività di analisi, ogni difficoltà viene assegnata a una specifica tematica, in modo da definire il contesto in cui essa può manifestarsi.

Le soluzioni proposte derivano da combinazioni di soluzioni parziali che possono essere riutilizzate per risolvere altre difficoltà. Una volta completata l'analisi, vengono individuate le aree che richiedono maggiori ricerche per migliorare la metodologia. I risultati ottenuti vengono utilizzati per rafforzare le motivazioni a favore della diffusione di DevSecOps.

1.3 Organizzazione del lavoro

Nel primo capitolo della tesi si definisce il contesto e l'obiettivo del lavoro di tesi, seguito dalla descrizione in dettaglio della metodologia DevSecOps e dall'integrazione della sicurezza in DevOps nel secondo capitolo. Nel terzo capitolo, vengono identificate e analizzate le sfide che possono sorgere durante l'adozione di DevSecOps, con una valutazione della loro importanza, la descrizione delle relative soluzioni e la presentazione dei risultati in forma tabellare. Inoltre, vengono definite le aree che richiedono una maggiore ricerca. Nel capitolo conclusivo, vengono dedotte le conclusioni sulla metodologia DevSecOps, rafforzando le ragioni per le quali un'organizzazione, che produce software, dovrebbe investire in tale approccio.

Capitolo 2 – Stato dell’arte

2.1 DevOps

DevOps [3] rappresenta una metodologia di sviluppo software, utilizzata nell’ambito dell’informatica, che si focalizza sulla comunicazione, la collaborazione e l’integrazione tra sviluppatori e responsabili delle operazioni IT.

Le aziende che adottano DevOps sono generalmente orientate a effettuare rilasci frequenti del software. Questo ciclo di rilascio viene chiamato *Continuous Deployment*, in cui il software modificato viene automaticamente trasferito nell’ambiente di produzione, ovvero un’area accessibile agli utenti che possono utilizzare il software.

La metodologia DevOps supporta le aziende nella gestione dei rilasci attraverso la standardizzazione degli ambienti di sviluppo. L’integrazione di tale metodologia ha l’obiettivo di automatizzare il rilascio del prodotto, testarlo, e agevolare la sua manutenzione mediante il monitoraggio, la correzione di eventuali bug e il rilascio di versioni minori. L’obiettivo è di ridurre i cicli di vita del software, favorendo rilasci rapidi e tempestivi.

DevOps si propone di ottimizzare l’interazione tra il gruppo di sviluppo e quello delle operazioni, attraverso l’implementazione di un insieme di processi e metodi, volti a promuovere la comunicazione e la collaborazione reciproca, al fine di garantire il successo del progetto.

Eseguire un’adeguata transizione a DevOps non è un processo semplice, in quanto richiede un cambiamento di mentalità all’interno dell’organizzazione, soprattutto per quanto riguarda i gruppi di lavoro, che devono essere interfunzionali. Ciò implica la costituzione di gruppi composti da individui con differenti competenze ed esperienze, i quali collaborano per raggiungere un obiettivo comune.

Come mostrato nella figura 1, DevOps richiede la collaborazione tra il gruppo di Sviluppo, il gruppo di Operazioni IT ed il gruppo di QA, ovvero della garanzia di qualità.

L’adozione della metodologia DevOps richiede un cambiamento organizzativo, che comporta diversi vantaggi, tra cui un miglioramento nel corso del tempo da parte dei membri dei gruppi. Ovvero, molti membri dei gruppi, in particolare gli sviluppatori, talvolta si interfacciano ad alcune tecnologie a cui non sono avvezzi, di conseguenza apprendono tramite la pratica. Difatti, in DevOps non è necessario che tutti i membri del gruppo abbiano capacità complete, nell’utilizzo degli strumenti disponibili, sin dal principio.



Figura 1 : DevOps [4]

2.2 DevSecOps

La metodologia DevOps mira a velocizzare tutte le fasi del ciclo di vita del software, favorendo rilasci rapidi e qualitativi, tuttavia c'è un aspetto del software che viene trascurato, ovvero la sua sicurezza.

In campo IT, l'insorgenza di problemi di sicurezza può causare danni considerevoli, soprattutto se si verificano perdite o divulgazioni di dati sensibili e inaccessibilità ai servizi forniti. La gravità dei danni causati da questi problemi varia a seconda del settore aziendale, ma può comportare anche gravi conseguenze economiche e legali. Inoltre, la recente introduzione del Regolamento Generale sulla Protezione dei Dati (GDPR), ha reso la sicurezza informatica un tema sempre più importante per le aziende. Pertanto, è essenziale che le aziende siano consapevoli dei rischi associati a tali incidenti di sicurezza e adottino misure per salvaguardare le loro risorse finanziarie e la reputazione.

La metodologia DevSecOps è raccomandata per le organizzazioni che intendono rilasciare software in modo veloce e sicuro, poiché mira ad integrare la sicurezza in ogni fase del ciclo di vita del software.

Gli sviluppatori sono restii ad eseguire controlli del codice, in quanto ritengono che sia un'attività dispendiosa, che provoca un rallentamento dello sviluppo del software e degli aggiornamenti. Tuttavia, questa negligenza causa spesso la presenza di vulnerabilità all'interno del codice, che possono persistere anche nelle fasi successive, di conseguenza c'è il rischio di rilasciare un software che presenta vulnerabilità all'interno, questo comporta gravi danni per l'azienda, esposizione dell'utente a dei rischi di sicurezza e perdita di tempo, necessaria a individuare e mitigare le falte di sicurezza.

2.2.1 Cultura orientata alla sicurezza

DevSecOps, affinché possa essere adottata da un'azienda, prevede che si stabilisca un cambio culturale, ovvero occorre impostare una cultura orientata alla sicurezza [5]. Questa è una pratica fondamentale, in quanto non è auspicabile imporre una metodologia, senza che si predisponga prima di cambiamenti necessari che favoriscano la sua adozione, nel caso in questione la cultura deve essere orientata alla sicurezza.

La cultura orientata alla sicurezza è caratterizzata dai seguenti punti:

- Collaborazione: per integrare i principi di sicurezza, è necessario promuovere la collaborazione tra i gruppi di sviluppo, di operazioni IT e di sicurezza dell'azienda. È auspicabile costituire un unico gruppo di lavoro, composto da membri dei gruppi principali, in modo da favorire il lavoro di squadra e la collaborazione tra i membri di tutte le squadre, indipendentemente dalla loro specializzazione come sviluppatori o esperti di sicurezza;
- Conoscenza condivisa: è essenziale condividere la conoscenza in materia di sicurezza, poiché DevSecOps utilizza numerosi strumenti di automazione della sicurezza, che non garantiscono la sicurezza se non vengono utilizzati da personale specializzato. Pertanto, gli esperti del gruppo di sicurezza devono condividere le loro conoscenze anche con gli altri gruppi. È consigliabile nominare dei "Security Champions", ovvero programmati che rappresentano i membri dei rispettivi gruppi con maggiori conoscenze in materia di sicurezza. Il gruppo di sicurezza ha il compito di insegnare le pratiche di sicurezza a questi campioni, che dovranno a loro volta diffondere la conoscenza acquisita anche al resto dei loro gruppi.
- Feedback: deve essere immediato e continuo. È di fondamentale importanza che tutti i membri dei diversi gruppi siano a conoscenza di eventuali problematiche, rilevate generalmente da membri esperti di sicurezza, in modo che possano essere informati tempestivamente.
- Continuo miglioramento: abilità di includere attività, che non solo si occupano di garantire la qualità, ma che integrano anche la sicurezza, senza dover rallentare la produzione del software. Il monitoraggio continuo della sicurezza delle applicazioni, l'utilizzo di strumenti di sicurezza, l'analisi delle minacce e soprattutto l'utilizzo di misure che consentano di valutare la performance del software;
- Comunicazione: in DevSecOps il gruppo di sicurezza ha un ruolo chiave nella comunicazione, in quanto si rende disponibile nei confronti degli altri gruppi, che ne usufruiscono fin dalle prime fasi. La comunicazione è molto semplice, gli esperti di sicurezza chiedono agli sviluppatori oppure ai membri delle operazioni IT se hanno bisogno di aiuto e in che cosa; tuttavia, il loro intervento non deve in alcun modo rallentare troppo il loro lavoro;

- Responsabilità: ciascun membro di ciascun gruppo è responsabile delle proprie attività, in particolar modo quelle che integrano la sicurezza; tuttavia, non è richiesto che tutti siano esperti di sicurezza. Ciononostante, è importante che nell'organizzazione si diffonda un adeguato livello di conoscenza, in maniera tale che ognuno sia in grado di potersi prendere carico di queste responsabilità;
- Fiducia reciproca: Il gruppo di sicurezza gioca un ruolo cruciale in DevSecOps; tuttavia, affinché il loro lavoro sia efficace, è fondamentale che gli altri gruppi ripongano fiducia in loro. Gli sviluppatori potrebbero trovarsi spesso nella posizione di ringraziare gli esperti di sicurezza e, allo stesso tempo, i membri del gruppo di sicurezza dovrebbero trattare gli sviluppatori con rispetto. È sconsigliabile criticare eccessivamente il lavoro degli sviluppatori, a meno che non sia strettamente necessario.
- Sperimentazione: sperimentare in DevSecOps è fondamentale, in quanto non esiste una standardizzazione di strumenti da utilizzare, ciascun membro può esplorare diversi strumenti prima di trovare quello adatto, questo è molto utile per fare pratica e per imparare strumenti nuovi;
- Leadership: Per promuovere l'adozione e il supporto della cultura DevSecOps, sono necessarie azioni di leadership. Tuttavia, affinché questo approccio possa essere diffuso in tutta l'organizzazione aziendale, sono richieste ulteriori iniziative.
- Incolpevolezza: In un'organizzazione che adotta DevSecOps, ogni membro è responsabile delle proprie azioni. Tuttavia, non si dovrebbe mai incolpare o colpevolizzare chi ha commesso un errore. Criticare aspramente il lavoro di uno sviluppatore non porta a vantaggi, se non a possibili conflitti e tensioni all'interno dell'organizzazione.
- Personale competente: è necessario introdurre nell'organizzazione persone che abbiano conoscenze pratiche e teoriche di DevSecOps, ad esempio persone in grado di utilizzare alcuni strumenti di analisi statica del codice e che riescano ad individuare i falsi positivi, spesso prodotti dagli stessi strumenti;
- Trasparenza: è importante che tutti i gruppi abbiano la completa visione dei componenti del software, oltre che di tutte le attività programmate o in esecuzione.

Riuscire a soddisfare pienamente queste caratteristiche rappresenta una sfida per tutte le organizzazioni aziendali. Nel capitolo successivo, saranno analizzate le sfide che DevSecOps può presentare, molte delle quali rappresentano ostacoli per le caratteristiche della cultura incentrata sulla sicurezza. Nell'adozione di questa metodologia, è fondamentale rispettare il principio caratterizzante, che definisce il modo di integrare la sicurezza nelle fasi del ciclo di vita del software, comunemente noto come *shift-left*, che sarà esaminato nella sezione successiva.

2.2.2 Shift-Left

Uno dei principi fondamentali della metodologia DevSecOps è il principio *shift-left* [6], ovvero l'idea che i gruppi, dovrebbero garantire la sicurezza dell'applicazione sin dalle prime fasi del ciclo di vita del software.

Shift-left consente di integrare le misure di sicurezza, tipicamente integrate solamente alla fine, attraverso tutto il ciclo di vita di sviluppo del software. Ne consegue che, il software è progettato fin dall'inizio con le migliori pratiche di sicurezza, favorendo l'individuazione preventiva di vulnerabilità, su cui verranno eseguite le giuste contromisure. Ciò consente di limitare al massimo i rallentamenti e i problemi, spesso causati dalla tardiva integrazione della sicurezza.

Il principio *shift-left* di DevSecOps afferma che l'integrazione della sicurezza deve essere considerata fin dalle prime fasi del ciclo di vita del software, in modo da individuare e risolvere precocemente le vulnerabilità. Sebbene gli strumenti di sicurezza siano importanti, la componente fondamentale è rappresentata dalle persone coinvolte nel processo di sviluppo. DevOps ha l'obiettivo di migliorare la collaborazione tra il gruppo di sviluppo e quello delle operazioni, mentre DevSecOps ha l'ulteriore responsabilità di integrare la sicurezza in tutto il processo. Questo ha portato a un maggiore coinvolgimento dei gruppi di sviluppo nella gestione delle pratiche di sicurezza, di cui devono prendersi le responsabilità.

2.2.3 CI/CD

CI/CD è un metodo fondamentale utilizzato in questo contesto, [7] l'obiettivo è quello di velocizzare il rilascio del software.

CI/CD è un acronimo che indica la combinazione di *Continuous Integration (CI)* e *Continuous Delivery/Deployment (CD)*:

1. *Continuous Integration*: è un approccio di sviluppo software che fa parte della metodologia DevOps e DevSecOps, che prevede che gli sviluppatori integrino continuamente il codice in un *repository* centralizzato e che le *build* e i *test* siano eseguiti automaticamente. L'integrazione continua riguarda principalmente la fase di creazione di *build* e l'integrazione del processo di rilascio del software. Tipicamente, la CI è composta da due componenti: un componente di automazione, come la CI stessa o un servizio di creazione di *build*, e un componente culturale, come la decisione di integrare il nuovo codice più frequentemente.

Nel contesto della metodologia DevOps e DevSecOps, gli sviluppatori utilizzano un sistema di controllo di versione come Git per eseguire i *commit* su un *repository* condiviso. Prima dell'integrazione effettiva, gli sviluppatori possono eseguire i test unitari in locale.

L'integrazione continua implica la creazione automatica di una build e l'esecuzione di test unitari per identificare eventuali errori. In particolare, l'integrazione continua riguarda le fasi di creazione di build e di test unitari del processo di rilascio del software. Ad ogni nuova versione rilasciata, viene creata una nuova build e viene eseguita un'attività di testing.

2. *Continuous Delivery/Deployment*: questa fase, in ambito di sviluppo software, avviene dopo l'integrazione continua e richiede che il codice sia stato buildato e testato in diverse ambientazioni. Tale fase permette l'implementazione automatica degli aggiornamenti al codice, nel caso di *Continuous Deployment*, senza la necessità di una verifica manuale, che invece è richiesta nella *Continuous Delivery*.

Incorporare l'implementazione continua implica la rapida consegna di nuove funzionalità per gli utenti senza compromettere la qualità del software. Tuttavia, nei settori critici come quello finanziario, in cui è richiesta una maggiore attenzione alla sicurezza e alla stabilità del sistema, la *Continuous Delivery* risulta preferibile, in quanto richiede l'intervento umano per controllare e procedere con l'implementazione, tramite un click di un bottone.

In generale, la fase di *Continuous Integration* (CI) e *Continuous Delivery/Deployment* (CD) rappresentano un processo complesso di sviluppo software che richiede un elevato grado di competenze tecniche e organizzative per garantire la qualità del software. La fase di CI si concentra sull'integrazione continua del codice e la sua validazione tramite test automatizzati. D'altra parte, la fase di CD si occupa della distribuzione dell'applicazione sul server principale, che può avere conseguenze finanziarie significative per l'azienda.

In particolare, la fase di CD rappresenta una sfida tecnica e organizzativa significativa, in quanto qualsiasi rilascio errato potrebbe causare la perdita di clienti e danni economici per l'azienda. Ad esempio, quando si passa da una versione di un'applicazione a una versione successiva, il rilascio deve essere gestito in modo accurato, e qualora ci fossero dei problemi con la nuova versione, sarebbe necessario effettuare un roll back alla versione precedente. Pertanto, la gestione della fase di CD richiede un alto livello di attenzione, monitoraggio costante e procedure di roll-back ben definite per garantire il successo dell'implementazione.

Gli strumenti di *Continuous Integration/Continuous Deployment* (CI/CD) possono essere utilizzati per automatizzare lo sviluppo, il *deployment* e i test del software. Esistono strumenti specifici per l'integrazione continua, lo sviluppo e il deployment continuo e il testing continuo.

Tra gli strumenti *open source* più noti per il CI/CD, *Jenkins* è un server di automazione progettato per gestire l'intero processo, dalla semplice integrazione continua all'hub di distribuzione continua. *Tekton Pipelines*, invece, è un framework di CI/CD specifico per piattaforme *Kubernetes*.

Di seguito vengono elencati alcuni degli strumenti *open source* per CI/CD, che possono essere considerati oltre a *Jenkins* e *Tekton Pipelines*: *Spinnaker*, una piattaforma CD pensata per ambienti multi-cloud; *GoCD*, un server CI/CD incentrato sulla modellazione e sulla visualizzazione; *Concourse*, una risorsa continua open source; *Screwdriver*, una piattaforma di creazione progettata per CD.

Inoltre, i gruppi possono valutare l'utilizzo di strumenti CI/CD offerti da diversi fornitori, tra cui i principali *provider* di cloud pubblico, *GitLab*, *CircleCI*, *Travis CI* e altri.

Va notato che gli strumenti essenziali per DevOps, come quelli per l'automazione della configurazione (*Ansible*, *Chef* e *Puppet*), i *runtime* dei container (*Docker*, *rkt* e *cr-io*) e l'orchestrazione dei container (*Kubernetes*), sebbene non siano strettamente strumenti CI/CD, fanno spesso parte dei flussi di lavoro CI/CD.

Nella figura 2 è possibile notare che la fase di CI precede quella di CD, inoltre viene mostrata la sottile differenza tra *Continuous Deployment* e *Continuous Delivery*.

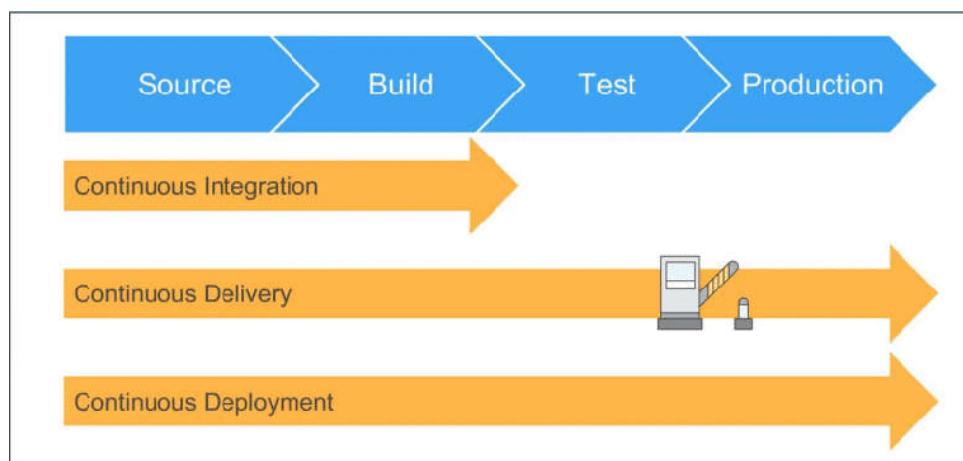


Figura 2: CI/CD [8]

2.2.4 Pipeline DevSecOps

La pipeline DevSecOps [9] incorpora tutte le fasi del processo DevOps, ma richiede l'integrazione di specifiche attività di sicurezza in ciascuna di esse. La Tabella 1 rappresenta tutte le fasi del processo, il loro svolgimento e le attività di sicurezza integrate in ciascuna fase.

Fase	Svolgimento	Attività di sicurezza da integrare
<i>Plan</i>	Durante questa fase, viene effettuata la raccolta dei requisiti, definita una strategia e pianificate le attività da svolgere prima del passaggio alla successiva fase.	Questa fase comprende la valutazione delle possibili minacce di sicurezza e l'analisi dei rischi per l'identificazione e la gestione delle minacce e dei rischi. Inoltre, vengono raccolti i requisiti di sicurezza e si analizzano gli impatti sulla sicurezza, al fine di stabilire le modifiche da apportare.
<i>Code</i>	Questa fase è dedicata all'implementazione delle funzionalità del sistema, durante la quale il codice viene caricato in un <i>repository</i> centrale condiviso.	Questa fase comprende la revisione del codice da parte di esperti di sicurezza, l'utilizzo di linee guida per la sicurezza del codice.
<i>Build</i>	In questa fase il codice viene compilato e incapsulato in un pacchetto o un'immagine.	Durante questa fase si utilizzano strumenti SAST (<i>Static Analysis Security Testing</i>) per individuare le possibili falle di sicurezza nel codice sorgente, e strumenti SCA (<i>Software Composition Analysis</i>) per identificare, in modo automatizzato, le dipendenze di terze parti e individuare eventuali vulnerabilità nel codice esterno.

<i>Test</i>	Fase apposita per la definizione e l'esecuzione di test in un ambiente dedicato, aggiornato alla nuova versione del sistema	In questa fase del processo viene effettuata un'analisi delle vulnerabilità del sistema e viene eseguita l'attività di <i>penetration testing</i> , la quale ha lo scopo di individuare eventuali punti deboli del sistema e, se possibile, sfruttarli per testare l'efficacia delle contromisure di sicurezza adottate.
<i>Release</i>	Dopo che il software è stato sottoposto ai test, esso può essere considerato pronto per essere distribuito negli ambienti di produzione.	In queste fasi vengono adottate le adeguate misure per garantire una distribuzione sicura del software, mediante l'uso di <i>repository</i> di <i>artifact</i> che consentono di archiviare tutti gli <i>artifact</i> generati durante la fase di <i>deployment</i> .
<i>Deploy</i>	Il software viene rilasciato in un ambiente di produzione ed è a disposizione dei clienti.	Si utilizzano diversi strumenti per garantire la sicurezza del sistema: il <i>Security Information and Event Management</i> (SIEM), che offre il monitoraggio e l'analisi degli eventi in tempo reale; il <i>Security Orchestration, Automation and Response</i> (SOAR), che si occupa di rispondere ad incidenti di sicurezza con operazioni automatizzate; infine, l' <i>Intrusion Detection Systems</i> (IDS), che è un dispositivo software o hardware che individua gli accessi non autorizzati ai computer o alle reti locali.
<i>Operate</i>	Fase che prevede un'attività di manutenzione, in caso di eventuali malfunzionamenti o errori imprevisti.	
<i>Monitor</i>	Fase che prevede un'attività di monitoraggio attraverso la quale vengono raccolti dati, metriche e viene valutata la performance del software, questa fase è molto importante in quanto consente di migliorare il prodotto; pertanto, dopo questa fase si riparte ciclicamente dalla fase di pianificazione.	

Tabella 1: Fasi di DevSecOps e integrazione delle attività di sicurezza

La pipeline DevSecOps concerne tutte le fasi e le attività di sicurezza integrate, viene rappresentata nella Figura 3.

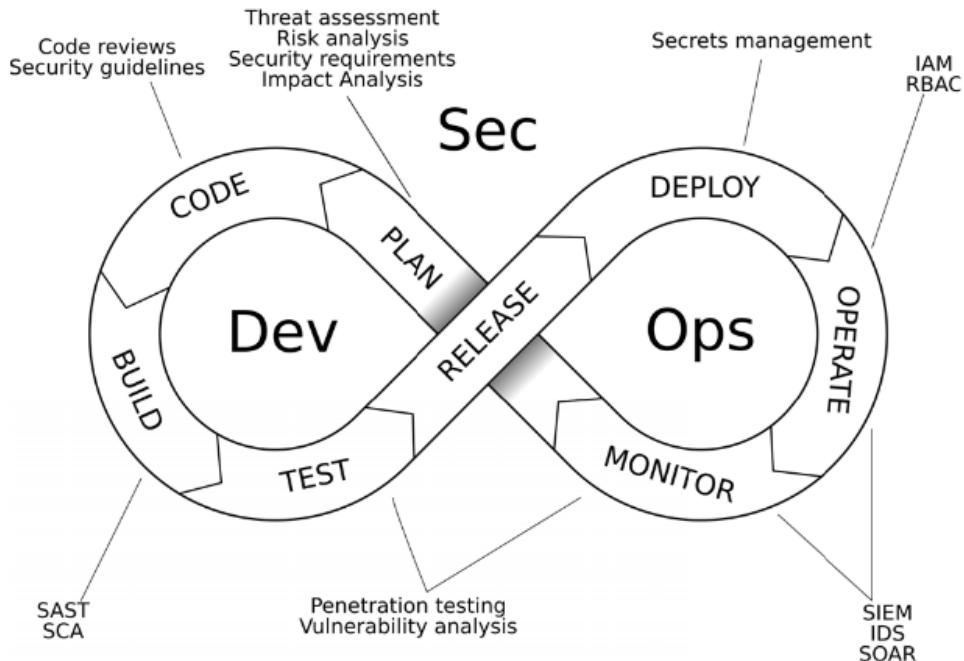


Figura 3: Pipeline DevSecOps [9]

2.2.5 Elementi di DevSecOps

Prima di procedere all'attività di analisi della metodologia, è necessario introdurre alcuni elementi distintivi di DevSecOps, poiché anch'essi potrebbero presentare possibili problematiche che verranno esaminate in seguito. Gli elementi sono i seguenti:

- **Container:** [10] rappresentano unità eseguibili di software che consentono di impacchettare codice applicativo, librerie e relative dipendenze in un formato che può essere eseguito su qualsiasi piattaforma, sia desktop che cloud. Poiché i container richiedono una forma di virtualizzazione del sistema operativo, le funzionalità di tale sistema operativo sono utilizzate per isolare e gestire le risorse dei processi. Uno dei principali vantaggi dei container è la portabilità, poiché non richiedono un sistema operativo ospite, ovvero una copia virtuale dell'hardware necessario per eseguire il sistema operativo. In ambito DevOps, l'utilizzo dei container spesso si riferisce a *Docker*, un software popolare progettato specificamente per l'esecuzione di processi nei container, e *Kubernetes*, un sistema *open source* per l'orchestrazione e la gestione di carichi di lavoro e servizi containerizzati, i quali possono funzionare in sinergia.

L'immagine del container può essere soggetta a due pratiche di sicurezza, la scansione dell'immagine e la minimizzazione dell'immagine. La scansione dell'immagine viene effettuata per individuare componenti software non aggiornati, scaduti o vulnerabili che potrebbero essere presenti nell'immagine del container. La minimizzazione dell'immagine viene eseguita per ridurre la possibilità di attacchi, mediante la riduzione del numero di file e componenti contenuti nel container. Inoltre, l'immagine del container potrebbe contenere informazioni sensibili, come password, certificati e chiavi crittografiche, che sono state immesse involontariamente, e potrebbero verificarsi malfunzionamenti, dovuti ad errori di configurazione o ad attacchi che avrebbero dovuto essere prevenuti.

- Strumenti per la sicurezza: Esistono vari strumenti disponibili per identificare e mitigare i problemi di sicurezza fin dalle prime fasi del ciclo di vita del software. È particolarmente importante evitare che tali problemi si trasmettano alle fasi successive, e pertanto DevSecOps fa uso di diverse tecnologie e strumenti di sicurezza per raggiungere questo obiettivo.

- *Static application security testing (SAST)*: è una tecnica di analisi del codice sorgente di un'applicazione, che viene eseguita senza che l'applicazione stessa sia in esecuzione. In pratica, un software di analisi SAST valuta il codice sorgente dell'applicazione per individuare eventuali vulnerabilità di sicurezza.

Durante l'analisi, il software SAST può identificare problemi di sicurezza come le vulnerabilità di input non valido, le vulnerabilità di accesso non autorizzato, le vulnerabilità di codice non sicuro e altre vulnerabilità comuni, che potrebbero essere utilizzate da un attaccante per compromettere l'applicazione o accedere ai dati sensibili. Il vantaggio di SAST è che può individuare i problemi di sicurezza prima che l'applicazione venga eseguita, consentendo agli sviluppatori di correggere le vulnerabilità prima che l'applicazione venga rilasciata. Tuttavia, SAST potrebbe anche produrre falsi positivi, ovvero identificare come vulnerabilità alcuni comportamenti del codice sorgente che in realtà sono sicuri.

- *Dynamic application security testing (DAST)*: è un metodo di testing che prevede l'analisi dell'applicazione durante la sua esecuzione. A differenza di SAST, che richiede l'accesso al codice sorgente o la conoscenza della struttura interna o dei meccanismi dell'applicazione, DAST non ne fa uso. In DAST, vengono simulate attività di attacco per valutare il comportamento dell'applicazione e individuare eventuali vulnerabilità di sicurezza.

I test DAST possono essere eseguiti in modo manuale o automatizzato, e sono in grado di valutare diverse aree dell'applicazione. In particolare, possono rilevare vulnerabilità come le iniezioni di SQL, le cross-site scripting (XSS), le vulnerabilità di autenticazione, le vulnerabilità di autorizzazione e altre vulnerabilità comuni, che potrebbero consentire ad un attaccante di compromettere l'applicazione o di accedere ai dati sensibili. Tuttavia, i test DAST possono produrre falsi positivi in quanto si basano su una simulazione di attacco e potrebbero identificare come vulnerabilità alcuni comportamenti sicuri dell'applicazione.

- *Software composition analysis* (SCA): è un approccio metodologico utilizzato per l'identificazione e la gestione dei rischi, associati all'utilizzo di software di terze parti e librerie *open source* all'interno di un'applicazione. Poiché l'uso di software di terze parti è diventato sempre più comune, è fondamentale comprendere le dipendenze software e i rischi associati ad esse. SCA impiega strumenti analitici per valutare le librerie utilizzate nell'applicazione e fornisce raccomandazioni per mitigare i rischi. Inoltre, SCA aiuta a garantire la conformità alle licenze *open source* utilizzate, in maniera tale da evitare problemi legali associati ad esse.
- *Interactive application security testing* (IAST): è una tecnica di testing dinamico dell'applicazione, che combina il testing della sicurezza con il testing funzionale dell'applicazione. È a metà tra SAST e DAST, ed è in grado di individuare e segnalare le vulnerabilità di sicurezza dell'applicazione durante l'esecuzione, fornendo una rapida identificazione dei problemi e riducendo il rischio di falsi positivi.

IAST funziona incorporando agenti di testing all'interno dell'applicazione, i quali monitorano l'attività dell'applicazione durante l'esecuzione, rilevando eventuali vulnerabilità di sicurezza e segnalando tali vulnerabilità in tempo reale. Inoltre, i dati raccolti dagli agenti di testing possono essere utilizzati per fornire informazioni dettagliate sui problemi di sicurezza, come ad esempio la causa radice della vulnerabilità. IAST offre diversi vantaggi rispetto ad altre tecniche di testing della sicurezza, come ad esempio il ridotto rischio di falsi positivi, l'identificazione in tempo reale delle vulnerabilità e la possibilità di fornire informazioni dettagliate sulla causa radice della vulnerabilità.

- *Threat modeling*: tecnica di sicurezza informatica finalizzata all'identificazione e alla valutazione dei rischi e delle minacce, che possono presentarsi all'interno di sistemi informatici e applicazioni. Tale tecnica mira ad analizzare le vulnerabilità del sistema, nonché le possibili minacce, per comprendere come un attaccante potrebbe compromettere il sistema e le conseguenze di tale attacco. Il processo di *threat modeling* coinvolge diverse fasi, tra cui l'identificazione degli asset da proteggere, l'individuazione delle possibili minacce e la valutazione delle vulnerabilità, la definizione dei controlli di sicurezza e la documentazione dei risultati. Tale tecnica può essere applicata in varie fasi del ciclo di vita del software e può essere svolta manualmente o con l'ausilio di strumenti software. Inoltre, può essere integrata con altre tecniche di sicurezza, come il *testing* di sicurezza dinamico e statico, per migliorare ulteriormente la sicurezza dei sistemi informatici e delle applicazioni.
- *Infrastructure as Code* (IaC): [12] è una pratica che mira a definire e automatizzare l'ambiente di test e rilascio del software. Al posto di configurare manualmente una macchina virtuale, con le librerie e le specifiche necessarie per il sistema software, si definiscono le configurazioni dell'infrastruttura una sola volta, utilizzando strumenti automatizzati per generare l'immagine della macchina virtuale. Il suffisso "*as-code*" si riferisce alla pratica di sviluppare i file di specifica dell'infrastruttura, utilizzando linguaggi specifici come *Puppet* e *Chef*. L'IaC consente di ridurre al minimo gli interventi manuali, automatizzando il processo, ma può richiedere attività di manutenzione per garantire l'efficacia e la sicurezza dell'infrastruttura.

Capitolo 3 – Analisi della metodologia

In questo capitolo viene condotta un'analisi della metodologia DevSecOps. Sono state raccolte e analizzate le sfide che tale metodologia presenta e, per ognuna di esse, è stata definito il livello di criticità ed è stata proposta una soluzione. Successivamente, i risultati ottenuti sono stati presentati in forma tabellare. Infine, sono state identificate le aree della metodologia che necessitano di ulteriori ricerche.

L'interesse per la metodologia DevSecOps sta crescendo sia tra le organizzazioni che desiderano adottarla, sia nel mondo accademico, dove la sua novità e complessità suscitano interesse e presentano alcune sfide. Nella pratica, DevSecOps [13] rappresenta una sfida a causa della sua complessità e della necessità di effettuare cambiamenti nella cultura organizzativa, soprattutto per l'adozione delle corrette pratiche di sicurezza.

È necessario che un'organizzazione che intenda adottare DevSecOps consideri molti fattori, per evitare errori nell'adattamento, nell'impiego degli strumenti e nell'applicazione delle pratiche di sicurezza, che potrebbero peggiorare la situazione aziendale. Pertanto, è cruciale essere consapevoli di tutte le sfide legate alla metodologia, nonché avere le competenze necessarie per affrontarle attraverso le soluzioni appropriate.

Per classificare tutte le difficoltà della metodologia, sono state definite quattro tematiche principali:

- **Strumenti:** tema che affronta le difficoltà legate agli strumenti di sicurezza in DevSecOps, tra cui l'incorretto impiego e la mancanza di una standardizzazione degli stessi;
- **Pratiche:** tema che affronta le difficoltà connesse a DevOps, tra cui l'integrazione delle pratiche di sicurezza manuali in DevSecOps, oltre alle problematiche riscontrate nella pipeline CI/CD.
- **Infrastruttura:** tema che affronta tutte le problematiche che potrebbero sorgere nell'implementazione di DevSecOps in ambienti complessi.
- **Personale e cultura organizzativa:** tema che affronta le problematiche legate al personale e la cultura organizzativa, tra cui la carenza di competenze in materia di sicurezza, i problemi di comunicazione e collaborazione all'interno del gruppo e tra i gruppi, la suddivisione dei ruoli e la difficoltà di trasformazione della cultura organizzativa.

I livelli di criticità delle difficoltà sono assegnati con il seguente criterio:

- Livello basso: la difficoltà non compromette l'adozione della metodologia, di conseguenza l'organizzazione non è tenuta ad affrontarla;
- Livello medio: la difficoltà può compromettere l'adozione della metodologia, di conseguenza è fortemente raccomandato che l'organizzazione se ne occupi;
- Livello alto: la difficoltà compromette l'adozione della metodologia, di conseguenza è fondamentale che l'organizzazione l'affronti.

Le sezioni successive prevedono la descrizione delle difficoltà relative a ciascuna tematica sopra elencata, accompagnata dalla valutazione del loro grado di criticità e dalla presentazione di una soluzione appropriata.

3.1 – Difficoltà e soluzioni relative agli strumenti di sicurezza

Come riportato nel capitolo precedente, DevSecOps utilizza una politica *shift-left*, volta a imporre l'utilizzo degli strumenti di sicurezza, fin dalle prime fasi del ciclo di sviluppo, al fine di individuare e mitigare prontamente le potenziali minacce alla sicurezza.

Gli strumenti sono utilizzati allo scopo di monitorare, identificare e risolvere tutte le problematiche che si incontrano attraverso la pipeline.

Un ulteriore aspetto cruciale di DevSecOps riguarda la standardizzazione dei processi, ne consegue che gli strumenti automatizzati sono da preferire, in quanto consentono di predisporre all'interno della pipeline dei *triggers*, il cui compito è quello di eseguire determinate azioni automatiche oppure valutazioni sulla sicurezza, il più delle volte senza che ci sia l'intervento umano.

Gli strumenti di sicurezza sono molto importanti per DevSecOps, ne consegue che è fondamentale che un'organizzazione sia al corrente di quali possano essere le problematiche associate e come poter agire di conseguenza.

A tale scopo, in questa sezione vengono trattate ed esaminate tutte le problematiche relative agli strumenti di sicurezza, inoltre viene assegnato un grado di criticità e una soluzione per ciascuna di esse.

3.1.1 – D1: difficoltà nella scelta degli strumenti

L'utilizzo degli strumenti è essenziale in DevSecOps, difatti per ogni fase della metodologia sono disponibili svariati strumenti; tuttavia, risulta complesso stabilire una standardizzazione degli strumenti.

Le variazioni nei set di strumenti [14], costituiscono un problema organizzativo, in quanto la coordinazione e la collaborazione tra le diverse squadre di sicurezza, sviluppo e operazioni IT, sono caratteristiche molto importanti per la gestione efficace della sicurezza. Tuttavia, tali differenze causano una barriera nell'implementare la sicurezza in maniera appropriata.

Ogni programmatore possiede preferenze individuali per gli strumenti da utilizzare in ogni fase dello sviluppo, e mentre molti di questi strumenti possono essere integrati con altri, richiedono una configurazione specifica e un'attenzione alle risorse per il loro mantenimento. Una soluzione semplice a questa problematica sarebbe quella di imporre un insieme di strumenti standard per tutti i programmati. Tuttavia, dato che ogni strumento ha i propri vantaggi e svantaggi, potrebbe verificarsi una situazione in cui lo strumento ideale non fa parte dell'insieme standard.

La mancanza di standardizzazione degli strumenti [15], costituisce una delle cause della scarsa consapevolezza della sicurezza. Alcuni sviluppatori sostengono che, in un contesto di DevSecOps, sarebbe preferibile che tutti adottassero strumenti facili e comuni a tutti. Emerge come la complessità dell'automazione delle pratiche di sicurezza sia amplificata da questa questione degli strumenti.

Un ulteriore fattore da considerare riguarda la variazione degli strumenti utilizzati in diverse fasi della pipeline DevSecOps: potrebbero esserci aggiunte di strumenti in fasi successive. [16] Tuttavia questa scelta è assolutamente da evitare; pertanto, è fondamentale selezionare i giusti strumenti all'inizio del progetto, anche se ciò comporta un periodo di formazione per apprenderne l'utilizzo, evitando così di interrompere lo sviluppo più avanti per effettuare la formazione.

Infine, un aspetto da tenere in considerazione nel momento in cui si devono scegliere gli strumenti è che potrebbero essere impiegati degli strumenti di distribuzione inadeguati. [17] Infine, va sottolineato che la scelta di strumenti di distribuzione impropri può provocare una distribuzione insicura del software. Sebbene il rilascio automatico del software possa essere vantaggioso per la sicurezza del sistema, l'uso improprio degli strumenti di distribuzione o la mancanza di adozione di giuste pratiche di sicurezza possono causare problemi. È pertanto importante, che coloro che lavorano in tutte le fasi del ciclo di vita del software, prestino la massima attenzione nella selezione e nell'utilizzo degli strumenti di distribuzione.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio, in quanto nella maggior parte dei casi, le organizzazioni riescono comunque a produrre un software sicuro e in tempi rapidi. Tuttavia, tale livello di successo potrebbe essere compromesso in caso di integrazione di strumenti nel corso del progetto o nell'utilizzo di strumenti inappropriati.

Soluzione

La difficoltà principale è che ci sono troppi strumenti a disposizione ad ogni fase della pipeline DevSecOps, è stato osservato come gli sviluppatori non siano in grado di avere un'inquadratura generale su cosa utilizzare [14], ed è inoltre importante evitare che si verifichi un uso improprio degli strumenti e soprattutto modifiche o aggiunte nelle fasi avanzate della pipeline, in quanto porterebbero a perdite di tempo e costi evitabili. Quasi tutti gli sviluppatori convengono che per risolvere questo problema occorrerebbe che la *community* convergesse verso degli strumenti standardizzati, in questo modo le organizzazioni saranno molto più propense a seguire le linee guida di selezione e utilizzo di questi strumenti standardizzati e allo stesso tempo i membri dei gruppi avrebbero fin da subito una visione generale di cosa utilizzare, anche e soprattutto riguardo gli strumenti di distribuzione.

3.1.2 – D2: Problemi di sicurezza legati alla complessità degli strumenti e problemi di integrazione

La complessità degli strumenti utilizzati in una *toolchain* [18], spesso costituiti da file di codice dipendenti tra loro e in diversi formati di file, può rendere difficoltosa la comprensione e l'utilizzo degli stessi. Inoltre, la documentazione sugli strumenti tende ad essere carente, in particolare per quanto riguarda le impostazioni di sicurezza del principio di privilegio minimo, il quale di default assegna permessi ampi che non sono ideali per la sicurezza.

È quindi difficile stabilire le politiche minime da rispettare per garantire un corretto funzionamento dello strumento. Nonostante ciò, le organizzazioni possono imporre l'uso di uno strumento anche se gli sviluppatori non hanno le competenze e la conoscenza necessaria per utilizzarlo, aumentando il rischio di un utilizzo scorretto degli strumenti.

In considerazione della complessità degli strumenti e della scarsa documentazione disponibile, diventa necessario che gli sviluppatori che intendono costruire un sistema sicuro, possiedano una conoscenza avanzata in diverse aree per configurare e integrare correttamente gli strumenti.

L'integrazione degli strumenti è un aspetto cruciale per garantire il corretto funzionamento del modello DevSecOps. Tuttavia, gli sviluppatori incontrano difficoltà nell'integrazione di strumenti di

testing [19], poiché l'offerta di tali strumenti è limitata. La mancanza di strumenti automatizzati per l'integrazione dei test di sicurezza, rappresenta una barriera all'adozione di DevOps da parte delle organizzazioni, poiché questo problema può avere un impatto negativo sulla sicurezza del sistema.

Integrare gli strumenti [15] è un'attività piuttosto difficoltosa, poiché non c'è tempo per effettuare manualmente il testing e la verifica. Pertanto, l'integrazione degli strumenti è un'attività dispendiosa, manuale e difficoltosa.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello basso, in quanto nella maggior parte dei casi, le organizzazioni riescono comunque a produrre un software sicuro e in tempi rapidi. La mancanza di documentazione e la complessità non costituiscono un grave problema.

Soluzione

La principale problematica da affrontare riguarda la mancanza di documentazione dettagliata, concerne la corretta configurazione degli strumenti, il che rende difficile la loro adeguata impostazione. Un'attività di scansione delle vulnerabilità, attraverso l'uso di scripts [20], ha individuato delle criticità di sicurezza, in cui un alta percentuale di script è concorde sulla presenza di criticità specifiche, come ad esempio l'utilizzo di deboli algoritmi di crittografia e l'utilizzo di HTTP senza TLS, per un'istanza di quest'ultima criticità, i professionisti evidenziano la scarsa documentazione e supporto degli strumenti, affermando che è probabile che tutto ciò potrebbe essere stato evitato se ci fosse stata una documentazione adeguata e un supporto migliore degli strumenti. Pertanto, si ritiene che una documentazione più dettagliata e un miglior supporto degli strumenti siano necessari per garantire una configurazione corretta, ad esempio per l'adozione di HTTP con TLS.

Attraverso la documentazione, gli sviluppatori saranno in grado di configurare gli strumenti di sicurezza correttamente, oltre a gestire queste configurazioni nel migliore dei modi.

Tuttavia, anche se tali documentazioni sono disponibili, non è ancora stata definita una pratica che definisca come tutti possano accedere ad esse. [21] Una buona documentazione consente di attuare le giuste impostazioni di sicurezza e configurazioni degli strumenti, ciononostante è necessario che sia accessibile indistintamente a tutti i membri dei gruppi.

In ottica di gestione documentale, è importante garantire che la documentazione sia tracciabile e accessibile in modo sicuro e affidabile da tutti gli stakeholder del progetto, tra cui i membri dei gruppi. A tal fine, si raccomanda di evitare di condividere i dati su mezzi non tracciabili, come ad esempio le chat dei clienti, e di adottare invece un *repository* centralizzato dove memorizzare tutte le

informazioni necessarie. In questo modo, si può garantire un adeguato livello di sicurezza e di controllo sull'accesso e la gestione dei dati, favorendo la condivisione e la collaborazione tra tutti i membri del progetto.

3.1.3 – D3: problemi di gestione della configurazione degli strumenti

In ambiente DevSecOps, la gestione delle configurazioni degli strumenti rappresenta una critica questione, poiché una configurazione errata può introdurre vulnerabilità nella sicurezza del software. Gli sviluppatori [22], spesso negligenti, possono commettere errori di configurazione del software e della relativa infrastruttura, compromettendo così il sistema.

L'adozione di DevOps introduce una maggiore complessità e velocità nel rilascio del software attraverso meccanismi come il *Continuous Delivery*, pertanto, il problema della gestione delle configurazioni assume un'importanza ancora maggiore. Per evitare problemi di sicurezza, è necessario garantire che la configurazione degli strumenti sia effettuata in modo accurato, inoltre è importante garantire la sicurezza del software attraverso meccanismi come il *Continuous Security*.

In ambito DevOps, la negligenza degli sviluppatori risulta evidente nelle configurazioni di default [23], che spesso non sono appropriate e rappresentano un rischio per la sicurezza dell'applicazione.

Nel caso specifico del Cloud, molte soluzioni di sicurezza si basano sugli strumenti di sicurezza con le configurazioni di default, portando a uno spreco di risorse inutili e alla mancata protezione di alcune parti dell'applicazione. È quindi necessario un cambio di configurazione, per adattarsi ai requisiti di sicurezza specifici di ciascun componente dell'applicazione, al fine di garantire la sicurezza e l'efficienza del sistema.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto, poiché l'utilizzo delle configurazioni di default rappresenta un pericolo per il software, in quanto numerosi componenti dell'applicazione potrebbero risultare vulnerabili ed esposti ad attacchi. Di conseguenza, risulta essenziale che gli sviluppatori configurino gli strumenti adottati per la sicurezza in modo appropriato, evitando di affidarsi alle impostazioni di default, al fine di garantire la sicurezza dell'intera applicazione.

Soluzione

La gestione delle configurazioni degli strumenti rappresenta una difficoltà che può essere affrontata attraverso la documentazione e il supporto degli strumenti [20], che consente di evitare l'utilizzo delle

configurazioni di default o errate. Inoltre, è fondamentale adottare le migliori pratiche per l'uso degli strumenti al fine di garantire un adeguato livello di sicurezza.

È importante notare che l'utilizzo delle configurazioni di default [23], può compromettere la sicurezza del software; pertanto, è necessario configurare ogni strumento in base al contesto in cui si opera.

Per affrontare questa difficoltà, sono stati condotti studi sull'utilizzo di ASCAT, ovvero gli strumenti di analisi statica del codice automatici [24], che consentono di definire i controlli specifici da effettuare sul progetto, escludendo quelli non pertinenti (ad esempio, i controlli web non avrebbero senso in un progetto non basato su siti web). Inoltre, consentono di escludere il codice non pertinenti dall'ambiente di produzione dalla scansione degli strumenti, di configurare gli strumenti in modo appropriato sulla base delle linee guida interne e di evitare la duplicazione dei controlli.

3.1.4 – D4: limiti degli strumenti di analisi statica che colpiscono i cicli di distribuzione rapida

Gli strumenti che fanno parte della famiglia SAST (*Static Application Security Testing*) come, ad esempio gli strumenti di analisi statica [25], sono problematici, in quanto si occupano di ispezionare il codice sorgente, i byte e il codice binario senza che il software sia in esecuzione.

Questo tipo di strumenti rappresenta un'importante risorsa per individuare precocemente difetti, vulnerabilità e inconsistenze presenti nel codice. Tuttavia, questi strumenti presentano delle limitazioni a causa della complessità degli algoritmi di analisi semantica e dei tempi di esecuzione richiesti per l'analisi del codice, soprattutto quando si tratta di codice particolarmente complesso. Ad esempio, l'uso di puntatori o di strutture di codice ambigue può rendere difficile l'analisi del codice e compromettere l'efficacia degli strumenti di analisi.

In molti casi, gli strumenti di analisi statica del codice possono mancare di comprensione su alcuni comportamenti del codice, il che può portare alla generazione di falsi negativi. Tuttavia, allo stesso tempo, tali strumenti possono anche fare assunzioni errate sul comportamento del codice, causando così la generazione di falsi positivi. Questi ultimi possono essere attribuiti alla scarsa conoscenza e alle abilità degli sviluppatori in materia di sicurezza [14]. Pertanto, è importante che gli sviluppatori aumentino la loro conoscenza e abilità in materia di sicurezza, in modo da poter utilizzare correttamente questi strumenti e interpretare i loro risultati, poiché gli strumenti non sono in grado di individuare tutti gli errori da soli e possono generare falsi positivi.

Purtroppo, questa situazione conduce all'insoddisfazione di molti sviluppatori che, invece di approfondire e migliorare la loro conoscenza, tendono ad ignorare completamente gli strumenti SAST, in quanto richiedono una maggiore conoscenza e sono difficili da integrare nell'ambiente

DevSecOps, che necessita rapidità. Il problema principale di SAST riguarda il tempo necessario affinché vengano rilevati e scansionati tutti i falsi positivi che questi strumenti generano, causando ritardi che spesso non sono accettabili in un sistema che adotta DevSecOps, in cui i rilasci sono molteplici e in brevi periodi. Di conseguenza, i gruppi, soprattutto gli sviluppatori, dovrebbero essere in grado di rilevare i falsi positivi durante la scrittura del codice, in maniera tale da evitare sprechi di tempo causati dall'uso degli strumenti.

In un contesto DevSecOps, in cui la velocità è un fattore critico, l'impiego degli strumenti SAST risulta problematico a causa del lungo tempo necessario per la scansione del codice e del loro alto consumo di risorse [16]. Gli sviluppatori tendono ad eseguire frequenti e piccoli *commit*, tuttavia se dovessero essere eseguite scansioni SAST per ogni *commit*, ci sarebbe un notevole spreco di tempo. In pratica, non è possibile eseguire scansioni complete del codice per ogni minima modifica apportata, poiché questo sarebbe inefficiente.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello basso, poiché gli strumenti SAST rimangono utilizzabili ed efficaci in DevSecOps, a condizione che siano impiegati correttamente e che i falsi positivi e i falsi negativi vengano gestiti in modo adeguato.

Soluzione

L'adozione delle migliori pratiche per l'utilizzo degli strumenti SAST [16], risulta essenziale per costruire una pipeline ottimizzata. Al fine di risolvere il problema del tempo di scansione del codice, ad esempio in caso di scansione profonda, gli strumenti SAST potrebbero essere eseguiti contemporaneamente senza interrompere il controllo del codice.

Un'altra soluzione per gestire questa difficoltà è rappresentata dall'utilizzo dei servizi Cloud [25], che costituiscono una tendenza popolare nell'ambito attuale. I clienti che utilizzano i servizi Cloud hanno la possibilità di evitare molti inconvenienti causati dall'utilizzo di strumenti SAST autonomi. Questa soluzione è particolarmente efficace, in quanto consente di risolvere la problematica del grande numero di falsi positivi, oltre alle difficoltà di configurazione degli strumenti SAST in un ambiente DevSecOps.

Gli strumenti IAST [26], ovvero *Interactive Application Security Testing*, rappresentano un'alternativa ai tradizionali strumenti SAST. Questo tipo di strumento è particolarmente adatto al paradigma di sviluppo del software adottato in DevSecOps.

IAST combina il testing dinamico e il testing statico, trovandosi a metà strada tra SAST e DAST. In genere, gli strumenti IAST vengono eseguiti durante i test unitari funzionali, ovvero parti critiche del

codice che si occupano di una specifica funzionalità del software. Tuttavia, per utilizzare gli strumenti IAST, è necessario creare appositamente test di unità che possano essere eseguiti con questi strumenti. Molti di questi strumenti, che si adattano molto bene all'ambiente DevSecOps, sono disponibili sia in versione commerciale che *open source*.

3.1.5 – D5: limitato utilizzo degli strumenti DAST in DevSecOps

Nel contesto di DevSecOps, l'utilizzo di strumenti DAST (*Dynamic Application Security Testing*) [27] e di strumenti per eseguire *penetration testing* è frequente. Lo scopo di questi strumenti è individuare e, talvolta, sfruttare le vulnerabilità di sicurezza all'interno di un programma. Essi coprono una vasta gamma di vulnerabilità, tra cui errori di sicurezza di memoria come buffer overflow e utilizzo scorretto della memoria dinamica, ma anche problemi legati alla sanificazione degli input, come *SQL injection* e *command injection*. Inoltre, tali strumenti vengono spesso utilizzati manualmente dagli sviluppatori e dagli esperti di sicurezza per individuare falle di sicurezza all'interno del programma, prima che vengano sfruttate da hacker malintenzionati. Tuttavia, per utilizzare questi strumenti, è necessario che il software sia in esecuzione. Ciò presenta un'altra problematica, ovvero che questi strumenti richiedono che il software sia buildato, installato e configurato insieme a tutti i software necessari, come ad esempio una base di dati o un web server [25]. Inoltre, è richiesta una suite di test che venga eseguita insieme al software, il che rappresenta un'attività molto dispendiosa in termini di tempo.

In una configurazione di sistema, che adotta DevSecOps, in cui il codice viene rilasciato frequentemente, condurre tutti questi step ad ogni rilascio risulta molto complesso. Un altro problema che potrebbe rallentare il rilascio è causato dalla modalità d'uso di questi strumenti, ossia che richiedono di essere configurati ed eseguiti manualmente, inoltre è comune che i casi di test occorra definirli manualmente [27].

Come gli strumenti SAST, anche i DAST tipicamente richiedono un tempo più lungo di esecuzione [28], ciononostante è molto importante il loro impiego anche all'interno dei container Docker, in quanto aiutano ad individuare comportamenti malevoli nel container, come la presenza di immagini malevoli o file che non erano presenti prima.

Tutte queste problematiche relative agli strumenti DAST, minano i rilasci frequenti di DevOps, sebbene siano molti utili in ambito di sicurezza, purtroppo tutti questi svantaggi ne limitano l'uso in DevSecOps.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello basso, poiché gli strumenti DAST, per la stessa ragione dei SAST, rimangono utilizzabili ed efficaci in DevSecOps, a patto che vengano adottate le giuste pratiche di utilizzo.

Soluzione

Un’alternativa agli strumenti DAST, che è stata proposta anche per sostituire gli strumenti SAST, è l’utilizzo degli strumenti IAST [26], di cui è stata già esaminata la loro funzione e i vantaggi. Qualora l’organizzazione volesse continuare ad utilizzare strumenti DAST, allora, in tal caso è doveroso adottare le migliore pratiche di sicurezza per l’utilizzo di tali strumenti, come è stato visto in precedenza con gli strumenti SAST.

3.1.6 – D6: limiti di sicurezza o vulnerabilità che colpiscono l’ecosistema dei Container

Nel contesto di DevSecOps, l’uso di container è ampiamente diffuso, ma comporta problematiche inerenti alla sicurezza e alla complessità [29]. In particolare, l’aumento dei livelli di astrazione del software, causato dall’uso di container, comporta un aumento della complessità, la quale, a sua volta, può generare un maggior numero di possibili vettori di attacco.

DevOps fornisce degli elementi che sono riutilizzabili, tra cui le immagini del container; tuttavia, queste possono essere compromesse e presentare delle vulnerabilità; nonostante si incoraggi la sicurezza di tali elementi, spesso è un aspetto che viene trascurato.

I container utilizzano degli elementi di terze parti, tipicamente in esecuzione su diverse piattaforme, fornite da fornitori differenti, questa situazione solleva dei dubbi sull’integrità del codice, in quanto se il codice fosse compromesso, vorrebbe dire che sarebbero presenti delle vulnerabilità, potenzialmente sfruttabili da hacker malevoli.

L’utilizzo inappropriato dei container può portare ad altre implicazioni di sicurezza. L’adozione di configurazioni non sicure e di impostazioni del controllo degli accessi inadeguate, possono portare a spaccature nella sicurezza, che minano l’integrità dei file sorgente, che si trovano nel container [30].

Infine, altre vulnerabilità potrebbero sorgere nel momento in cui i container vengono lanciati come macchine virtuali dagli sviluppatori, ad esempio tramite la virtualizzazione [31]. Tale pratica espone risorse ben definite al sistema ospite (risorse dell’hardware virtuale).

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. Nel contesto di DevSecOps, l'utilizzo dei container rappresenta un pilastro essenziale, pertanto è di vitale importanza garantirne la sicurezza e implementare le migliori pratiche di sicurezza.

Soluzione

Molti studi suggeriscono l'utilizzo di piattaforme di orchestrazione, in quanto sono in grado di gestire le problematiche relative ai container. Questi orchestratori tipicamente riescono a correggere l'utilizzo improprio dei container [31], in particolare di Docker, che è molto utilizzato in questo ambito. Nello specifico, risulta che i container sono scarsamente isolati.

Questo stato di isolamento [30], potrebbe essere migliorato introducendo alti livelli di astrazione, (controller di replicazione, memorizzazione remota persistente ecc.) che eliminano completamente la dipendenza dall'*host*, ottenendo un isolamento elevato. Inoltre, gli orchestratori elevano l'automazione come fattore chiave, allo scopo di ottenere una ripetibile, prevedibile e verificabile sicurezza continuamente migliorata.

Per garantire la distribuzione sicura dei servizi, la piattaforma di orchestrazione deve soddisfare standard elevati di sicurezza, al fine di prevenire e rilevare intrusioni [32]. Tra le funzionalità di sicurezza fondamentali di una piattaforma di orchestrazione si possono includere la sanificazione dell'immagine del container, la riduzione della superficie di attacco del container, la restrizione degli accessi degli utenti e l'irrobustimento dell'*host*.

Un ulteriore pratica che aiuta a risolvere le problematiche legate ai container, consiste nell'utilizzo di strumenti per la valutazione continua delle vulnerabilità [27]. Questi strumenti sono considerati una soluzione chiave nell'ambito di DevSecOps, in quanto permettono di eseguire controlli continui sulle vulnerabilità di sicurezza ogni volta che viene caricato del nuovo codice in un *repository*. Ciò consente di allertare immediatamente gli sviluppatori, qualora ci fossero delle falle di sicurezza che sono state appena individuate.

Infine, un ulteriore apporto consiste nell'utilizzo di multipli strumenti DAST e SAST per la verifica continua delle vulnerabilità dei container [28]. Ne consegue che gli inconvenienti di alcuni strumenti come, ad esempio, i falsi positivi possono essere mitigati.

3.1.7 – D7: vulnerabilità che riguardano i sistemi di *Continuous Integration*

In ambito DevOps, la fase di *Continuous Integration* (CI) è di fondamentale importanza, tuttavia, diversi studi evidenziano la maggior vulnerabilità degli strumenti utilizzati in questa fase rispetto ad altri [33]. Ciò comporta un elevato numero di vettori di attacco associati a tali sistemi. Durante la build del codice, è necessario eseguire il codice fornito dagli sviluppatori, che può presentare diversi problemi, tra cui l'eccessivo consumo di risorse, che può causare malfunzionamenti del sistema, o problemi più gravi come attacchi DoS, virus e altri tipi di attacchi malevoli.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. Nel contesto di DevSecOps la fase di CI è cruciale; pertanto, è di vitale importanza che questa venga condotta in maniera sicura.

Soluzione

Questa soluzione è stata sviluppata per affrontare i problemi di attacchi alla fase di *Continuous Integration* (CI) o ai server di build [33]. L'uso di strumenti di virtualizzazione svolge un ruolo chiave nell'incapsulamento del processo di compilazione, al fine di creare un server di build sicuro. L'obiettivo è di mantenere lo stato del server di build non compromesso dopo ogni processo di compilazione, prevenendo potenziali attacchi.

3.1.8 – D8: Limitazioni degli script e strumenti IaC o CaC.

La pratica di *Configuration as Code* (CaC) [34] implica la definizione delle configurazioni della rete e del sistema tramite il codice sorgente, che può essere gestito attraverso i sistemi di controllo di versione per garantire la riproducibilità e il testing. Questa tecnica è ampiamente utilizzata per automatizzare il rilascio di nuove versioni del software.

Anche se gli strumenti CaC, in particolare *Puppet*, sono molto utili in questo contesto, è stato riscontrato che gli sviluppatori possono avere difficoltà nell'utilizzo di tali strumenti, ne consegue che l'utilizzo improprio di essi può portare a problemi di sicurezza.

Infrastructure as Code (IaC) [20] è una tecnica, nonché un elemento importante di DevSecOps, spesso utilizzata attraverso strumenti e script come *Chef* e *Puppet*. Molte organizzazioni utilizzano questi script IaC per gestire i propri server, in modo da fornire ambienti di sviluppo in modo rapido e affidabile. Tuttavia, tali script sono vulnerabili a diverse problematiche di sicurezza, inclusi sette

problemi di sicurezza comuni, come la presenza di amministratori di default, la mancanza di password, l'integrazione di informazioni riservate nel codice (*hard-coded secret*), l'utilizzo di *binding IP* non validi, l'utilizzo di commenti sospetti da parte degli sviluppatori, l'utilizzo di HTTP senza TLS e l'utilizzo di algoritmi crittografici scarsi.

In generale, è stato evidenziato un insufficiente interesse riguardante la sicurezza degli strumenti e degli script di IaC e CaC, nonostante la loro ampia diffusione. Di conseguenza, si richiede una maggiore attenzione da parte degli sviluppatori in questo ambito.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio. IaC è un elemento spesso utilizzato in DevSecOps; pertanto, è molto importante che venga posta più attenzione sulla sicurezza, al fine di evitare attacchi informatici.

Soluzione

La soluzione proposta mira a risolvere i problemi di sicurezza associati agli script IaC e può essere adattata anche agli script CaC. [35] Tali script sono spesso utilizzati come input per gli strumenti IaC. È stato sviluppato uno strumento di analisi statica, denominato *Security Linter for Infrastructure as Code scripts* (SLIC), che identifica automaticamente i problemi di sicurezza negli script IaC.

3.1.9 – D9: limiti e vulnerabilità di sicurezza che colpiscono la pipeline CD

Uno dei problemi rilevanti per la parte di *Continuous Delivery* (CD) del software riguarda le vulnerabilità e le limitazioni presenti nella pipeline di *Continuous Delivery* (CDP). La maggior parte delle CDP non è stata progettata per garantire un alto livello di sicurezza [36].

Per prima cosa, sarebbe auspicabile consentire l'attribuzione di differenti tipi di accesso alla pipeline in base ai ruoli all'interno del gruppo di sviluppo o di operazioni. Ad esempio, uno sviluppatore non dovrebbe avere la possibilità di rilasciare direttamente sull'ambiente di produzione, senza che le sue modifiche al codice siano passate attraverso la pipeline. Alcuni compiti di testing e di build dovrebbero essere eseguiti solo da determinate persone con ruoli specifici.

In secondo luogo, è fondamentale che gli ambienti di testing e di produzione siano completamente isolati, poiché spesso i guasti nei sistemi si verificano quando un componente dell'ambiente di test viene accidentalmente connesso alla base di dati dell'ambiente di produzione. Infine, in una pipeline di distribuzione continuata (CDP) mal configurata, è possibile che siano presenti codici malevoli o

semplicemente errati che confluiscano nell'ambiente di produzione. Nonostante l'importanza della CDP, è molto comune progettarla senza considerare tutti i possibili problemi di sicurezza.

La maggior parte dei componenti di una CDP vengono eseguiti su un ambiente che utilizza delle librerie online, difatti spesso fonti di vulnerabilità [37]. Tra i componenti della CDP ci sono il repository (di solito GitHub), il server principale e il server di *Continuous Integration* (CI). Il *repository* è esposto al rischio di accesso non autorizzato da parte di utenti malevoli, che possono accettare o rifiutare richieste di *pull*, effettuare *commit* e altre attività, ciò potrebbe verificarsi in caso di uso di password deboli o a causa di un utente interno con intenti malevoli. Per quanto riguarda il server principale, spesso si utilizza solo una password per l'autenticazione, ma sarebbe consigliabile implementare ulteriori sistemi di autenticazione per evitare accessi non autorizzati. Infine, per il server di *Continuous Integration* (CI) spesso si utilizza *Jenkins*, che offre supporto per la distribuzione, l'automazione e la compilazione dei progetti. Tuttavia, *Jenkins*, nel momento in cui viene installato, permette l'accesso libero a tutti, ma sarebbe consigliabile assegnare dei ruoli e limitare l'accesso solo a determinate persone.

Tramite uno studio in cui sono state utilizzate due pipeline CD [38], è stato riscontrato che la sicurezza delle pipeline non ha un'alta priorità per il gruppo di sviluppo, malgrado la maggior parte dei membri abbia accesso alle configurazioni della pipeline CD. La mancanza di accortezze di sicurezza può porre a serio rischio il business dell'organizzazione aziendale.

Questo studio ha mostrato che entrambe le pipeline presentano delle vulnerabilità che oscillano tra il medio rischio e l'alto rischio. Alcune vulnerabilità sono presenti perché il gruppo che lavora al progetto dipende dall'infrastruttura imposta dal cliente, per entrambe le pipeline, infatti, il gruppo deve fidarsi della sicurezza dell'infrastruttura esterna e dei loro stessi membri del gruppo.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. La pipeline di *Continuous Delivery* (CDP) rappresenta un elemento cruciale all'interno di DevSecOps; pertanto, risulta essenziale garantire la sua sicurezza. A tal proposito, è necessario promuovere un maggiore interesse tra i membri dei gruppi di sviluppo riguardo alla CDP.

Soluzione

È consigliabile adottare le best practice di sicurezza, in maniera tale da migliorare il livello di sicurezza della pipeline CD [37]. In particolare, sono state studiate due CDP differenti, in una delle quali sono state integrate cinque pratiche di sicurezza, ed è stato dimostrato che il livello di sicurezza della CDP in questione è migliorato notevolmente. Tali pratiche sono le seguenti: rendere il *repository* sicuro tramite accessi controllati, in modo tale da avere il controllo su chi può eseguire i *commit* su

determinati *branch* del *repository*; rendere sicura la connessione al server principale tramite l'utilizzo di una chiave privata sul *Secure Shell* (SSH), che è un protocollo che permette di stabilire una sessione sicura e cifrata, tramite un'interfaccia a riga di comando con un altro host della rete; utilizzare i ruoli sul server principale per controllare gli accessi tramite l'ausilio dell'IAM di AWS; configurare il server CI per avviare una macchina virtuale con uno stato pulito, sfruttando il plugin di macchina virtuale di *Jenkins*; infine, utilizzare un *plugin* di *Jenkins* che consente di effettuare l'assegnazione dei ruoli all'interno del server CI. In questo modo il *repository*, il server principale e il server CI sono sicuri.

Affianco a queste misure di sicurezza, è consigliabile adottare un modello di analisi delle minacce [38] che è in grado di identificare, comunicare, comprendere le minacce e fornire dei metodi di mitigazione.

Nel presente caso, è stato impiegato il modello STRIDE, il quale rappresenta un framework utilizzato per la valutazione delle vulnerabilità in ambito informatico. Esso consente di classificare le minacce in sei categorie distinte, ossia spoofing, manomissione, non ripudio, divulgazione di informazioni, attacco *Denial-of-Service* e acquisizione di privilegi più elevati.

3.2 – Difficoltà e soluzioni relative alle pratiche

In questa sezione verranno esaminate le problematiche relative alle pratiche continue, come ad esempio CI/CD, e alle pratiche di integrazione della sicurezza, come ad esempio le difficoltà nell'integrare le pratiche manuali di sicurezza all'interno del contesto DevSecOps.

3.2.1 – D10: difficoltà ad automatizzare le tradizionali pratiche di sicurezza manuali

L'automazione è un aspetto fondamentale in DevOps, ma la sua estensione in DevSecOps presenta alcune problematiche, poiché molte delle tradizionali pratiche di sicurezza vengono ancora eseguite manualmente.

Il principale problema sorge quando un'organizzazione cerca di mantenere la stessa velocità di produzione di beni e servizi, ma allo stesso tempo aderisce a standard di sicurezza delle informazioni, *framework* e migliori pratiche. [37]

Ci sono alcune pratiche complesse da integrare, come ad esempio la sicurezza e *privacy by design* [39], che è un approccio per lo sviluppo di software e hardware che cerca di proteggere i sistemi fin

dalle prime fasi di sviluppo, mediante l'implementazione di adeguate misure di sicurezza e metodologie di gestione dei rischi. Queste pratiche rappresentano una sfida, in quanto non sono facili da automatizzare e richiedono un grande investimento economico, a causa della necessità di esperti di sicurezza nel gruppo di sviluppo, affiancata alla scarsità di strumenti automatici che esaminano i rischi e valutano la sicurezza nelle fasi di progettazione e di funzionamento.

Un'altra pratica di sicurezza complessa da integrare è l'analisi dei rischi dell'architettura, [40] che è una delle più potenti attività di sicurezza del software; ciononostante, molti progetti che adottano DevOps non se ne prendono cura adeguatamente, difatti non mettono a disposizione architetti del software adeguati a gestire la sicurezza.

Molte organizzazioni che adottano DevOps spesso trascurano l'importanza dell'architettura del software, anche se la sua presenza è fondamentale. Infatti, le falte di sicurezza nell'architettura del software possono causare gravi danni e, pertanto, è importante effettuare l'analisi dei rischi dell'architettura senza interferire con il corretto funzionamento di DevOps.

Infine, anche il *threat modeling*, che è un elemento fondamentale di sicurezza, presenta problemi di integrazione. È difficile da automatizzare, inoltre la situazione si complica se gli sviluppatori non sono in grado di eseguire la pratica.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio. Sebbene sia fondamentale automatizzare le pratiche di sicurezza, questa problematica non impedisce la produzione di software sicuri. In alternativa, l'organizzazione potrebbe scegliere di effettuare alcune pratiche manualmente, anche se ciò comporterebbe un allungamento dei tempi di produzione.

Soluzione

Viene proposta una soluzione per automatizzare alcune pratiche di sicurezza, come test di conformità, gestione dei rischi, sicurezza e privacy by design, attraverso l'adattamento di standards, politiche, modelli e contratti di servizio su criteri testabili. Ad esempio, i contratti di servizio possono essere utilizzati per definire una metodologia di *security-by-design* integrabile in DevSecOps [39].

I contratti di servizio permettono di modellare e valutare i requisiti di sicurezza, specificando le capacità di sicurezza dell'applicazione e quantificando il livello di sicurezza fornito. Inoltre, consentono di automatizzare tecniche di analisi dei rischi e valutazioni di sicurezza. La chiave per attuare questa soluzione è l'adattamento dei criteri testabili. Questi criteri devono essere adattati ai vari strumenti e metodologie utilizzati nell'organizzazione.

I contratti di servizio rappresentano un meccanismo importante, in quanto consentono di garantire la sicurezza e la privacy dei dati anche in ambienti multi-cloud [41], ovvero in contesti in cui le applicazioni o i servizi fanno uso di componenti distribuiti in servizi cloud diversi. In particolare, questa soluzione consente di gestire il ciclo di vita delle applicazioni cloud in modo completo, basandosi sulla prioritizzazione dei rischi, attraverso l'utilizzo dei contratti di servizio.

Per eseguire pratiche di test di conformità alla velocità di DevOps [42], gli sviluppatori possono usare gli strumenti, forniti dall'organizzazione, per personalizzare gli standard di sicurezza delle informazioni sui criteri testabili. Questi strumenti sono in grado di eseguire test di conformità, che abilitano la validazione del codice all'interno di una pipeline di rilascio/distribuzione automatizzata, in tempi efficienti. L'attuazione di meccanismi automatici e continui di test di conformità è fondamentale nell'approccio DevSecOps [22]. Infatti, l'attuazione di questi test di conformità nelle prime fasi di *deployment*, supporta un feedback rapido e tempestivo sulla compatibilità dell'ambiente ideale con la più recente versione del software.

3.2.2 – D11: inabilità ad effettuare una rapida valutazione dei requisiti di sicurezza

L'adozione di DevSecOps in un progetto rende complessa la valutazione rapida dei requisiti di sicurezza. A causa della velocità delle fasi automatizzate, è difficile verificare completamente i requisiti di sicurezza prima del passaggio del software all'ambiente di produzione.

Malgrado l'importanza di questa pratica, spesso non viene effettuata a livello concreto. Una volta che i requisiti di sicurezza vengono valutati e sviluppati, diventa necessario testare tutti gli oggetti relativi alla sicurezza in un ambiente apposito, ma gli sviluppatori sono spesso riluttanti ad eseguire test manuali. La mancanza di strumenti e metodi adeguati rappresenta il principale ostacolo per la corretta esecuzione di questa pratica [43]. Infatti, la maggior parte degli strumenti si concentra sul testing delle vulnerabilità dei sistemi attuali, invece di valutare l'architettura e la progettazione orientata ai principi e alle proprietà di sicurezza.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio. L'analisi e la valutazione dei requisiti di sicurezza sono di fondamentale importanza in questo contesto, in particolare quando non vengono effettuati test manuali. In questi casi, la problematica diventa critica.

Soluzione

Per superare la difficoltà di valutare in modo completo i requisiti di sicurezza nel contesto DevSecOps, è necessario adottare metodi di automazione dell'analisi dei requisiti di sicurezza. A tal fine, sono state sviluppate *toolchain* specifiche per convertire automaticamente i documenti dei requisiti in modelli ontologici, analizzarli e fornire risultati utili [43]. Una volta ottenuti questi risultati, gli sviluppatori possono intervenire sui problemi fin dalle prime fasi del processo.

3.2.3 – D12: difficoltà relative alle pratiche di misura di sicurezza in DevOps

La valutazione della sicurezza in un sistema risulta una sfida, soprattutto quando si adotta DevOps, poiché richiede di tenere il passo con i rilasci rapidi e continui del software [44].

Risulta impossibile prevedere tutti i possibili attacchi a cui il sistema potrebbe essere esposto, poiché molte organizzazioni conservano vulnerabilità nascoste per anni, anche quando esposte a spionaggi. Inoltre, la misurazione dell'insicurezza del software è un compito complesso a causa della mancanza di metriche di sicurezza adeguate, il che rappresenta il principale ostacolo alla valutazione della sicurezza del software.

In molte organizzazioni, i metodi tradizionali di raccolta dati sono ancora ampiamente utilizzati, nonostante la loro lentezza, questo rappresenta una sfida per l'adozione di metodologie più agili come DevOps. La causa è insita nella resistenza al cambiamento della cultura aziendale. [45] Tuttavia, metodi di raccolta dati più adatti a DevOps devono essere identificati, poiché le tecniche tradizionali, come interviste e sondaggi, non si adattano bene alla metodologia DevOps, che richiede il rilascio continuo e rapido del codice.

La tracciabilità è una proprietà cruciale per individuare prontamente i problemi e risolverli successivamente. Per mantenere tale proprietà in un sistema, è essenziale creare *feedback loop* brevi e rapidi [15], ovvero parti del sistema in cui alcune porzioni dell'output vengono utilizzate come input per future operazioni. Questo aiuta a mantenere tutti i membri del gruppo allineati, favorendo una comunicazione continua con gli stakeholder del progetto.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio. La valutazione di sicurezza del software è importante, in particolar modo è necessario che sia al passo con i tempi di DevOps.

Soluzione

Per superare le limitazioni dei tradizionali metodi di raccolta delle informazioni, è possibile adottare nuovi approcci [45], basati sull'integrazione di tecniche di analisi comportamentale e soluzioni Big Data, al fine di sfruttare i vantaggi di queste tecnologie nel contesto DevSecOps. Tali approcci consentono di acquisire una maggiore comprensione delle tipologie di utenti che usano l'applicazione, in questo modo è anche più facile ottenere dei riscontri veloci da parte dell'utente finale. L'adozione di queste tecniche comportamentali fornisce tre principali benefici al gruppo HCI: l'utilizzo di profili utente per monitorare il comportamento dell'utente, l'analisi predittiva delle tendenze basata sul comportamento dell'utente e la realizzazione di *benchmark* per la misura di diverse variabili.

A causa della velocità dei rilasci del software nel paradigma DevOps, gli esperti non prioritizzano la documentazione e i processi di *logging*, in quanto è molto probabile che rallentino la pipeline. Tuttavia, questi processi sono importantissimi per lo sviluppo di un software, di conseguenza devono essere eseguiti a prescindere dalla velocità di rilascio. [21]

Al fine di individuare ed eliminare facilmente le anomalie nel processo, nonché utilizzare i risultati dei test come prova in caso di verifiche, è necessario documentare la distribuzione automatizzata e i relativi test. Per raggiungere questo obiettivo, è possibile impiegare strumenti specifici per la registrazione dei registri dei rilasci e delle attività di test, la registrazione degli accessi dell'utente ai dati e l'utilizzo di *repository* per i documenti. Queste pratiche consentono di elaborare rapidamente i documenti e di adattarsi a paradigmi come DevOps.

La definizione di metriche di sicurezza del software è una necessità, con particolare attenzione alle metriche che valutano le caratteristiche degli sviluppatori e le loro attività [14]. La misurazione di tali metriche è importante, poiché le squadre hanno riscontro costante su quello che stanno svolgendo in merito alla sicurezza. In questo modo, nella maggior parte dei casi, si prevengono eventuali problemi nelle fasi finali dello sviluppo del software.

Le metriche sono le seguenti: Il numero di sviluppatori che hanno avuto un periodo di formazione di sicurezza; il numero di errori di sicurezza trovati secondo la classificazione di sicurezza nota, come ad esempio OWASP top 10; il tempo speso per correggere gli errori in ciascuna categoria; il numero di sistemi che sono toccati da *penetration testing* interni e esterni; il numero di *commit* in un determinato intervallo di tempo.

3.2.4 – D13: difficoltà relative all’analisi della sicurezza continua

L’analisi continua della sicurezza è una pratica raccomandata in DevSecOps, [46] tuttavia i processi relativi a questa pratica non sono ampiamente adottati. La sicurezza continua cerca di prioritizzare la sicurezza come aspetto importante attraverso tutte le fasi del ciclo di vita di sviluppo, talvolta anche oltre il rilascio; ne consegue che è una pratica fondamentale, ciononostante ha una priorità più bassa rispetto ad altri aspetti.

La pratica dell’analisi continua delle vulnerabilità, pur essendo importante in DevSecOps [47], spesso non viene adottata a causa della mancanza di controlli periodici da parte degli sviluppatori, scarsa conoscenza degli strumenti utilizzati per eseguire tale processo.

Affinché questa pratica possa essere correttamente attuata all’interno di DevSecOps, occorre che ci siano chiare direttive riguardo le specifiche sezioni della pipeline, che dovranno includere misure di sicurezza. Ciononostante, è presente uno scarso consenso (ad esempio una metodologia standardizzata) su come devono essere integrate le misure di sicurezza nella pipeline DevOps [48]. Ad esempio, alcune definizioni di DevSecOps pongono l’accento sull’integrazione di pratiche di sicurezza all’interno di un sistema cloud, mentre altre enfatizzano l’importanza di coinvolgere gli sviluppatori e i manager nella preoccupazione concerne il tempo di produzione.

In generale, la mancata standardizzazione dell’integrazione delle misure di sicurezza in DevSecOps, rende difficile l’adozione di questa pratica in modo uniforme e coerente.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio. Le pratiche di sicurezza continue sono raccomandate in DevSecOps, pertanto anche se non sono considerate indispensabili, le organizzazioni dovrebbero comunque prestare la necessaria attenzione.

Soluzione

L’analisi continua delle vulnerabilità, che è un importante pratica di sicurezza continua, è attuabile in DevSecOps. A tal proposito, si utilizzano strumenti dedicati che valutano costantemente la presenza di vulnerabilità nel software [27]. Questi strumenti consentono di eseguire controlli continui di vulnerabilità di sicurezza, in particolar modo ogni volta che viene aggiunto codice nel *repository* principale, al fine di assistere gli sviluppatori con il riconoscimento e la mitigazione delle falle di sicurezza.

L'approccio di sicurezza *shift-left* è fondamentale per DevSecOps. Integrare pratiche e attività di sicurezza fin dalle prime fasi è un grande vantaggio, in quanto consente di assegnare una priorità maggiore alla sicurezza. A tal proposito, gli sviluppatori riescono ad individuare le vulnerabilità fin da subito, favorendo la copertura dei costi che si avrebbero nelle fasi avanzate. Questo approccio ha un notevole impatto nella risoluzione della difficoltà in questione.

Infine, a supporto delle precedenti, è possibile implementare una pratica di valutazione di sicurezza continua. Secondo questa pratica [46], la sicurezza viene trasformata da un requisito funzionale generico a un concetto chiave, in quanto deve essere presente in tutte le fasi del ciclo di vita di sviluppo, incluso il post-rilascio. Ciò è supportato da un approccio intelligente e intuitivo che identifica le vulnerabilità di sicurezza.

Per integrare la sicurezza in tutte le fasi di DevOps, è necessario innanzitutto raggiungere un consenso sulle pratiche di sicurezza da adottare e dove associarle [48], insieme ad un chiaro obiettivo di sicurezza da raggiungere.

Successivamente, tutti gli strumenti necessari per attuare queste pratiche di sicurezza devono essere coinvolti nel processo [49]. Tra le pratiche di sicurezza, il monitoraggio continuo è ampiamente utilizzato in ambienti altamente regolamentati [50], in cui tutte le impostazioni, gli eventi, le attività, i registri e le notifiche devono essere monitorati costantemente.

Il monitoraggio continuo rappresenta una pratica fondamentale anche nei sistemi IoT, poiché consente di ottenere feedback costanti e tempestivi dai gruppi di sviluppo e operazioni, favorendo l'implementazione delle pratiche di sicurezza in DevOps. Tale pratica consiste nella rilevazione di minacce e anomalie di sicurezza nei registri delle attività, e consente agli sviluppatori di risolvere i problemi con maggior facilità e rapidità.

3.2.5 – D14: incompatibilità tra sicurezza e pratiche DevOps

DevOps ha come obiettivo principale quello di velocizzare il rilascio; tuttavia, molte pratiche di test di sicurezza richiedono l'intervento umano. Ad esempio, il *penetration testing* [44], è un'attività che richiede che qualcuno si occupi di configurare, controllare ed eseguire gli strumenti che eseguono il testing. La complessità di queste pratiche richiede notevoli risorse, pertanto l'adozione di rilasci rapidi potrebbe ostacolare l'esecuzione di approfonditi programmi di testing.

Inoltre, l'aumentare della complessità, le vulnerabilità e le dipendenze da componenti di terze parti hanno reso la garanzia di sicurezza ancora più difficoltosa [15]. In linea generale, l'uso di componenti

di terze parti, sebbene sia vantaggioso, può introdurre nuove vulnerabilità all'interno dell'applicazione. Pertanto, è necessario garantire la sicurezza anche di tutti i componenti esterni.

Gli sviluppatori si trovano spesso ad affrontare i compromessi tra la velocità di rilascio e l'integrazione di sicurezza [51], questi possono essere identificati come: garanzia; sicurezza; flessibilità. Con garanzia si intende il compromesso tra l'incremento del valore aggiunto che test aggiuntivi forniscono e il costo necessario per eseguire tali test; con sicurezza invece si intende il compromesso tra l'incremento di misure di sicurezza e l'abilità di accedere e modificare il sistema CI; infine, la flessibilità descrive il compromesso tra la necessità di avere sistemi potenti e altamente configurabili, e la necessità di avere sistemi semplici e facili da usare.

Molte organizzazioni purtroppo sono riluttanti a trasformare i loro processi di sviluppo e operazioni in DevSecOps [21], in quanto prevedono molte incompatibilità tra sicurezza e DevOps; tuttavia, queste incompatibilità vengono risolte nel momento in cui si adottano le corrette pratiche.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. È cruciale che un'organizzazione prima di adottare DevSecOps, sia al corrente delle sfide di integrazione delle pratiche, altrimenti l'imposizione della metodologia sarebbe fallimentare.

Soluzione

Le incompatibilità tra sicurezza e DevOps vengono risolte tramite l'ausilio delle giuste pratiche, su tutte è importante adottare l'approccio di sicurezza *shift-left*. A supportare il paradigma DevSecOps, possono essere implementate le pratiche di valutazione di sicurezza continua come, ad esempio, il monitoraggio continuo.

Per ridurre le incompatibilità tra sicurezza e DevOps si può usufruire di *patch* di sicurezza che vengono gestite da pratiche DevOps [44]. Un aspetto importante da tenere in conto quando si parla di questa metodologia è la rapidità; i rilasci del software, infatti, sono rapidi così come le fasi di sviluppo, ne consegue la possibilità di presenza di vulnerabilità nell'ambiente di produzione, in cui il software è disponibile per l'utilizzo.

Nonostante la presenza involontaria di vulnerabilità, è fondamentale affrontarle il prima possibile attraverso l'applicazione di *patch* di sicurezza, queste vengono distribuite rapidamente utilizzando le pratiche di DevOps.

Per promuovere l'adozione di DevSecOps, è importante facilitare la comunicazione e la collaborazione nel gruppo e tra i gruppi. In particolare, è molto utile costituire un gruppo multidisciplinare [42] all'interno dell'organizzazione, ovvero un gruppo in cui i membri hanno ruoli

e caratteristiche differenti ma che vogliono raggiungere lo stesso scopo. Inoltre, è importante diffondere la passione per la sicurezza tra gli sviluppatori e coinvolgerli attivamente nelle attività di sicurezza, compresi i compiti nella gestione degli incidenti. Questa collaborazione, a sua volta, consente di integrare le pratiche di sicurezza in ogni aspetto della pipeline continua di distribuzione, inoltre aiuta a ridurre notevolmente i problemi di vulnerabilità prima che il software venga rilasciato.

Il gruppo di sicurezza può contribuire e migliorare la collaborazione con gli altri gruppi, tramite attività di automazione esistenti di DevOps all'interno dell'organizzazione, [17] in particolare modificando gli strumenti di sicurezza esistenti, in modo tale da garantire un corto ciclo di *feedback* tra il gruppo di sicurezza e gli altri.

Inoltre, è importante che all'interno dell'organizzazione si diffonda la passione per la sicurezza tra gli sviluppatori [52], in quanto favorirebbe la risoluzione rapida dei problemi in caso di incidenti di sicurezza; tuttavia, è necessario che tali sviluppatori partecipino alle attività di sicurezza, in modo da avere dei compiti da svolgere, a tal proposito è importante includerli nel ciclo di vita della gestione degli incidenti.

Per garantire una maggiore collaborazione, è necessario adottare controlli e standard appropriati [21]. Ad esempio, è necessario stabilire una chiara separazione dei compiti tra i membri dei vari gruppi, inoltre è importante definire una chiara comunicazione tra i membri dei vari gruppi.

I metodi manuali di comunicazione come le e-mail non dovrebbero essere abusati, in quanto potrebbero causare ancora più spreco di tempo. I metodi automatici invece sono ideali per poter informare i membri del gruppo delle attività svolte o che sono in corso nei processi. Ad esempio, gli stakeholders dovrebbero essere notificati automaticamente da un sistema, invece che da un amministratore manualmente.

In ambienti altamente regolamentati, è necessario comunicare in modo continuo agli stakeholders informazioni sui rischi, i risultati delle valutazioni e dei processi logici [53]. A tal fine, è fondamentale che tutti gli stakeholders dispongano di un profilo che permetta la loro identificazione, e che siano progettati appositi canali di comunicazione per la diffusione di tali informazioni.

3.3 – Difficoltà e soluzioni relative alle infrastrutture

Questa sezione affronta le difficoltà di adozione di DevSecOps in diverse tipologie di infrastrutture complesse.

3.3.1 – D15: difficoltà nell'adottare DevSecOps in ambienti altamente regolamentati

Negli ambienti altamente regolamentati, come l'industria farmaceutica, l'adozione di pratiche continue di sicurezza come DevSecOps può essere una sfida complessa, poiché potrebbe essere difficile integrare la distribuzione continua di *patch* di sicurezza per i dispositivi medici [54]. L'aumento degli attacchi informatici in questa industria ha portato a un aumento della domanda di prodotti medici sicuri, inoltre ci sono leggi e regolamentazioni che richiedono ai produttori di progettare prodotti sicuri, oltre che valutare correttamente i rischi. Pertanto, per distribuire patch di sicurezza in modo continuo, gestire le vulnerabilità in modo trasparente e integrare la sicurezza nel ciclo di vita dello sviluppo, è necessaria una strategia di sicurezza informatica che includa controlli di sicurezza integrati, e la distribuzione sicura dei prodotti.

In questo contesto, risulta difficile integrare le pratiche continue di sicurezza di DevSecOps in ambienti di produzione isolati, come evidenziato da [55]. Ciò è dovuto alla mancanza di soluzioni DevOps appropriate per tali infrastrutture, che rendono difficile l'integrazione della sicurezza in tutte le fasi del ciclo di vita del software. Le architetture di sicurezza *zero-trust*, che caratterizzano questi ambienti, includono ambienti separati, politiche di accesso temporaneo e comunicazioni ristrette con gli stakeholders, che rappresentano una sfida significativa per l'implementazione delle pratiche di sicurezza di DevSecOps. Di conseguenza, l'adozione di misure di sicurezza continua in ambienti altamente regolamentati o isolati risulta complessa.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. La difficoltà descritta è rilevante solo per alcune organizzazioni, ovvero quelle che adottano infrastrutture altamente regolamentate o isolate. Tuttavia, per le aziende che intendono adottare DevSecOps su tali infrastrutture, è essenziale prendere in considerazione questo problema.

Soluzione

In ambienti altamente regolamentati, l'adozione di DevSecOps risulta complessa, ma può essere resa fattibile attraverso l'utilizzo delle giuste pratiche di sicurezza. Una soluzione parziale efficace è rappresentata dall'implementazione di pratiche di valutazione di sicurezza continua, in particolare il monitoraggio continuo, che risulta molto utilizzato in questi ambienti.

Inoltre, un modo efficace per incentivare l'adozione di DevSecOps in tali ambienti, consiste nel facilitare la collaborazione e la comunicazione all'interno e tra i gruppi, soprattutto in contesti altamente regolamentati dove ciò è particolarmente importante.

Per garantire la sicurezza nelle pipeline di lavoro condivise, è necessario assegnare ruoli e configurare i permessi di accesso, soprattutto in ambienti altamente regolamentati [55]. In questi casi, le politiche di gestione degli accessi limitati sono fondamentali, ad esempio l'assegnazione di privilegi minimi in base alle necessità dei singoli sviluppatori, che accedono alle configurazioni degli ambienti altamente regolamentati. Inoltre, è una buona pratica automatizzare i cambiamenti nell'ambiente di produzione; pertanto, potrebbe essere una buona idea rimuovere l'accesso degli sviluppatori all'ambiente di produzione.

È altamente consigliabile utilizzare IaC (*Infrastructure as Code*) in infrastrutture complesse e altamente regolamentate. L'utilizzo di IaC [53] permette di mantenere l'infrastruttura sotto controllo di versione, di essere testata, costruita e distribuita in modo affidabile, rappresentando spesso una soluzione preferita in ambienti complessi con requisiti di sicurezza rigorosi, come ad esempio ambienti fisicamente isolati e sicuri. Un gruppo di sviluppo può creare un servizio automatico centralizzato che fornisca e distribuisca sistemi e software. Inoltre, all'inizio del progetto è essenziale che gli stakeholder concordino sui sistemi e gli ambienti necessari per la compilazione e il test del progetto, in modo da risolvere il problema della distribuzione del software, attraverso un numero limitato di processi centralizzati, che rappresenta una sfida comune in ambienti altamente regolamentati.

In ambienti altamente regolamentati, è cruciale che le attività di gestione delle vulnerabilità siano sistematiche e trasparenti per garantire la sicurezza dei prodotti, come ad esempio dispositivi medici [54]. In particolare, è importante progettare una valutazione dei rischi consapevole per l'industria, ad esempio per progettare un sistema con configurazioni altamente regolamentate, data la complessità dei software che spesso includono componenti open source. Pertanto, è necessario valutare sistematicamente le vulnerabilità delle componenti utilizzate, anche in ambienti complessi come quelli altamente regolamentati.

In aggiunta, è consigliato creare ambienti di simulazione in questi contesti complessi, in cui gli sviluppatori possono condurre pratiche di testing, utilizzando il sistema di simulazione come un ambiente temporaneo [53]. In seguito, possono riportare i risultati ai vari stakeholder e gli utenti finali possono testare le versioni distribuite anche nell'ambiente di simulazione, fornendo ulteriori feedback anche in infrastrutture isolate, se necessario. Infine, per garantire la sicurezza del sistema, è consigliabile costruire politiche di gestione degli accessi limitati, assegnando ruoli e configurando i permessi di accesso, in modo da limitare gli accessi in base ai bisogni, soprattutto per gli sviluppatori che hanno bisogno di accedere alle configurazioni di tali ambienti.

Il framework ENACT [56] per sistemi IoT affidabili, contiene strumenti di testing e simulazione in uno dei suoi livelli. Tali strumenti consentono di creare scenari di test per le applicazioni in base alle

condizioni programmate, con lo scopo di fornire indicazioni sulle prestazioni, la resistenza al test e la gestione dei rischi. In particolare, questi strumenti mirano a fornire linee guida per valutare la capacità del sistema di fronteggiare i problemi di sicurezza.

3.3.2 – D16: difficoltà nell'adottare DevSecOps in ambienti con risorse limitate

In genere, quando si discute di ambienti con risorse limitate, ci si riferisce spesso a due tipi di infrastrutture: i sistemi *Internet of Things* (IoT) e i sistemi *embedded*. L'IoT è definito come una rete di oggetti, tra cui sensori e altre tecnologie che scambiano informazioni con altri dispositivi sulla rete. Questi oggetti possono includere dispositivi domestici o industriali [57]. Tuttavia, adottare DevSecOps su un'infrastruttura così eterogenea è una sfida, poiché esiste un elevato rischio di attacchi informatici, ed è altamente complesso mantenere e aggiornare l'infrastruttura nel tempo.

La complessità nell'implementare pipeline di distribuzione sicure e monitorare gli eventi di sicurezza nei sistemi IoT, a causa della loro natura eterogenea e distribuita, rappresenta una sfida significativa [50]. Questo può ostacolare lo sviluppo rapido e rendere difficile le operazioni e il monitoraggio dei sistemi di produzione, inclusi gli eventi di sicurezza informatica. Inoltre, l'infrastruttura deve essere altamente flessibile per consentire ai gruppi di configurare e monitorare la sicurezza, in base ai loro criteri specifici.

Non si dispone di molti approcci e strumenti progettati per sistemi IoT che adottano DevSecOps [56], a causa della diffusione dei sistemi Cloud che hanno a disposizione più strumenti.

I sistemi *embedded* rappresentano un'altra infrastruttura complessa per adottare DevSecOps [58]. Questi sono costituiti da sistemi elettronici di elaborazione a microprocessore, progettati per un utilizzo specifico e non riprogrammabili dagli utenti. Tuttavia, l'integrazione continua (CI) nei software sviluppati per questi sistemi risulta difficoltosa, a causa dell'importanza della conformità con gli standard, che non si concentra sulla rapidità del rilascio del software. Inoltre, la restrizione dell'accesso all'informazione è una problematica correlata ai problemi di sicurezza in questo settore. In sintesi, entrambe le infrastrutture hanno difficoltà a adottare DevSecOps e alcune pratiche di sicurezza.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. La difficoltà descritta è rilevante solo per alcune organizzazioni, ovvero quelle che adottano IoT oppure

sistemi embedded. Tuttavia, per le aziende che intendono adottare DevSecOps su tali infrastrutture, è essenziale prendere in considerazione questo problema.

Soluzione

Infrastrutture complesse come IoT e sistemi embedded hanno bisogno di pratiche di valutazione di sicurezza continua, al contempo è raccomandabile adottare IaC per gestire questa complessità, inoltre è di fondamentale importanza avere ambienti simulativi per effettuare attività di testing.

L'adozione di DevSecOps nelle infrastrutture IoT può essere supportata da modelli guidati [57]. Tali modelli dovrebbero essere in grado di affrontare le sfide specifiche delle infrastrutture IoT, come l'eterogeneità, e di supportare la definizione dei requisiti di sicurezza e privacy, nonché la distribuzione automatica e la gestione delle funzionalità di sicurezza.

GENESIS è un modello ingegneristico che adotta questo approccio per supportare DevSecOps nell'ambito dell'IoT. Inoltre, l'uso di un framework adatto è fondamentale per lo sviluppo di sistemi IoT intelligenti, che coinvolgono sensori e attuatori e richiedono elevati livelli di sicurezza, privacy, resistenza, affidabilità e protezione. ENACT [56], è un *framework* DevOps che offre un approccio per adattare le tecniche DevOps alle esigenze specifiche dei sistemi IoT intelligenti.

3.3.3 – D17: difficoltà nell'adottare DevSecOps in complessi ambienti Cloud

I principi e le pratiche di DevSecOps sono difficili da adottare in ambienti cloud complessi. Ad esempio, la produzione rapida di un software sicuro è difficoltosa in un ambiente cloud, in cui il sistema è in una forma *system-of-systems* (SoS) [59]. Tale sistema, comprende altri sistemi (i sistemi costituenti), che hanno indipendenza manageriale e operazionale. L'integrazione di applicazioni e servizi cloud all'interno di un tale sistema complesso comporta diverse difficoltà, specialmente riguardo l'aspetto della sicurezza. Pertanto, è fondamentale definire un framework adeguato che pianifichi e definisca le fasi di requisiti di sicurezza per un SoS, utilizzando l'approccio DevSecOps.

Gli ambienti multi-cloud, ovvero applicazioni che combinano servizi di cloud multipli ed eterogenei, rappresentano un'altra infrastruttura complessa [41]. Garantire la sicurezza in sistemi multi-cloud non è semplice, poiché è necessario controllare non solo i componenti di un singolo sistema e i rischi a cui sono esposti, ma anche quelli dei fornitori di cloud. La sicurezza, la privacy e la protezione dei dati continuano a rappresentare le maggiori barriere per l'adozione di infrastrutture cloud, soprattutto se si intende applicare un approccio DevSecOps.

In ambienti multi-cloud, l'utilizzo di microservizi è comune [60]. Questi possono essere definiti come una raccolta di servizi indipendenti che comunicano tra loro, mediante semplici meccanismi di comunicazione interna. La suddivisione di un'applicazione in microservizi consente di sviluppare e rilasciare rapidamente applicazioni cloud, tuttavia, l'intercomunicazione tra questi servizi espone il sistema a rischi di sicurezza, quali spionaggio di informazioni e altri possibili attacchi. Pertanto, l'adozione di DevSecOps risulta complessa in questo contesto.

Un altro aspetto fondamentale, che riguarda la sicurezza nelle infrastrutture cloud, è la sicurezza dei dati [61]. Produrre rapidamente il software e allo stesso tempo occuparsi di garantire la sicurezza dei dati, rappresenta un compito molto complesso; ciononostante, è fondamentale che la sicurezza sia garantita in questi ambienti.

La sicurezza dei dati è importante non solo durante l'esecuzione dell'applicazione cloud, ma anche durante tutte le altre fasi del ciclo di vita del software. Pertanto, è fondamentale che i dati siano protetti sia da attaccanti remoti che locali, sia quando il sistema è in esecuzione che quando è spento.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. La difficoltà descritta è rilevante per molte organizzazioni, ovvero quelle che adottano Cloud, in quanto è l'infrastruttura più diffusa. Pertanto, le aziende che intendono adottare DevSecOps su tali infrastrutture, è fondamentale che prendano in considerazione questo problema.

Soluzione

Gli ambienti cloud possono essere molto complessi, a tal proposito è raccomandato utilizzare IaC, inoltre è fondamentale, per il paradigma DevSecOps, che ci sia una comunicazione e collaborazione facilitata all'interno dei gruppi e tra i gruppi.

Un aspetto importante delle soluzioni cloud è la sicurezza dei dati. Questo aspetto è complesso, difatti assicurare la sicurezza dei dati risulta difficoltoso, soprattutto in una metodologia che mira a rilasciare versioni del software rapidamente. Per affrontare questa problematica occorre concentrarsi sulla sicurezza dei dati in parallelo con il ciclo di sviluppo [61]. A tal fine, è stato realizzato il *Software as a Service (SaaS) security life cycle* (SSLC), che combina la sicurezza dei dati e i cicli di vita di sviluppo del software. Questa soluzione include la raccolta e l'analisi dei requisiti, la progettazione, l'implementazione, il testing, la distribuzione e la manutenzione.

Infine, anche per gli ambienti cloud è ideale adottare dei *framework* appositi per DevSecOps. Sono stati realizzati diversi framework per gli ambienti cloud, in quanto sono l'infrastruttura più utilizzata dalla metodologia.

Per gli ambienti cloud che utilizzano SoS è stato realizzato un *framework* incentrato sulla sicurezza [59], ovvero SoS *governance*. Questo, supporta l'integrazione di pratiche di sicurezza, come ad esempio l'ingegneria sicura, la separazione dei compiti, la distribuzione e le operazioni sicure, la reperibilità e la continuità di business, formazione e revisione.

È stato sviluppato il *framework* MUSA [62], per garantire la sicurezza e la privacy nei sistemi multi-cloud, unificando l'approccio preventivo e reattivo alla sicurezza. In termini di prevenzione, il framework supporta le pratiche di *Security by Design* durante lo sviluppo dell'applicazione, così come l'integrazione di meccanismi di sicurezza richiesti nell'applicazione stessa. Per quanto riguarda la sicurezza reattiva, MUSA adotta pratiche di monitoraggio per rilevare, notificare e mitigare gli incidenti di sicurezza, in modo che i fornitori delle applicazioni multi-cloud siano informati e pronti a rispondere rapidamente ad attacchi e problemi di sicurezza.

Un altro *framework* di sicurezza utilizza le funzioni di virtualizzazione della rete (NFV) [63] e pattern progettuali di microservizi. Questo framework è usato per ambienti cloud eterogenei e distribuiti.

Infine, viene proposto lo *Unicorn Framework* [61], il quale mira ad affrontare le sfide legate alla progettazione di applicazioni scalabili e sicure, che sono basate su architetture di microservizi in ambienti multi-cloud. Il framework fornisce un insieme completo di strumenti che aiutano i gruppi nello sviluppo, nella progettazione e nella gestione di applicazioni containerizzate scalabili e sicure in ambienti multi-cloud.

3.4 – Difficoltà e soluzioni relative al personale e alla cultura organizzativa

Questa sezione tratta delle difficoltà legate alla conoscenza, alle competenze e alla collaborazione tra i membri dei diversi gruppi DevSecOps, nonché delle sfide legate alla cultura dell'organizzazione.

3.4.1 – D18: problemi di collaborazione tra i gruppi

La comunicazione e la collaborazione efficaci tra i membri dei gruppi sono un fattore critico di successo in DevSecOps. Tuttavia, la corretta collaborazione tra i gruppi non sempre viene praticata e spesso sorgono conflitti tra il gruppo di sviluppo e quello di sicurezza [14].

Tipicamente i programmati spendono molto tempo nella scrittura del codice che, nella gran parte dei casi, subisce il giudizio da parte dei professionisti di sicurezza, che valutano il loro lavoro. Ciò può causare una perdita di autonomia da parte degli sviluppatori, che si sentono a volte inferiori e

incapaci di prendere decisioni riguardo al codice che hanno scritto, dovendo invece adattarsi alle direttive e ai consigli dei professionisti di sicurezza. Inoltre, l'opinione degli sviluppatori, anche se competenti in materia di sicurezza, potrebbe essere ritenuta di minor valore rispetto a quella degli esperti del team di sicurezza.

Storicamente l'ingegneria del software e la sicurezza del software lavorano separatamente. Queste separazioni impediscono comunicazioni e collaborazioni tra gli stakeholders, di cui fanno parte i membri del gruppo di sicurezza. Gli altri gruppi devono comunicare e collaborare frequentemente con i membri del gruppo di sicurezza, inoltre è fondamentale che si predisponga una responsabilità condivisa, tuttavia spesso ciò non accade.

È necessario disporre di strumenti che agevolino la collaborazione con il gruppo di sicurezza [64]. In particolare, nell'ambito delle fasi iniziali del processo di sviluppo, l'uso di strumenti potrebbe essere vantaggioso per la gestione e la fornitura di modelli di progettazione, in modo da consentire una comprensione collaborativa della motivazione progettuale del team di sicurezza o di altri componenti di altri gruppi.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio. La collaborazione e la comunicazione nel gruppo e tra i gruppi è importante; tuttavia, eventuali divergenze nell'organizzazione raramente comprometteranno la produzione del software, ciononostante l'azienda deve occuparsi di garantire un ambiente sano e collaborativo.

Soluzione

Per migliorare la collaborazione e la comunicazione tra i gruppi all'interno di DevSecOps, è necessario avere un ruolo di "campione di sicurezza" che funga da ponte tra il gruppo di sicurezza e gli altri [14]. I membri con questo ruolo sono tipicamente dei programmati che hanno ricevuto la migliore formazione in ambito di sicurezza, questi devono occuparsi di facilitare la comunicazione tra i programmati e gli altri gruppi, ossia saranno le prime persone ad essere chiamate quando un problema di sicurezza è identificato esternamente. Pertanto, la squadra riceve un messaggio da uno della propria squadra e non dall'esterno, inoltre si occupano di comunicare con gli altri membri e si assicurano che la sicurezza venga integrata in tutte le fasi del processo.

In genere, è preferibile che i membri con il ruolo di campione di sicurezza siano integrati nel gruppo di sviluppo [44]. Ciò può agevolare l'accettazione delle loro indicazioni e dei loro suggerimenti da parte degli sviluppatori, in quanto i campioni di sicurezza fanno parte dello stesso gruppo e hanno una maggiore conoscenza sulla sicurezza rispetto agli sviluppatori.

A supporto del ruolo di campione di sicurezza, è possibile migliorare la comunicazione e la collaborazione tra i gruppi mediante l'adozione di appropriate norme e standard.

Inoltre, è possibile utilizzare un servizio Cloud, come ad esempio CAIRIS [64], per consentire agli esperti di sicurezza e agli altri gruppi di lavorare insieme su modelli di progettazione, promuovendo così la collaborazione interna tra i vari gruppi.

Infine, l'approccio fondamentale di DevSecOps, ovvero la sicurezza *shift-left*, che aiuta molto a migliorare la collaborazione interna e tra i gruppi, in quanto fin dalle prime fasi i membri dei gruppi devono collaborare e comunicare in maniera tale da integrare la sicurezza.

3.4.2 – D19: differenze di conoscenza in sicurezza

È stato dimostrato che DevSecOps supporta gli sviluppatori nell'affrontare le attività di sicurezza; tuttavia, non tutti gli sviluppatori hanno le stesse competenze in materia di sicurezza, e spesso le conoscenze e le abilità non sono sufficienti per eseguire determinati compiti. In particolare, la mancanza di formazione, addestramento e competenze in materia di sicurezza da parte degli sviluppatori, costituisce una sfida significativa per l'adozione di DevSecOps [14].

In ambito DevSecOps, il principale problema è rappresentato dalla separazione delle formazioni di ingegneria del software e sicurezza del software. Gli sviluppatori non sono solitamente adeguatamente formati in sicurezza, il che li rende impreparati nel settore.

Secondo un'altra teoria [18], la sicurezza di DevOps è complessa e richiede personale altamente qualificato per contrastare gli attacchi. Poiché le pratiche DevOps sono diffuse nell'organizzazione, potrebbe essere difficile reperire personale con le abilità richieste per gestire tutte queste pratiche. Di conseguenza, sarebbe opportuno semplificare gli aspetti di sicurezza di DevOps, in maniera tale da facilitare il lavoro agli sviluppatori che sono poco specializzati.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello basso. In un'organizzazione che adotta DevSecOps, le differenze di conoscenza tra gli sviluppatori non rappresentano un problema significativo, poiché è diffuso che i membri del gruppo migliorino le loro abilità nel tempo e attraverso l'esperienza pratica.

Soluzione

Attuare una formazione sulla sicurezza e metodi di conoscenza condivisa, potrebbe favorire il miglioramento in ambito di sicurezza, dei membri del gruppo di sviluppo, in maniera tale da poter eseguire dei compiti importanti. Ad esempio, una buona comprensione delle questioni di sicurezza è essenziale per scegliere e utilizzare strumenti appropriati [14]; se il gruppo di sviluppo utilizza strumenti di analisi statica del codice (SAST) per identificare vulnerabilità, deve essere consapevole che questi strumenti possono generare falsi positivi e spetta ai membri del gruppo identificarli.

Tuttavia, non è auspicabile fornire formazione specialistica in materia di sicurezza del software, a tutti gli sviluppatori [44]. Ciononostante, è possibile formare sufficientemente ciascuno di essi, in modo che siano in grado di identificare le aree in cui è necessario il supporto di un esperto. Si ritiene che se gli sviluppatori acquisissero una conoscenza sufficiente sulla sicurezza del software, il numero di anomalie di sicurezza potrebbe diminuire del 50%.

Una strategia efficace per mitigare le vulnerabilità di sicurezza consiste nell'offrire una formazione sulla sicurezza [17], finalizzata a preparare gli sviluppatori a integrare la sicurezza all'interno delle organizzazioni DevOps. Questo processo di formazione potrebbe includere diverse attività, tra cui il completamento di corsi online pertinenti, la partecipazione a *bootcamp* per sviluppatori e l'organizzazione di incontri aziendali.

Per quanto riguarda l'uso di metodi di conoscenza condivisa [14], è comune condurre una retrospettiva, ovvero una revisione dell'apprendimento, al fine di valutare i risultati. È importante notare che la retrospettiva è incolpevole, cioè che ogni individuo ha fatto del suo meglio in base alle risorse disponibili, alle proprie capacità e alle circostanze in cui si trovava, indipendentemente dai risultati ottenuti.

3.4.3 – D20: difficoltà nella cultura organizzativa

Per adottare con successo DevSecOps, come precedentemente evidenziato, è necessario apportare cambiamenti comportamentali e promuovere una cultura della sicurezza. Tuttavia, questo processo può portare a competizione tra i membri del gruppo [59], e alla paura di essere sostituiti. Al contrario, questo meccanismo dovrebbe rappresentare un'opportunità per crescere e ampliare la propria conoscenza in ambito di sicurezza, senza perdere la propria posizione. È necessario che i membri dei gruppi collaborino e si comportino in modo corretto all'interno del proprio gruppo, oltre che con gli altri gruppi.

Un problema ancora più significativo, che impedisce l'adozione di DevSecOps, è la reticenza nel dare la priorità alla sicurezza all'interno dell'organizzazione [14]. In particolare, molti sviluppatori e altri membri del gruppo non vogliono assumersi le responsabilità relative alla sicurezza, in quanto ritengono che lo sforzo richiesto sia eccessivo e il rischio associato sia troppo elevato.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello medio. In un'organizzazione che adotta DevSecOps, la difficoltà nella cultura organizzativa rappresenta una sfida significativa, poiché la cultura orientata sulla sicurezza è fondamentale per adottare tale metodologia.

Soluzione

Una soluzione per facilitare l'adozione di DevSecOps, che porti ad una cultura orientata sulla sicurezza, consiste nel portare avanti una formazione sulla sicurezza e metodi di conoscenza condivisa, i cui benefici sono stati già esaminati nella presente sezione.

Il cambiamento di cultura per adottare DevSecOps può essere difficoltoso per alcune persone dell'organizzazione. Per affrontare questo problema, è raccomandabile portare avanti dei programmi di gestione delle risorse umane [59] (HRM), in parallelo alle trasformazioni dei progetti DevSecOps.

In genere, i programmi di formazione sulla sicurezza dovrebbero mirare a indirizzare sentimenti comuni tra i membri del gruppo, come la paura di perdere il controllo del proprio lavoro o di essere sostituiti. È fondamentale che l'ambiente sia favorevole al cambiamento, inoltre DevSecOps deve essere percepita come un'opportunità di migliorare le proprie competenze.

3.4.4 – D21: Minacce interne

Nel contesto di DevOps, il ruolo dello sviluppatore è cambiato e spesso ha accesso ai servizi di produzione. Questo ampliamento dei permessi ha aumentato il rischio di insider, cioè di utenti con accesso all'ambiente di produzione, che possono utilizzare strumenti per danneggiare il sistema o addirittura prenderne il controllo [65].

Una collaborazione senza limitazioni tra i gruppi può portare a problemi di sicurezza come accessi inappropriati alle risorse del sistema [17]. In altre parole, quando i gruppi collaborano troppo strettamente, i singoli membri possono abusare del loro potere di accesso alle risorse, e cambiare accidentalmente o intenzionalmente le proprietà del sistema.

Livello di criticità

In base alla valutazione della criticità, questa difficoltà può essere considerata di livello alto. È di fondamentale importanza per le organizzazioni che adottano DevSecOps, prendere in considerazione

questa problematica, poiché l'abuso dei permessi di accesso alle risorse del sistema può causare gravi violazioni della sicurezza.

Soluzione

In una prospettiva di mitigazione delle minacce interne, è stato proposto un *framework* di protezione dell'integrità dei dati e del sistema [65]. In particolare, è stato realizzato un insieme di requisiti di protezione dell'integrità basato sulle possibili minacce, e successivamente è stato sviluppato un *framework* mirato alla protezione dell'integrità globale all'interno dei sistemi basati su microservizi.

Un esempio di misura adottata per contrastare le minacce interne è l'impiego di un componente di votazione sicura, in cui i membri più affidabili dei gruppi sono chiamati a votare se accettare o meno i frequenti cambiamenti apportati agli *artifact* (ovvero scripts e files di configurazione). In questo modo, anche se tali modifiche sono state effettuate da *insider* malevoli, è comunque possibile controllarle e scartarle.

3.5 – Risultati ottenuti

L'analisi delle sfide e delle soluzioni fornisce una panoramica esaustiva della metodologia. È stato evidenziato che le sfide più significative, nell'implementazione della metodologia DevSecOps, riguardano principalmente gli strumenti, che sono nove in totale. Inoltre, sono state identificate cinque problematiche relative alle pratiche da integrare, quattro sfide relative al personale e alla cultura organizzative e infine tre sfide riguardanti le infrastrutture. Ogni sfida individuata durante l'analisi può avere un impatto negativo sull'utilizzo corretto della metodologia DevSecOps. Tuttavia, è importante classificare le difficoltà in base al loro livello di criticità e concentrarsi principalmente su quelle più rilevanti. Di conseguenza, per un'organizzazione che vuole adottare la metodologia, risolvere la maggior parte o tutte le problematiche identificate risulta essere di fondamentale importanza. Le soluzioni proposte sono efficaci per raggiungere questo obiettivo. Le difficoltà, affiancata dal livello di criticità, e le relative soluzioni sono rappresentate nella tabella 2.

Numero difficoltà	livello di criticità	Difficoltà	Soluzioni
Strumenti			
D1	medio	Difficoltà nella scelta degli strumenti.	- Standardizzazione degli strumenti.
D2	basso	Problemi di sicurezza legati alla complessità degli strumenti e problemi di integrazione.	- Documentazione con supporto alla sicurezza.
D3	alto	Problemi di gestione della configurazione degli strumenti.	- Documentazione con supporto alla sicurezza. - Utilizzo delle migliori pratiche per l'uso dello strumento.
D4	basso	Limiti degli strumenti di analisi statica che colpiscono i cicli di distribuzione rapida.	- Utilizzo delle migliori pratiche per l'uso dello strumento. - Utilizzo di strumenti IAST. - Muoversi verso soluzioni cloud.
D5	basso	Limitato utilizzo degli strumenti DAST in DevSecOps.	- Utilizzo di strumenti IAST. - Utilizzo delle migliori pratiche per l'uso dello strumento.
D6	alto	Limiti di sicurezza o vulnerabilità che colpiscono l'ecosistema dei Container.	- Utilizzo di piattaforme di orchestrazione. - Adozione di strumenti per la valutazione continua delle vulnerabilità.
D7	alto	Vulnerabilità che riguardano i sistemi di <i>Continuous Integration</i> .	- Utilizzare uno strumento di virtualizzazione per incapsulare parte del sistema.
D8	medio	Limitazioni degli script e strumenti IaC o CaC.	- Analisi statica degli script IaC.
D9	alto	Limiti e vulnerabilità di sicurezza che colpiscono la CDP.	- Frammenti di progettazione riutilizzabili. - Utilizzo di pratiche di analisi di minacce di sicurezza.

Pratiche			
D10	medio	Difficoltà ad automatizzare le tradizionali pratiche di sicurezza manuali.	<ul style="list-style-type: none"> - Adattare standards, politiche, modelli, contratti di servizio su criteri testabili.
D11	medio	Inabilità ad effettuare una rapida valutazione dei requisiti di sicurezza.	<ul style="list-style-type: none"> - Rilevamento automatizzato di vulnerabilità attraverso l'analisi dei requisiti.
D12	medio	Difficoltà relative alle pratiche di misura di sicurezza in DevOps.	<ul style="list-style-type: none"> - Utilizzo di Big Data e tecniche di analisi comportamentali. - Elaborazione efficace della documentazione e strategie di registrazione. - Definizione di metriche di sicurezza.
D13	medio	Difficoltà relative all'analisi della sicurezza continua.	<ul style="list-style-type: none"> - Utilizzo di strumenti per la valutazione continua delle vulnerabilità. - Usare l'approccio di sicurezza <i>shift-left</i>. - Implementare pratiche di valutazione continua della sicurezza.
D14	alto	Incompatibilità tra sicurezza e pratiche DevOps.	<ul style="list-style-type: none"> - Adottare l'approccio di sicurezza <i>shift-left</i>. - Gestione delle patch di sicurezza. - Implementare pratiche di valutazione di sicurezza continua. - Facilitare la comunicazione e la collaborazione all'interno e tra i gruppi.

Infrastrutture			
D15	alto	Difficoltà nell'adottare DevSecOps in ambienti altamente regolamentati.	<ul style="list-style-type: none"> - Implementare pratiche di valutazione di sicurezza continua. - Facilitare la comunicazione e la collaborazione interna. - Attuare politiche di gestione di accessi limitati. - Adottare IaC. - Valutazione delle vulnerabilità specifiche per prodotto. - Creare degli ambienti duplicati o simulativi per eseguire test.
D16	alto	Difficoltà nell'adottare DevSecOps in ambienti con risorse limitate.	<ul style="list-style-type: none"> - Implementare pratiche di valutazione di sicurezza continua. - Adottare IaC. - Ingegneria guidata dal modello per supportare DevSecOps. - Creare ambienti simulativi o duplicati per eseguire test. - Adottare framework che supporti DevSecOps.
D17	alto	Difficoltà nell'adottare DevSecOps in complessi ambienti Cloud.	<ul style="list-style-type: none"> - Adottare IaC. - Facilitare la comunicazione e la collaborazione nel gruppo e tra i gruppi. - Adottare un framework per DevSecOps. - Cicli di vita ibridi che mirano alla sicurezza dei dati.

Personale e cultura organizzativa			
D18	medio	Problemi di collaborazione tra i gruppi.	<ul style="list-style-type: none"> - Avere campioni di sicurezza nei gruppi. - Facilitare la comunicazione e collaborazione interna dei gruppi con gli appropriati controlli o standards. - Usare l'approccio di sicurezza <i>shift-left</i>. - Muoversi verso delle soluzioni cloud.
D19	basso	Differenze di conoscenza in sicurezza.	<ul style="list-style-type: none"> - Attuare una formazione sulla sicurezza e metodi di conoscenza condivisa.
D20	medio	Difficoltà nella cultura organizzativa.	<ul style="list-style-type: none"> - Attuare una formazione sulla sicurezza e metodi di conoscenza condivisa. - Portare avanti programmi di gestione delle risorse umane (HRM) in parallelo.
D21	alto	Minacce interne.	<ul style="list-style-type: none"> - Framework di protezione delle integrità.

Tabella 2 : Riepilogo delle difficoltà e delle soluzioni associate

L'analisi ha rilevato che su un totale di 21 sfide individuate, nove di esse presentano un elevato livello di criticità, otto un livello di criticità medio e quattro un basso livello di criticità. È importante sottolineare che le difficoltà relative alle infrastrutture, sono considerate altamente critiche solo se l'organizzazione fa uso di tali infrastrutture. La diffusione dei livelli di criticità è raffigurata nel grafico 1.

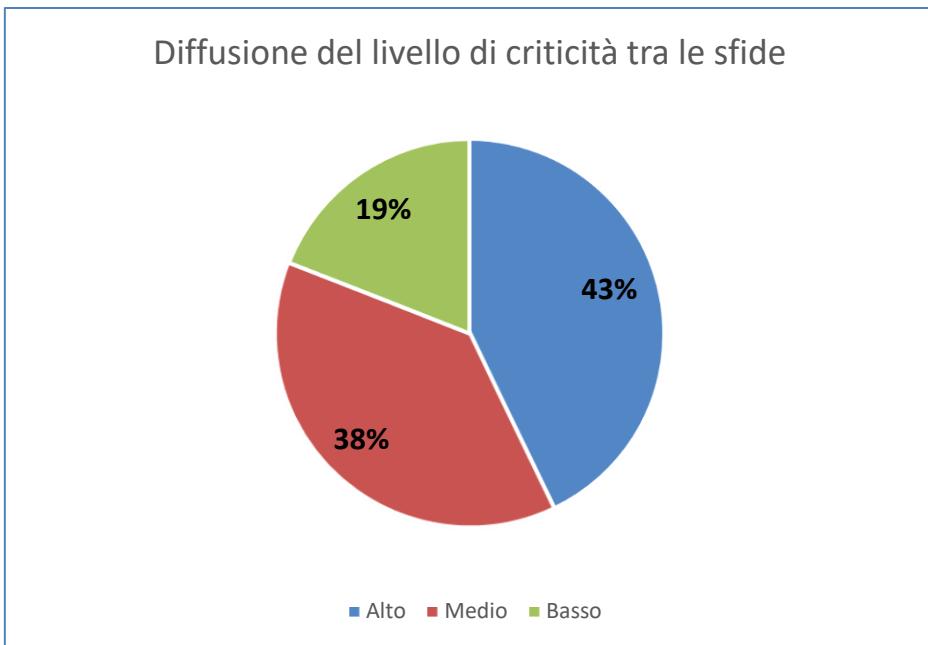


Grafico 1: criticità delle sfide di DevSecOps

3.6 – Ricerca e miglioramenti

In precedenza, tutte le possibili problematiche che possono emergere durante l'adozione della metodologia DevSecOps sono state categorizzate ed esaminate, e ad ognuna di esse è stata associata almeno una soluzione ed un livello di criticità. Dalle analisi effettuate risulta che DevSecOps rappresenta l'approccio ideale per le organizzazioni che desiderano rilasciare frequentemente software sicuro, nonostante la complessità della metodologia stessa. Pertanto, seguendo tutte o la maggior parte delle soluzioni proposte, qualsiasi organizzazione interessata può adottare la metodologia in modo adeguato. Tuttavia, è importante sottolineare che nella metodologia ci sono ancora aree che richiedono ulteriore ricerca. In seguito, vengono riportate tutte queste aree:

- 1) Occorre una maggiore ricerca su strumenti e metodi di sicurezza dell'applicazione, che sono ad hoc per DevSecOps: Nelle soluzioni è stato discusso di strumenti SAST e DAST che sono spesso inadeguati, in quanto rallentano molto i rilasci del software, a tal proposito è stato trovato un compromesso con gli strumenti IAST, che esistono e sono disponibili, ma una maggiora ricerca su strumenti adeguati sarebbe ottimale;
- 2) Occorre una maggiore ricerca su come altri metodi di testing di sicurezza dell'applicazione, possano essere forniti come servizi (ad esempio DAST): È presente solo uno studio che dimostra che gli strumenti SAST se utilizzati e forniti come servizi possono risolvere i problemi di rallentamento;

- 3) È necessario realizzare degli strumenti di sicurezza incentrati sugli sviluppatori: la maggior parte degli strumenti a disposizione delle organizzazioni sono complicati per gli sviluppatori, che spesso sono disincentivati a impiegarli;
- 4) Gli sviluppatori spesso non sanno come trovare un equilibrio tra la sicurezza e la rapidità di DevOps: Quando si individuano delle vulnerabilità, è necessario trovare il giusto compromesso per agire senza rallentare eccessivamente i processi DevOps, ma senza trascurare la sicurezza.
- 5) Occorre una maggiore ricerca su consensi per l'approccio di sicurezza *shift-left*: ci sono solo pochi studi che offrono delle soluzioni su come poter adottare questo approccio, e come riuscire ad automatizzare le tradizionali pratiche manuali; pertanto, occorrerebbe una maggiore ricerca su consensi e pratiche;
- 6) Occorre una maggiore ricerca sulle metriche di sicurezza: ci sono pochi studi che hanno stabilito delle metriche di sicurezza adattabili a DevSecOps;
- 7) Rendere chiari i ruoli in DevSecOps: i ruoli decisionali degli sviluppatori che si occupano di compiti di sicurezza non sono ben definiti e chiari;
- 8) Occorre una maggiore ricerca sulle difficoltà del personale nell'adozione di DevSecOps: le difficoltà del personale risultano essere poche in confronto a quelle che riguardano gli strumenti, bisognerebbe avere degli ulteriori studi sociotecnici che si concentrino su queste difficoltà e di come impattano su DevSecOps;
- 9) Validazione empirica di *framework*: la maggior parte dei *framework* proposti rappresentano dei casi di studio o delle soluzioni proposte dalla stessa azienda che ne aveva bisogno; pertanto, occorrerebbero maggiori studi per comprendere l'applicabilità di queste soluzioni, specialmente in differenti contesti.

Capitolo 4 – Conclusione

Dall’attività di analisi delle difficoltà e delle soluzioni e dai risultati ottenuti, si può concludere che DevSecOps è una metodologia valida per tutte le organizzazioni che vogliono produrre software sicuri e rapidi. Gli strumenti di sicurezza rappresentano un elemento fondamentale per il successo di DevSecOps. Mentre sono state individuate diverse problematiche, tutte possono essere gestite in modo efficace attraverso la standardizzazione degli strumenti, l’adozione di strumenti IAST, la fornitura di documentazione sugli strumenti, l’utilizzo di soluzioni Cloud e di strumenti IaC.

Per quanto riguarda le pratiche di sicurezza, è stato osservato quanto possa essere difficile attuarle in questa metodologia; tuttavia, anch’esse sono gestite con, ad esempio, l’utilizzo di metriche di sicurezza, l’analisi dei requisiti di sicurezza, la facilitazione della collaborazione e comunicazione nel gruppo e tra i gruppi, l’utilizzo dell’approccio di sicurezza *shift-left*.

Un’altra tematica esaminata è stata l’infrastruttura, in particolar modo si è osservato che per alcune infrastrutture come, ad esempio, gli ambienti cloud complessi (multi-cloud, SoS ecc.), gli ambienti altamente regolamentati e gli ambienti con risorse limitate, è molto complesso adottare la metodologia DevSecOps; ciononostante, per ciascuna di esse ci sono delle soluzioni che facilitano l’adozione, in particolare tutte condividono la necessità di un *framework* apposito, la facilitazione della collaborazione e comunicazione tra i gruppi e nei gruppi e l’utilizzo di IaC.

Infine, è stato esaminato un aspetto di fondamentale importanza per la metodologia DevSecOps, riguardante il personale e la cultura organizzativa. In particolare, si sono prese in considerazione le difficoltà riscontrate nella collaborazione tra i membri del gruppo, le differenze di conoscenze in materia di sicurezza, le problematiche nell’implementare una cultura orientata alla sicurezza e le possibili minacce interne. Tuttavia, anche queste sfide sono state affrontate attraverso l’adozione di soluzioni che migliorano le competenze del personale in materia di sicurezza, favoriscono una comunicazione efficace tra i membri del team, riducono le differenze di conoscenza in ambito di sicurezza, gestiscono le minacce interne e promuovono il cambiamento culturale verso un approccio centrato sulla sicurezza.

DevSecOps rappresenta una metodologia complessa che richiede ulteriori ricerche in alcune aree. Tuttavia, data l’incremento degli attacchi informatici nel corso dell’ultimo anno, diventa sempre più importante per le organizzazioni integrare il concetto di sicurezza, che spesso viene trascurato o affrontato con superficialità. Un’organizzazione che produce software sicuro e veloce ha un vantaggio competitivo sul mercato e garantisce la sicurezza degli utenti finali, nonché della stessa azienda, riducendo l’esposizione ai possibili attacchi informatici.

Riferimenti

- [1] Jabbari, R., Ali, N. bin, Petersen, K., & Tanveer, B. (2016). What is DevOps? A systematic mapping study on definitions and practices. *ACM International Conference Proceeding Series, 24-May-2016*.
- [2] Zaheeruddin Ahmed, & Shoba. C. Francis. (2019). *Integrating Security with DevSecOps: Techniques and Challenges*. Proceeding of 2019 International Conference on Digitization: Landscaping Artificial Intelligence.
- [3] Török, M., & Pataki, N. (2020). *DevOps Dashboard with Heatmap*. Proceedings of the 11th International Conference on Applied Informatics, Eger, Hungary, January 29–31, 2020.
- [4] Stoyanova, M. (2019). SMART CONCEPT FOR PROJECT MANAGEMENT – TRANSITION TO DevOps. In *KNOWLEDGE-International Journal* (Vol. 34).
- [5] Sánchez-Gordón, M., & Colomo-Palacios, R. (2020). *Security as Culture A Systematic Literature Review of DevSecOps*.
- [6] Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A multivocal literature review. *Communications in Computer and Information Science, 770*, 17–29.
- [7] Nandgaonkar, S., & Khatavkar, V. (2022). CI-CD Pipeline For Content Releases. *2022 IEEE 3rd Global Conference for Advancement in Technology, GCAT 2022*.
- [8] Sethi, F. (2020). Automating Software Code Deployment Using Continuous Integration and Continuous Delivery Pipeline for Business Intelligence Solution. *International Journal of Innovation Scientific Research and Review, 02(10)*, 445–449.
- [9] Dupont, S., Ginis, G., Malacario, M., Porretti, C., Maunero, N., Ponsard, C., & Massonet, P. (2021). Incremental Common Criteria Certification Processes using DevSecOps Practices. *Proceedings - 2021 IEEE European Symposium on Security and Privacy Workshops, Euro S and PW 2021*, 12–23.
- [10] Yang, Y., Shen, W., Ruan, B., Liu, W., & Ren, K. (2021). Security Challenges in the Container Cloud. *Proceedings - 2021 3rd IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2021*, 137–145.
- [11] Chen, S. J., Pan, Y. C., Ma, Y. W., & Chiang, C. M. (2022). The Impact of the Practical Security Test during the Software Development Lifecycle. *International Conference on Advanced Communication Technology, ICACT, 2022-February* 313–316.

- [12] Jiang, Y., & Adams, B. (2015). Co-evolution of infrastructure and source code - An empirical study. *IEEE International Working Conference on Mining Software Repositories, 2015-August*, 45–55.
- [13] Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. In *Information and Software Technology* (Vol. 141). Elsevier B.V.
- [14] Tomas, N., Li, J., & Huang, H. (2019, June 1). An empirical study on culture, automation, measurement, and sharing of DevSecOps. *2019 International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2019*.
- [15] Yasar, H., & Kontostathis, K. (2017). Where to Integrate Security Practices on DevOps Platform. *International Journal of Secure Software Engineering*, 7(4), 39–50.
- [16] Soenen, T., van Rossem, S., Tavernier, et al. (2018). Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps methodology. *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*, 1–6.
- [17] Rahman, A. A. U., & Williams, L. (2016). Software security in DevOps: Synthesizing practitioners' perceptions and practices. *Proceedings - International Workshop on Continuous Software Evolution and Delivery, CSED 2016*, 70–76.
- [18] Wilde, N., Eddy, B., Patel, K., Cooper, N., Gamboa, V., Mishra, B., & Shah, K. (2016). Security for Devops Deployment Processes: Defenses, Risks, Research Directions. *International Journal of Software Engineering & Applications*, 7(6), 01–16.
- [19] Rafi, S., Yu, W., & Akbar, M. A. (2020). Towards a Hypothetical Framework to Secure DevOps Adoption: Grounded Theory Approach. *ACM International Conference Proceeding Series*, 457–462.
- [20] Rahman, A., Parnin, C., & Williams, L. (2019). The Seven Sins: Security Smells in Infrastructure as Code Scripts. *Proceedings - International Conference on Software Engineering, 2019-May*, 164–175.
- [21] Mohan, V., ben Othmane, L., & Kres, A. (2018). BP: Security concerns and best practices for automation of software deployment processes: An industrial case study. *Proceedings - 2018 IEEE Cybersecurity Development Conference, SecDev 2018*, 21–28.
- [22] Steffens, A., Licher, H., & Moscher, M. (n.d.). *Towards Data-driven Continuous Compliance Testing*. (2018). CSE 2018: 3rd Workshop on Continuous Software Engineering.

- [23] Kritikos, K., Papoutsakis, M., Ioannidis, S., & Magoutis, K. (2019). Towards configurable cloud application security. *Proceedings - 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2019*, 684–689.
- [24] Zampetti, F., Scalabrino, S., Oliveto, R., Canfora, G., & di Penta, M. (2017). How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines. *IEEE International Working Conference on Mining Software Repositories*, 334–344.
- [25] Kupsch, J. A., Miller, B. P., Basupalli, V., & Burger, J. (n.d.). *From Continuous Integration to Continuous Assurance*. 2017 IEEE 28th Annual Software Technology Conference (STC).
- [26] Siewruk, G., Mazurczyk, W., & Karpiński, A. (2019). Security assurance in Devops methodologies and related environments. *International Journal of Electronics and Telecommunications*, 65(2), 211–216.
- [27] Lescisin, M., Mahmoud, Q. H., & Cioraca, A. (2019). Design and implementation of SFCI: A tool for security focused continuous integration. *Computers*, 8(4).
- [28] Brady, K., Moon, S., Nguyen, T., & Coffman, J. (2020). Docker Container Security in Cloud Computing. *2020 10th Annual Computing and Communication Workshop and Conference, CCWC 2020*, 975–980.
- [29] Bjørgeengen, J. (2019). A Multitenant Container Platform with OKD, Harbor Registry and ELK. *ISC High Performance 2019: High Performance Computing* pp 69–79.
- [30] Combe, T., Martin, A., & di Pietro, R. (2016). To Docker or Not to Docker: A Security Perspective. *IEEE Cloud Computing*, 3(5), 54–62.
- [31] Martin, A., Raponi, S., Combe, T., & di Pietro, R. (2018). Docker ecosystem – Vulnerability Analysis. *Computer Communications*, 122, 30–43.
- [32] Khan, A. (2017). *Key Characteristics of a Container Orchestration Platform to Enable a Modern Application*. IEEE Cloud Computing Vol.4 Issue 5.
- [33] Gruhn, V., Hannebauer, C., & John, C. (2013). *Security of Public Continuous Integration Services*. WikiSym '13: Proceedings of the 9th International Symposium on Open Collaboration.
- [34] Rahman, A., Partho, A., Morrison, P., & Williams, L. (2018). What questions do programmers ask about configuration as code? *Proceedings - International Conference on Software Engineering*, 16–22.
- [35] Guerriero, M., Marconi, F., et al., (2018). Towards DevOps for Privacy-by-Design in Data-Intensive Applications: A Research Roadmap. ICPE '17 Companion: Proceedings of the 8th ACM/SPEC.

- [36] Rimba, P., Zhu, L., Bass, L., Kuz, I., & Reeves, S. (2016). Composing Patterns to Construct Secure Systems. *Proceedings - 2015 11th European Dependable Computing Conference, EDCC 2015*, 213–224.
- [37] Ullah, F., Johannes Raft, A., Shahin, M., Zahedi, M., and Babar, M.A (2017) Security Support in Continuous Deployment Pipeline, *Proceedings of 12th International Conference on Evaluation of Novel Approaches to Software Engineering*
- [38] Paule, C., Dullmann, T. F., & van Hoorn, A. (2019). Vulnerabilities in Continuous Delivery Pipelines? A Case Study. *Proceedings - 2019 IEEE International Conference on Software Architecture - Companion, ICSA-C 2019*, 102–108
- [39] Casola, V., de Benedictis, A., Rak, M., & Villano, U. (2020). A novel Security-by-Design methodology: Modeling and assessing security by SLAs with a quantitative approach. *Journal of Systems and Software*, 163.
- [40] Jaatun, M. G. (2019). Architectural risk analysis in agile development of cloud software. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, 2019-December*, 295–300.
- [41] Rios, E., Iturbe, E., et al., (2019). Service level agreement-based GDPR compliance and security assurance in (multi)Cloud-based systems. *IET Software*, 13(3), 213–222.
- [42] Abrahams, M., Langerman, J., (2018). Compliance at Velocity within a DevOps Environment. 2018 Thirteenth International Conference on Digital Information Management (ICDIM).
- [43] Atighetchi, M., Simidchieva, B., & Olejnik, K. (2019). Security Requirements Analysis - A Vision for an Automated Toolchain. *Proceedings - Companion of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS-C 2019*, 97–104.
- [44] Jaatun, M. G., Cruzes, D. S., & Luna, J. (2017). DevOps for better software security in the cloud. *ACM International Conference Proceeding Series, Part F130521*.
- [45] Brewer, J., Joyce, G., & Dutta, S. (2017). Converging data with design within agile and continuous delivery environments. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10288 LNCS, 533–542.
- [46] Fitzgerald, B., & Stol, K. J. (2014). Continuous software engineering and beyond: Trends and challenges. *1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014 - Proceedings*, 1–9.

- [47] Nassereddine, M. (2020). DevSecOps practices for an agile and secure it service management. In *Journal of Management Information and Decision Sciences* (Vol. 23, Issue 2).
- [48] Larrucea, X., Berreteaga, A., & Santamaria, I. (2019). Dealing with Security in a Real DevOps Environment. *Communications in Computer and Information Science*, 1060, 453–464.
- [49] Vijayakumar, K., & Arun, C. (2019). Continuous security assessment of cloud based applications using distributed hashing algorithm in SDLC. *Cluster Computing*, 22, 10789–10800.
- [50] Díaz, J., Pérez, J. E., Lopez-Peña, M. A., Mena, G. A., & Yagüe, A. (2019). Self-service cybersecurity monitoring as enabler for DevSecops. *IEEE Access*, 7, 100283–100295.
- [51] Hilton, M., Nelson, N., Tunnell, T., Marinov, D., & Dig, D. (2017). Trade-offs in continuous integration: Assurance, security, and flexibility. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Part F130154*, 197–207.
- [52] Gilje Jaatun, M. (2018). *Software Security Activities that Support Incident Management in Secure DevOps*. ARES 2018: Proceedings of the 13th International Conference on Availability, Reliability and Security.
- [53] Morales, A. R. F. J. A. A., Yasar, H., & Volkman, A. (2018). *Implementing DevOps Practices in Highly Regulated Environments* (Vol. 4). ACM.
- [54] Stockhausen, H. M. von, & Rose, M. (2020). Continuous security patch delivery and risk management for medical devices. *Proceedings - 2020 IEEE International Conference on Software Architecture Companion, ICSA-C 2020*, 204–209.
- [55] Zheng, E. (2018). *Building a Virtually Air-gapped Secure Environment in AWS: with principles of devops security program and secure software delivery*. HotSoS'18.
- [56] Lavirotte, S., Metzger, A., et al., (2020). *Development and Operation of Trustworthy Smart IoT Systems: The ENACT Framework*. DEVOPS 2019: Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment pp 121–138.
- [57] Ferry, N., Nguyen, P. H., Song, H., Rios, E., Iturbe, E., Martinez, S., & Rego, A. (2020). Continuous deployment of trustworthy smart IoT systems. *Journal of Object Technology*, 19(2), 1–23.
- [58] Mårtensson, T., Ståhl, D., & Bosch, J. (2016). Continuous integration applied to software-intensive embedded systems – Problems and experiences. *Lecture Notes in Computer Science (Including*

Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10027 LNCS, 448–457.

- [59] Carturan, S. B. O. G., & Goya, D. H. (2019). A systems-of-systems security framework for requirements definition in cloud environment. *ACM International Conference Proceeding Series*, 2, 235–240.
- [60] Pallis, G., Trihinas, D., Tryfonos, A., & Dikaiakos, M. (2018). DevOps as a Service: Pushing the Boundaries of Microservice Adoption. *IEEE Internet Computing*, 22(3), 65–71.
- [61] Weber, I., Nepal, S., & Zhu, L. (2016). Developing Dependable and Secure Cloud Applications. *IEEE Internet Computing*, 20(3), 74–79.
- [62] Rios, E., Iturbe, E., & Palacios, M. C. (2017). Self-healing multi-cloud application modelling. *ACM International Conference Proceeding Series, Part F130521*.
- [63] Thanh, T. Q., Covaci, S., Magedanz, T., Gouvas, P., & Zafeiropoulos, A. (2016). Embedding security and privacy into the development and operation of cloud applications and services. *2016 17th International Telecommunications Network Strategy and Planning Symposium, Networks 2016 - Conference Proceedings*, 31–36.
- [64] Faily, S., & Jacob, C. (2017). Design as code: Facilitating collaboration between usability and security engineers using CAIRIS. *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, 76–82.
- [65] Ahmadvand, M., Pretschner, A., Ball, K., & Eyring, D. (2018). Integrity protection against insiders in microservice-based infrastructures: From threats to a security framework. *Lecture Notes in Computer Science*, 11176 LNCS, 573–588.