Alma Mater Studiorum · Università di Bologna

DIPARTIMENTI DI INFORMATICA - SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria informatica

ADOZIONE DI DEVSECOPS COME PROCESSO DI SVILUPPO SOFTWARE: PASSATO, PRESENTE E FUTURO

Elaborato in
SICUREZZA INFORMATICA

Relatore
Prof. PRANDINI MARCO

Corelatori
Prof. MELIS ANDREA
Dott. Mag. BERARDI DAVIDE

Presentata da
PAGLIA FRANCESCO

Anno Accademico 2020-2021

Indice

Introduzione

Il mercato informatico ha attraversato varie trasformazioni ed evoluzioni negli ultimi anni. La proliferazione delle software house, lo ha reso un settore altamente competitivo. Inoltre, l'avanzamento tecnologico ha incrementato esponenzialmente la complessità delle applicazioni informatiche, con il conseguente invecchiamento precoce dei tradizionali metodi di sviluppo software. I responsabili aziendali delle organizzazioni informatiche, uniti agli sviluppatori e agli operatori IT, necessitavano di un nuovo modello di sviluppo per allinearsi ai moderni standard di velocità e complessità.

In questo clima di metamorfosi, nasce il movimento DevOps. Esso definisce un nuovo modello di sviluppo che rinnova il ciclo di vita del software, snellendolo e velocizzandolo senza però comprometterne la qualità. DevOps rivoluziona persino la cultura aziendale mediante opportune best practice. Queste ultime sono essenziali per le organizzazioni perché consentono loro di superare le sfide e i cambiamenti architetturali, tecnologici e organizzativi imposti dall'adozione di DevOps.

I vantaggi del modello DevOps sono concreti e visibili. Tutte le organizzazioni che hanno integrato correttamente le sue metodologie sono diventate più competitive, veloci ed efficienti. Purtroppo, parallelamente ai cambiamenti nel campo dello sviluppo software, anche il settore del cybercrimine è cresciuto. La sua grandezza è dovuta principalmente alla diffusione di internet, agli utenti finali inesperti ma soprattutto alla scarsa sicurezza delle applicazioni. In un mondo sempre più connesso e digitale, è fondamentale introdurre la cultura della sicurezza informatica all'interno dei processi or-

ganizzativi.

Tradizionalmente, il team della sicurezza inseriva i meccanismi di protezione solo dopo il rilascio di una nuova versione software. Però, con l'adozione di DevOps, la rapidità degli sviluppatori si scontrò fin da subito con la lentezza dei datati processi di difesa. Era necessario rinnovare e integrare le attività del team di sicurezza all'interno del modello DevOps. Da questa esigenza nasce il nuovo modello DevSecOps.

Esso integra le attività di controllo e di protezione nella pipeline DevOps, favorendo la creazione di software sicuro in maniera rapida e affidabile. Naturalmente, l'integrazione comporta una rivoluzione culturale al livello aziendale e lavorativo. Tutte le figure professionali coinvolte diventano responsabili della sicurezza del prodotto software. Di conseguenza, i team devono formarsi sulle tecniche di difesa e sull'uso dei nuovi strumenti di protezione.

Il processo di adozione di DevSecOps deve essere lento e graduale, altrimenti si rischia di creare solo malcontento generale. Le sfide da affrontare sono molteplici ma la loro difficoltà è nulla in confronto ai danni economici e di immagine che può subire un'organizzazione dopo un moderno attacco informatico. Fortunatamente, le tendenze future in campo informatico ridurranno le risorse e l'impegno necessario per aderire al modello DevSecOps. Le piattaforme cloud, l'intelligenza artificiale, il machine learning e l'automazione sono solo alcune delle novità che agevoleranno il lavoro degli sviluppatori, degli operatori IT e degli addetti alla sicurezza. In futuro, il settore dello sviluppo informatico non potrà prescindere dalla sicurezza informatica.

Capitolo 1

DevOps

DevOps è una metodologia di sviluppo software che permette a sviluppatori, operatori IT e addetti alla sicurezza di coordinarsi e collaborare per fornire prodotti migliori e più affidabili. Nel corso degli ultimi anni, DevOps è diventato il punto di riferimento per tutte le organizzazioni nel settore dello sviluppo software. Però, per apprezzare pienamente le novità e i benefici introdotti da questo nuovo modello, è necessario fare un salto nel passato e analizzare i modelli di sviluppo software tradizionali, con i relativi limiti. [1]

1.1 Modelli di sviluppo software tradizionali

Nel mondo dello sviluppo software, tradizionalmente, lo sviluppatore e l'addetto alle operazioni IT lavorano in modo sequenziale. Lo sviluppatore concepisce, sviluppa e rilascia il software. L'addetto alle operazioni IT ha la responsabilità di far funzionare correttamente l'intera applicazione nell'ambiente di produzione. Questa gestione a cascata del ciclo di vita del software, unita alla mancanza di strumenti di automazione, comporta continui ricircoli, disservizi e malcontento a livello organizzativo. [2]

1.1.1 Modello a cascata

Il modello a cascata fu il primo a essere applicato quando lo sviluppo delle applicazioni cominciò a essere concepito come attività industriale.

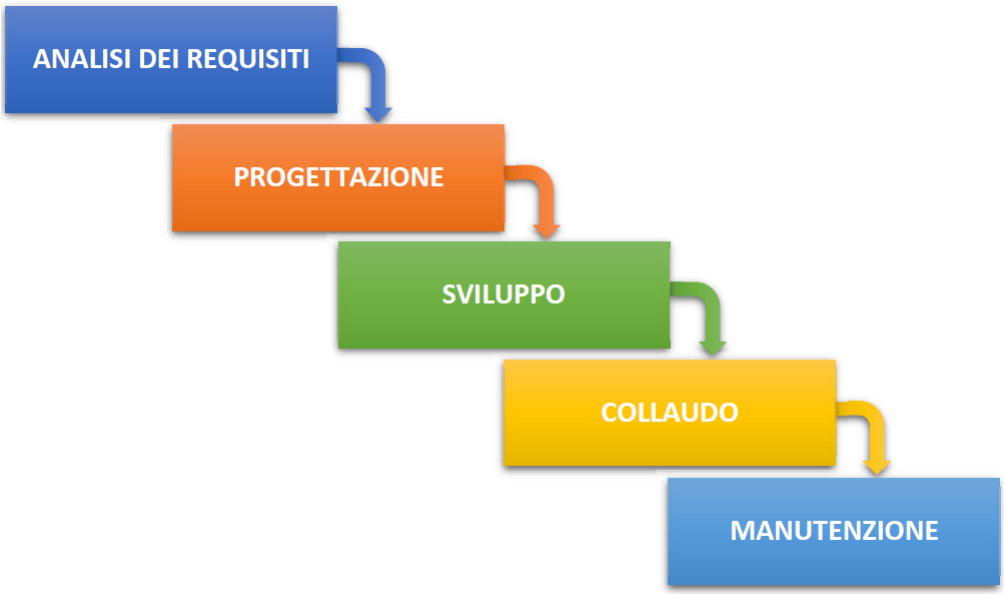

Figura 1.1: Rappresentazione schematica del modello a cascata

Come mostrato nella Figura 1.1, esso struttura il ciclo di vita del software in cinque fasi che si susseguono sequenzialmente:

- **Analisi dei requisiti:** intervista con il cliente per capire quali funzionalità e quali vincoli dovrà avere l'applicazione. Il risultato di questa fase è il documento di specifica dei requisiti software.
- **Progettazione:** partendo dal documento dei requisiti, si definiscono l'architettura logica e fisica dell'applicazione. Il risultato di questa fase è il documento di specifiche di progetto.
- **Sviluppo:** sviluppo del software seguendo lo schema architetturale precedentemente definito.

- **Collaudo:** collaudo del funzionamento e della correttezza dei moduli software sviluppati.
- **Manutenzione:** il software viene consegnato al cliente. Viene effettuata la manutenzione correttiva, adattativa e perfettiva quando ritenuto necessario.

Questo modello si basa sul presupposto che l'introduzione di cambiamenti sostanziali nelle fasi avanzate dello sviluppo ha costi troppo elevati. Pertanto, ogni fase deve essere svolta in maniera esaustiva prima di passare alla successiva, evitando le retroazioni.

Ovviamente, la rigidità di questo modello si scontra con la variabilità dei requisiti software che, in qualsiasi momento, possono subire variazioni. Inoltre, il cliente può ricevere il software solo dopo che l'intero ciclo di sviluppo è stato completato. Se dopo la consegna si scoprissero funzionalità implementate in maniera non corretta o che non soddisfano il cliente, si dovrebbero ripetere tutte le fasi dall'inizio per correggere il software.

In conclusione, il modello a cascata può essere applicato con successo solo nei casi in cui le specifiche sono ben definite e non variano nel tempo. In tutti gli altri casi, si rischia di rilasciare un prodotto software con significativi ritardi, costi elevati e scarsa affidabilità.

1.1.2 Modello agile

L'aumento di complessità nel mercato informatico, rese antiquato il modello a cascata e favorì la nascita dei modelli agili. Questi ultimi proponevano un approccio meno strutturato e più focalizzato sull'interazione con il cliente, per consegnare in tempi brevi software funzionante e di qualità.

Nel modello agile si parte da specifiche molto astratte e si sviluppa un primo prototipo da sottoporre al cliente e da perfezionare successivamente. Applicando le metodologie agili, il prototipo iniziale evolve gradualmente verso il prodotto finito attraverso un certo numero di sprint. Al termine di ogni sprint, al cliente vengono mostrate le modifiche che il prototipo ha subito.

Così facendo, il cliente può direttamente rendersi conto dell'avanzamento del progetto e può verificare l'effettiva implementazione dei requisiti richiesti. Ogni sprint comprende quattro fasi (Figura 1.2):

- **Plan:** decisione, insieme al cliente, di quali funzionalità implementare durante lo sprint.
- **Build:** implementazione delle funzionalità stabilite, avvalendosi della prototipazione in caso di specifiche poco chiare.
- **Test:** verifica che ciò che è stato implementato sia privo di errori, spesso facendo uso di test automatici.
- **Review:** il cliente fornisce il suo parere sul lavoro svolto, indicando eventuali miglioramenti implementabili nel prossimo sprint.

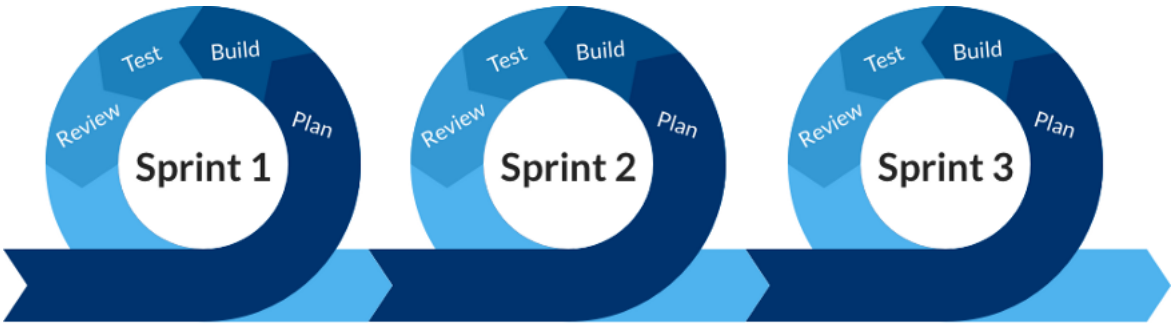

Figura 1.2: Rappresentazione schematica del modello agile [3]

Nonostante l'impiego delle metodologie agili rappresenti un sostanziale miglioramento rispetto alla rigidità del modello a cascata, alcune problematiche sono ancora presenti. Se il prodotto non viene testato nell'ambiente in cui si troverà in produzione ma solo sulle macchine degli sviluppatori, è molto probabile che dopo il deployment si riscontrino degli errori. Questo problema nasce dal fatto che il team degli sviluppatori e quello delle operazioni IT lavorano senza nessun tipo di collaborazione. Infatti, una volta terminato lo sprint, il software viene semplicemente messo nelle mani dei sistemisti ed

è compito loro trovare il modo di farlo funzionare stabilmente nell'ambiente di destinazione. Naturalmente questo compito non è semplice: l'ambiente di produzione potrebbe avere diverse configurazioni, librerie mancanti o software di terze parti non installato. Il team di operatori si trova quindi a far fronte a una grossa sfida: risolvere problemi senza avere un'adeguata conoscenza del codice.

Per agevolare la messa in produzione del software, risolvendo tutti i problemi sopraelencati, è necessaria una stretta cooperazione tra il mondo degli sviluppatori e quello degli operatori. Questi presupposti gettano le basi per la cultura DevOps e il suo modello di sviluppo.

1.2 Modello DevOps

Concepito a partire dal modello agile, DevOps è una metodologia di sviluppo software che punta a favorire la comunicazione, collaborazione e integrazione tra sviluppatori e operatori IT. Non a caso, il termine DevOps nasce dalla combinazione delle parole Development (sviluppo) e Operations (operazioni).
Come mostrato nella Figura 1.3, sono presenti otto fasi che si susseguono ciclicamente.

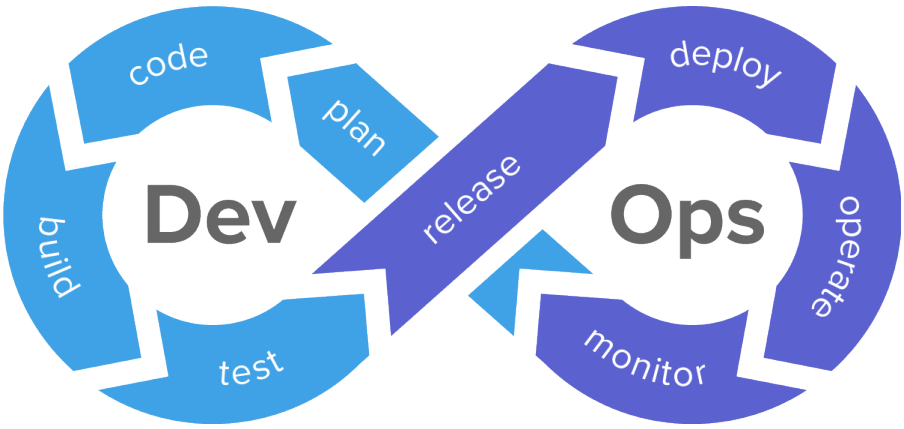

Figura 1.3: Rappresentazione schematica del modello DevOps [4]

Le prime quattro sono associate al team di sviluppatori (Dev); le restanti a quello delle operazioni IT (Ops):

- **Plan:** pianificazione delle prossime funzionalità da implementare.
- **Code:** sviluppo delle nuove funzionalità facendo uso di strumenti di controllo della versione.
- **Build:** compilazione dei singoli componenti sviluppati o modificati.
- **Test:** test dei vari componenti utilizzando strumenti automatici e non.
- **Release:** integrazione dei componenti sviluppati dai singoli team.
- **Deploy:** installazione dell'intero sistema nell'ambiente di produzione.
- **Operate:** configurazione dell'ambiente di produzione per fare in modo che il prodotto funzioni correttamente.
- **Monitor:** supervisione dell'ambiente di produzione per verificare l'impatto sull'utente finale e la presenza di errori.

Per sfruttare al meglio il modello DevOps non basta che i due team (Dev e Ops) lavorino a stretto contatto, ma è necessario anche utilizzare gli strumenti giusti nelle diverse fasi:

- **Monitor - Plan:** strumenti di monitoraggio del software in produzione e gestione di eventuali feedback e problematiche.
- **Code:** strumenti di gestione del codice che tengono traccia delle modifiche realizzate nel tempo e delle diverse versioni, richiamandole qualora necessario, e permettendo agli sviluppatori di lavorare in modo concorrenziale e parallelo.
- **Build - Test:** strumenti per automatizzare le fasi di integrazione del codice e di testing funzionale e prestazionale, ogni qualvolta si ha una nuova modifica software.

- **Release - Deploy:** strumenti che eseguono autonomamente procedure standardizzate di configurazione delle infrastrutture e distribuzione del software.
- **Operate:** strumenti che agevolano la gestione della manutenzione del software e dell'hardware, evitando la creazione di disservizi per l'utente finale.

Grazie all'impiego delle metodologie definite da DevOps è possibile ottenere i seguenti vantaggi:

- **Velocità:** processi più agili per adattarsi meglio ai cambiamenti dettati dal mercato e dai clienti.
- **Distribuzione rapida:** aumenta la frequenza delle nuove release, velocizzando il rilascio di nuove funzionalità e la correzione dei bug.
- **Affidabilità:** verifica che gli aggiornamenti delle applicazioni e le modifiche dell'infrastruttura siano sempre conformi agli standard di qualità, per garantire affidabilità e produttività senza sacrificare l'esperienza degli utenti finali.
- **Scalabilità:** l'introduzione dell'automazione rende possibile gestire la scalabilità di sistemi complessi, riducendo i rischi connessi.
- **Sicurezza:** possibilità di utilizzare policy di conformità automatizzate, controlli granulari e tecniche di analisi delle configurazioni per monitorare costantemente la sicurezza.

Tutti i vantaggi sopraelencati non soltanto semplificano il lavoro di sviluppatori e operatori IT, ma sono persino a beneficio del cliente. Difatti, la frequenza maggiore di release rende più semplice ottenere un feedback continuo da parte del cliente. Mediante quest'ultimo si riescono a correggere più semplicemente gli errori individuati, si ottengono maggiori informazioni sulle funzionalità da implementare e si riesce a sviluppare un'applicazione che

rispecchia le effettive volontà del suo richiedente. I miglioramenti a livello di collaborazione e produttività sono essenziali anche per raggiungere obiettivi aziendali come: accelerazione del time-to-market, adattamento al mercato e alla competizione, conservazione della stabilità e dell'affidabilità del sistema, miglioramento del tempo medio per il ripristino. [5, 6, 7]

1.3 Pratiche della cultura DevOps

La transizione verso l'approccio DevOps richiede un'evoluzione di mentalità e cultura aziendali. Per agevolare l'adozione di questo modello, è indispensabile seguire le seguenti best practice:

- Integrazione continua.
- Distribuzione continua.
- Microservizi.
- Infrastruttura come codice.
- Monitoraggio continuo.
- Comunicazione e collaborazione.

Il valore di business di un'organizzazione, che adotta le pratiche DevOps, aumenta agli occhi dei clienti. Questo valore può esprimersi sotto forma di release, funzionalità o aggiornamenti dei prodotti più frequenti. Può riguardare la rapidità con cui la release di un prodotto o di una nuova funzionalità diventa accessibile ai clienti, senza tuttavia comprometterne la qualità e la sicurezza. Oppure, può concentrarsi sulla rapidità con cui viene identificato e risolto un problema o un bug prima di una nuova release.
L'infrastruttura organizzativa deve supportare le pratiche DevOps adoperando dei software che ottimizzino le performance, la disponibilità e l'affidabilità dei processi di sviluppo e test, prima che il software venga inviato all'ambiente di produzione. [5, 6, 7]

1.3.1 Integrazione continua

L'integrazione continua è un metodo di sviluppo software DevOps in cui gli sviluppatori aggiungono regolarmente modifiche al codice salvato in un repository centralizzato, eseguendo automaticamente anche la creazione della build e i relativi test. Le frequenti modifiche al codice (numero di commit elevato) rendono necessaria l'adozione di un sistema di controllo della versione come Git.

In passato, gli sviluppatori di un team lavoravano separatamente per un lungo periodo di tempo e integravano tutte le modifiche soltanto una volta completate. Purtroppo, questa pratica rendeva difficoltosa e dispendiosa l'unione di diversi pezzi di codice, con il conseguente accumulo di bug. L'insieme di questi fattori rendeva molto complicato offrire aggiornamenti rapidi e affidabili ai propri clienti. Fortunatamente, l'integrazione continua risolve questi problemi e garantisce i seguenti vantaggi:

- **Maggiore produttività per gli sviluppatori:** il team di sviluppo migliora la propria produttività sostituendo le attività manuali con attività automatizzate, che favoriscono la riduzione del numero di errori e bug nel software distribuito ai clienti.
- **Bug individuati e risolti con maggiore prontezza:** aumentando la frequenza del testing, è più facile individuare e risolvere tempestivamente i bug prima che diventino problemi gravi.
- **Aggiornamenti più rapidi:** l'integrazione continua consente il rilascio di aggiornamenti più rapidamente e più frequentemente.

1.3.2 Distribuzione continua

Fondamento dello sviluppo moderno di applicazioni ed estensione dell'integrazione continua; la distribuzione continua distribuisce, dopo la fase di creazione della build, tutte le modifiche del codice all'ambiente di testing e/o di produzione. Se implementata correttamente, gli sviluppatori avranno

sempre a disposizione una build pronta per la distribuzione. Ogni modifica apportata deve essere sottoposta a una serie di test automatici che verificano il funzionamento del software su più livelli, prima di distribuirlo ai clienti. Queste prove possono includere test dell'interfaccia, test di caricamento, test di integrazione, test di affidabilità delle API e così via. In questo modo è più semplice per gli sviluppatori analizzare gli aggiornamenti più approfonditamente e rilevare preventivamente eventuali problemi.

Ai vantaggi dati dall'integrazione continua, si aggiunge il vantaggio di avere un processo di rilascio del software automatizzato. Una volta ogni aggiornamento doveva essere approvato manualmente; oggi l'automazione rende più efficiente e veloce testare e inoltrare in produzione una nuova build.

1.3.3 Pipeline CI/CD

La pipeline CI/CD (Continuous Integration/Continuous Delivery) consiste in una serie di passaggi che devono essere eseguiti per fornire una nuova versione software. Basata sull'integrazione continua e sulla distribuzione continua, la pipeline CI/CD è pensata appositamente per ottimizzare l'erogazione di software attraverso l'approccio DevOps. Infatti, le fasi e le attività della pipeline sono le stesse già viste nel modello DevOps.

L'ottimizzazione è possibile grazie all'introduzione del monitoraggio e dell'automazione nelle fasi di integrazione, test, distribuzione e deployment. L'automazione riduce il lavoro manuale dei team e i tempi di rilascio. Il monitoraggio controlla l'affidabilità e la sicurezza dei processi aziendali. La loro unione fornisce delle fondamenta solide per costruire e supportare il modello DevOps. [8]

1.3.4 Microservizi

Utilizzando un'architettura basata su microservizi, è possibile realizzare un'applicazione mediante l'unione di diversi componenti indipendenti: i microservizi. Ogni microservizio è:

- **Autonomo:** ciascun servizio può essere sviluppato, distribuito, eseguito e ridimensionato senza influenzare il funzionamento degli altri componenti. I servizi non devono condividere alcun codice o implementazione con gli altri. Qualsiasi comunicazione tra i componenti individuali avviene attraverso API ben definite.
- **Specializzato:** ciascun servizio è progettato per fornire una serie di funzionalità e si concentra sulla risoluzione di un problema specifico. Se gli sviluppatori aggiungono del codice a un servizio rendendolo più complesso, il servizio può anche essere scomposto in servizi più piccoli.

L'impiego di un'architettura basata sui microservizi apporta i seguenti vantaggi: scalabilità e flessibilità nello sviluppo, semplicità di distribuzione, libertà tecnologica ed elevata riusabilità del codice.

1.3.5 Infrastruttura come codice

L'infrastruttura come codice definisce le risorse e le topologie del sistema in maniera descrittiva, consentendo ai team di gestire tali risorse con lo stesso approccio usato per il codice. Queste definizioni vengono spesso archiviate e sottoposte a sistemi di controllo della versione, all'interno dei quali è possibile effettuare operazioni di revisione e ripristino. L'infrastruttura come codice contribuisce anche all'automazione del codice e riduce il rischio di errore umano, in particolare negli ambienti complessi di grandi dimensioni. Questa soluzione aiuta i team a distribuire le risorse di sistema in modo affidabile, ripetibile e controllato; garantendo loro il mantenimento di ambienti di sviluppo e test identici a quelli di produzione. Anche la duplicazione degli ambienti in data center diversi e piattaforme cloud diverse risulta più semplice e più efficiente.

1.3.6 Monitoraggio continuo

Per monitoraggio continuo si intende l'analisi completa in tempo reale delle prestazioni e dell'integrità dell'intero stack applicativo; partendo dal-

l'infrastruttura che esegue l'applicazione fino ai componenti software di livello superiore. Il controllo attivo è di cruciale importanza soprattutto per tutti quei servizi critici, la cui disponibilità non deve presentare interruzioni. Il monitoraggio continuo torna utile anche quando si deve stimare l'impatto che nuove modifiche e aggiornamenti software hanno avuto sugli utenti finali.

Il monitoraggio parte dalla raccolta di dati, metadati e log generati dal software e dall'infrastruttura. Le informazioni raccolte vengono categorizzate e analizzate per aiutare i team ad attenuare i problemi in tempo reale e scoprire come migliorare l'applicazione nei cicli di sviluppo futuri. È possibile configurare persino degli allarmi che si attivano quando nel sistema succedono degli eventi che necessitano l'attenzione di un operatore specializzato.

1.3.7 Comunicazione e collaborazione

Il modello DevOps punta alla rimozione delle barriere tra due team che, in genere, non comunicano tra loro: sviluppatori e operatori IT. Essi collaborano a stretto contatto per ottimizzare sia la produttività dello sviluppo sia l'affidabilità delle operazioni. Anche i team dedicati al controllo della qualità e sicurezza spesso sono coinvolti più direttamente nei compiti di questi team. In alcune realtà aziendali, è possibile arrivare persino a una fusione delle due figure professionali. Indubbiamente, nell'approccio DevOps qualsiasi forma di comunicazione e collaborazione è effettuata per soddisfare al meglio le esigenze del cliente finale.

Capitolo 2

Sviluppare software sicuro

Per sfruttare tutta l'agilità e la reattività dell'approccio DevOps, occorre tener conto anche di un altro elemento cruciale all'interno del ciclo di vita delle applicazioni: la sicurezza informatica.

2.1 Importanza della sicurezza informatica

Oggigiorno la maggior parte delle attività quotidiane sono svolte digitalmente. Di conseguenza, ci sono moltissime applicazioni che raccolgono i nostri dati sensibili: dati personali, finanziari e sanitari.

La raccolta di queste informazioni non interessa solamente le organizzazioni, ma anche tutti i cybercriminali che sanno di poterle rivendere sul dark web a cifre molto elevate. Basti pensare che il recente ransomware Sodinokibi, secondo uno studio condotto da IBM, è riuscito a esfiltrare 21.6 terabyte di dati per un guadagno complessivo di 123 milioni di dollari solo nel 2020.

Naturalmente, è importante che le organizzazioni aumentino la loro sensibilità verso la protezione e la sicurezza informatica tanto quanto gli utenti finali. Molto spesso gli attacchi informatici hanno successo proprio grazie a utenti inesperti che non prendono i giusti accorgimenti per proteggersi. Una pessima gestione della sicurezza informatica può essere fatale persino per le

organizzazioni: perdita di fiducia da parte dei clienti, ingenti danni economici e persino fallimento.

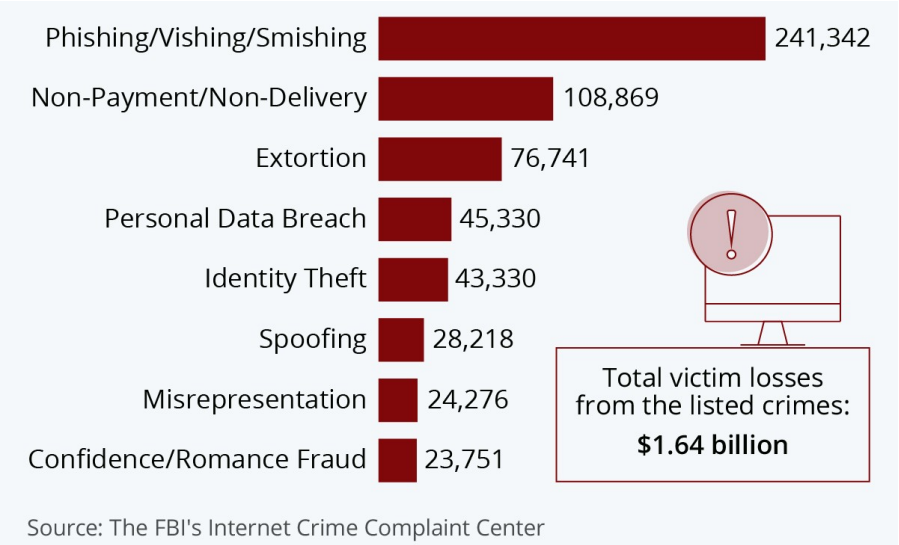

Figura 2.1: Grafico che mostra il numero di americani che sono stati vittime di attacchi informatici nel 2020 [9]

Esistono dieci regole che ognuno dovrebbe seguire per proteggersi nel mondo digitale:

1. Effettuare giornalmente i backup dei dati su dispositivi esterni come hard disk, flash drive e altri.

2. Utilizzare password robuste di almeno otto caratteri e con caratteri speciali, soprattutto quando si condividono i dati con l'esterno.

3. Installare sistemi di antivirus sia sui computer sia sui dispositivi mobili, avendo cura di aggiornarli continuamente.

4. Fare attenzione agli allegati contenuti in email ingannevoli. Molto probabilmente si tratta di phishing.

5. Non condividere informazioni sensibili con terzi non autorizzati. Questi dati potrebbero essere usati per effettuare attacchi di ingegneria sociale.

6. Usare sempre connessioni sicure (VPN) tra i computer personali e il server contenente i dati sensibili.

7. Non usare strumenti pubblici di condivisione dati ma cluod privati, proteggendo i dati con password robuste.

8. Utilizzare strumenti di crittografia simmetrica o asimmetrica quando avviene lo scambio di dati sensibili.

9. Implementare e seguire le procedure di sicurezza, in termini di software da utilizzare e azioni da intraprendere in caso di data breach (violazione dei dati personali).

10. Tracciare i propri accessi, attivando meccanismi di notifica ogni volta che si accede a un sito o a una applicazione.

In conclusione, non si può vivere in un mondo digitale facendo finta che non esistano i problemi legati alla sicurezza. Bisogna prenderne atto e assumere un atteggiamento di cautela e attenzione, informandosi sui possibili rischi e le relative contromisure. [10]

2.2 Cos'è la sicurezza informatica

La sicurezza informatica è l'insieme dei mezzi, delle tecnologie e delle procedure utilizzate per proteggere la disponibilità, la riservatezza e l'integrità dei beni gestiti mediante sistemi informatici.

La disponibilità è l'attitudine di un sistema a essere in grado di svolgere una funzione richiesta a un dato instante in determinate condizioni. La riservatezza è la proprietà delle informazioni di non essere rese disponibili o divulgate a individui, entità o processi non autorizzati. L'integrità è la protezione delle informazioni da modifiche accidentali o volontarie fatte da soggetti non autorizzati. Oltre alle tre fondamentali proprietà, ne esistono altre: autenticità, non ripudiabilità, responsabilità, affidabilità.

Garantire la sicurezza di sistemi informatici complessi non è affatto semplice. Per farlo correttamente è necessario il team della sicurezza, il quale comprende sia esperti in ambito tecnico-informatico sia esperti in campo giuridico. Il loro compito è individuare le minacce, le vulnerabilità e i rischi associati agli asset informatici, al fine di proteggerli da possibili attacchi che potrebbero provocare danni diretti o indiretti di impatto superiore a una determinata soglia di tollerabilità (economico, politico-sociale, di reputazione) per l'organizzazione. Inoltre, la sicurezza informatica può assumere diverse sfaccettature, dipendentemente dal contesto in cui deve essere applicata: sicurezza dei programmi, sicurezza dei sistemi informatici e sicurezza nelle reti.

Nei contesti in cui la sicurezza gioca un ruolo di fondamentale importanza (militare e governativo), il team della sicurezza può anche essere suddiviso in Red team e Blue team. Il Red team ricopre il ruolo di attaccante e il suo obiettivo è quello di analizzare il sistema, trovare delle vulnerabilità e sfruttarle per veicolare gli attacchi. Il Blue team funge da difensore e il suo scopo è quello di amministrare il sistema per rilevare, rispondere e mitigare gli attacchi del Red team. Grazie alle attività di penetration test, è possibile effettuare una valutazione delle capacità difensive dell'organizzazione e migliorare i suoi sistemi.

Una sottoclasse della sicurezza informatica piuttosto rilevante è la cybersecurity. Essa si focalizza sulle qualità di resilienza, robustezza e reattività che una tecnologia deve possedere per fronteggiare attacchi cyber mirati a comprometterne il suo corretto funzionamento e le sue performance. [11]

2.3 Sviluppo di software sicuro

In passato, lo sviluppo software era maggiormente focalizzato sul creare applicazioni che dovevano essere veloci e facili da utilizzare. La sicurezza dei programmi ricopriva un ruolo secondario e, generalmente, veniva introdotta a software ultimato. Con l'aumentare della complessità delle applicazioni

divenne sempre più dispendioso introdurre dei meccanismi di protezione a posteriori.

Oggigiorno, viene applicato il principio di security by design: un software sicuro deve essere progettato fin dall'inizio per essere tale. Le funzionalità di sicurezza devono essere parte integrante del software e non delle parti accessorie. Per perseguire questo obiettivo, è importante indurre gli sviluppatori ad adottare pratiche di sviluppo affidabili e sicure come la defensive programming e il secure coding. [12]

2.3.1 Defensive programming e secure coding

La defensive programming è una forma di progettazione difensiva usata per garantire il funzionamento continuo di un componente software in circostanze impreviste. Le pratiche di defensive programming vengono spesso utilizzate dove è necessaria alta disponibilità, sicurezza e protezione. Esse lavorano sul codice sorgente per garantire miglioramenti di:

- **Qualità generale:** riduzione del numero di bug e problemi nel software.
- **Comprensione del codice sorgente:** codice più leggibile e comprensibile per individuare facilmente errori.
- **Previsione del comportamento:** fare in modo che il software si comporti in modo prevedibile, nonostante gli input o le azioni impreviste dell'utente.

La defensive programming si divide in due categorie: programmazione sicura e programmazione offensiva. La prima è strettamente connessa con la sicurezza informatica e ha come obiettivo la riduzione della superficie di attacco, con la conseguente diminuzione del numero di bug che potrebbero essere sfruttati in modo dannoso. La seconda è focalizzata sull'utilizzo di controlli che intervengono o bloccano l'utente quando effettua azioni potenzialmente dannose.

Il secure coding è la pratica di sviluppare software proteggendolo dall'introduzione accidentale di vulnerabilità di sicurezza. Disattenzioni, bug e difetti logici sono la causa principale di questi problemi che vengono abitualmente sfruttati per attaccare un programma. Attraverso l'analisi di migliaia di vulnerabilità segnalate, i professionisti della sicurezza hanno scoperto che la maggior parte dei punti di attacco deriva da un numero relativamente piccolo di errori di programmazione. Identificando le pratiche di codifica sbagliate ed educando gli sviluppatori a usare alternative più sicure, le organizzazioni possono adottare misure proattive per aiutare a ridurre o eliminare in modo significativo le vulnerabilità nel software prima della sua distribuzione.
Le vulnerabilità più comuni sono:

- **Injection:** si verifica quando dei dati malevoli vengono inviati a un interprete come parte di un comando o di una query. L'attaccante induce l'interprete a eseguire comandi involontariamente e ad accedere ai dati sensibili senza un'adeguata autorizzazione. Ad esempio, sfruttando le SQL injection è possibile iniettare dei comandi in grado di manipolare interamente i dati e la struttura di un database.

- **Broken authentication:** spesso le funzioni relative all'autenticazione e alla gestione delle sessioni sono implementate in modo errato, consentendo agli aggressori di compromettere password, chiavi o token di sessione. Sfruttando questa vulnerabilità, gli attaccanti riescono ad assumere l'identità di altri utenti temporaneamente o permanentemente.

- **Sensitive data exposure:** i dati sensibili degli utenti non sono protetti adeguatamente: informazioni identificative personali, dati sanitari, dati finanziari e credenziali di accesso a servizi. Vulnerabilità di questo tipo nascono quando un dato non è riconosciuto come sensibile, non si individuano tutte le copie da proteggere e non si utilizzano metodi di difesa sufficientemente robusti sia durante le trasmissioni sia per i salvataggi locali. Grazie all'esposizione dei dati sensibili, gli ag-

gressori commettono furti di identità, frodi con carte di credito altrui, ingegneria sociale e tanto altro.

- **Broken access control:** si verifica quando non si definiscono o applicano correttamente le restrizioni utente sull'accesso alle risorse disponibili. Gli attaccanti usano queste lacune per accedere agli account di altri utenti, visualizzare file riservati, modificare i dati altrui e manipolare i diritti di accesso. Per evitare tale vulnerabilità, è necessario implementare delle politiche di accesso default deny e dei solidi meccanismi di controllo.

- **Security misconfiguration:** l'errata configurazione della sicurezza è comunemente il risultato di configurazioni predefinite non sicure, configurazioni incomplete o ad hoc, intestazioni HTTP mal formate e messaggi di errore contenenti informazioni riservate. Oltre a essere configurati in modo sicuro, le applicazioni e i sistemi devono essere sempre aggiornati all'ultima versione software disponibile.

- **Insecure deserialization:** lo scambio di informazioni tra i vari microservizi di un'applicazione può avvenire anche utilizzando la serializzazione. Il trasferimento di un oggetto serializzato in chiaro e la mancanza di controlli di integrità nel deserializzatore sono alla base di una deserializzazione insicura. Avvalendosi di questa vulnerabilità, l'attaccante è in grado di eseguire codice malevolo per effettuare injection e privilege escalation.

- **Known vulnerabilities:** i componenti software (librerie, framework, moduli vari) sono sempre eseguiti con gli stessi privilegi dell'applicazione. Se un componente vulnerabile viene sfruttato come vettore di attacco, il software e i suoi dati potrebbero cadere completamente nelle mani degli attaccanti. Prima di includere componenti esterni, è buona norma controllare se presentano delle vulnerabilità conosciute non ancora risolte.

- **Insufficient logging e monitoring:** senza un'adeguata tracciabilità degli accessi falliti e delle transazioni terminate con errori e con una scarsa protezione e dettaglio dei log, il lavoro dell'attaccante è altamente facilitato. L'archiviazione corretta dei log e il monitoraggio sono fondamentali per rilevare e bloccare tentativi di forza bruta, riuscire a ricostruire la dinamica di un attacco per eliminare la vulnerabilità utilizzata e riscontrare i danni subiti per riparali.

Non lavorare secondo i principi del secure coding può portare alla realizzazione di un software scadente e facilmente attaccabile. Un software robusto è vantaggioso anche per l'organizzazione che eviterà di subire danni economici, blocchi di attività e perdita di dati a seguito di attacchi informatici. [12]

2.3.2 Linee guida per lo sviluppo di software sicuro

Le problematiche di sicurezza del software possono essere introdotte in qualsiasi fase del ciclo di sviluppo, ad esempio:

- Non identificando in anticipo i requisiti di sicurezza.
- Progettando il software senza sufficienti difese.
- Creando progetti concettuali con errori logici.
- Integrando codice da fonti non attendibili.
- Testando superficialmente i nuovi componenti software.
- Utilizzando standard obsoleti, che introducono vulnerabilità tecniche.
- Distribuendo un software in modo errato.
- Inserendo errori durante la manutenzione o l'aggiornamento.
- Scrivendo codice difficilmente leggibile.

Esistono delle vere e proprie linee guida essenziali per sviluppare software sicuro. Ogni fase contiene diverse regole:

1. **Design**

    - **Politiche di sicurezza:** individuare le politiche di sicurezza da adottare e lavorare al loro design e alla loro architettura. Le politiche sono importanti tanto quanto i requisiti software. Infatti, individuarle fin da subito permette di implementare un sistema software che non avrà la necessità di subire rattoppi per integrarle a posteriori, risparmiando tempo e risorse.
    - **Requisiti di sicurezza:** identificare e documentare i requisiti di sicurezza nelle prime fasi del ciclo di sviluppo e assicurarsi che gli elementi sviluppati, successivamente, siano conformi a tali requisiti. Quando i requisiti di sicurezza non sono definiti, la sicurezza del sistema risultante non può essere valutata in modo efficace.
    - **Difesa a cipolla:** gestire il rischio con più strategie difensive affinché, se uno strato di difesa risulta inadeguato, un altro livello di difesa possa impedire che un difetto di sicurezza diventi una vulnerabilità sfruttabile. Ad esempio, la combinazione di tecniche di programmazione sicura abbinate ad ambienti di esecuzione sicuri può ridurre la possibilità che le vulnerabilità rimanenti nel codice possano essere sfruttate.

2. **Architettura**

    - **Privilegi minimi:** ogni processo deve essere eseguito con il minimo insieme di autorizzazioni necessarie. Accedere a un privilegio elevato deve essere possibile, per il tempo strettamente necessario, solo se indispensabile. Questo approccio riduce la possibilità che un utente o processo attaccante possa eseguire codice arbitrario altamente dannoso.

- **Negazione di default:** concedere permessi di accesso in base a condizioni e non a utenze. Ciò significa che, per impostazione predefinita, l'accesso è sempre negato. Solo nel caso in cui vengono rispettate tutte le condizioni di specifici controlli, si ottiene il permesso per accedere a una determinata risorsa.
- **Semplice e leggero:** rendere il software il più semplice e leggero possibile: gli elementi complessi aumentano la probabilità di errori durante l'implementazione, configurazione, utilizzo e manutenzione. Lo sforzo richiesto per raggiungere un livello adeguato di sicurezza aumenta notevolmente con la complessità dei meccanismi di sicurezza.
- **Standard sicuri:** adottare gli standard di codifica disponibili più sicuri, ponderando anche in base al contesto in cui deve operare.

3. **Programmazione**

- **Convalida dell'input:** è importantissimo convalidare tutti gli input con origine non attendibile come: fonti di dati esterne, argomenti della riga di comando, interfacce di rete, variabili d'ambiente, file e input dell'utente. La corretta convalida dell'input può eliminare la maggior parte delle vulnerabilità del software. L'input non accettabile può essere respinto o 'sanificato'.
- **Sanificazione dei dati:** tutti i dati usati devono essere controllati e ripuliti, sostituendo o eliminando i caratteri del testo che possono essere pericolosi. Un attaccante potrebbe sfruttare una mancata sanificazione per effettuare operazioni malevole attraverso l'uso di attacchi di injection.
- **Strumenti di analisi:** durante la scrittura del codice, lo sviluppatore sarà assistito da avvisi generati in automatico dal compilatore; molto importanti per la correzione degli errori di programmazione. Generalmente, è di notevole aiuto utilizzare ulteriori

strumenti di analisi statici e dinamici per rilevare ed eliminare le problematiche di sicurezza più comuni.

4. **Test**

   - **Garanzia di qualità:** i test per garantire la qualità del prodotto software necessitano di tecniche efficaci per individuare ed eliminare le vulnerabilità. I fuzz test, penetration test e i controlli del codice sorgente dovrebbero essere incorporati all'interno di un efficace programma di verifica e validazione.
   - **Strategie di mitigazione:** utilizzare la modellazione delle minacce al fine di gestire le possibili minacce. Questo processo comporta l'identificazione e la categorizzazione di tutto ciò che può essere minacciato ed essere vettore di attacchi di diverso genere. Per ciascun asset e componente devono essere individuate le possibili minacce, classificandole per rischio. L'attività di analisi deve portare allo sviluppo di strategie di mitigazione del rischio, all'implementazione e verifica delle contromisure.

La responsabilità di creare software sicuro e affidabile ricade soprattutto sul team della sicurezza, il quale ha il compito di aiutare gli altri team a rispettare le linee guida soprastanti. [12]

Capitolo 3

DevSecOps

In passato, quando i cicli di sviluppo duravano mesi o anni, la sicurezza non aveva la stessa importanza che ha oggi. Alcuni meccanismi di protezione venivano inseriti da un team specializzato solamente nelle fasi finali dello sviluppo. Persino nella metodologia DevOps la sicurezza ricopriva ancora un ruolo secondario.

Negli ultimi anni, la necessità di avere applicazioni e sistemi sempre più sicuri è cresciuta talmente tanto da dar vita al nuovo modello DevSecOps. La sicurezza deve diventare una responsabilità di tutti, a partire dagli sviluppatori fino ad arrivare agli operatori IT. [13]

3.1 Sicurezza in DevOps

La sicurezza informatica è di fondamentale importanza anche per le applicazioni sviluppate utilizzando il modello DevOps. Quest'ultimo è stato concepito per sviluppare, testare e rilasciare software in maniera molto efficiente, veloce e il più possibile automatizzata; senza focalizzarsi troppo sulla sicurezza del prodotto ottenuto.

Quest'ultimo aspetto è lasciato al team della sicurezza che lavora sulle nuove versioni software appena rilasciate per renderle sicure. Il loro operato, quindi, inizia solo alla fine del flusso di attività DevOps. Il loro compito

è quello di controllare la presenza di vulnerabilità in librerie di terze parti, problemi di licenza, esposizione di dati sensibili, errori di configurazione, vulnerabilità nell'ambiente di test e molto altro. Una volta completata la fase di analisi, stilano un report contente tutte le vulnerabilità, ordinate in base alla criticità, e i relativi accorgimenti per eliminarle. Il report viene poi passato agli sviluppatori che si preoccuperanno di rilasciare nuove versioni del software, correggendo i problemi segnalati.

Questo processo sembra funzionare in maniera idilliaca, ma la realtà è ben diversa. I software odierni sono complessi ed estesi perché spesso basati su un'architettura a microservizi. Ogni microservizio è eseguito in un apposito container e può comunicare con altri microservizi. Rendere sicuro un software così strutturato non è affatto semplice perché la superficie d'attacco è veramente vasta. Sfortunatamente, il team della sicurezza impiega giorni o settimane per trovare tutte le vulnerabilità presenti in una specifica versione dell'applicazione. In questo lungo lasso di tempo, gli sviluppatori rilasciano nuove versioni del software senza attendere la fine dell'operato del team della sicurezza. Questo porta ad avere nuove versioni che potenzialmente hanno le stesse vulnerabilità delle precedenti e porta a sovraccaricare di lavoro gli addetti alla sicurezza che non riescono a star dietro a tutte le nuove release.

La mancanza di una stretta collaborazione tra team della sicurezza, team degli sviluppatori e team delle operazioni IT è dannosa sia per il prodotto finale sia per l'organizzazione. È sicuramente necessaria una sincronizzazione tra il lavoro di questi gruppi.

Una prima soluzione sarebbe quella di adattare i tempi del modello DevOps a quelli del team della sicurezza. Ogni volta che viene rilasciata una nuova versione dell'applicazione, il team della sicurezza lavora per renderla sicura. Gli sviluppatori non possono rilasciare nuove versioni del software fino a quando non ricevono il report sulla sicurezza e correggono le vulnerabilità presenti. Così facendo, potremmo semplicemente risolvere tutti i problemi di collaborazione tra sviluppo e sicurezza. Sciaguratamente, questo espediente non è effettivamente applicabile perché crea un collo di bottiglia che rallenta

significativamente il processo di sviluppo DevOps. Il suo vantaggio è proprio quello di essere ottimizzato per rispondere tempestivamente alle richieste del mercato e dei clienti. I tempi del team di sicurezza sono troppo lenti per essere applicabili al modello DevOps.

La seconda soluzione è rendere la sicurezza parte integrante del ciclo di sviluppo DevOps. Il team della sicurezza non sarà più isolato ma sarà completamente integrato tra gli sviluppatori e gli operatori IT. La responsabilità di realizzare un software sicuro sarà nelle mani di tutte le figure professionali coinvolte nel ciclo di vita del software. Naturalmente il team della sicurezza ricoprirà sempre il ruolo di guida nel suo ambito. Grazie al connubio tra la Security e il Development-Operations nasce il modello DevSecOps, che mantiene l'immediatezza del modello DevOps e garantisce un adeguato livello di sicurezza per il software rilasciato. [14]

3.2 Modello DevSecOps

Il modello DevSecOps fonde la sicurezza informatica all'interno del flusso di lavoro DevOps. Infatti, come mostrato nella Figura 3.1, DevSecOps nasce dall'unione dei termini Development (sviluppo), Security (sicurezza) e Operations (operazioni IT).

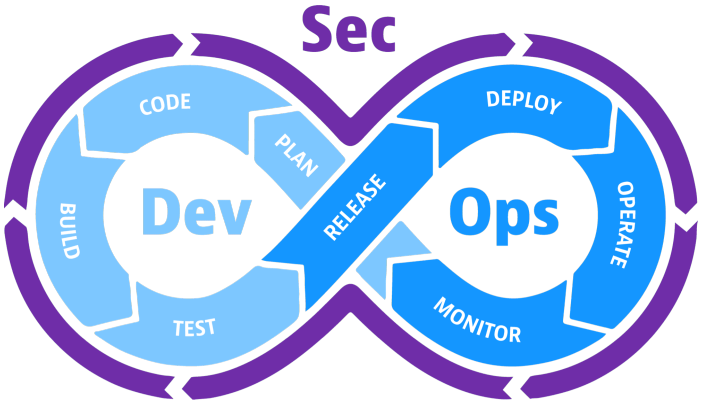

Figura 3.1: Rappresentazione schematica del modello DevSecOps [15]

Il ciclo di vita del software viene ristrutturato per supportare le valutazioni della sicurezza e i test di vulnerabilità in ogni punto della pipeline CI/CD (Figura 3.2). Grazie a queste modifiche diventa possibile sfruttare la velocità e la flessibilità del modello DevOps per sviluppare un'applicazione sicura fin dalle prime fasi della pipeline.

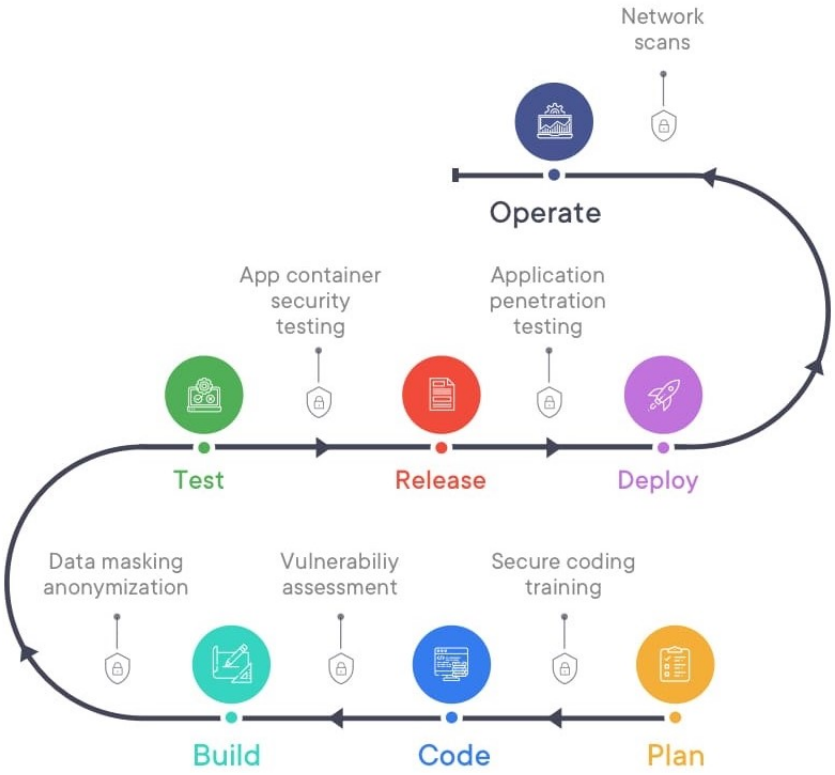

Figura 3.2: Integrazione delle attività di sicurezza nel modello DevOps [16]

La mentalità stabilita da DevSecOps punta a creare un sistema cooperativo in cui ogni figura professionale coinvolta, oltre a svolgere i propri compiti correttamente, deve acquisire anche la capacità di prendere decisioni sulla sicurezza in modo rapido e consapevole. Ognuno, nell'esercizio delle proprie attività, diventa responsabile della sicurezza del software. Per perseguire questo obiettivo, gli ingegneri della sicurezza creano e mettono a punto degli strumenti e dei processi che aiutano gli altri team nel processo decisionale sulla sicurezza. Il team della sicurezza gioca un ruolo fondamentale nel

cambiamento introdotto dalla cultura DevSecOps: non solo deve guidare le attività di controllo, ma deve persino istruire gli altri team per migliorare le loro competenze nel campo della sicurezza informatica.

Più in generale, gli esperti di sicurezza dovrebbero collaborare con il consiglio di amministrazione per cercare di radicare la cultura della sicurezza in tutti i processi, le strategie e gli strumenti aziendali. Sviluppo, operazioni IT e sicurezza devono unirsi in un unico processo semplice, trasparente e armonioso. La collaborazione tra i diversi team non dovrà più essere forzata come nel passato, quando i meccanismi di protezione erano inseriti difficoltosamente dopo una release.

Grazie all'adozione del modello DevSecOps possiamo ottenere i seguenti vantaggi:

- **Maggiore fiducia dei clienti:** i clienti potrebbero non essere in grado di dire se un'azienda sta implementando una strategia DevSecOps all'inizio, ma diventa evidente nel tempo. Massicce violazioni di sicurezza fanno sì che un prodotto perda molti, se non tutti, i suoi utenti poiché nessuno si fida di un prodotto di scarsa qualità.

- **Miglioramento della cultura del lavoro:** quando tutti nell'organizzazione comprendono l'importanza della sicurezza aziendale, diventa più facile collaborare. Il lavoro di squadra riesce persino a far emergere maggiormente le qualità e i valori fondamentali di un'organizzazione o di un prodotto.

- **Riduzione dei costi:** l'implementazione del flusso DevSecOps permette agli sviluppatori di risolvere i problemi di sicurezza non appena vengono rilevati. L'eliminazione di questi problemi costa meno nelle fasi di sviluppo, rispetto alla correzione effettuata dopo il rilascio del software. Inoltre, grazie all'adozione di strumenti di sicurezza automatizzati, è anche possibile ridurre il carico di lavoro sia per gli sviluppatori sia per gli operatori IT.

- **Approccio olistico:** l'applicazione di DevSecOps influisce su tutte le fasi del modello DevOps. Questo garantisce la creazione di una difesa completa ed end-to-end in tutto l'ambiente di produzione.

Purtroppo, investire sulla sicurezza informatica è molto dispendioso economicamente e temporalmente. A causa di ciò, inizialmente l'adozione del modello DevSecOps non fu semplice. Acquisire nuovi addetti alla sicurezza non è un'operazione banale ed economica. La formazione dei team sull'uso di nuovi strumenti e sui concetti basilari di sicurezza informatica sottrae tempo alla distribuzione di nuove applicazioni. Infatti, spesso gli sviluppatori trovano delle scorciatoie per aggirare i test di sicurezza e rilasciare più velocemente nuove versioni software. L'implementazione di meccanismi di sicurezza generalmente comporta la diminuzione della facilità d'utilizzo da parte degli utenti finali, con la conseguente irritazione dei clienti.

Sfortunatamente, le pratiche di sicurezza tradizionali non sono più efficaci nel pericoloso ambiente odierno. La natura dei recenti attacchi rende necessaria una soluzione integrata e profonda per ottenere un prodotto sicuro. DevSecOps è la risposta. Non è uno strumento, una metodologia o un progetto; è un cambiamento radicale dei processi aziendali e delle figure professionali nei confronti della cultura DevOps. L'inclusione dei controlli di sicurezza all'interno dell'intero ciclo di vita del software è un processo lungo e complesso, ma i vantaggi sono immediatamente tangibili: software consegnato di migliore qualità, prodotti più sicuri, meno problemi nella produzione e mitigazione di molti altri rischi connessi all'ingegneria del software. [14, 16, 17]

3.3 Pratiche della cultura DevSecOps

Creare software sicuro tenendo il passo con i requisiti di velocità e scalabilità del mercato è un paradosso per molte delle moderne organizzazioni IT. Spesso il passaggio da DevOps a DevSecOps comporta dei cambiamenti

non semplici da affrontare. Per fortuna, ci sono diverse best practice che agevolano questa evoluzione:

- Promuovere la cultura e la mentalità DevSecOps.
- Aiutare i team a integrare la sicurezza.
- Scegliere quando inserire i controlli.
- Automatizzare strumenti e processi.
- Testare piccole parti.
- Trattare ugualmente vulnerabilità della sicurezza e difetti del software.
- Valutare continuamente.
- Imparare dai fallimenti.
- Perseguire una gestione scalabile.

Adattare un'organizzazione al modello DevSecOps è un'impresa enorme con molte sfide note e sconosciute. Le best practice servono principalmente per gettare le basi di questa rivoluzione; sarà poi compito di tutti i team radicare realmente la mentalità DevSecOps in tutte le decisioni e i processi aziendali. [18]

3.3.1 Promuovere la cultura e la mentalità DevSecOps

Le caratteristiche più importanti della cultura DevSecOps sono: collaborazione, automazione, apprendimento, misurazione e condivisione. L'obiettivo primario è quello di far prosperare una cultura e una mentalità in cui i diversi team cooperano per creare continuamente software sicuro.

I team più motivati e allineati ai principi della cultura DevSecOps devono diventare un esempio da seguire per gli altri, mostrando loro i vantaggi che derivano dalla radicalizzazione della sicurezza nei processi di sviluppo.

Ovviamente, questa integrazione deve partire da piccoli progetti e avvenire nel rispetto delle funzioni quotidiane, bilanciando velocità, scalabilità e sicurezza. Dopo aver preso familiarità con questa mentalità, sarà possibile espanderla all'intera organizzazione.

3.3.2 Aiutare i team a integrare la sicurezza

Una delle principali sfide che i team devono affrontare è la mancanza di strumenti e processi per rendere il software sicuro. La protezione di un'applicazione deve essere garantita ancor prima della scrittura del codice. Attività come la modellazione delle minacce e la revisione dell'architettura sono utilissime per individuare i requisiti e i controlli di sicurezza da implementare durante il ciclo dello sviluppo software. Chiaramente, il team di sviluppo deve anche possedere una formazione sufficiente su come scrivere codice sicuro e risolvere i problemi di sicurezza. In caso di problemi, gli sviluppatori possono sempre contare sul supporto fornito dal team della sicurezza e dai diversi strumenti integrati nell'IDE per scannerizzare il codice, trovare le vulnerabilità ed eliminarle.

3.3.3 Scegliere quando inserire i controlli

Il segreto per implementare con successo un'attività di sicurezza è capire quando deve essere eseguita all'interno del ciclo di vita del software. Ogni organizzazione si approccia al modello DevSecOps sulla base del settore in cui opera, della sua maturità aziendale e della cultura interna. Da ciò deriva che il posizionamento dei controlli di sicurezza sarà unico. Una volta individuati i punti nevralgici in cui posizionare le attività di protezione, l'organizzazione può iniziare a pensare a quali attività seguire e quale tipo di strumenti acquistare.

3.3.4 Automatizzare strumenti e processi

Innanzitutto, è importante identificare quali attività di sicurezza possono essere completamente automatizzate e quali richiedono ancora l'intervento manuale. Ad esempio, l'esecuzione di test statici può essere completamente automatizzata; mentre, la modellazione delle minacce e i test di penetrazione richiedono sforzi manuali. Lo stesso vale per i processi. L'invio di feedback alle parti interessate può essere automatizzato; tuttavia, un'approvazione di sicurezza richiede l'intervento manuale.
L'automazione è fondamentale per:

- Creare il giusto compromesso tra sicurezza, velocità e scalabilità.
- Seguire più facilmente le best practice di DevSecOps.
- Garantire che gli strumenti e i processi vengano utilizzati in modo coerente, ripetibile e affidabile.

Una strategia di automazione di successo dipende anche dagli strumenti e dalla tecnologia utilizzati. Generalmente, gli strumenti migliori sono quelli altamente configurabili e che dispongono di diverse interfacce per consentirne l'integrazione con altri sottosistemi.

3.3.5 Testare piccole parti

Solitamente, quando i team iniziano a integrare le attività di sicurezza nella pipeline DevSecOps, tendono ad abilitare contemporaneamente un ampio set di regole e varie tipologie di scansioni. Questo modus operandi ostacola l'adozione di DevSecOps perché il team di sviluppo viene improvvisamente sommerso da troppe segnalazioni sulla sicurezza, rendendo impossibile correggerle tutte in un breve lasso di tempo. Questo può scoraggiare gli sviluppatori e minacciare l'adozione dell'intera cultura DevSecOps.

È fondamentale, quindi, eseguire i test di sicurezza in maniera gradualmente crescente. Ad esempio, invece di abilitare scansioni complete, i team

dovrebbero utilizzare dei controlli limitati solamente alle vulnerabilità più gravi. Dopo queste prime attività di sicurezza, se ritenuto necessario, saranno effettuate delle scansioni e revisioni più approfondite per garantire una maggiore sicurezza prima del rilascio.

3.3.6 Trattare ugualmente vulnerabilità della sicurezza e difetti del software

La maggior parte delle organizzazioni gestisce la segnalazione delle vulnerabilità di sicurezza in maniera differente rispetto ai difetti funzionali e di qualità. Infatti, le due tipologie di avvisi vengono persino memorizzati in luoghi distinti. Questa separazione limita la visione generale dei team sui problemi dell'applicazione. Mantenere i risultati di sicurezza e qualità in un unico posto aiuta i team a trattare entrambi i tipi di problemi allo stesso modo e con la stessa importanza.

Tuttavia, è importante porre particolare attenzione ai risultati sulla sicurezza provenienti da strumenti di scansione automatica perché potrebbero essere dei falsi positivi. Per evitare questo problema è necessario adottare degli strumenti il più possibile configurabili, in maniera tale da personalizzarli con regole che gestiscano i casi di falso positivo correttamente.

3.3.7 Valutare continuamente

Durante tutte le fasi della pipeline è necessario raccogliere dei dati per valutare il successo o il fallimento della strategia aziendale usata per aderire al modello DevSecOps. Le informazioni possono essere raccolte dai team coinvolti, dai processi utilizzati, dai componenti software integrati e da tante altre fonti. Logicamente, per velocizzare il processo di raccolta, vengono utilizzati degli strumenti automatici.

Tutti i dati raccolti devono essere analizzati e usati dai team per creare delle metriche che riducano i difetti software, i problemi di sicurezza e il tempo medio di implementazione e produzione. Le metriche servono anche

per posizionare strategicamente i controlli di sicurezza, valutare la qualità del software, valutare i compiti svolti dai team, gestire i tempi della pipeline e verificare il corretto impiego della strategia scelta.

3.3.8 Imparare dai fallimenti

Durante l'integrazione di DevSecOps e le misurazioni continue, possono verificarsi dei guasti. Idealmente, questi dovrebbero essere usati dai team per imparare e migliorare. Se una pipeline non segnala mai un errore, allora non sta veramente abbracciando la cultura DevSecOps.
In linea di massima, esistono tre principi:

1. Per garantire il successo del sistema end-to-end, non trasferire mai un errore noto o un difetto di sicurezza critico nel lavoro a valle.

2. Abbreviare i cicli di feedback, segnalando qualsiasi tipo di errore appena si verifica.

3. Costruire una cultura dello sviluppo più sicura imparando dai fallimenti passati, iterando e affrontando i nuovi fallimenti/difetti proprio come delineato nei precedenti punti.

3.3.9 Perseguire una gestione scalabile

I modelli di governance tradizionali ostacolano in modo significativo il modello DevSecOps, che vuole garantire la consegna rapida, sicura e protetta del software. Così come i test di sicurezza, anche le attività di governance dovrebbero essere automatizzate ove possibile.

La governance dovrebbe essere integrata all'interno della pipeline con l'obiettivo di gestire le eccezioni, l'implementazione e la scalabilità della distribuzione software. Un esempio di governance sono i processi di approvazione che l'organizzazione segue prima di rilasciare una nuova versione software. Per ricevere l'approvazione, l'applicazione deve superare diversi controlli diretti e indiretti di sicurezza. In questi casi, la collaborazione tra sviluppatori

e operatori IT è fondamentale e solo l'adozione di un modello di governance inclusivo può garantirla.

3.4 Strumenti DevSecOps

DevSecOps è un nuovo paradigma e non dispone ancora di un set di strumenti ben definito; però è possibile individuare otto tipi di strumenti che sono sempre più utilizzati dalle organizzazioni per integrare la sicurezza nei processi di sviluppo, test e distribuzione: [19]

- Scansione delle vulnerabilità open source.
- Test statici della sicurezza delle applicazioni (Static Application Security Testing).
- Test dinamici della sicurezza delle applicazioni (Dynamic Application Security Testing).
- Scansione delle immagini.
- Strumenti di automazione dell'infrastruttura.
- Dashboard e strumenti di visualizzazione.
- Strumenti di modellazione delle minacce.
- Strumenti di allerta.

3.4.1 Scansione delle vulnerabilità open source

La maggior parte dei progetti software contiene migliaia di dipendenze esterne. Molte di queste sono componenti open source che potrebbero essere stati creati senza le migliori pratiche di sicurezza e, di conseguenza, contenere vulnerabilità o problemi di licenza una volta incorporati nel progetto.

La scansione delle vulnerabilità open source, nota anche come analisi della composizione software, analizza i componenti open source, le librerie e le

dipendenze per comprendere se sono potenzialmente dannosi. Gli artefatti open source rilevati sono identificati dalla loro versione, distribuzione, origine, enumerazione della piattaforma comune e altre caratteristiche distintive. Tutti questi dati vengono raccolti e confrontati con un database che contiene gli avvisi di sicurezza dei fornitori e le vulnerabilità note relative ai diversi componenti open source.

Grazie a questo confronto, è possibile capire se il codice che abbiamo incluso ha delle vulnerabilità conosciute, la loro gravità, il potenziale impatto se venissero sfruttate e dei suggerimenti per eliminarle. Nel processo DevSecOps, la scansione delle vulnerabilità viene effettuata in quasi tutte le fasi. Così facendo, è possibile ottenere i seguenti vantaggi:

- **Scansione in fase di sviluppo:** gli sviluppatori sono automaticamente informati sui problemi di sicurezza relativi ai componenti che stanno includendo. Grazie a questo aiuto, possono decidere di fermare la distribuzione del software fino a quando non riescono a risolvere la vulnerabilità emersa.

- **Scansione nei test di sicurezza:** se un componente ha una vulnerabilità che supera una soglia di rischio predefinita, viene generato un avviso per il team della sicurezza. Quest'ultimo, assieme al team di sviluppo, deve ispezionare il componente e cercare di ripararlo prima della distribuzione.

- **Scansione in produzione e pre-produzione:** utilizzata per rilevare tutte le vulnerabilità entrate nel progetto attraverso mezzi diversi dalla pipeline CI/CD. Generalmente questo tipo di scansione individua le vulnerabilità zero-day o i malware.

3.4.2 Test statici della sicurezza delle applicazioni

I test statici della sicurezza delle applicazioni consentono agli sviluppatori di eseguire la scansione del codice sorgente per ricercare codici deboli o non

sicuri, identificando potenziali punti di attacco da eliminare. Ogni problema rilevato ha un livello di gravità. Ovviamente, gli sviluppatori dovranno organizzare il proprio lavoro per risolvere prima le vulnerabilità più gravi e poi tutte le restanti.

SAST è un metodo di white-box testing. Quando i test statici vengono integrati nel ciclo di sviluppo software, i team possono definire il numero massimo di problemi e il loro livello di gravità sufficiente per causare il fallimento della build o impedire a un componente di passare alle fasi successive della pipeline. Inoltre, gli sviluppatori integrano nel proprio IDE dei meccanismi che evidenziano i punti deboli del codice mentre lo stanno scrivendo, aiutandoli a creare codice sicuro fin dall'inizio.

3.4.3 Test dinamici della sicurezza delle applicazioni

Gli strumenti per i test dinamici possono eseguire automaticamente test di sicurezza sulle applicazioni in esecuzione, testando una varietà di minacce senza richiedere l'accesso al codice sorgente. In genere, questi strumenti testano le interfacce HTTP e HTML di un'applicazione web.

DAST è un metodo di black-box testing, in grado di identificare le vulnerabilità delle applicazioni dal punto di vista di un utente malintenzionato, simulando i più comuni vettori di attacco e ricreando il modo in cui un aggressore può scoprire e sfruttare le vulnerabilità. Infine, poiché i test dinamici sono automatizzati, è facile integrarli con gli altri strumenti automatici DevOps.

3.4.4 Scansione delle immagini

I team DevOps distribuiscono i componenti software mediante immagini e contenitori Docker. Una delle preoccupazioni principali nell'ambiente DevSecOps è identificare delle vulnerabilità nelle immagini dei container, che spesso vengono estratte da repository pubblici o da altre fonti non attendibili. Le immagini Docker possono anche contenere dei componenti software

obsoleti, potenzialmente pieni di vulnerabilità non ancora risolte.

La scansione delle immagini viene impiegata proprio per verificare che le immagini contengano solo codice e artefatti affidabili e conformi alle migliori pratiche di configurazione sicura. I processi DevSecOps che impiegano i contenitori devono includere la scansione e la correzione delle immagini in ogni fase della pipeline CI/CD.

3.4.5 Strumenti di automazione dell'infrastruttura

DevSecOps si basa fortemente sull'automazione della configurazione e della sicurezza dell'infrastruttura. Gli strumenti automatici implementano una gestione basata su eventi e gestiscono sia la configurazione sia l'infrastruttura come codice, rilevando e riparando autonomamente varie vulnerabilità di sicurezza e problemi di configurazione.

3.4.6 Dashboard e strumenti di visualizzazione

Nel modello DevSecOps è importante adottare degli strumenti che agevolino la visualizzazione e la condivisione delle informazioni sulla sicurezza tra tutti i team coinvolti: sviluppo, operazioni IT e sicurezza. Spesso si utilizza un unico pannello di controllo, dal quale è possibile gestire anche i rischi di sicurezza esistenti. Strumenti efficaci mostrano persino le tendenze di mercato, gli indicatori di performance aziendali, la crescita o la riduzione nel tempo delle vulnerabilità per un'applicazione specifica e tanto altro.

Nei momenti in cui è necessario effettuare dei controlli approfonditi o prendere delle decisioni sul futuro dell'applicazione, i dashboard personalizzati tornano molto utili perché consentono ai team di mantenere sempre un quadro completo sulla situazione inerente a uno specifico progetto software.

3.4.7 Strumenti di modellazione delle minacce

Gli strumenti di modellazione delle minacce aiutano il team della sicurezza a prevedere, rilevare e valutare le minacce sull'intera superficie di attacco.

L'obiettivo è fornire ai team più dati possibili per prendere rapidamente decisioni proattive e ridurre al minimo l'esposizione del software ai rischi di sicurezza. Oggigiorno sono disponibili molti strumenti con un'ampia gamma di funzionalità, tra cui la creazione automatica di modelli delle minacce e la visualizzazione in tempo reale dei dati raccolti.

3.4.8 Strumenti di allerta

Gli strumenti di allerta aiutano i team DevSecOps a rispondere rapidamente agli eventi di sicurezza. Uno strumento di allerta ideale avvisa il team di sicurezza solo dopo che l'evento anomalo è stato analizzato, categorizzato e ritenuto degno dell'attenzione di un operatore specializzato. Questo procedimento è fondamentale per ridurre al minimo i falsi allarmi del sistema ed evitare di distogliere l'attenzione dei team dagli ordinari flussi di lavoro DevSecOps. Una volta che i team vengono informati, possono indagare rapidamente sull'evento e applicare le correzioni necessarie.

Capitolo 4

Futuro di DevSecOps

Negli ultimi anni, DevSecOps ha preso piede nel mondo dell'informatica e sempre più organizzazioni cercano di adottarlo. Gli scenari futuri per questo modello sono svariati, ma tutti si concentrano su quattro pilastri: [20]

- **Persone:** le risorse umane di un'organizzazione sono un importante catalizzatore nella crescita di DevSecOps poiché aiutano a rompere le barriere tradizionali. Innovare con un team che valorizza l'obiettivo condiviso serve come fonte di ispirazione per l'intera organizzazione. Solo con una profonda comprensione degli approcci alla sicurezza informatica, il personale potrà adottare pratiche di codifica sicure ed efficienti.

- **Processi:** le organizzazioni dovranno incorporare nei loro processi velocità, qualità e compattezza. Per limitare al minimo le rielaborazioni progettuali, riducendo significativamente le violazioni di sicurezza, è fondamentale migliorare le pratiche di implementazione, design specifico per i clienti, modellazione delle minacce e scansione incrementale del codice.

- **Tecnologia:** i software di sicurezza informatica dovranno fornire sempre più strumenti di sicurezza come codice, test come codice e infra-

struttura come codice per sradicare dalla pipeline CI/CD le attività di sicurezza manuali e aumentarne la velocità.

- **Governance:** le organizzazioni utilizzeranno maggiormente dei framework che agevolano la governance aziendale. Questi framework, basati su una struttura gerarchica, saranno formati da un insieme di strumenti e processi usati per incrementare l'efficienza e la velocità dei compiti da svolgere.

4.1 Sfide da affrontare

Il passaggio dal modello DevOps a DevSecOps è pieno di sfide e difficoltà da superare. Non si tratta solamente di utilizzare nuovi strumenti o maggiori accortezze durante lo sviluppo software; la rivoluzione deve avvenire nella mentalità dei team e nella gestione di tutte le fasi della pipeline CI/CD.

Tutte le organizzazioni che, nel futuro prossimo, vorranno adottare il modello DevSecOps dovranno sicuramente affrontare le seguenti sfide:

- Riluttanza al cambiamento culturale.
- Mancanza di conoscenza.
- Integrazione di strumenti complessi.
- Scontro tra strumenti di sicurezza tradizionali e moderni.
- Gestione dei falsi positivi.

Le sfide soprastanti sono ardue da superare e tante organizzazioni si arrendono nel tentativo di far funzionare tutte le cose perfettamente. L'adozione di DevSecOps è un processo lungo ma, una volta concluso, può migliorare significativamente l'intera organizzazione. Cercare di ottenere fin da subito una sicurezza perfetta in ogni fase dello sviluppo, ostacolerà e demoralizzerà solamente i team coinvolti. [21, 22]

4.1.1 Riluttanza al cambiamento culturale

L'ostacolo più significativo che la maggior parte delle organizzazioni deve affrontare nell'adozione delle metodologie DevSecOps è la resistenza al cambiamento da parte dei team. Siccome le persone coinvolte sono già profondamente abituate agli attuali processi di sviluppo, può essere difficile rompere il vecchio sistema e adottare nuovi metodi di lavoro. Il comfort tende a trionfare sull'esplorazione di ciò che non è familiare. Inoltre, in molti vedono ancora la sicurezza informatica come un optional che limita la creatività e la velocità degli sviluppatori.

Per superare questi problemi, il cambiamento deve partire dall'alto e deve essere graduale. I responsabili aziendali non solo dovranno rivedere l'intera gestione del processo di sviluppo, ma dovranno anche trovare dei modi per radicare il nuovo cambiamento nella cultura lavorativa aziendale. Con i giusti accorgimenti e strumenti, i risultati positivi non tarderanno ad arrivare e i team inizieranno veramente a comprendere la validità di DevSecOps.

4.1.2 Mancanza di conoscenza

Nonostante negli ultimi anni il tema della sicurezza informatica sia tra i più discussi, sono ancora in molti a non conoscere le giuste pratiche e tecniche per proteggersi. Da un lato, troviamo gli sviluppatori e gli operatori IT che, probabilmente, non hanno ricevuto un'adeguata preparazione in merito e non riescono autonomamente a creare e amministrare il software in maniera sicura. Dall'altro lato, abbiamo gli utenti finali che utilizzano l'applicazione senza i giusti accorgimenti, mettendo a rischio i propri dati personali.

Un'organizzazione che decide di adottare DevSecOps dovrà sicuramente scegliere un team di sicurezza all'altezza delle lacune presenti. Il team dovrà far comprendere l'importanza della sicurezza informatica, radicare la cultura della protezione nelle attività organizzative e istruire gli altri team su come sviluppare e gestire del software sicuro. Naturalmente, l'organizzazione deve supportare l'operato del team della sicurezza e promuovere l'adozione della

cultura DevSecOps.

4.1.3 Integrazione di strumenti complessi

La maggior parte degli strumenti usati in DevOps proviene da vari fornitori. I team, in base alle proprie esigenze, scelgono gli strumenti più adatti alla gestione del codice sorgente, dell'integrazione continua, della distribuzione continua, della compilazione, dei repository binari, dell'automazione dei test e i sistemi di risoluzione dei problemi. L'integrazione di tutti questi strumenti è ragionevolmente facile in una pipeline DevOps ben oliata.

Purtroppo, la scelta di DevSecOps comporta anche l'integrazione di nuovi strumenti per garantire la sicurezza lungo tutto il ciclo di vita del software. Innanzitutto, bisogna comprendere dove posizionare i nuovi tool affinché svolgano correttamente il lavoro per cui sono stati pensati. Successivamente, è fondamentale capire come integrarli con gli altri strumenti e imparare a interpretare i dati che questi generano, al fine di rendere tutti i team autonomi nella gestione della sicurezza del progetto software. La migliore soluzione è riuscire a trovare uno strumento in grado di risolvere tutti i problemi di sicurezza, semplificando sia l'integrazione sia il lavoro degli sviluppatori.

4.1.4 Scontro tra strumenti di sicurezza tradizionali e moderni

I tradizionali strumenti di sicurezza erano stati pensati per i modelli di sviluppo software precedenti a DevOps. In passato, il team della sicurezza iniziava il proprio lavoro solo dopo il rilascio di una nuova versione. Durante la ricerca delle vulnerabilità, il team di sviluppatori continuava parallelamente lo sviluppo di altre parti dell'applicazione. Questa lenta e distaccata interazione fra i due team non è più tollerabile nei fulminei modelli DevOps e DevSecOps.

Questi ultimi necessitano l'adozione di strumenti più veloci e affidabili. Infatti, gli strumenti di sicurezza moderni sfruttano ampiamente l'automa-

zione per garantire la velocità e l'agilità richiesta. I test odierni sono attivati da eventi generati dal ciclo di vita del software, sono eseguiti in background senza l'intervento umano e applicano autonomamente le policy di sicurezza per aiutare gli sviluppatori a concentrarsi solamente sui rischi più elevati.

Sfortunatamente, per godere di tutti i benefici degli strumenti moderni, bisogna configurarli correttamente e questa non è un'operazione banale. Inoltre, il team della sicurezza deve essere in grado di scegliere gli strumenti più adatti al tipo di contesto in cui devono essere integrati.

4.1.5 Gestione dei falsi positivi

La configurazione degli strumenti di sicurezza è molto complessa e può variare per ogni progetto software. Durante i test, uno dei problemi principali è la presenza di falsi positivi. Purtroppo, non esiste un approccio generico per individuarli. L'unica soluzione è eseguire tanti test, limitando o modificando di volta in volta le regole settate sui diversi strumenti.

Il team della sicurezza deve conoscere anche la tecnica multi-strumento. Essa consiste nell'utilizzo di vari strumenti, ognuno dei quali adotta diversi metodi per rilevare le vulnerabilità. L'approccio multi-strumento aiuta a far emergere i falsi negativi che, solitamente, restano sepolti nel rumore dei falsi positivi. Questi modus operandi richiedono molto tempo però fanno maturare al team della sicurezza molta esperienza nella configurazione degli strumenti e nell'affinare le regole di controllo.

4.2 Tendenze future

L'espansione della metodologia DevSecOps è agevolata anche dalle tendenze future nei seguenti quattro campi:

1. **Architettura**
    - Da monolitica a microservizi.
    - Cloud come impostazione predefinita.

2. **Infrastruttura**
    - Kubernetes.
    - Infrastruttura agile.
3. **Tecnologia**
    - Rilascio e distribuzione automatizzati.
    - Malware avanzati e APT (Advanced Persistent Threat).
    - Intelligenza Artificiale e Machine Learning.
4. **Organizzazione**
    - Impatto economico.
    - Distinzione tra sviluppo e test.

In generale, l'obiettivo futuro di DevSecOps è quello di diventare meno costoso, più automatizzato e facilmente fruibile da tutte le organizzazioni, persino le meno innovative. [23, 24]

4.2.1 Da architettura monolitica a microservizi

Il passaggio da applicazioni massicce e inflessibili a software agile e leggero ha accelerato l'adozione di DevOps e DevSecOps. Le architetture a microservizi basate su container si diffonderanno a macchia d'olio e faciliteranno l'installazione, esecuzione e mantenimento dei sistemi futuri. Più in generale, questa tendenza segna il passaggio dai server centralizzati a quelli basati su microservizi, che permettono la distribuzione di diversi servizi in maniera indipendente. I team riusciranno a collaborare più facilmente fra di loro e velocizzeranno la codifica, il debug e la distribuzione di nuovi componenti software mediante gerarchie di container. L'adozione dell'architettura a microservizi è inevitabile per gestire la moderna complessità del software. Sarà compito dei team DevSecOps sfruttare al massimo le sue potenzialità, ponendo sempre attenzione alla sicurezza complessiva.

4.2.2 Cloud come impostazione predefinita

Il passaggio ai microservizi e alla containerizzazione ha anche contribuito ad accelerare una tendenza correlata, in corso da quasi un decennio: lo sviluppo e la gestione di software nel cloud come approccio predefinito. Quando DevOps iniziò a essere adottato, uno di problemi principali era la creazione e la transizione verso servizi cloud a cui poter accedere in modo efficiente e sicuro. Oggigiorno, grazie alla presenza di servizi cloud ben progettati, i diversi team (sviluppatori, operatori IT e servizio clienti) possono già cooperare, sviluppare ed eseguire software sul cloud come impostazione predefinita. In futuro, le piattaforme cloud si diffonderanno e perfezioneranno i loro strumenti talmente tanto da diventare indispensabili per le organizzazioni.

4.2.3 Kubernetes

Adottato su larga scala nel settore tecnologico, Kubernetes è un software open source progettato per ottimizzare la gestione, l'implementazione e la scalabilità dei container. Il suo successo è dovuto proprio ai cambi architetturali sopra elencati.

Grazie a Kubernetes, le organizzazioni possono produrre software più rapidamente e sicuramente, la gestione delle risorse e la trasparenza sono notevolmente migliorate, il tempo necessario per correggere i bug si riduce e l'adozione dei container consente a più team di sviluppare un progetto contemporaneamente.

In futuro, Kubernetes e il sistema di gestione dei container elimineranno la necessità di qualsiasi interazione umana e creeranno uno scenario completamente automatizzato, il cosiddetto NoOps. Tutto ciò sarà possibile sfruttando degli strumenti di intelligenza artificiale e machine learning in grado di analizzare i dati provenienti dalla pipeline CI/CD e decidere autonomamente quale operazione effettuare, senza l'ausilio dell'essere umano. Kubernetes imposta un'infrastruttura basata su registrazione, rilevamento continuo, coerenza e auto-riparazione che renderà più efficienti i sistemi im-

piegati nel ciclo DevSecOps e semplificherà la sua adozione nella maggior parte delle organizzazioni.

4.2.4 Infrastruttura agile

L'infrastruttura agile è diventata il nuovo standard nel campo dello sviluppo software e sta influenzando notevolmente l'adozione di DevSecOps. Essa si concentra sulla collaborazione, organizzazione e diversificazione delle competenze per ottenere resilienza, distribuzione rapida e processi di lavoro auto-miglioranti. DevSecOps può essere visto come una sorta di framework per questo tipo di infrastruttura.

Adattarsi a un'infrastruttura agile significa creare dei team composti da membri esperti in vari campi, creando così una conoscenza completa. Ogni versione consegnata è seguita da un ciclo di feedback. I team agili devono essere trasparenti, collaborativi e aperti ai cambiamenti aziendali interni ed esterni. In sostanza, l'infrastruttura agile rappresenta le basi che un'organizzazione deve avere se vuole implementare DevSecOps con successo.

4.2.5 Rilascio e distribuzione automatizzati

Ogni volta che vengono rilasciati degli aggiornamenti o viene creata una nuova applicazione, è necessario spostare manualmente il software tra gli ambienti di test e produzione. Questa attività inefficiente e faticosa amplifica significativamente il divario tra CI e CD, rallentando l'intera pipeline.

In futuro si punta ad automatizzare interamente il rilascio e la distribuzione, semplificando e velocizzando completamente questa transizione. Il codice, dopo essere stato testato, entrerà autonomamente in produzione e sarà distribuito sui controlli di versione o sui servizi di hosting. Sfoca virtualmente la linea tra CI e CD dal punto di vista dello sviluppatore. L'automazione diminuisce notevolmente il lavoro delle organizzazioni e agevola l'impiego delle metodologie DevSecOps.

4.2.6 Malware avanzati e APT

La crescente sensibilità verso la sicurezza informatica ha favorito la creazione di ambienti di sviluppo e produzione più sicuri ma, allo stesso tempo, anche di malware avanzati, spesso implementati mediante APT (minaccia complessa e avanzata). Questa nuova classe di minacce riesce a sfruttare la natura distribuita delle moderne architetture software per attaccare i componenti e i dati dei moderni sistemi cloud.

I malware avanzati minacceranno soprattutto i team che lavoreranno ancora con l'approccio DevOps, spingendoli a migrare verso il modello DevSecOps per integrare la sicurezza fin dalle prime fasi e creare un software maggiormente robusto e sicuro. Logicamente non basta solo un'applicazione sicura; è fondamentale porre la sicurezza al primo posto anche nella progettazione dell'infrastruttura.

4.2.7 Intelligenza Artificiale e Machine Learning

L'evoluzione dell'intelligenza artificiale e dell'apprendimento automatico cambierà notevolmente lo sviluppo software. Con la crescente automazione, per i sistemi di machine learning diventa più facile elaborare grandi quantità di dati, fare previsioni affidabili ed effettuare analisi estremamente accurate. L'intelligenza artificiale può utilizzare questi dati e queste previsioni per amministrare macchine intelligenti, in grado di determinare le operazioni da effettuare senza l'ausilio umano.

Oggi questa tendenza è applicata in minima parte perché i diversi team stanno ancora studiando come utilizzarla su larga scala. Tuttavia, in futuro, l'AI e il ML potrebbero rivoluzionare profondamente DevSecOps, favorendo l'avvento del ciclo NoOps. All'interno di quest'ultimo tutte le operazioni IT sono automatizzate e non necessitano dell'intervento umano. In realtà, uno scenario futuro più realistico è quello in cui gli operatori IT non vengono sostituiti ma sono affiancati ai sistemi intelligenti per ridurre al minimo il lavoro ripetitivo e svolgere i propri compiti con maggiore sicurezza e affidabilità.

4.2.8 Impatto economico

Assumere addetti alla sicurezza, acquistare e integrare nuovi strumenti, ridefinire i processi aziendali e formare gli altri team sul tema della protezione informatica sono solo alcuni dei costi che un'organizzazione deve sostenere quando decide di adottare il modello DevSecOps. Naturalmente, in futuro la sicurezza informatica diventerà indispensabile e si troveranno dei metodi sempre più economici per implementarla. Tuttavia, è importante adoperarsi già da ora per difendere la propria organizzazione e i software sviluppati.

La digitalizzazione è una tendenza futura inarrestabile e porta con sé anche l'incremento degli attacchi informatici. Questi ultimi diventeranno talmente complessi da essere difficilmente arrestabili senza adeguate difese aziendali. Basti pensare ai recenti malware WannaCry e NotPetya che sono riusciti a bloccare aziende energetiche, trasporti pubblici, siti governativi, banche e strutture sanitarie. La maggior parte delle organizzazioni attaccate ha dovuto affrontare delle spese non pianificate per correggere le lacune di sicurezza, perdendo persino il proprio vantaggio competitivo in alcuni casi. Con i dovuti meccanismi di difesa, questi problemi potevano essere tranquillamente evitati.

Nonostante sia costoso adottare il modello DevSecOps, i vantaggi che ne derivano surclassano tutte le spese sostenute. Del resto, investire oggi sulla difesa digitale è sempre più economico che dover rimediare un domani a violazioni di privacy, furto di dati, frodi e blocchi di sistemi.

4.2.9 Distinzione tra sviluppo e test

La crescente complessità delle applicazioni necessita di figure professionali altrettanto competenti. Purtroppo, non tutte le organizzazioni possono permettersi di assumere personale esperto. Fortunatamente, è possibile esternalizzare alcune parti del processo di sviluppo software presso apposite società specializzate. Una delle fasi più cruciali e problematiche è quella dei test. Questi ultimi non solo hanno bisogno di tanto tempo per essere con-

figurati ed eseguiti, ma distolgono persino il team degli sviluppatori e della sicurezza dalle loro normali mansioni. Quindi, delegare i test a società esterne può essere la giusta soluzione. Il loro compito è testare il funzionamento e la sicurezza dei componenti software, valutando anche la qualità complessiva e la correttezza delle configurazioni proposte. Tutte queste operazioni sono eseguite da esperti del settore, in grado di perfezionare i processi di test in base al software da analizzare.

A godere dei vantaggi dell'esternalizzazione sono soprattutto gli sviluppatori e gli addetti alla sicurezza. I primi potranno dedicarsi principalmente alla scrittura di codice ottimizzato e sicuro, tralasciando la ricerca spasmodica di bug. I secondi potranno focalizzarsi maggiormente sullo studio delle nuove vulnerabilità, la creazione di meccanismi di protezione e la formazione degli altri team. Gli esperti dei test possono addirittura instaurare un rapporto di collaborazione con i team interni all'organizzazione, con l'obiettivo di aumentare l'efficienza del processo di sviluppo software e distribuire applicazioni di qualità elevata.

Dato l'elevato numero di benefici, l'esternalizzazione e la distinzione tra sviluppo e test rappresenteranno sicuramente una delle tendenze future maggiormente impiegate, soprattutto dalle software house di piccole dimensioni.

Conclusioni

Lo scopo della tesi è analizzare l'adozione di DevSecOps come processo di sviluppo software nel passato, presente e futuro. Dallo studio del passato emerge soprattutto il ruolo secondario della sicurezza informatica. L'obiettivo delle software house era rilasciare rapidamente software funzionante e poco costoso. Nonostante fossero accaduti gravi attacchi informatici, la necessità di proteggersi nel mondo digitale non era molto sentita nemmeno tra gli utenti finali. Il team della sicurezza era visto come un'entità esterna, non sempre essenziale.

Per fortuna, oggigiorno molte delle organizzazioni hanno cambiato questa tendenza e hanno iniziato a radicare la sicurezza informatica all'interno del proprio processo di sviluppo software. DevSecOps rappresenta il giusto punto di partenza per difendere concretamente le applicazioni odierne; le quali gestiscono e conservano quantità spropositate di dati sensibili, in cui l'utente è il prodotto. Purtroppo, dalla ricerca sullo stato attuale di integrazione emerge una situazione abbastanza critica: poche sono le organizzazioni che riescono effettivamente ad adottare DevSecOps. La colpa è principalmente della mentalità restia degli sviluppatori e degli operatori IT, degli strumenti macchinosi e dalla mancanza di una buona conoscenza sulla sicurezza informatica. Tra qualche anno, sia le istituzioni sia gli utenti più consapevoli inizieranno a pretendere una maggiore protezione. Il futuro delle organizzazioni che non vorranno uniformarsi ai prossimi standard di sicurezza sicuramente non sarà roseo.

Esaminando le tendenze future, è possibile notare due scenari paralle-

li. Nel primo, la semplificazione di diverse attività grazie all'introduzione dell'automazione, della diffusione del cloud, dell'intelligenza artificiale e del machine learning. Nel secondo, un aumento della complessità nell'amministrazione globale del processo di sviluppo, data proprio dalla necessità di formarsi sulle nuove architetture, infrastrutture e tecnologie. Persino le sfide da affrontare per adottare DevSecOps si evolveranno e diventeranno sempre più ardue da superare senza le opportune basi sulla sicurezza informatica.

Il manifesto DevSecOps è stato pubblicato solo pochi anni fa. Essendo un modello relativamente nuovo, personalmente penso che sia il momento migliore per adottarlo. La sicurezza informatica diventerà un must have nelle prossime applicazioni ed è fondamentale partire ora per guadagnare sufficiente esperienza ed essere in grado di fronteggiare le trasformazioni future. In conclusione, DevSecOps è il migliore investimento sulla sicurezza che un'organizzazione possa fare per sé stessa e per tutti i suoi clienti futuri.

Bibliografia

[1] wikipedia.org, DevOps
https://it.wikipedia.org/wiki/DevOps

[2] osservatori.net, Metodologia DevOps: come funziona e perché è fondamentale per le Direzioni IT
https://blog.osservatori.net/it_it/devops-come-funziona-fondamentale-per-direzione-it

[3] medium.com, Agile Methodology: Incremental and Iterative way of development
https://medium.com/@ashutoshagrawal1010/agile-methodology-incremental-and-iterative-way-of-development-a6614116ae68

[4] nitroacademy.it, Perché DevOps?
https://nitroacademy.it/2019/05/22/dev-ops/

[5] azure.microsoft.com, Cos'è DevOps?
https://azure.microsoft.com/it-it/overview/what-is-devops/

[6] aws.amazon.com, Cos'è DevOps?
https://aws.amazon.com/it/devops/what-is-devops/

[7] netapp.com, Cos'è DevOps?
https://www.netapp.com/it/devops-solutions/what-is-devops/

[8] redhat.com, Cos'è una pipeline di CI/CD?
https://www.redhat.com/it/topics/devops/what-cicd-pipeline

[9] statista.com, The most common types of cyber crime
https://www.statista.com/chart/24593/most-common-types-of-cyber-crime/

[10] dgroove.it, Quali sono le attività fondamentali per la sicurezza informatica personale?
https://www.dgroove.it/quali-sono-le-attivita-fondamentali-per-la-sicurezza-informatica-personale/5753/

[11] wikipedia.it, Sicurezza informatica
https://it.wikipedia.org/wiki/Sicurezza_informatica

[12] cybersecurity360.it, Secure coding: regole e linee guida per lo sviluppo software sicuro
https://www.cybersecurity360.it/cybersecurity-nazionale/secure-coding-regole-e-linee-guida-per-lo-sviluppo-software-sicuro/

[13] redhat.com, Cos'è la metodologia DevSecOps?
https://www.redhat.com/it/topics/devops/what-is-devsecops

[14] devsecops.org, What is DevSecOps?
https://www.devsecops.org/blog?offset=1442901285190&tag=DevSecOps+Explained

[15] dynatrace.com, What is DevSecOps? And what you need to do it well
https://www.dynatrace.com/news/blog/what-is-devsecops/

[16] netsolutions.com, What is DevSecOps?
https://www.netsolutions.com/insights/what-is-devsecops/

[17] medium.com, What is DevSecOps?
https://medium.com/@aditi.chaudhry92/what-is-devsecops-cb14cfd457b2

[18] synopsys.com, DevSecOps best practices for building secure software https://www.synopsys.com/blogs/software-security/devsecops-best-practices/

[19] aquasec.com, DevSecOps tools https://www.aquasec.com/cloud-native-academy/devsecops/devsecops-tools/

[20] opensenselabs.com, Are you ready for a future with DevSecOps? https://opensenselabs.com/blog/articles/future-DevSecOps

[21] microtica.com, Common DevSecOps challenges and how to overcome theminutes https://microtica.com/blog/common-devsecops-challenges-and-how-to-overcome-them/

[22] qentelli.com, DevSecOps - Challenges at scale https://www.qentelli.com/thought-leadership/insights/devsecops-challenges-scale

[23] infoq.com, Nine trends that are influencing the adoption of Devops and Devsecops https://www.infoq.com/articles/devops-secure-trends/

[24] cdotrends.com, 7 Trends influencing DevOps and DevSecOps adoption https://www.cdotrends.com/story/15251/7-trends-influencing-devops-and-devsecops-adoption

Elenco delle figure