Secure Software Development Life Cicle (SSDLC)

L'SSDLC è un processo di sviluppo software che rende il software più sicuro.

Un Secure SDLC richiede l'aggiunta di test di sicurezza in ogni fase di sviluppo del software, dalla progettazione, allo sviluppo, al deploy e oltre.

La sicurezza si applica a tutte le fasi del ciclo di vita dello sviluppo del software (SDLC) e deve essere in primo piano nella mente degli sviluppatori mentre implementano i requisiti del software.

L'obiettivo di Secure SDLC non è eliminare completamente i controlli di sicurezza tradizionali, come i pen test, ma piuttosto includere la sicurezza nell'ambito delle responsabilità degli sviluppatori e metterli in grado di creare applicazioni sicure fin dall'inizio.

Secure Software Development Life Cicle (SSDLC)

La sicurezza delle applicazioni è spesso responsabilità dei team di **sicurezza informatica** dedicati al supporto delle applicazioni. Tempo fa le applicazioni venivano testate solo dopo il loro rilascio. I test avvenivano in ambienti di produzione. Sfortunatamente, ciò significava che qualsiasi potenziale vulnerabilità sarebbe rimasta *"allo scoperto"* per essere sfruttata dagli aggressori per un certo numero di settimane o addirittura mesi prima di poter essere notata e affrontata. Di conseguenza, la maggior parte delle aziende ha scelto di integrare i *test di produzione con test di sicurezza pre-rilascio*. Le applicazioni dovevano superare il controllo di sicurezza prima di distribuire il codice in produzione.

Problema

Questa fase di test di sicurezza spesso richiede diverse settimane per essere completata, ***allungando il ciclo di rilascio***. Inoltre il suo esito è del tutto impossibile da pianificare: Un test di sicurezza può trovare solo poche vulnerabilità che possono essere risolte in pochi giorni o può trovare decine o addirittura centinaia di vulnerabilità. La correzione delle vulnerabilità riscontrate potrebbe richiedere modifiche significative al codice che sostituiscono interi componenti sottostanti, che dovranno poi essere riverificati rispetto ai requisiti dell'applicazione e a un altro test di sicurezza.

Questo può - e spesso lo fa - far arretrare di settimane gli sviluppatori di applicazioni che continuano a cercare di rispettare scadenze di rilascio ormai impossibili. Questo crea un forte attrito all'interno delle organizzazioni e porta le aziende a scegliere tra due opzioni sbagliate: ***rilasciare un'applicazione con vulnerabilità*** o ***non rispettare gli obiettivi di consegn***a (o entrambe le cose).

Può costare fino a 100 volte di più risolvere un problema scoperto a questo punto del ciclo di vita dell'SDLC che risolverlo semplicemente all'inizio del processo.

Secure Software Development Life Cicle (SSDLC)

L'implementazione della sicurezza del ciclo di vita del software riguarda ogni ***fase del processo di sviluppo del software***. Richiede un mindset incentrato sulla distribuzione sicura, sollevando i problemi nelle fasi di requisiti e sviluppo non appena vengono scoperti. Questo è molto più efficiente, e molto più economico, che aspettare che i problemi di sicurezza si manifestino nell'applicazione online

Secure Software Development Life Cicle (SSDLC)

Le 5 fasi

Ogni fase del ciclo di vita del software deve contribuire alla sicurezza dell'applicazione complessiva. Ciò avviene in modi diversi per ogni fase dell'SDLC, con una nota critica: la sicurezza del ciclo di vita dello sviluppo del software deve essere in primo piano nella mente di tutto il team.

1. **Requirements**
2. **Design**
3. **Development**
4. **Verification**
5. **Maintenance and Evolution**

Esempio
Vediamo un esempio di ciclo di vita di sviluppo del software sicuro per un team che crea un portale per il rinnovo delle iscrizioni dei soci

Secure Software Development Life Cicle (SSDLC)

Requirements

In questa prima fase si raccolgono i requisiti per le nuove funzionalità da parte dei vari stakeholder. È importante identificare tutte le considerazioni sulla sicurezza per i requisiti funzionali che si stanno raccogliendo per la nuova release.

Esempio di requisito funzionale: l'utente deve poter verificare le proprie informazioni di contatto prima di poter rinnovare l'iscrizione.

Esempio di considerazione sulla sicurezza: gli utenti devono poter vedere solo le proprie informazioni di contatto e quelle di nessun altro.

Secure Software Development Life Cicle (SSDLC)

Design

In questa fase trasformiamo i requisiti in un piano di come devono apparire in un applicativo reale. In questo caso, i requisiti funzionali descrivono tipicamente ciò che deve accadere, mentre i requisiti di sicurezza si concentrano su ciò che non deve accadere.

Esempio di requisito di progettazione: la pagina deve recuperare il nome, l'e-mail, il telefono e l'indirizzo dell'utente dalla tabella CUSTOMER_INFO del database e visualizzarli sullo schermo.

Esempio di problema di sicurezza: dobbiamo verificare che l'utente abbia un token di sessione valido prima di recuperare le informazioni dal database. In caso di assenza, l'utente deve essere reindirizzato alla pagina di login.

Secure Software Development Life Cicle (SSDLC)

Development

Fase di scrittura del codice dove le preoccupazioni di solito si spostano sulla necessità di assicurarsi che il codice sia ben scritto dal punto di vista della sicurezza. Di solito vengono stabilite ***linee guida per la codifica sicura*** e vengono effettuate revisioni del codice per verificare che tali linee guida siano state seguite correttamente. Queste revisioni del codice possono essere **manuali** o **automatizzate** utilizzando tecnologie come ***static application security testing (SAST).***

Molto spesso ci si affidano alle funzionalità esistenti, di solito fornite da componenti open source gratuiti, per offrire nuove funzionalità e quindi valore all'organizzazione il più rapidamente possibile. Oltre il 90% delle moderne applicazioni utilizzano codice open-source. Questo codice viene solitamente verificato con dei ***Software Composition Analysis (SCA) tools.***

Secure Software Development Life Cicle (SSDLC)

SAST

SAST è una tecnica di ***scansione delle vulnerabilità*** che si concentra sul codice sorgente, sul bytecode o sul codice assembly. Lo scanner può essere eseguito all'inizio della pipeline CI o anche come plugin IDE durante la codifica. Gli strumenti SAST monitorano il codice, garantendo la protezione da problemi di sicurezza come per esempio il salvataggio di una password in chiaro.

- **Scansioni all'inizio dello sviluppo**: La maggior parte degli strumenti SAST lavora esclusivamente sul codice sorgente, verificandolo rispetto alle best practice. Ciò significa che SAST può essere applicato mentre si scrive il codice.

- **Indica le posizioni problematiche del codice e spiega il problema riscontrato**: SAST mostra la posizione esatta di ogni vulnerabilità e e spiega il flusso di dati.

- **Non richiede l'esecuzione di applicazion**i: SAST lavora sul codice sorgente prima dell'esecuzione dell'applicazione, pertanto le scansioni di SAST sono molto più rapide di altre suite di test delle applicazioni.

- **Facile da automatizzare**: I file del codice sorgente possono essere analizzati automaticamente in qualsiasi punto dell'SDLC. Ciò significa che SAST può essere utilizzato come gateway di sicurezza in qualsiasi momento.

Secure Software Development Life Cicle (SSDLC)

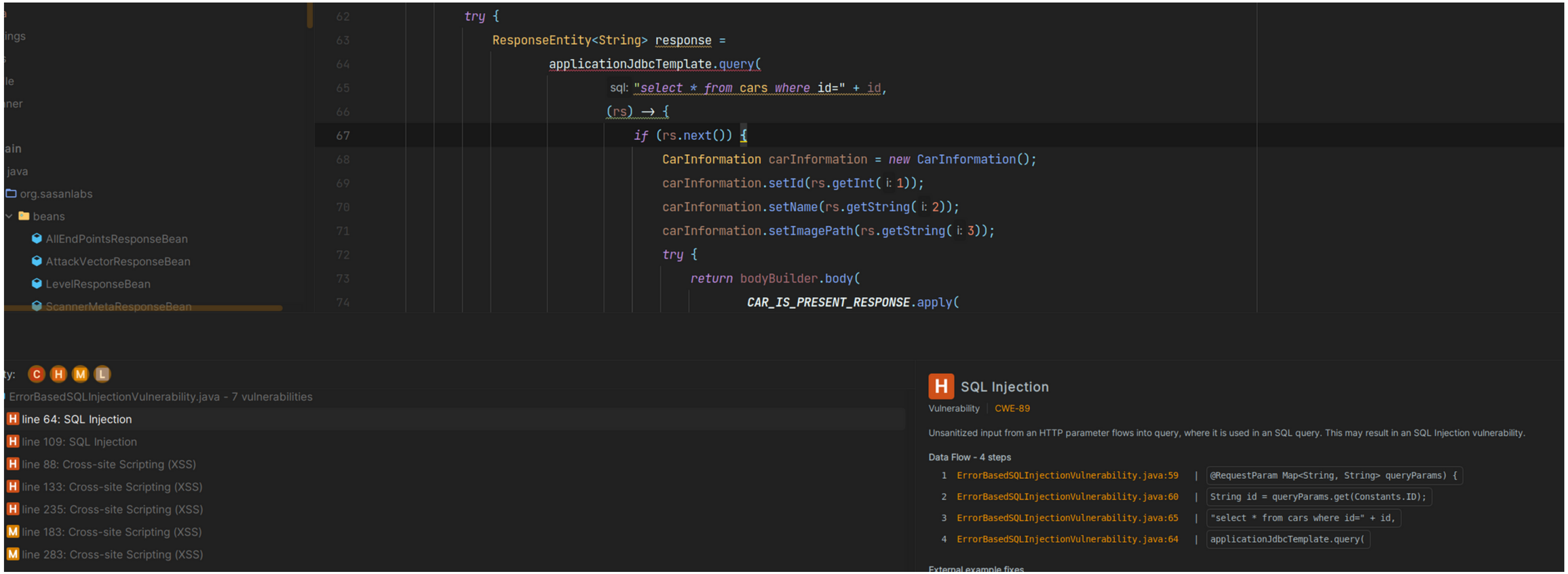
SAST - fasi

1. ***Esaminare*** il codice mentre viene scritto.
2. ***Dare priorità*** in base alla gravità e all'impatto delle vulnerabilità riscontrate.
3. ***Comprendere*** la natura delle vulnerabilità trovate esaminando i dati della scansione e valutando il livello di rischio associato.
4. ***Imparare*** dai risultati della scansione per prevenire vulnerabilità simili in futuro. Ciò include il miglioramento della qualità del codice, l'adozione di pratiche di codifica sicure e l'implementazione della formazione sulla sicurezza degli sviluppatori.
5. ***Risolvere*** le vulnerabilità riscontrate nella scansione, applicando una patch al codice o implementando altre misure di rimedio.
6. ***Eseguire*** nuovamente la scansione per verificare che la correzione abbia funzionato.
7. ***Continuare*** la codifica integrando la sicurezza nel processo di sviluppo per evitare che le vulnerabilità vengano introdotte nel codice futuro.

Secure Software Development Life Cicle (SSDLC)

SAST

Code Security Scanning



Secure Software Development Life Cicle (SSDLC)

SCA

La ***Software Composition Analysis*** (SCA) è una metodologia di sicurezza applicativa per la gestione dei componenti ***open source***. Utilizzando la SCA, i team di sviluppo possono tracciare e analizzare rapidamente qualsiasi componente open source inserito in un progetto. Gli strumenti SCA possono scoprire tutti i componenti correlati, le loro librerie di supporto e le loro dipendenze dirette e indirette. Gli strumenti SCA possono anche rilevare le licenze software, le dipendenze deprecate, nonché le vulnerabilità e i potenziali exploit.

Secure Software Development Life Cicle (SSDLC)

SCA

Il 90% delle organizzazioni si affida oggi all'open source per le proprie applicazioni.

L'open source è solo una parte delle moderne applicazioni. Oltre ai pacchetti open source, sono assemblate da codice proprietario, container e Infrastructure as a Code, per citare solo alcuni dei blocchi di costruzione utilizzati per la ***software supply chain***, tutti potenziali punti di **ingresso** per gli attori malintenzionati.

Una vulnerabilità sfruttata in una parte della catena può essere utilizzata per infettare l'intera applicazione, ampliando così la superficie di attacco che richiede protezione.

Octopus Scanner
L'attacco del malware Octopus Scanner è un attacco supply chain che ha preso di mira gli sviluppatori di applicazioni Java. Il malware è stato inserito in almeno 26 progetti collegati ad Apache NetBeans.
Gli attaccanti avevano compromesso i repo.
Quando uno sviluppatore scaricava un progetto Java infetto da GitHub, NetBeans IDE scaricava anche i file infetti. Questi file venivano quindi eseguiti quando lo sviluppatore eseguiva la build del progetto.

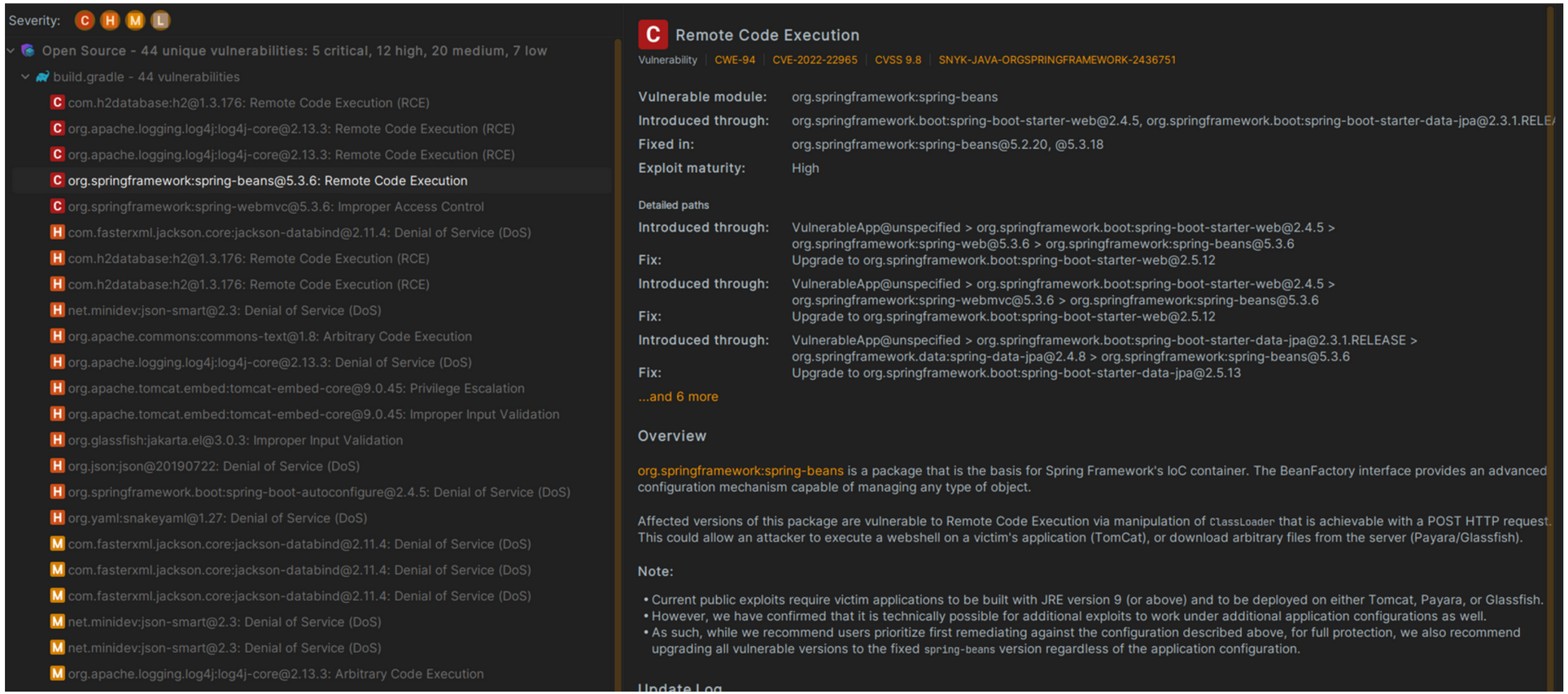
In questo modo, il malware Octopus Scanner veniva installato sul computer dello sviluppatore.

Gli autori dell'attacco sono sconosciuti e non si sa con precisazione se e/o quante librerie sono affette.

Secure Software Development Life Cicle (SSDLC)

SCA

Open Source Scanning



Secure Software Development Life Cicle (SSDLC)

Scanner utili

- https://owasp.org/www-community/Source_Code_Analysis_Tools
- https://snyk.io/product/snyk-code/
- https://www.sonarsource.com/products/sonarqube/downloads/

Secure Software Development Life Cicle (SSDLC)

Development - Secure coding guidelines

- Utilizzo di query paramettrizzate e read only per leggere i dati dal database e ridurre al minimo le possibilità che qualcuno possa appropriarsi di queste query per scopi illeciti (prevenire quindi qualsiasi tipo di data disclosure).
- Convalidare gli input dell'utente prima di elaborare i dati.
- Sanitizzazione di tutti i dati inviati all'utente dal database.
- Verificare la presenza di vulnerabilità nelle librerie open source prima di utilizzarle.

Secure Software Development Life Cicle (SSDLC)

Verification

La fase di verifica è quella in cui le applicazioni vengono sottoposte a un ciclo di test approfondito per garantire la conformità al progetto e ai requisiti originali. Qui si introducono anche i ***test di sicurezza automatizzati***. L'applicazione non viene distribuita se i test non vengono superati.

Questa fase spesso include strumenti automatizzati come le ***pipeline CI/CD*** per controllare la verifica e il rilascio.

La verifica in questa fase può includere

- Test automatizzati che esprimono i percorsi critici dell'applicazione
- Esecuzione automatica degli ***unit/integration/e2e/smoke/regression/performance etc... tests*** dell'applicazione che verificano la correttezza dell'applicazione sottostante.
- Strumenti di deployment automatizzati che scambiano dinamicamente i secrets dell'applicazione da utilizzare in un ambiente di produzione.

Secure Software Development Life Cicle (SSDLC)

Verification - test di sicurezza automatizzati

- ***Vulnerability test***: I test di vulnerabilità sono progettati per identificare le vulnerabilità di sicurezza in un'applicazione o un sistema. .
- ***Penetration test***: I test di penetrazione sono progettati per simulare un attacco reale da parte di un aggressore. Questi test sono eseguiti da esperti di sicurezza che utilizzano una varietà di tecniche per tentare di compromettere un'applicazione o un sistema.
- ***Stress test***: I test di stress sono progettati per verificare la capacità di un'applicazione o un sistema di resistere a carichi elevati. Questi test possono essere utilizzati per identificare le vulnerabilità che possono essere sfruttate in caso di attacco DDoS.

Secure Software Development Life Cicle (SSDLC)

KMS & Secrets Management

- **Hashicorp Vault:** https://www.hashicorp.com/products/vault
- **AWS Secrets Manager:** https://docs.aws.amazon.com/secretsmanager/latest/userguide/
- **Azure Key Vault:** https://docs.microsoft.com/en-us/azure/key-vault/
- **Google Cloud Secret Manager:** https://cloud.google.com/secret-manager/docs/
- **Kubernetes Secrets:** https://kubernetes.io/docs/concepts/configuration/secret/

Keys Rotation

La rotazione delle chiavi è il processo di sostituzione di una chiave crittografica con una nuova chiave. Questo processo viene eseguito regolarmente per prevenire attacchi informatici che potrebbero sfruttare la conoscenza della chiave corrente.
Quando una chiave viene compromessa, gli attaccanti possono utilizzare la chiave per decrittografare i dati che sono stati crittografati con quella chiave. La rotazione delle chiavi riduce il rischio che ciò accada.

Esistono due tipi principali di rotazione delle chiavi:

- **Rotazione manuale**: Questa è la rotazione delle chiavi eseguita manualmente dagli amministratori di sistema.
- **Rotazione automatica**: Questa è la rotazione delle chiavi eseguita automaticamente da un sistema.

Secure Software Development Life Cicle (SSDLC)

Maintenance and Evolution

Le vulnerabilità che sono ***sfuggite al controllo*** possono essere trovate nell'applicazione molto tempo dopo il suo rilascio, possono essere presenti nel codice scritto dagli sviluppatori o trovarsi nelle librerie di terze parti. Questo può portare ad avere degli ***zero-days***, ovvero di vulnerabilità precedentemente sconosciute.

Queste vulnerabilità devono poi ***essere corrette dal team di sviluppo***, un processo che in alcuni casi può richiedere una significativa riscrittura delle funzionalità dell'applicazione. Le vulnerabilità in questa fase possono provenire anche da altre fonti, come i ***penetration test*** esterni condotti da hacker etici o le segnalazioni attraverso programmi ***bug bounty***. La risoluzione di questi tipi di problemi di produzione deve essere pianificata e prevista nelle versioni future.

Secure Software Development Life Cicle (SSDLC)

Vantaggi

- Il SSDLC è un esempio dell'approccio ***shift-left***, che si riferisce all'integrazione dei controlli di sicurezza il più presto possibile nell'SDLC.
- Aiuta i team di sviluppo a pianificare correttamente i rilasci, rendendo più facile individuare e risolvere i problemi che potrebbero influire sulla tempistica del rilascio.
- Ha come fulcro il fatto che gli sforzi per la sicurezza siano guidati dal team di sviluppo stesso. In questo modo, i problemi possono essere risolti dagli esperti del settore che hanno scritto il software, anziché affidare a un altro team la correzione dei bug a posteriori.
- Nonostante SSDLC possa sembrare molto impegnativo e costoso da realizzare, oggi la maggior parte dei processi viene automatizzata.
- Richiede una collaborazione frequente tra DevOps e gli ingegneri che implementano le funzionalità dell'applicazione.
- Risolvendo questi problemi nelle prime fasi del processo, i team di sviluppo possono ridurre il costo di sviluppo e manutenzione delle applicazioni. Scoprire un problema a posteriore può avere un costo fino a 100 volte maggiore.

Race Conditions

Tipo comune di vulnerabilità strettamente correlato alle falle nella ***logica di business***. Si verificano quando i siti web elaborano le richieste in modo concorrente senza adeguate protezioni. Questo può portare a più thread distinti che interagiscono con gli stessi dati nello stesso momento, dando luogo a una ***collisione*** che causa un comportamento indesiderato nell'applicazione. Un attacco di tipo race condition utilizza richieste accuratamente temporizzate per causare collisioni intenzionali e sfruttare questo comportamento indesiderato per scopi malevoli.

Il periodo di tempo in cui è possibile una collisione si chiama ***race window***. Potrebbe trattarsi di una frazione di secondo tra due interazioni con il database, ad esempio.

Come altri difetti logici, l'impatto di una race conditions dipende dall'applicazione e dalla funzionalità specifica in cui si verifica.

Il tipo più noto di race conditions consente di superare una sorta di limite imposto dalla logica di business dell'applicazione.

Race Conditions

- Riscattare più volte una gift-card
- Valutazione di un prodotto più volte
- Prelevare o trasferire denaro contante in eccesso rispetto al saldo del vostro conto
- Bypassare un limite di velocità anti-brute-force.

I superamenti dei limiti sono un sottotipo delle falle "time-of-check to time-of-use" (TOCTOU).

Race Conditions

Mitigation

- Utilizzare un lock per proteggere l'accesso a una risorsa condivisa.
- Utilizzare una queque per garantire che i works vengano eseguiti una alla volta
- Utilizzare un semaphore per limitare il numero di thread che possono accedere a una risorsa condivisa.

Race Conditions

Pessimistic Read and Write

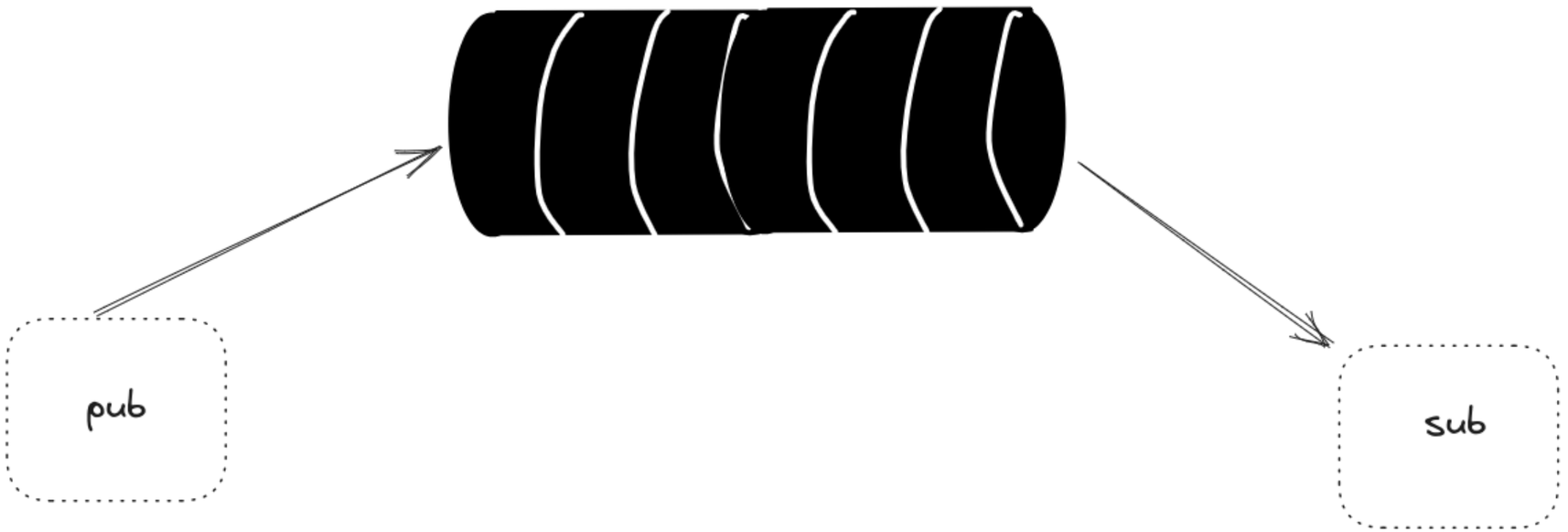
```
private final EntityManager entityManager;

@Autowired
public UserService(UserRepository userRepository, EntityManager entityManager) {
   this.userRepository = userRepository;
   this.entityManager = entityManager;
}

@Transactional
public int withdrawCredits(WithdrawDto params) {
   UserEntity user = userRepository.findOneById(params.getUserId());

   entityManager.lock(user, LockModeType.PESSIMISTIC_READ);

   if (user.getCredits() < params.getCredits()) {
      throw new ResponseStatusException(HttpStatus.CONFLICT, "Insufficient credits");
   }

   user.setCredits(user.getCredits() - params.getCredits());
   userRepository.save(user);

   entityManager.flush();

   return params.getCredits();
}
```

Race Conditions

Queue

Un altro modo per mitigare la race conditions potrebbe essere l’utilizzo delle code.
Si allocano in una coda di tipo FIFO i vari job e/o le info da elaborare e la si smaltisce uno ad uno.

Redis
Mephis
SQS
Kafka



Threat modeling

Il threat modeling è un processo di analisi delle minacce che possono essere utilizzate per compromettere un sistema o un'applicazione. Il processo di threat modeling viene utilizzato per identificare e mitigare le minacce, in modo da proteggere gli asset aziendali.

Ci sono moltissime minacce, quindi è importante utilizzare un metodo strutturato per l'identificazione di quelle più pertinenti al contesto, che può avvenire utilizzando uno o più approcci:

- ***Approccio incentrato sugli asset***: Questo metodo utilizza i risultati dell'asset valuation e tenta di identificare le minacce connesse agli asset più di valore per il contesto.

- ***Approccio incentrato sugli attacker***: Metodo che si basa sull'identificazione di potenziali attacker, individuando le minacce in base agli obiettivi che questi si possono prefiggere. Riconoscere ciò che gli attaccanti vogliono ottenere, permette di spostare il proprio focus sull'identificare e proteggere gli asset più rilevanti.

- ***Approccio incentrato sul software***: Se un'organizzazione sviluppa software, può prendere in considerazione potenziali minacce che potrebbero colpire il software stesso.

Threat modeling

Ecco i passaggi generali per eseguire il threat modeling:

1. ***Identificare gli asset*** Il primo passo del threat modeling consiste nell'identificare gli asset che devono essere protetti. Gli asset possono essere fisici, come un edificio o un server, o digitali, come dati o applicazioni.
2. ***Identificare le minacce*** Una volta identificati gli asset, è necessario identificare le minacce che potrebbero essere utilizzate per comprometterli. Le minacce possono essere interne, come un dipendente malintenzionato, o esterne, come un utente malintenzionato.
3. ***Valutare le minacce*** Dopo aver identificato le minacce, è necessario valutarle per determinare la loro gravità e probabilità. La gravità di una minaccia è la potenziale perdita o danno che può causare. La probabilità di una minaccia è la possibilità che si verifichi.
4. ***Implementare le contromisure*** Una volta valutate le minacce, è necessario implementare le contromisure per mitigarle. Le contromisure possono essere tecniche, come l'installazione di un firewall, o procedurali, come l'aggiornamento regolare del software.

Threat modeling

Metodologie

STRIDE sviluppato da Microsoft, ma è stato rilasciato al pubblico in modo che chiunque possa utilizzarlo.

- **Spoofing** falsificazione dell’identità
- **Tampering** qualsiasi azione che comporti modifiche o manipolazioni non autorizzate dei dati
- **Repudiation** la capacità di un utente o di un attaccante, di negare di aver eseguito una certa azione o attività
- **Information** disclosure ivelazione o distribuzione di informazioni private, riservate o controllate, a entità esterne o non autorizzate
- **Denial of service** attacco che tenta di impedire l’uso autorizzato di una risorsa
- **Elevation of privilege** un attacco in cui un account utente limitato, ottiene maggiori privilegi

STRIDE identifica le minacce in base a queste sei categorie. Ogni categoria rappresenta un tipo diverso di minaccia che può essere utilizzata per compromettere un sistema o un'applicazione.

Threat modeling

Metodologie

PASTA è flessibile, può essere utilizzato per analizzare le minacce a qualsiasi tipo di sistema o applicazione.
È composto da sette fasi:

1. **Preparazione** consiste nell'identificare gli obiettivi dell'analisi delle minacce e della simulazione di attacchi. È inoltre necessario identificare le risorse che saranno utilizzate per l'analisi e la simulazione.
2. **Definizione degli asset** consiste nell'identificare gli asset che devono essere protetti. Gli asset possono essere fisici, come un edificio o un server, o digitali, come dati o applicazioni.
3. **Identificazione delle minacce** consiste nell'identificare le minacce che potrebbero essere utilizzate per compromettere gli asset. Le minacce possono essere interne, come un dipendente malintenzionato, o esterne, come un utente malintenzionato.
4. **Valutazione delle minacce** consiste nel valutare le minacce per determinare la loro gravità e probabilità. La gravità di una minaccia è il potenziale danno che può causare. La probabilità di una minaccia è la possibilità che si verifichi.
5. **Simulazione degli attacchi** La fase di simulazione degli attacchi consiste nel simulare gli attacchi che potrebbero essere utilizzati per compromettere gli asset. La simulazione degli attacchi può essere utilizzata per testare le contromisure e per identificare eventuali vulnerabilità.
6. **Analisi dei risultati** consiste nell'analizzare i risultati della simulazione degli attacchi per identificare eventuali vulnerabilità e per valutare l'efficacia delle contromisure.
7. **Documentazione e presentazione dei risultati** consiste nel documentare i risultati dell'analisi delle minacce e della simulazione di attacchi.

Threat modeling

Metodologie

VAST è l’acronimo di Visual, Agile e Simple Threat. Metodologia di threat modeling basata sui principi di project management e programming Agile. L’obiettivo è quello di integrare threat e risk management all’interno di programmi di sviluppo Agile.

Alla base di questa metodologia c’è l’idea che la modellazione delle minacce è utile solo se comprende l’intero ciclo di vita di sviluppo del software (SDLC), incorporando tre diversi pilastri quali l’automazione, l’integrazione, e la collaborazione.

Standard Iso

Gli standard ISO in sicurezza informatica sono un insieme di linee guida e requisiti che le organizzazioni possono adottare per migliorare la sicurezza delle proprie informazioni. Questi standard sono sviluppati dall'International Organization for Standardization (ISO), un'organizzazione non governativa che si occupa di stabilire standard internazionali.

ISO 27035
Rappresenta lo standard per gli aspetti di gestione degli incidenti informatici.
prevede le fasi di
- Pianificazione: vengono definite le policy per gestire gli incidenti e viene creato l'IRT (incident response team)
- Individuazione: vengono raccolte le informazioni e documentati gli ISE (information security event)
- Riposta: serve per contenere i dati ed eliminare l'ISI (information security incident)
- Apprendimento: segue la fase di risposta, si apprende la lezione appresa e viene valutato l'IRT

ISO 27004:
Questo standard fornisce linee guida per la misurazione e la valutazione della sicurezza delle informazioni.

ISO 27017
Questo standard fornisce linee guida per la sicurezza delle informazioni nei servizi cloud.

JWT libraries

Un JSON Web Token (JWT) è un token di sicurezza che può essere utilizzato per autenticare e autorizzare gli utenti.
È un formato di dati aperto e standard che può essere utilizzato da qualsiasi applicazione.

Un JWT è composto da tre parti:

- Header: contiene le informazioni sul tipo di token e l'algoritmo di firma utilizzato.
- Payload: contiene le informazioni sull'utente, come il nome utente, l'ID utente o le autorizzazioni.
- Signature: viene utilizzata per verificare l'integrità del token.

JWT libraries

https://jwt.io/libraries