| Vulnerabilità | | Descrizione | L'applicazione è vulnerabile se | Misure per prevenire la vulnerabilità |
|---|---|---|---|---|
| RW1 | **A1:2017 Injection** | Le Injection Flaws, come SQL , NoSQL, OS e LDAP Injection, si verificano quando dati non validati sono inviati come parte di un comando o di una query al loro interprete. Il dato infetto può quindi ingannare tale interprete, eseguendo comandi non previsti o accedendo a dati per i quali non si ha l'autorizzazione. | • I dati forniti dall'utente non sono convalidati, filtrati o sanitizzati dall' applicazione.<br>• Le query dinamiche o le chiamate non parametrizzate, senza context-aware escaping, vengono utilizzate direttamente nell'interprete.<br>• I dati dannosi vengono utilizzati nei parametri di ricerca di object-relational mapping (ORM) per estrarre ulteriori record sensibili.<br>• I dati dannosi vengono utilizzati direttamente o concatenati , in modo tale che SQL o un comando contengono sia la struttura che dati dannosi nella query dinamica, nei comandi o nelle stored procedure | Per prevenire l'iniezione è necessario mantenere separati i dati da comandi e query.<br>• L'opzione preferita è utilizzare delle API sicure, che evitino l'uso di un interprete o forniscano delle interfacce parametrizzate, o migrare per utilizzare gli ORM (Object Relational Mapping Tools).<br>Nota: anche se parametrizzate, le stored procedure possono ancora introdurre SQL injection se PL / SQL o T-SQL concatena query e dati oppure processa dati dannosi con EXECUTE IMMEDIATO o exec ().<br>• Utilizzare lato server la convalida positiva o una "whitelist" dell'input . Questa non è una difesa completa in quanto molte applicazioni richiedono caratteri speciali, così come aree di testo o API per applicazioni mobili.<br>• Per eventuali query dinamiche residue, evitare i caratteri speciali usando la sintassi di escape specifica per quell'interprete.<br>Nota: strutture SQL come nomi di tabella, nomi di colonna e così via non possono essere evitate, e quindi i nomi delle strutture forniti dall'utente sono pericolosi. Questo è un problema comune nei software di report-writing.<br>• Utilizzare LIMIT e altri controlli SQL all'interno delle query per impedire divulgazione di massa di record in caso di SQL Injection. |
| RW2 | **A2:2017 Broken Authentication** | Le procedure applicative relative all'autenticazione e alla gestione della sessione sono spesso implementate in modo non corretto, permettendo agli attaccanti di compromettere password, chiavi, token di sessione o sfruttare debolezze implementative per assumere l'identità di altri utenti, temporaneamente o permanentemente. | La conferma dell'identità dell'utente, l'autenticazione e la gestione della sessione sono fondamentali per la protezione verso attacchi relativi all'autenticazione.<br>Potrebbero esserci dei punti deboli nell'autenticazione se l'applicazione:<br>• Permette attacchi automatici come ad esempio credential stuffing, laddove l'attaccante abbia un elenco di nomi utente e password validi.<br>• Permette il brute force o altri attacchi automatici.<br>• Consente l'utilizzo di password predefinite, deboli o note, come ad esempio "Password1" o "admin / admin".<br>• Utilizza processi deboli o inefficaci di recupero delle credenziali e di password dimenticate come ad esempio "risposte basate sulla conoscenza", che non possono essere rese sicure.<br>• Utilizza password di testo semplice, crittografate o con hash debole (vedere A3:2017-Sensitive Data Exposure).<br>•Non ha un'autenticazione a più fattori o questa è inefficace.<br>• Espone gli ID sessione nell'URL (ad es. URL rewriting).<br>• Non ruota gli ID di sessione dopo un accesso avvenuto con successo.<br>• Non invalida correttamente gli ID sessione. Non vengono invalidati correttamente le sessioni utente o i token di autenticazione (in particolare token Single Sign-On (SSO)) durante il logout o un periodo di inattività. | Laddove possibile, implementare l'autenticazione a più fattori per prevenire il completamento automatico, credential stuffing, brute force e attacchi di riutilizzo delle credenziali rubate.<br>• Non inviare o deployare con credenziali di default, in particolare per gli utenti amministratori.<br>• Implementare controlli per le password deboli, così come ad esempio implementare test per le password nuove o modificate confrontandole con un elenco delle prime 10000 peggiori password.<br>• Allineare le politiche di lunghezza, complessità e rotazione della password con le Linee guida del NIST 800-63 B nella sezione 5.1.1.<br>• Garantire che la registrazione, il recupero delle credenziali e i pathways delle API siano hardenizzati contro gli attacchi di account enumeration usando gli stessi messaggi per tutti gli esiti.<br>• Limitare il numero di tentativi di accesso non riusciti o incrementare l'attesa tra i vari tentativi. Registrare tutti gli errori e avvisare gli amministratori in caso di credential stuffing, forza bruta o altri attacchi che vengono rilevati.<br>• Utilizzare, lato server, un gestore di sessioni integrato e sicuro che, dopo l'accesso, generi un nuovo ID di sessione random e con elevata entropia . l' ID di sessione non deve essere nell'URL e deve essere archiviato in modo sicuro e invalidato dopo il logout, l'inattività o il timeout. |
| RW3 | **A3:2017 Sensitive Data Expos** | Molte applicazioni web e API non proteggono adeguatamente i dati sensibili, come dati finanziari,<br>dati sanitari e informazioni personali. Gli attaccanti possono rubare o modificare tali dati debolmente protetti per ottenere frodi con carte di credito, furto di identità o altri reati. I dati sensibili possono essere compromessi se non hanno misure di protezione extra, come la crittografia per i dati memorizzati o in transito, e richiedono precauzioni speciali quando<br>scambiati via browser. | La prima cosa è determinare le esigenze di protezione dei dati in transito e memorizzati. Ad esempio password, numeri di carta di credito, cartelle cliniche, informazioni personali e segreti aziendali richiedono una protezione aggiuntiva, in particolare se tali dati rientrano nell'ambito dellle norme sulla Privacy, come il Regolamento generale sulla protezione dei dati dell'UE<br>(GDPR), o rientrano nell'ambito di regolamentazioni come ad es. il PCI Data Security Standard (PCI DSS) per la protezione dei dati finanziari.<br>Per tutti questi va verificato se:<br>• I dati sono trasmessi in chiaro. Ciò riguarda i protocolli come HTTP, SMTP e FTP. Il traffico Internet esterno è<br>particolarmente pericoloso. Verifica tutto il traffico interno, ad es. fra bilanciatori di carico, server Web o sistemi back-end.<br>• I dati sensibili sono archiviati in chiaro, inclusi i backup.<br>• Sono utilizzati algoritmi crittografici vecchi o deboli per default o nel vecchio codice.<br>• Sono in uso chiavi crittografiche di default, chiavi crittografiche deboli generate o riutilizzate oppure manca la corretta gestione o rotazione delle chiavi.<br>• La crittografia non viene rispettata, ad es. ci sono user agent (browser) che mancano di direttive di sicurezza o intestazioni.<br>•Lo user agent (ad es. App, client di posta) non verifica la validità del certificato ricevuto dal server. | Effettuare almeno le seguenti operazioni e consultare i riferimenti:<br>• Classificare i dati elaborati, archiviati o trasmessi da un' applicazione. Identificare quali dati sono sensibili in base alla normativa privacy, ai requisiti normativi o alle esigenze aziendali.<br>• Applicare i controlli secondo la classificazione.<br>• Non archiviare inutilmente dati sensibili; eliminarli non appena<br>possibile o utilizzare token conformi alle direttive PCI DSS oppure<br>troncarli per evitare di risalire al dato. I dati che non vengono conservati non possono essere rubati.<br>• Assicurati di crittografare tutti i dati sensibili memorizzati.<br>• Garantire algoritmi, protocolli standard aggiornati ed efficaci, chiavi robuste e un corretto meccanismo di gestione delle chiavi.<br>• Crittografare tutti i dati in transito con protocolli sicuri come TLS,<br>con perfect forward secrecy (PFS). Rafforzare la crittografia<br>utilizzando direttive come HTTP Strict Transport Security (HSTS).<br>• Disabilitare la memorizzazione nella cache per le risposte che contengono dati riservati.<br>• Archiviare le password utilizzando funzioni strong adaptive e salted hashing<br>con un fattore di lavoro (fattore di ritardo), come Argon2, scrypt, bcrypt o PBKDF2.<br>• Verificare in modo indipendente l'efficacia della configurazione e delle<br>impostazioni. |

| | | | | |
|---|---|---|---|---|
| RW4 | **A4:2017 External Entities (XX** | Molti processori XML meno recenti o mal configurati valutano i riferimenti ad entità esterne nei documenti XML. Le entità esterne possono essere utilizzate per divulgare file interni utilizzando il gestore URI di file, condivisioni interne di file, scansione delle porte interne, esecuzione di codice in modalità remota e attacchi denial of service. | Applicazioni e in particolare servizi XML-based web services o downstream integrations potrebbero essere vulnerabili agli attacchi se:<br>• L'applicazione accetta direttamente XML o XML uploads, soprattutto da fonti non attendibili o inserisce dati non attendibili in documenti XML, che vengono quindi elaborati da un processore XML.<br>• Qualsiasi processore XML nell'applicazione o i SOAP based web services hanno abilitate le Document Type Definitions (DTDs) .<br>Poiché il meccanismo esatto per disabilitare l'elaborazione dei DTDs dipende dal processore, è buona norma consultare un riferimento come ad esempio il foglio informativo OWASP "XXE Prevention".<br>•L'applicazione utilizza SAML per l'elaborazione delle identità per lafederated security o per il single sign on (SSO). SAML utilizza XML per le identity assertions (processo di scambio di dati di autenticazione e autorizzazione tra domini di sicurezza distinti) e può essere vulnerabile.<br>• L'applicazione utilizza versioni di SOAP precedenti alla versione 1.2. In questo caso è probabile che l'applicazione sia suscettibile agli attacchi di tipo XXE se le entità XML vengono passate al framework SOAP.<br>• Essere vulnerabili agli attacchi di tipo XXE significa probabilmente che l'applicazione è vulnerabile agli attacchi di tipo Denial of Service come ad esempio il Billion Laughs attack. | La formazione degli sviluppatori è essenziale per riuscire ad identificare e mitigare la XXE.<br>Oltre a ciò, per prevenire la XXE è necessario:<br>• Quando possibile, utilizzare formati di dati meno complessi come JSON, evitando la serializzazione di dati sensibili.<br>• Patchare o aggiornare tutti i processori e le librerie XML utilizzati dall'applicazione o dal sistema operativo sottostante. Usare dependency checkers. Aggiornare SOAP alla verisione 1.2 o successiva.<br>• Disabilitare l'entità esterna XML e l'elaborazione DTD in tutti gli XML parser nell'applicazione, come indicato nel foglio informativo di OWASP "XXE prevenzione".<br>• Implementare la convalida del positive ("whitelisting") server-side input ,il filtraggio o la sanitizzazione per impedire dati dannosi all'interno di documenti, intestazioni o nodi XML.<br>• Verificare che la funzionalità di caricamento dei file XML o XSL validi questi tipi di file in arrivo utilizzando la convalida XSD o similare.<br>•I tools SAST possono aiutare a rilevare XXE nel codice sorgente, sebbene la revisione manuale del codice è la migliore alternativa nelle applicazioni grandi e complesse con molte integrazioni.<br>Se questi controlli non sono possibili, considerare l'utilizzo di patching virtuale, gateway di sicurezza API o Web Application Firewalls (WAF) per rilevare, monitorare e bloccare gli attacchi XXE. |
| RW5 | **A5:2017 Broken Access Contr** | Le restrizioni su ciò che gli utenti autenticati sono autorizzati a fare, spesso non vengono applicate correttamente. Gli attaccanti possono sfruttare questi difetti per accedere a funzionalità e / o dati non autorizzati, come accedere agli account di altri utenti, visualizzare file sensibili, modificare dati di altri utenti, modificare diritti di accesso, ecc. | Il controllo degli accessi fa si che gli utenti non possano agire al di fuori delle autorizzazioni previste. Falle nel controllo accessi possono implicare divulgazione, modifica o distruzione di informazioni non autorizzate di tutti i dati, oppure consentire una business function al di fuori del ruolo dell'utente. Le vulnerabilità comuni nel controllo degli accessi includono:<br>• Bypass delle verifiche di controllo accessi modificando l'URL, lo stato interno dell'applicazione o la pagina HTML o semplicemente usando un tool di attacco API personalizzato.<br>• Consentire la modifica della chiave primaria nel record di un altro utente, consentendo di visualizzare o modificare l'account di qualcun altro.<br>• Elevazione del privilegio. Agire come utente senza aver effettuato l'accesso o agire da amministratore quando si accede come utente.<br>• Manipolazione dei metadati, come ad esempio la riproduzione o la manomissione con un token di controllo accesso di tipo JSON Web Token (JWT) o un cookie o un campo nascosto manipolato per elevare i privilegi o abusare della JWT invalidation.<br>• La configurazione errata di CORS consente l'accesso alle API non autorizzato.<br>• Forzare la navigazione verso pagine autenticate come utente non autenticato o a pagine privilegiate come utente standard. Accesso alle API senza controlli di accesso per POST, PUT e DELETE. | Il controllo dell'accesso è efficace solo se applicato nel codice attendibile server-side o nelle server-less API, dove l'attaccante non può modificare il file di controllo accessi o i metadati.<br>• Ad eccezione delle risorse pubbliche, negare l'accesso per default.<br>• Implementare i meccanismi di controllo accessi una volta e riutilizzarli in tutta l'applicazione, riducendo al minimo l'utilizzo di CORS.<br>• Il modello di controllo accessi dovrebbe imporre la proprietà dei record, piuttosto che accettare che l'utente possa creare, leggere, aggiornare o eliminare qualsiasi record.<br>• I modelli di dominio dovrebbero prevedere requisiti univoci per gli application business limit.<br>• Disabilitare l'elenco delle directory del server Web e garantire che i metadati del file (ad es. .git) e i file di backup non siano presenti nelle web root.<br>• Effettuare il log degli errori del sistema di controllo degli accessi e avvisare gli amministratori quando opportuno (ad esempio guasti ripetuti).<br>• Stabilire un rate limit API e un controller access per ridurre al minimo il danno derivante da tool di attacco automatizzati.<br>• I token JWT devono essere invalidati sul server dopo il logout.<br>Gli sviluppatori e il personale addetto al controllo qualità dovrebbero includere un'unità funzionale di controllo accessi e test di integrazione. |
| RW6 | **A6:2017 Security Misconfiguration** | La configurazione errata della sicurezza è il problema più frequente. Questo è comunemente il risultato di configurazioni predefinite che risultano insicure, configurazioni incomplete o ad hoc, open storage cloud, configurazione errata degli header HTTP e messaggi di errore dettagliati contenenti informazioni riservate. Tutti i sistemi operativi, framework, librerie e applicazioni non solo devono essere configurati in modo sicuro, ma devono essere aggiornati e patchati in modo tempestivo. | • Manca un adeguato hardening di sicurezza in qualsiasi parte dell'application stack oppure ci sono permessi configurati in modo errato nei servizi cloud.<br>• Le funzionalità non necessarie sono abilitate o installate (ad es. porte, servizi, pagine, account o privilegi non necessari).<br>• Gli account di default sono ancora abilitati e le loro password non sono state ancora cambiate.<br>• Il sistema di gestione dei messaggi di errore rende evidenti “stack trace” o altre informazioni nei messaggi di errore non necessarie all’utenza.<br>• Per i sistemi aggiornati, le ultime funzionalità di sicurezza sono disabilitate o non configurate in modo corretto.<br>• Le impostazioni di sicurezza negli application server, negli application framework (ad es. Struts, Spring, ASP.NET), nelle librerie, nei database, ecc. non sono impostate su valori sicuri.<br>• Il server non invia security headers o direttive di sicurezza o non sono impostati su valori sicuri.<br>• Il software è out of date o è vulnerabile (vedere A9: 2017-Using Components with Known Vulnerabilities).<br>Senza coerenti e ripetibili processi di configurazione della sicurezza<br>delle applicazioni, i sistemi sono a rischio elevato. | È necessario implementare processi di installazione sicuri, tra cui:<br>• Un processo di hardening ripetibile che renda più veloce e facile realizzare un ambiente opportunamente protetto. Sviluppo, QA e ambiente di produzione devono essere configurati nello stesso modo (con l’uso di password diverse in ogni ambiente). Questo processo dovrebbe essere automatizzato per minimizzare l’impegno richiesto per la realizzazione di un ambiente sicuro.<br>• Una piattaforma minimale senza funzionalità, componenti, documentazione ed esempi non necessari. Rimuovere o non installare funzionalità e framework inutilizzati.<br>• Un task per rivedere e aggiornare le configurazioni appropriate per tutte le note sulla sicurezza, gli aggiornamenti e le patch come parte del processo di patch management (vedi A9: 2017-Utilizzo dei componenti con Vulnerabilità note). In particolare, rivedere le autorizzazioni per il cloud storage (ad es. S3 bucket permissions).<br>• Un'architettura applicativa segmentata che fornisca una reale e sicura separazione tra i diversi componenti o tra i diversi tenant, con segmentazione, contenierizzazione o gruppi di sicurezza cloud.<br>• Invio di direttive di sicurezza ai clients, ad es. Security Headers.<br>• Un processo automatizzato per verificare l'efficacia di configurazioni e impostazioni in tutti gli ambienti. |

| | | | | |
|---|---|---|---|---|
| RW7 | **A7:2017 Cross-Site Scripting (XSS)** | Le falle di tipo XSS si verificano quando un'applicazione web include dati non affidabili in una nuova pagina web senza una opportuna validazione e/o "escaping", o quando l'applicazione aggiorna una pagina web esistente che contiene dati forniti dall'utente mediante un browser API che può creare HTML o JavaScript. Il XSS permette agli attaccanti di eseguire degli script malevoli sui browser delle vittime; tali script possono dirottare la sessione dell'utente, defacciare il sito web o re-indirizzare l'utente su un sito malevolo. | Esistono tre forme di XSS, mirate ai browser degli utenti:<br>• XSS riflesso: l'applicazione o l'API includono input non convalidati e sviste dell'utente come parte dell'output HTML. Un attacco riuscito può permettere all'attaccante di eseguire HTML e JavaScript arbitrari nel browser della vittima. In genere l'utente dovrà interagire con alcuni link dannosi che rimandano a una pagina controllata dagli attaccanti, ad esempio siti Web dannosi, annunci pubblicitari o simili.<br>• XSS memorizzato: l'applicazione o l'API memorizza input non sanitizzato dell'utente che viene visualizzato successivamente da un altro utente o da un amministratore. L'XSS memorizzato è spesso considerato rischio alto o critico.<br>• DOM XSS: framework JavaScript, applicazioni single-page ed API che dinamicamente includono dati controllabili dagli attaccanti in una pagina sono vulnerabili a DOM XSS. Idealmente, l'applicazione non dovrebbe inviare dati che potrebbero essere controllati dagli attaccanti ad API JavaScript non sicure.<br>Gli attacchi XSS tipici includono il furto di sessioni, l'acquisizione di account, bypass MFA, replacement o defacement del nodo DOM (come pannelli di login trojan), attacchi contro il browser dell'utente mediante download di software dannoso, registrazione delle chiavi e altri attacchi lato client. | Prevenire XSS richiede la separazione dei dati non attendibili dal contenuto del browser attivo. Ciò può essere ottenuto mediante:<br>• Utilizzo di framework che sfuggono automaticamente XSS grazie alla progettazione, come l'ultimo Ruby on Rails, React JS. Conoscere i limiti della protezione XSS di ciascun framework e gestire in modo appropriato i casi non coperti.<br>•Un appropriato escaping del contesto HTML dove i dati non affidabili sono contenuti (body, attribute, JavaScript, CSS, or URL) risolverà le vulnerabilità Reflected and Stored XSS. Per maggiori informazioni sulle tecniche di escaping fare riferimento al foglio informativo 'OWASP XSS Prevention'.<br>•Applicare una codifica context-sensitive quando si modifica un documento nel browser, lato client, contrasta DOM XSS. Quando ciò non può essere evitato, è possibile applicare simili tecniche di escape context-sensitive alle API del browse,r come descritto nel foglio informativo OWASP "Prevenzione XSS basata sul DOM".<br>• Abilitare una Content Security Policy (CSP) è un controllo di "difesa in profondità" o Defense-in-Depth contro XSS. È efficace se non esistono altre vulnerabilità che consentano l'inserimento di codice dannoso tramite inclusioni di file locali. |
| RW8 | **A8:2017 Insecure Deserialization** | La deserializzazione non sicura spesso porta all'esecuzione di codice in modalità remota. Anche se i difetti di deserializzazione non comportano l'esecuzione di codice in modalità remota, possono essere utilizzati per eseguire attacchi, inclusi replay attacks, injection attacks e attacchi di escalation di privilegi. | Le applicazioni e le API saranno vulnerabili se deserializzano oggetti dannosi o manomessi forniti da un attaccante.<br>Ciò può comportare due tipi principali di attacchi:<br>• Attacchi correlati a oggetti e strutture di dati in cui l'attaccante modifica la logica dell'applicazione o ottiene l'esecuzione di un codice remoto arbitrario.<br>• Tipici attacchi di manomissione dei dati, come attacchi relativi al controllo degli accessi, in cui vengono utilizzate strutture di dati esistenti ma il cui contenuto è cambiato.<br>La serializzazione può essere utilizzata in applicazioni per:<br>• Comunicazione remota e tra processi (RPC / IPC)<br>• Wire protocol, servizi web, message brokers<br>• Caching/Persistence<br>• Databases, cache servers, file systems<br>• HTTP cookies, HTML form parameters, API authentication tokens | L'unico modello architetturale sicuro è non accettare oggetti serializzati da fonti non attendibili o utilizzare mezzi di serializzazione che accettano solo tipi di dati primitivi.<br>Se ciò non è possibile, prendere in considerazione uno dei seguenti elementi:<br>• Implementare controlli di integrità come firme digitali su qualsiasi oggetto serializzato per impedire la creazione di oggetti dannosi o manomissione di dati.<br>• Applicare vincoli rigorosi durante la deserializzazione prima della creazione dell'oggetto, dato che il codice prevede in genere un set definibile di classi. Questa tecnica é stata Bypassata, quindi non é consigliabile affidarsi esclusivamente ad essa.<br>• Quando possibile isolare ed eseguire di codice che diserializza in ambienti con privilegi bassi.<br>•logging delle eccezioni e degli errori di deserializzazione, ad esempio come quando il tipo di oggetto ??? in arrivo non è del tipo previsto, o quando la deserializzazione genera eccezioni.<br>• Limitazione o monitoraggio della connettività di rete in entrata e in uscitada containers o server che deserializzano.<br>• Monitoraggio della deserializzazione, avvisando se un utente deserializza costantemente. |
| RW9 | **A9:2017 Using components with Known Vulnerabilities** | Le componenti di un'applicazione come librerie, framework e altri moduli software, funzionano con gli stessi privilegi dell'applicazione. Se viene sfruttata una componente vulnerabile, tale attacco può facilitare una grave perdita di dati o l'acquisizione del server. Applicazioni e API che utilizzano componenti con vulnerabilità note possono minare le loro difese e consentire vari attacchi e impatti. | • Se non si conoscono le versioni di tutti i componenti utilizzati (sia lato client che lato server). Questo include sia componenti usate in modo diretto sia componenti nidificate.<br>• Se il software è vulnerabile, non supportato o non aggiornato. Questo riguarda il sistema operativo, il web /application server, il database management system (DBMS), le applicazioni,le API e tutti i componenti, gli ambienti di runtime e le librerie.<br>• Se non si esegue una scansione periodica di vulnerabilità e se non si sottoscrive la ricezione di bollettini sulla sicurezza relativi ai componenti utilizzati.<br>• Se non si corregge o non si aggiorna la piattaforma sottostante, i frameworks e le dipendenze in modo tempestivo e basato sul rischio.<br>• Se gli sviluppatori software non testano la compatibilità delle librerie aggiornate o patchate.<br>• Se non si proteggono le configurazioni dei componenti (vedi A6: 2017 - Security Misconfiguration). | Dovrebbe esserci un processo di gestione delle patch in atto per:<br>• Rimuovere dipendenze non utilizzate, funzionalità non necessarie, componenti, file e documentazione.<br>• Inventario continuo delle versioni dei componenti sia lato client che lato server (ad esempio framework, librerie) e loro dipendenze utilizzando strumenti come versioni, DependencyCheck, retire.js, ecc. Monitorare continuamente fonti come CVE e NVD per le vulnerabilità nei componenti. Usare tools di analisi della composizione del software per automatizzare il processo. Configurare e-mail per alert di vulnerabilità di sicurezza relative ai componenti usati.<br>• Ottenere componenti solo da fonti ufficiali tramite collegamenti sicuri.<br>Preferire i pacchetti firmati per ridurre la possibilità di includere un componente modificato e dannoso.<br>• Monitorare le librerie ed i componenti non manutenuti o non creare patch di sicurezza per le versioni precedenti. Se il patching è impossibile, prendere in considerazione la distribuzione di una patch virtuale per monitorare, rilevare o proteggere dal problema rilevato.<br>Ogni organizzazione deve garantire l'esistenza in corso di un piano per il monitoraggio, il triaging e l'applicazione di aggiornamenti o change di configurazione per la durata dell'applicazione o del portafoglio. |

| | | | | |
|---|---|---|---|---|
| RW10 | **A10:2017 Insufficient Logging & Monitoring** | Insufficient logging and monitoring, coupled with missing or ineffective integration with incident<br>response, allows attackers to further attack systems, maintain persistence, pivot to more systems,<br>and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over<br>200 days, typically detected by external parties rather than internal processes or monitoring.<br><br>Logging e monitoraggio insufficienti, associati ad una mancante o inefficace integrazione con l'incident response, consentono agli attaccanti di aggredire ulteriormente i sistemi, mantenere la persistenza, ruotare su più sistemi, e manomettere, estrarre o distruggere i dati. La maggior parte degli studi sulle violazioni mostra che il tempo per rilevare una violazione è di oltre 200 giorni e in genere è rilevata da soggetti esterni anziché da processi interni o dal monitoraggio. | Logging, detection, monitoraggio e risposta attiva insufficienti<br>si verificano ogni volta che:<br>• Eventi verificabili, come accessi, accessi non riusciti e transazioni di valore elevato non sono registrate.<br>• Warnings ed errori non generano log, oppure generano messaggi di log inadeguati o poco chiari.<br>• I log delle applicazioni e delle API non sono monitorati per attività sospette.<br>• I log sono memorizzati solo localmente.<br>• Non sono in atto Soglie di allarme appropriate e processi di escalation di risposta.<br>• Test di penetrazione e scansioni con strumenti DAST (come OWASP ZAP) non generano alert.<br>• L'applicazione non è in grado di rilevare,escalare o avvisare in caso di attacchi in tempo reale o in tempo quasi reale.<br>Sei vulnerabile alla perdita di informazioni se il logging e l' alerting sono visibili a un utente o ad un attaccante (vedere A3: 2017-Esposizione di informazioni sensibili). | A seconda del rischio dei dati archiviati o elaborati dall'applicazione:<br>• Garantire che tutti gli accessi, gli errori di controllo degli accessi e gli errori di validazione degli input lato server possano essere loggati con un contesto utente sufficiente ad identificare account sospetti o malevoli e possano essere conservati per un tempo sufficiente per consentire una successiva analisi forense.<br>• Assicurare che i log vengano generati in un formato che può essere compatibile con la soluzione di gestione dei log centralizzata.<br>• Garantire che le transazioni di alto valore abbiano un audit trail con controlli di integrità per impedire la manomissione o la cancellazione, come ad esempio tabelle di database append-only o simili.<br>• Implementare un monitoraggio e un alerting efficaci in modo tale che le attività sospette vengano rilevate e risposte in modo tempestivo.<br>• Stabilire o adottare un incident response and recovery plan, come ad esempio quello previsto nel NIST 800-61 rev 2 o successivo.<br>Esistono framework commerciali e open source per la protezione di applicazioni, come OWASP AppSensor, web application firewalls, come ModSecurity con OWASP ModSecurity Core Rule Set, e software di correlazione dei log con dashboard e alerting personalizzati. |