Application Security Verification Standard 4.0.3

Final

Ottobre 2021

Table of Contents

Frontespizio

Riguardo allo standard

La Application Security Verification Standard è una lista di requisiti o test per la sicurezza delle applicazioni che può essere utilizzata da architetti del software, sviluppatori, tester, security professionals, tool vendors, e utenti per definire, realizzare, testare e verificare la sicurezza delle applicazioni.

Copyright e Licenza

Version 4.0.3, October 2021

Leader del progetto

Andrew van der Stock	Daniel Cuthbert	Jim Manico
Josh C Grossman	Elar Lang	

Principali Contributori

Abhay Bhargav	Benedikt Bauer	Osama Elnaggar
Ralph Andalis	Ron Perris	Sjoerd Langkemper
Tonimir Kisasondi		

Altri Contributori e Revisori

Aaron Guzman	Alina Vasiljeva	Andreas Kurtz	Anthony Weems	Barbara Schachner
Christian Heinrich	Christopher Loessl	Clément Notin	Dan Cornell	Daniël Geerts
David Clarke	David Johansson	David Quisenberry	Elie Saad	Erlend Oftedal
Fatih Ersinadim	Filip van Laenen	Geoff Baskwill	Glenn ten Cate	Grant Ongers
hello7s	Isaac Lewis	Jacob Salassi	James Sulinski	Jason Axley
Jason Morrow	Javier Dominguez	Jet Anderson	jeurgen	Jim Newman
Jonathan Schnittger	Joseph Kerby	Kelby Ludwig	Lars Haulin	Lewis Ardern
Liam Smit	lyz-code	Marc Aubry	Marco Schnüriger	Mark Burnett
Philippe De Ryck	Ravi Balla	Rick Mitchell	Riotaro Okada	Robin Wood
Rogan Dawes	Ryan Goltry	Sajjad Pourali	Serg Belkommen	Siim Puustusmaa
Ståle Pettersen	Stuart Gunter	Tal Argoni	Tim Hemel	Tomasz Wrobel
Vincent De Schutter	Mike Jang	Riccardo Sirigu		

Se non vi ritrovate nella lista dei contributori relativa alla versione 4.0.3 qui sopra, si prega di aprire un ticket su GitHub per essere riconosciuti negli aggiornamenti futuri.

La Application Security Verification Standard si basa sul lavoro svolto da coloro che hanno partecipato dalla verisone ASVS 1.0 del 2008 fino alla versione 3.0 del 2016. Gran parte della struttura e degli elementi di verifica che sono ancora presenti nell'ASVS attuale, sono stati originariamente scritti da Mike Boberski, Jeff Williams e Dave Wichers, ma ci sono molti altri contributori. Grazie a tutti coloro che hanno partecipato in precedenza. Per un elenco completo di tutti coloro che hanno contribuito alle versioni precedenti, si prega di consultare ciascuna versione precedente.

Prefazione

Benvenuti alla versione 4.0 dello Standard di Verifica della Sicurezza delle Applicazioni (ASVS). L'ASVS è un'iniziativa guidata dalla comunità che mira a definire un framework di requisiti e controlli di sicurezza, concentrandosi sulle misure funzionali e non funzionali necessarie per la progettazione, lo sviluppo e il test di moderne applicazioni e servizi web.

La versione 4.0.3 rappresenta la terza revisione minore della versione 4.0, volta a correggere errori ortografici e a chiarire i requisiti, senza apportare modifiche sostanziali, come l'aggiornamento dei requisiti o l'introduzione di nuovi. Tuttavia, abbiamo leggermente allentato alcuni requisiti dove appropriato e rimosso quelli ormai superflui, mantenendo invariata la numerazione.

L'ASVS 4.0 è il risultato di un impegno collaborativo e dei feedback raccolti dal settore negli ultimi dieci anni. Abbiamo cercato di semplificarne l'adozione per una vasta gamma di casi d'uso, coprendo l'intero ciclo di vita dello sviluppo di software sicuro.

Siamo consapevoli che probabilmente non ci sarà mai un consenso unanime sul contenuto di uno standard per applicazioni web, inclusa l'ASVS. L'analisi del rischio è intrinsecamente soggettiva, il che rende complessa la creazione di uno standard universale. Tuttavia, speriamo che gli aggiornamenti introdotti in questa versione rappresentino un passo avanti nella direzione giusta, rafforzando i concetti chiave di questo importante standard industriale.

Cosa c'è di nuovo nella versione 4.0

Il cambiamento più rilevante in questa versione è l'adozione delle Linee Guida per l'Identità Digitale del NIST 800-63-3, che introduce controlli di autenticazione moderni, avanzati e basati su evidenze. Pur consapevoli che ci possa essere una certa resistenza nell'adeguarsi a uno standard di autenticazione così avanzato, riteniamo fondamentale che gli standard siano allineati, specialmente quando si basano su evidenze e sono supportati da un altro autorevole standard di sicurezza delle applicazioni.

Gli standard di sicurezza delle informazioni dovrebbero puntare a ridurre il numero di requisiti unici, per evitare che le organizzazioni, nel percorso verso la conformità, debbano scegliere tra controlli concorrenti o incompatibili. La OWASP Top 10 del 2017 e ora lo Standard di Verifica della Sicurezza delle Applicazioni OWASP sono allineati al NIST 800-63 per quanto riguarda l'autenticazione e la gestione delle sessioni. Invitiamo altri enti normativi a collaborare con noi, con il NIST e con altri, al fine di sviluppare un insieme di controlli di sicurezza delle applicazioni ampiamente accettato, che massimizzi la sicurezza e riduca al minimo i costi di conformità.

L'ASVS 4.0 è stato completamente rinumerato dall'inizio alla fine. Questo nuovo schema di numerazione ci ha consentito di colmare i vuoti lasciati da capitoli ormai scomparsi e di suddividere i capitoli più estesi, facilitando l'implementazione dei controlli per sviluppatori e team. Ad esempio, se un'applicazione non utilizza JWT, l'intera sezione relativa alla gestione delle sessioni tramite JWT non sarà applicabile.

Una delle principali novità della versione 4.0 è l'introduzione di una mappatura completa con le Common Weakness Enumeration (CWE), una delle funzionalità più richieste nell'ultimo decennio. Questa mappatura permette a produttori di strumenti e utilizzatori di software di gestione delle vulnerabilità di correlare i risultati di altri strumenti e versioni precedenti del ASVS con la versione 4.0 e successive. Per fare spazio alla mappatura CWE, abbiamo eliminato la colonna "Since", che, a seguito della rinumerazione totale, aveva perso rilevanza rispetto alle versioni precedenti del ASVS. Non tutti i controlli del ASVS hanno una CWE associata e, poiché i CWE presentano spesso duplicazioni, abbiamo scelto di utilizzare le corrispondenze più comuni, piuttosto che quelle strettamente più precise. Tuttavia, non sempre è possibile trovare una corrispondenza tra i controlli di verifica e le debolezze. Continuiamo a incoraggiare il dibattito con la comunità CWE e, più in generale, con il settore della sicurezza delle informazioni per affrontare questa sfida.

Abbiamo lavorato per soddisfare e superare in modo completo i requisiti necessari ad affrontare la OWASP Top 10 del 2017 e i Controlli Proattivi OWASP del 2018. Poiché la OWASP Top 10 del 2017 rappresenta il livello minimo per evitare negligenze in ambito sicurezza, abbiamo classificato come controlli di Livello 1 tutti i requisiti della Top 10, ad eccezione di quelli specifici al logging. Questo approccio facilita l'adozione di un vero standard di sicurezza per coloro che già utilizzano la OWASP Top 10.

Ci siamo inoltre impegnati a rendere l'ASVS 4.0 Livello 1 un insieme di controlli completo e più avanzato rispetto alla Sezione 6.5 dello standard PCI DSS 3.2.1, che copre aspetti come progettazione, sviluppo, test, revisioni sicure del codice e penetration test per le applicazioni. Per raggiungere questo obiettivo, abbiamo esteso i controlli di Livello 1 includendo la rilevazione di buffer overflow, operazioni di memoria non sicure (V5) e flag di compilazione associati a operazioni rischiose sulla memoria (V14), oltre ai già consolidati requisiti per la verifica di applicazioni e servizi web.

Abbiamo completato la transizione del ASVS da controlli monolitici per soli server ad un sistema che fornisce sicurezza per tutte le applicazioni e le API moderne. Nell'era della programmazione funzionale, delle API serverless, del mobile, del cloud, dei container, dell'integrazione continua e consegna continua (CI/CD), del DevSecOps, della federazione e di molte altre tecnologie, non possiamo più trascurare le architetture applicative moderne. Le applicazioni moderne sono progettate in modo molto diverso da quelle in voga al lancio del ASVS originale nel 2009. L'ASVS deve sempre guardare al futuro per offrire raccomandazioni solide al suo pubblico principale: gli sviluppatori. Per questo motivo, abbiamo chiarito o eliminato tutti i requisiti che presuppongono che le applicazioni vengano eseguite su sistemi di proprietà di una singola organizzazione.

A causa della complessità del ASVS 4.0 e del nostro obiettivo di renderlo lo standard di riferimento per tutti gli altri progetti correlati, abbiamo deciso di ritirare il capitolo dedicato al mobile, a favore del Mobile Application Security Verification Standard (MASVS). L'appendice relativa all'Internet of Things (IoT) verrà inclusa in un futuro standard ASVS specifico per l'IoT, curato dall'OWASP Internet of Things Project. Nell'Appendice C, abbiamo inserito un'anteprima iniziale del ASVS per l'IoT. Ringraziamo l'OWASP Mobile Team e l'OWASP IoT Project Team per il loro supporto all'ASVS e siamo entusiasti di collaborare con loro in futuro per offrire standard complementari.

Infine, abbiamo rimosso i controlli duplicati e quelli a basso impatto. Con il tempo, l'ASVS è diventato un insieme completo di controlli, ma non tutti hanno lo stesso valore nel garantire la sicurezza del software. Questo lavoro di eliminazione dei controlli a basso impatto potrebbe proseguire in futuro. Nella prossima edizione del ASVS, il Common Weakness Scoring System (CWSS) sarà utilizzato per aiutare a prioritizzare ulteriormente i controlli davvero rilevanti, distinguendoli da quelli da eliminare.

A partire dalla versione 4.0, l'ASVS si focalizzerà esclusivamente sul diventare lo standard di riferimento per le applicazioni e i servizi web, includendo sia architetture applicative tradizionali che moderne. Coprirà inoltre le pratiche di sicurezza agili e integrerà la cultura DevSecOps, garantendo un approccio completo e aggiornato alla sicurezza.

Utilizzo del ASVS

ASVS ha due obiettivi principali:

- aiutare le organizzazioni a sviluppare e mantenere applicazioni sicure.
- consentire ai fornitori di servizi di sicurezza, ai fornitori di strumenti di sicurezza e ai consumatori di allineare i propri requisiti e le loro offerte.

Livelli di verifica della sicurezza delle applicazioni

Lo standard di verifica della sicurezza delle applicazioni (ASVS) definisce tre livelli di verifica della sicurezza, ciascuno con un livello di approfondimento crescente.

- ASVS Livello 1 è pensato per bassi livelli di garanzie di sicurezza ed è completamente testabile tramite penetration test.
- ASVS Levello 2 è adatto per applicazioni che contengono dati sensibili che richiedono protezione, ed è il livello raccomandato per la maggior parte delle applicazioni.
- ASVS Levello 3 è riservato alle applicazioni più critiche, come quelle che eseguono transazioni di alto valore, contengono dati medici sensibili o qualsiasi applicazione che richieda il massimo livello di sicurezza.

Ciascun livello ASVS contiene un elenco di requisiti di sicurezza. Ognuno di questi requisiti può essere mappato anche su funzionalità e capacità specifiche per la sicurezza che devono essere integrate nel software dagli sviluppatori.

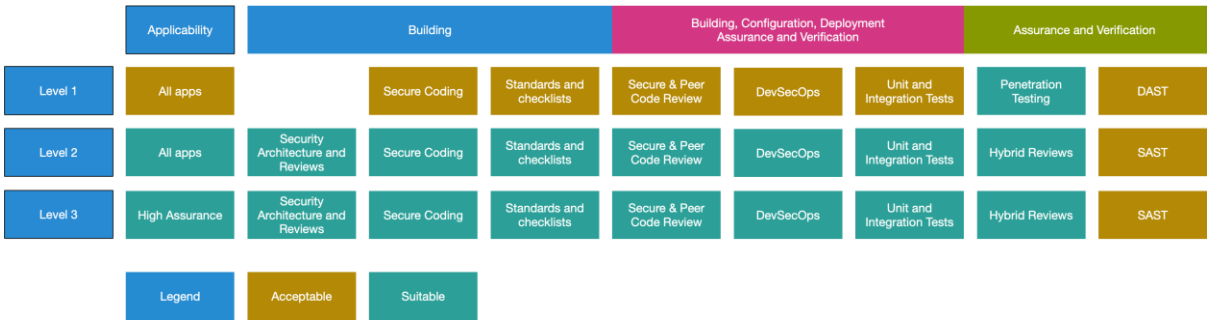

Figura 1 - Livelli dell'OWASP Application Security Verification Standard 4.0

Il Livello 1 del ASVS è l'unico che può essere testato completamente tramite penetration test eseguiti da personale umano. Tuttavia, tutti gli altri livelli richiedono accesso a documentazione, codice sorgente, configurazioni e interazione con il team di sviluppo. Sebbene il Livello 1 permetta test "black box" (senza accesso a documentazione e codice sorgente), questo tipo di verifica non è efficace e dovrebbe essere attivamente scoraggiato. Gli attori malintenzionati hanno tutto il tempo a disposizione per condurre attacchi, mentre la maggior parte dei penetration test si conclude in poche settimane. I difensori, invece, devono non solo integrare controlli di sicurezza, ma anche individuare e risolvere tutte le vulnerabilità e rilevare e rispondere alle minacce in tempi ragionevoli. Gli attaccanti, avendo potenzialmente tempo illimitato, necessitano solo di una singola falla, debolezza o errore di rilevamento per avere successo. I test "black box", spesso eseguiti alla fine dello sviluppo, in modo affrettato o persino trascurati, sono del tutto inadeguati per gestire questa disparità temporale tra difensori e attaccanti.

Negli ultimi 30 anni, i test "black box" hanno dimostrato ripetutamente la loro inefficacia nell'individuare problemi di sicurezza critici, portando a violazioni sempre più gravi. Per questo motivo, raccomandiamo vivamente di adottare una gamma più ampia di attività di verifica e misure di garanzia della sicurezza, inclusa la sostituzione dei tradizionali penetration test con test ibridi di Livello 1. Questi dovrebbero essere guidati dall'analisi del codice sorgente, accompagnati da un dialogo aperto con gli sviluppatori e un accesso continuo alla documentazione durante tutto il processo di sviluppo. Analogamente a come gli enti di regolamentazione finanziaria non accetterebbero un audit esterno senza accesso ai libri contabili, a transazioni campione o al personale responsabile dei controlli, il settore e i governi dovrebbero richiedere lo stesso livello di trasparenza nell'ingegneria del software.

Sosteniamo fortemente l'integrazione di strumenti di sicurezza direttamente nel processo di sviluppo. Strumenti come DAST (Dynamic Application Security Testing) e SAST (Static Application Security Testing) possono essere impiegati continuamente all'interno della pipeline di build, per identificare in modo proattivo vulnerabilità facili da rilevare, che non dovrebbero mai essere presenti nelle applicazioni.

Gli strumenti automatizzati e le scansioni online non possono coprire più della metà del ASVS senza supporto umano. Per automatizzare completamente i test su ogni build, si utilizzano unit test e test di integrazione personalizzati, insieme a scansioni online avviate dalla build. Tuttavia, le vulnerabilità nella logica di business e i controlli sugli accessi richiedono l'intervento umano e dovrebbero essere tradotti in unit test e test di integrazione.

Come utilizzare questo standard

Uno dei modi migliori per sfruttare lo standard di verifica della sicurezza delle applicazioni è usarlo come base per creare una checklist di sviluppo sicuro, adattata alla propria applicazione, piattaforma o organizzazione. Personalizzare l'ASVS in base ai propri casi d'uso permette di concentrarsi sui requisiti di sicurezza più rilevanti per i propri progetti e contesti operativi.

Livello 1 - Primi passi, automatizzato o visione d'insieme del portfolio

Un'applicazione raggiunge il Livello ASVS 1 se è sufficientemente protetta contro vulnerabilità applicative facilmente individuabili, come quelle presenti nella OWASP Top 10 o in elenchi simili.

Il Livello 1 rappresenta il requisito minimo che tutte le applicazioni dovrebbero soddisfare. È utile come punto di partenza in un percorso di sicurezza più ampio o per applicazioni che non trattano dati sensibili e non richiedono i controlli più rigorosi previsti dai Livelli 2 o 3. I controlli di Livello 1 possono essere verificati automaticamente tramite strumenti o manualmente senza accesso al codice sorgente. Consideriamo il Livello 1 come la soglia di sicurezza di base per qualsiasi applicazione.

Le minacce alle applicazioni spesso provengono da attaccanti che utilizzano tecniche semplici per identificare e sfruttare vulnerabilità facilmente rilevabili, a differenza di attori malevoli più determinati che impiegano risorse significative per colpire applicazioni specifiche. Se la vostra applicazione gestisce dati di alto valore, una verifica di Livello 1 non sarà quasi mai sufficiente.

Livello 2 - La maggior parte delle applicazioni

Un'applicazione raggiunge il Livello ASVS 2 (o Standard) se è in grado di proteggersi adeguatamente dalla maggior parte dei rischi associati al software moderno.

Il Livello 2 garantisce che i controlli di sicurezza siano implementati correttamente, funzionino in modo efficace e siano applicati in maniera appropriata all'interno dell'applicazione. È generalmente raccomandato per applicazioni che gestiscono transazioni business-to-business rilevanti, come quelle che trattano informazioni sanitarie, svolgono funzioni aziendali critiche o sensibili, o elaborano altre risorse sensibili. È anche fondamentale in settori dove l'integrità è essenziale, come l'industria dei videogiochi, per contrastare truffatori e attacchi hacker.

Le minacce per le applicazioni di Livello 2 provengono tipicamente da attaccanti qualificati e motivati, che mirano a obiettivi specifici utilizzando strumenti e tecniche avanzate per individuare e sfruttare le vulnerabilità delle applicazioni.

Livello 3 - Alto valore, alta affidabilità o alta sicurezza

Il Livello ASVS 3 rappresenta il più alto standard di verifica della sicurezza nell'ASVS, ed è generalmente destinato ad applicazioni che richiedono un livello significativo di protezione, come quelle dei settori della difesa, della sanità, della sicurezza e delle infrastrutture critiche.

Le organizzazioni possono adottare il Livello ASVS 3 per applicazioni che eseguono funzioni critiche, dove un eventuale fallimento potrebbe avere gravi conseguenze operative o compromettere persino la sopravvivenza dell'azienda. Un'applicazione raggiunge il Livello ASVS 3 (o Avanzato) se non solo protegge da vulnerabilità avanzate, ma dimostra anche una progettazione di sicurezza robusta e ben strutturata.

Le applicazioni di Livello ASVS 3 richiedono un'analisi più dettagliata dell'architettura, del codice e dei test rispetto ai livelli inferiori. Un'applicazione sicura a questo livello è organizzata in moduli distinti, in modo da

garantire resilienza, scalabilità e sicurezza. Ogni modulo, isolato tramite connessioni di rete o istanze fisiche, è responsabile della propria sicurezza, implementando una difesa stratificata. Queste responsabilità devono essere ben documentate e includere controlli per la riservatezza (ad esempio, crittografia), l'integrità (come la gestione delle transazioni e la validazione degli input), la disponibilità (ottimizzazione della gestione del carico), l'autenticazione (anche tra sistemi), l'autorizzazione e l'auditing (logging).

Applicazione pratica del ASVS

Gli attaccanti possono avere motivazioni diverse e alcuni settori dispongono di asset informatici e informativi unici, con requisiti normativi specifici.

Si raccomanda vivamente alle organizzazioni di valutare con attenzione le proprie caratteristiche di rischio in base alla natura del business e, in funzione di tali rischi e delle esigenze aziendali, determinare il livello ASVS più adatto per garantire la sicurezza delle applicazioni.

Come fare riferimento ai requisiti ASVS

Ciascun requisito dispone di un identificatore in formato <capitolo>.<sezione>.<requisito> dove ogni elemento è un numero, per esempio: 1.11.3.

- Il valore <capitolo> corrisponde al capitolo da cui proviene il requisito, per esempio: tutti i requisiti 1.#.# appartengono al capitolo Architettura.
- Il valore <sezione> corrisponde alla sezione all'interno di quel capitolo in cui compare il requisito, per esempio: tutti i requisiti, 1.11.# si trovano nella sezione Architettura logica di business del capitolo Architettura.
- Il valore <requisito> identifica il requisito specifico all'interno del capitolo e della sezione, per esempio: 1.11.3 che, nella versione 4.0.3 di questo standard è:

Verificare che tutti i flussi logici di business di alto valore, inclusi autenticazione, gestione delle sessioni e controllo degli accessi, siano thread-safe e resistenti alle race conditions quali time-of-check e time-of-use.

Gli identificatori possono cambiare tra le versioni dello standard, pertanto è preferibile che altri documenti, rapporti o strumenti utilizzino il formato v<version>-<chapter>.<section>.<requirement>, dove 'versione'è l'etichetta della versione ASVS. Per esempio: v4.0.3-1.11.3 indicherebbe specificatamente il 3° requisito nella sezione 'Architettura logica di business' del capitolo 'Architettura' della versione 4.0.3. (Questo può essere riassunto come v<version>-<requirement_identifier>.)

Nota: la v che precede la parte relativa alla versione deve essere minuscola.

Se gli identificatori vengono usati senza includere l'elemento v<version> si presuppone che facciano riferimento al contenuto più recente dello standard ASVS. Ovviamente, man mano che lo standard cresce e cambia, ciò diventa problematico. Ecco perché gli autori o gli sviluppatori dovrebbero sempre includere l'elemento versione.

Gli elenchi dei requisiti ASVS sono disponibili in CSV, JSON e altri formati che possono essere utili per riferimento o uso programmatico.

Valutazione e certificazione

La posizione di OWASP sulle certificazioni e i marchi di fiducia ASVS

OWASP, in quanto organizzazione no profit indipendente dai fornitori, attualmente non certifica alcun fornitore, ente verificatore o software.

Pertanto, tutte le dichiarazioni di garanzia, i marchi di fiducia o le certificazioni di questo tipo non sono ufficialmente esaminate, registrate o certificate da OWASP. Le organizzazioni che si affidano a tali elementi devono essere caute nel riporre fiducia in terze parti o marchi che dichiarano la certificazione ASVS.

Ciò non impedisce alle organizzazioni di offrire servizi di garanzia, purché non rivendichino ufficialmente una certificazione OWASP.

Linee guida per gli organismi di certificazione

Lo standard di verifica della sicurezza delle applicazioni può essere utilizzato come una verifica "open book", con accesso completo e senza restrizioni a risorse chiave, quali architetti del software e sviluppatori, documentazione di progetto, codice sorgente, e accesso autenticato ai sistemi di test (incluso l'accesso a uno o più account per ogni ruolo). Questo approccio è particolarmente importante per le verifiche di Livello 2 e Livello 3.

Storicamente, i penetration test e le revisioni del codice sicuro hanno incluso solo problematiche "per eccezione", ovvero solo i test falliti vengono riportati nel rapporto finale. Tuttavia, un organismo di certificazione deve includere nel rapporto l'intero ambito della verifica (soprattutto se un componente chiave, come l'autenticazione SSO, è fuori ambito), un riepilogo dei risultati, compresi i test superati e non, con indicazioni chiare su come risolvere eventuali test falliti.

Alcuni requisiti di verifica potrebbero non essere applicabili all'applicazione sotto test. Ad esempio, se si offre ai clienti un'API stateless a livello di servizio, senza una componente client, molti dei requisiti della sezione V3 (Gestione sessioni) non si applicheranno direttamente. In questi casi, un organismo di certificazione può comunque dichiarare la piena conformità ASVS, a patto che venga fornita una chiara motivazione per l'esclusione di tali requisiti nel rapporto finale.

La conservazione di rapporti di lavoro dettagliati, screenshot o filmati, script per riprodurre un problema in modo affidabile e log elettronici dei test (come i log di proxy e note di cleanup) è considerata una prassi standard nel settore. Questi elementi sono fondamentali per fornire prove solide dei risultati, soprattutto di fronte a sviluppatori scettici. Non è sufficiente eseguire uno strumento e segnalare semplicemente i fallimenti: questo non dimostra che tutti i problemi siano stati effettivamente testati in modo approfondito. In caso di controversia, devono esserci prove sufficienti per garantire che ogni singolo requisito sia stato effettivamente verificato.

Metodi di valutazione

Gli organismi di certificazione sono liberi di scegliere i metodi di valutazione più appropriati, ma devono documentarli in modo trasparente.

A seconda dell'applicazione sottoposta a test e dei requisiti di verifica, possono essere utilizzati vari metodi per garantire un'adeguata affidabilità dei risultati. Ad esempio, la convalida dell'efficacia dei meccanismi di verifica dell'input di un'applicazione può essere eseguita attraverso un penetration test manuale o tramite un'analisi del codice sorgente.

Ruolo degli strumenti di test di sicurezza automatizzati

Si incoraggia l'uso di strumenti di penetration testing automatizzati per garantire la copertura più ampia possibile.

Tuttavia, non è possibile completare interamente la verifica ASVS solo con strumenti automatizzati. Sebbene molti dei requisiti del Livello 1 possano essere soddisfatti tramite test automatizzati, gran parte dei requisiti non è adatta a essere verificata esclusivamente con penetration test automatizzati.

Va inoltre considerato che la distinzione tra test automatizzati e manuali si è ridotta con l'evoluzione del settore della sicurezza delle applicazioni. Gli strumenti automatizzati vengono spesso calibrati da esperti,

mentre i tester manuali sfruttano una varietà di strumenti automatizzati per migliorare l'efficacia delle loro analisi.

Ruolo del Penetration Testing

Nella versione 4.0, abbiamo deciso di rendere il Livello ASVS 1 completamente testabile tramite penetration testing, senza necessità di accedere a codice sorgente, documentazione o sviluppatori. Tuttavia, per soddisfare i requisiti di logging dell'OWASP Top 10 2017 A10, saranno necessarie interviste, screenshot o altre prove, come indicato anche nell'OWASP Top 10 2017. Nonostante ciò, condurre test senza l'accesso alle informazioni essenziali non è il metodo ideale per una verifica della sicurezza completa, poiché non permette di esaminare il codice sorgente per identificare minacce e controlli mancanti, limitando così la profondità dell'analisi.

Per le valutazioni di Livello 2 o 3, è preferibile avere accesso a sviluppatori, documentazione, codice sorgente e a un ambiente di test con dati non di produzione. Il penetration testing a questi livelli richiede questo tipo di accesso, definito "revisioni ibride" o "penetration test ibridi", per eseguire un'analisi più approfondita e accurata.

Altri utilizzi del ASVS

L'ASVS non è solo utile per valutare la sicurezza di un'applicazione, ma ha anche diversi altri potenziali impieghi.

Come guida completa per l'architettura della sicurezza

Uno degli utilizzi più comuni dell'Application Security Verification Standard (ASVS) è come risorsa per gli architetti della sicurezza. Lo Sherwood Applied Business Security Architecture (SABSA) non fornisce molte delle informazioni necessarie per una revisione completa dell'architettura di sicurezza delle applicazioni. L'ASVS può colmare queste lacune, aiutando gli architetti della sicurezza a selezionare controlli più efficaci per affrontare problemi comuni, come i modelli di protezione dei dati e le strategie di convalida degli input.

Come alternativa agli elenchi di controllo predefiniti per la codifica sicura

Molte organizzazioni possono beneficiare dell'adozione del ASVS, scegliendo uno dei tre livelli o modificandolo per adattare i requisiti a ciascun livello di rischio specifico per il proprio dominio. Incoraggiamo questo tipo di personalizzazione, a condizione che venga mantenuta la tracciabilità. In questo modo, se un'applicazione soddisfa il requisito 4.1, il risultato sarà coerente anche nelle versioni modificate rispetto allo standard originale, man mano che evolve.

Come guida per unit test e test di integrazione automatizzati

L'ASVS è progettato per essere altamente testabile, ad eccezione dei requisiti relativi all'architettura e al codice malevolo. Creando unit test e test di integrazione che includono fuzzing e scenari malevoli specifici e pertinenti, l'applicazione può diventare quasi auto-verificante a ogni build. Ad esempio, è possibile ampliare la suite di test di un controller di login, verificando il parametro *username* per nomi utente predefiniti comuni, enumerazione degli account, attacchi brute force, injection LDAP, SQL e XSS. Allo stesso modo, i test per il parametro *password* dovrebbero coprire password comuni, lunghezza minima, injection di byte null, rimozione completa del parametro, XSS e altre vulnerabilità.

Per la formazione allo sviluppo sicuro

L'ASVS può essere utilizzato anche per definire le caratteristiche di un software sicuro. Molti corsi di "programmazione sicura" tendono a concentrarsi principalmente su tecniche di hacking etico, offrendo solo alcuni suggerimenti di programmazione, il che potrebbe non essere sufficiente per aiutare gli sviluppatori a scrivere codice più sicuro. Invece, i corsi di sviluppo sicuro possono basarsi sull'ASVS, ponendo maggiore enfasi sui controlli proattivi descritti nello standard, piuttosto che focalizzarsi solo sui 10 principali rischi da evitare. Questo approccio fornisce agli sviluppatori una guida pratica e concreta per costruire applicazioni sicure fin dall'inizio.

Come catalizzatore per la sicurezza agile delle applicazioni

L'ASVS può essere utilizzato in un processo di sviluppo agile come framework per definire attività specifiche necessarie a garantire un prodotto sicuro. Un approccio potrebbe consistere nel partire dal Livello 1, verificando l'applicazione o il sistema rispetto ai requisiti ASVS per il livello selezionato, individuando i controlli

mancanti e creando ticket o attività specifiche nel backlog. Questo aiuta a dare priorità alle attività di sicurezza (o grooming) e rende la sicurezza una parte visibile e integrata del processo agile. L'ASVS può anche essere utilizzato per dare priorità ad attività di audit e revisione, dove un requisito specifico guida la revisione, il refactoring o l'audit da parte di un membro del team, e viene trattato come "debito" nel backlog, che dovrà essere risolto nel tempo.

Come framework per orientare l'approvvigionamento di software sicuro

L'ASVS è un ottimo framework per facilitare l'approvvigionamento di software sicuro o servizi di sviluppo personalizzati. L'acquirente può specificare come requisito che il software da acquistare sia sviluppato in conformità al Livello ASVS X, e richiedere al venditore di dimostrare che il software soddisfa tale livello. Questo approccio è particolarmente efficace se combinato con l'Allegato Contrattuale per Software Sicuro OWASP, garantendo che le aspettative di sicurezza siano chiaramente definite e concordate contrattualmente.

V1 Architettura, progettazione e Threat Modeling

Obiettivo del controllo

L'architettura di sicurezza in molte organizzazioni è diventata quasi un'arte perduta. Nell'era del DevSecOps, l'epoca degli architetti enterprise sembra superata. Il campo della sicurezza delle applicazioni deve evolversi, adottando i principi di sicurezza agile, pur reintroducendo i concetti di architettura della sicurezza più avanzati per i professionisti del software. L'architettura non riguarda l'implementazione, ma è un modo di affrontare i problemi che può portare a molte soluzioni diverse, senza una risposta "corretta" unica. Troppo spesso la sicurezza è percepita come rigida, richiedendo agli sviluppatori di correggere il codice in un modo specifico, quando potrebbero conoscere soluzioni migliori. Pensare che esista una soluzione unica per l'architettura è dannoso per l'ingegneria del software.

Un'applicazione web verrà probabilmente rivista più volte durante il suo ciclo di vita, ma la sua architettura complessiva cambierà raramente, evolvendosi lentamente. Lo stesso vale per l'architettura della sicurezza: oggi abbiamo bisogno dell'autenticazione, la richiederemo domani e tra cinque anni. Decisioni ponderate oggi possono far risparmiare tempo, sforzi e risorse in futuro, se si selezionano soluzioni conformi all'architettura esistente. Ad esempio, un decennio fa, l'autenticazione a più fattori era poco diffusa, ma oggi è una componente chiave.

Se gli sviluppatori avessero investito in un unico modello sicuro di provider di identità, come l'identità federata tramite SAML, il provider di identità potrebbe essere aggiornato per soddisfare nuovi requisiti, come la conformità al NIST 800-63, senza dover modificare le interfacce dell'applicazione originale. Se più applicazioni condividono la stessa architettura di sicurezza e, quindi, lo stesso componente, tutte beneficerebbero contemporaneamente di questo aggiornamento. Tuttavia, SAML non sarà sempre la soluzione di autenticazione migliore o più adatta: potrebbe essere necessario sostituirla con altre soluzioni in base ai nuovi requisiti. Tali cambiamenti possono essere molto complessi, costosi al punto da richiedere una riscrittura completa, o impossibili da gestire senza una solida architettura di sicurezza.

In questo capitolo, l'ASVS affronta i principi fondamentali di una buona architettura di sicurezza: disponibilità, riservatezza, integrità dell'elaborazione, non ripudio e privacy. Questi principi devono essere integrati in modo nativo in tutte le applicazioni. È essenziale "spostarsi a sinistra", iniziando con l'adozione di checklist di programmazione sicura, mentoring, formazione, test di programmazione, build, distribuzione, configurazione e operations, fino a concludere con test indipendenti per garantire che tutti i controlli di sicurezza siano attivi e funzionanti. Tradizionalmente, l'ultimo passaggio era l'unico svolto nel settore, ma non è più sufficiente quando il codice viene distribuito in produzione decine o centinaia di volte al giorno. I professionisti della sicurezza devono tenere il passo con le metodologie agili, adottare strumenti di sviluppo, imparare a programmare e collaborare con gli sviluppatori, piuttosto che criticare i progetti mesi dopo che il lavoro è stato concluso e il team è passato oltre.

V1.1 Ciclo di vita sicuro dello sviluppo software (S-SDLC)

#	Descrizione	L1	L2	L3	CWE
1.1.1	Verificare l'utilizzo di un ciclo di vita sicuro dello sviluppo software (S-SDLC) che tenga conto della sicurezza in tutte le fasi di sviluppo. (C1)		✓	✓	
1.1.2	Verificare l'utilizzo del threat modeling per ogni modifica del design o nella pianificazione dello sprint al fine di identificare le minacce, pianificare contromisure, facilitare risposte ai rischi appropriate e guidare i test di sicurezza.		✓	✓	1053
1.1.3	Verificare che tutte le user story e le funzionalità includano vincoli di sicurezza funzionali, come "Come utente, dovrei poter visualizzare e modificare il mio profilo. Non dovrei poter visualizzare o modificare il profilo di nessun altro"		✓	✓	1110

#	Descrizione	L1	L2	L3	CWE
1.1.4	Verificare che tutti i perimetri di trust, i componenti e i flussi di dati significativi dell'applicazione siano documentati e giustificati in modo accurato.		✓	✓	1059
1.1.5	Verificare che l'architettura di alto livello dell'applicazione e tutti i servizi remoti collegati siano definiti e sottoposti a un'analisi approfondita dal punto di vista della sicurezza. (C1)		✓	✓	1059
1.1.6	Verificare che siano implementati controlli di sicurezza centralizzati, semplici (secondo il principio dell'economia di progettazione), verificati, sicuri e riutilizzabili, per evitare controlli duplicati, mancanti, inefficaci o non sicuri. (C10)		✓	✓	637
1.1.7	Verificare che tutti gli sviluppatori e tester abbiano a disposizione una checklist di programmazione sicura, requisiti di sicurezza, linee guida o policy.		✓	✓	637

V1.2 Architettura di autenticazione

Quando si progetta un sistema di autenticazione, è irrilevante se si utilizza l'autenticazione a più fattori basata su hardware se un aggressore può reimpostare un account semplicemente chiamando un call center e rispondendo a domande facilmente reperibili. Tutti i percorsi di autenticazione devono avere la stessa efficacia nella verifica dell'identità.

#	Descrizione	L1	L2	L3	CWE
1.2.1	Verificare l'utilizzo di account di sistema operativo univoci o speciali con privilegi minimi per tutti i componenti, servizi e server dell'applicazione. (C3)		✓	✓	250
1.2.2	Verificare che le comunicazioni tra i componenti dell'applicazione, incluse le API, il middleware e il data layer, siano autenticate. I componenti devono avere i privilegi minimi necessari. (C3)		✓	✓	306
1.2.3	Verificare che l'applicazione utilizzi un unico meccanismo di autenticazione verificato noto per essere sicuro, estendibile per includere l'autenticazione forte e disponga di registri e monitoraggio sufficienti per rilevare abusi o violazioni dell'account.		✓	✓	306
1.2.4	Verificare che tutti i flussi di autenticazione e le API di gestione delle identità implementino dei controlli di autenticazione coerenti in efficacia, in modo tale che non ci siano alternative più deboli, in linea con il livello di rischio dell'applicazione.		✓	✓	306

V1.3 Architettura della gestione sessioni

Questo è un segnaposto per requisiti architetturali futuri.

V1.4 Architettura del Controllo Accessi

#	Descrizione	L1	L2	L3	CWE
1.4.1	Verificare che i punti di applicazione trusted, come gateway di controllo accessi, server e funzioni serverless, applichino controlli di accesso. Non implementare mai controlli di accesso sul client.		✓	✓	602
1.4.2	[ELIMINATO, NON APPLICABILE]				
1.4.3	[ELIMINATO, DUPLICATO DI 4.1.3]				

#	Descrizione	L1	L2	L3	CWE
1.4.4	Verificare che l'applicazione utilizzi un unico meccanismo di controllo accessi ben collaudato per accedere a dati e risorse protette. Tutte le richieste devono passare attraverso questo meccanismo per evitare copia e incolla o percorsi alternativi non sicuri. (C7)		✓	✓	284
1.4.5	Verificare che venga utilizzato il controllo di accesso basato su attributi o su funzionalità, in modo che il codice verifichi l'autorizzazione dell'utente per una funzionalità/elemento dati, non solo per il suo ruolo. Le autorizzazioni devono comunque essere assegnate utilizzando i ruoli. (C7)		✓	✓	275

V1.5 Architettura di Input e Output

Nella versione 4.0, abbiamo abbandonato il termine "lato server" poiché ambiguo in relazione al perimetro di trust. Tuttavia, il perimetro di trust rimane un aspetto cruciale: i controlli implementati su browser o dispositivi client non affidabili possono essere facilmente aggirati. Nelle moderne architetture tradizionali, il punto di contatto con un componente trusted è cambiato radicalmente. Pertanto, nell'ASVS, il termine "livello di servizio trusted" si riferisce a qualsiasi punto di applicazione considerato affidabile, indipendentemente dalla sua posizione. Questo include microservizi, API serverless, componenti lato server, API trusted su dispositivi client con secure boot, API di partner o esterne, e altre implementazioni simili.

Il termine "client non affidabile" si riferisce, in questo contesto, alle tecnologie lato client che gestiscono la presentazione, comunemente definite tecnologie "front-end". Il termine "serializzazione" non si limita al semplice trasferimento di dati in rete sotto forma di array di valori o alla lettura di strutture JSON, ma comprende anche il passaggio di oggetti complessi che possono includere logica.

#	Descrizione	L1	L2	L3	CWE
1.5.1	Verificare che i requisiti di input e output definiscano chiaramente come gestire ed elaborare i dati in base al tipo, al contenuto, alle leggi in vigore, ai regolamenti e alla conformità ad altre policy applicabili.		✓	✓	1029
1.5.2	Verificare che la serializzazione non venga utilizzata nelle comunicazioni con client non affidabili. Se inevitabile, applicare controlli di integrità adeguati (e possibilmente la crittografia se vengono inviati dati sensibili) per prevenire attacchi di deserializzazione, inclusa l'injection di oggetti.		✓	✓	502
1.5.3	Verificare che la convalida degli input venga applicata su un livello di servizio trusted. (C5)		✓	✓	602
1.5.4	Verificare che la codifica dell'output avvenga vicino o dall'interprete per il quale è destinata. (C4)		✓	✓	116

V1.6 Architettura Crittografica

Le applicazioni devono essere progettate con una solida architettura crittografica per proteggere i dati in base alla loro classificazione. Crittografare tutto indiscriminatamente è uno spreco, mentre non crittografare nulla può configurarsi come negligenza legale. È fondamentale trovare un equilibrio, generalmente durante la fase di progettazione architetturale, gli sprint di design o le analisi ad alto livello. Progettare la crittografia "al volo" o integrarla successivamente comporta costi significativamente maggiori per implementarla in modo sicuro rispetto a un'integrazione pianificata fin dall'inizio.

I requisiti architetturali riguardano l'intero sistema e sono quindi difficili da testare a livello di unità o integrazione. Questi requisiti devono essere considerati negli standard di programmazione durante tutta la fase di sviluppo e rivisti in occasione delle revisioni di sicurezza architetturale, nelle revisioni del codice (anche tra pari) o durante le retrospettive.

#	Descrizione	L1	L2	L3	CWE
1.6.1	Verificare che esista una policy esplicita per la gestione delle chiavi crittografiche e che il ciclo di vita delle chiavi crittografiche segua uno standard di gestione delle chiavi come il NIST SP 800-57.		✓	✓	320
1.6.2	Verificare che gli utilizzatori di servizi crittografici proteggano il materiale chiave e altri segreti utilizzando key vault o alternative basate su API.		✓	✓	320
1.6.3	Verificare che tutte le chiavi e le password siano sostituibili e facciano parte di un processo ben definito per la ri-crittografia dei dati sensibili.		✓	✓	320
1.6.4	Verificare che l'architettura consideri i segreti lato client, come chiavi simmetriche, password o token API, come non sicuri e non li utilizzi mai per proteggere o accedere a dati sensibili.		✓	✓	320

V1.7 Architettura di Errori, Logging e Audit

#	Descrizione	L1	L2	L3	CWE
1.7.1	Verificare che venga utilizzato un formato e un approccio di logging comune in tutto il sistema. (C9)		✓	✓	1009
1.7.2	Verificare che i log vengano trasmessi in modo sicuro a un sistema, preferibilmente remoto, per analisi, rilevamento, allerta e escalation. (C9)		✓	✓	

V1.8 Architettura per Protezione Dati e Privacy

#	Descrizione	L1	L2	L3	CWE
1.8.1	Verificare che tutti i dati sensibili siano identificati e classificati con adeguati livelli di protezione.		✓	✓	
1.8.2	Verificare che a tutti i livelli di protezione sia associato un set di requisiti di protezione, come requisiti di crittografia, requisiti di integrità, conservazione, privacy e altri requisiti di riservatezza, e che questi vengano applicati nell'architettura.		✓	✓	

V1.9 Architettura della Comunicazione

#	Descrizione	L1	L2	L3	CWE
1.9.1	Verificare che l'applicazione crittografi le comunicazioni tra componenti, in particolare quando questi componenti si trovano in container, sistemi, siti o provider cloud diversi. (C3)		✓	✓	319
1.9.2	Verificare che i componenti applicativi verifichino l'autenticità di ciascuna parte in un collegamento di comunicazione per prevenire attacchi man-in-the-middle. Ad esempio, i componenti dell'applicazione dovrebbero validare i certificati e le catene TLS.		✓	✓	295

V1.10 Architettura del Software Maligno

#	Descrizione	L1	L2	L3	CWE
1.10.1	Verificare l'utilizzo di un sistema di controllo del codice sorgente, con procedure per garantire che i check-in siano accompagnati da segnalazioni o ticket di modifica. Il sistema di controllo del codice sorgente deve disporre di controlli di accesso e utenti identificabili per consentire la tracciabilità di qualsiasi modifica.		✓	✓	284

V1.11 Architettura della Logica di Business

#	Descrizione	L1	L2	L3	CWE
1.11.1	Verificare la definizione e la documentazione di tutti i componenti applicativi in termini di funzioni aziendali o di sicurezza fornite.		✓	✓	1059
1.11.2	Verificare che tutti i flussi logici di business di alto valore, inclusi autenticazione, gestione sessioni e controllo degli accessi, sincronizzino correttamente lo stato.		✓	✓	362
1.11.3	Verificare che tutti i flussi logici di business di alto valore, inclusi autenticazione, gestione sessioni e controllo degli accessi, siano thread-safe e resistenti a race conditions quali "time-of-check" e "time-of-use".			✓	367

V1.12 Architettura del Caricamento Sicuro di File

#	Descrizione	L1	L2	L3	CWE
1.12.1	[ELIMINATO, DUPLICATO DI 12.4.1]				
1.12.2	Verificare che i file caricati dagli utenti - se devono essere visualizzati o scaricati dall'applicazione - vengano serviti tramite octet stream o da un dominio non correlato, come un bucket di archiviazione file cloud. Implementare una Content Security Policy (CSP) adeguata per ridurre il rischio di vettori XSS o altri attacchi.		✓	✓	646

V1.13 Architettura delle API

Questo è un segnaposto per requisiti architetturali futuri.

V1.14 Architettura della Configurazione

#	Descrizione	L1	L2	L3	CWE
1.14.1	Verificare la segregazione dei componenti con livelli di trust differenti attraverso controlli di sicurezza ben definiti, regole firewall, gateway API, proxy inverso, gruppi di sicurezza basati su cloud o meccanismi simili.		✓	✓	923
1.14.2	Verificare l'utilizzo di firme, connessioni affidabili e endpoint verificati per distribuire i file binari su dispositivi remoti.		✓	✓	494
1.14.3	Verificare che la pipeline di build avvisi di componenti obsoleti o non sicuri e intraprenda azioni appropriate.		✓	✓	1104

#	Descrizione	L1	L2	L3	CWE
1.14.4	Verificare che la pipeline di build contenga un passaggio di build per automatizzare la creazione e la verifica del deployment sicuro dell'applicazione, soprattutto se l'infrastruttura dell'applicazione è software-defined, come gli script di build per ambienti cloud.		✓	✓	
1.14.5	Verificare che le distribuzioni dell'applicazione implementino adeguatamente sandbox, containerizzazione e/o isolamento a livello di rete per rallentare e scoraggiare gli attori malevoli dall'attaccare altre applicazioni, specialmente quando eseguono azioni sensibili o pericolose come la deserializzazione. (C5)		✓	✓	265
1.14.6	Verificare che l'applicazione non utilizzi tecnologie client-side non supportate, non sicure o obsolete come plugin NSAPI, Flash, Shockwave, ActiveX, Silverlight, NACL o applet Java client-side.		✓	✓	477

Riferimenti

Per approfondimenti, consultare:

- OWASP Threat Modeling Cheat Sheet
- OWASP Attack Surface Analysis Cheat Sheet
- OWASP Threat modeling
- OWASP Software Assurance Maturity Model Project
- Microsoft SDL
- NIST SP 800-57

V2 Autenticazione

Obiettivo del controllo

L'autenticazione consiste nel confermare l'identità di qualcuno (o qualcosa) e nell'assicurarsi che le affermazioni fatte da una persona o su un dispositivo siano corrette, resistenti all'impersonazione e proteggano le password da furti o intercettazioni.

Quando l'ASVS è stato pubblicato per la prima volta, la combinazione di username e password era la forma più comune di autenticazione, eccetto nei sistemi ad alta sicurezza. L'Autenticazione Multi-Fattore (MFA) era generalmente riservata agli ambienti più critici e raramente richiesta altrove. Con l'aumento delle violazioni di password, l'idea che gli username fossero confidenziali e le password sconosciute è diventata obsoleta, invalidando molti controlli di sicurezza tradizionali. Ad esempio, il NIST 800-63 considera gli username e l'Autenticazione Basata sulla Conoscenza (KBA) come informazioni pubbliche, e classifica notifiche via SMS ed email come modalità di autenticazione "restricted", mentre le password sono considerate già compromesse. Ciò rende inutili gli autenticatori basati sulla conoscenza, il recupero tramite SMS o email, la cronologia delle password, la loro rotazione e complessità. Questi controlli, spesso poco efficaci, hanno portato gli utenti a creare password deboli ad ogni cambio, e con oltre 5 miliardi di username e password compromessi, è il momento di cambiare approccio.

Tra tutti i capitoli del ASVS, quelli relativi all'autenticazione e alla gestione delle sessioni hanno subito i cambiamenti più significativi. L'adozione di pratiche efficaci e basate su evidenze sarà una sfida per molti, ma è normale. La transizione verso un futuro post-password deve iniziare adesso.

NIST 800-63 - Standard di autenticazione moderno basato su evidenze

Lo standard NIST 800-63b è moderno, basato su evidenze e rappresenta un punto di riferimento per la sicurezza dell'autenticazione, indipendentemente dalla sua applicabilità specifica. È utile per tutte le organizzazioni a livello globale, ma è particolarmente rilevante per le agenzie statunitensi e per coloro che collaborano con esse.

La terminologia del NIST 800-63 può risultare inizialmente un po' complessa, soprattutto se si ha familiarità solo con l'autenticazione tradizionale tramite username e password. I progressi nell'autenticazione moderna richiedono l'introduzione di nuovi termini, che diventeranno comuni in futuro, ma comprendiamo che la loro adozione richiede tempo. Per semplificare la comprensione, abbiamo fornito un glossario alla fine di questo capitolo. Inoltre, molti requisiti sono stati riformulati per chiarirne meglio l'intento piuttosto che mantenere una corrispondenza letterale. Ad esempio, l'ASVS utilizza il termine "password", mentre lo standard NIST usa "segreto memorizzato".

Le sezioni sull'Autenticazione (V2), la Gestione delle Sessioni (V3) e, in misura minore, i Controlli di Accesso (V4) del ASVS sono state adattate per essere un sottoinsieme conforme ai controlli selezionati dal NIST 800-63b. Questi controlli sono mirati a contrastare le minacce comuni e le debolezze di autenticazione più sfruttate. Per ottenere la piena conformità con il NIST 800-63, è necessario fare riferimento direttamente a tale standard.

Selezione di un livello AAL NIST appropriato

L'Application Security Verification Standard (ASVS) ha tentato di mappare i requisiti ASVS L1 a NIST AAL1, L2 a AAL2 e L3 a AAL3. Tuttavia, l'approccio del Livello 1 del ASVS, che considera i controlli come "essenziali", potrebbe non sempre corrispondere perfettamente al livello AAL corretto per verificare un'applicazione o un'API. Ad esempio, se un'applicazione è di Livello 3 o ha requisiti normativi che richiedono il rispetto di AAL3, allora il Livello 3 dovrebbe essere selezionato per i capitoli V2 (Autenticazione) e V3 (Gestione delle sessioni). La scelta del livello di Authentication Assurance Level (AAL) conforme al NIST dovrebbe seguire le linee guida NIST 800-63b, come indicato nella sezione *Selecting AAL* NIST 800-63b Sezione 6.2.

Legenda

Le applicazioni possono sempre superare i requisiti del livello attuale, specialmente se nella roadmap è previsto un meccanismo di autenticazione più moderno. In precedenza, l'ASVS richiedeva l'MFA obbligatorio, mentre il NIST non lo impone sempre. Perciò, in questo capitolo abbiamo introdotto indicazioni facoltative per

evidenziare dove l'ASVS incoraggia ma non richiede un determinato controllo. Le seguenti indicazioni sono utilizzate in tutto il documento:

Simbolo	Descrizione
	Non richiesto
o	Consigliato, ma non necessario
✓	Necessario

V2.1 Sicurezza delle Password

Le password, definite "segreti memorizzati" nel NIST 800-63, comprendono anche PIN, sequenze di sblocco, la scelta di immagini corrette (come gattini o altri elementi visivi) e frasi d'accesso. Questi sono generalmente considerati "qualcosa che sai" e vengono spesso utilizzati come meccanismo di autenticazione a singolo fattore. Tuttavia, ci sono sfide significative legate all'uso continuato dell'autenticazione a singolo fattore, tra cui la divulgazione online di miliardi di username e password validi, l'uso di password predefinite o deboli, e l'accessibilità di tabelle rainbow o dizionari delle password più comuni.

Le applicazioni dovrebbero incoraggiare fortemente gli utenti a utilizzare l'autenticazione a più fattori (MFA) e permettere loro di riutilizzare token già in loro possesso, come token FIDO o U2F, oppure collegarsi a un *Credential Service Provider* (CSP) che offre l'autenticazione a più fattori.

I *Credential Service Providers* (CSP) offrono identità federate agli utenti, che spesso possiedono più identità con diversi CSP, come Azure AD, Okta, Ping Identity o Google per identità aziendali, oppure Facebook, Twitter, Google o WeChat per identità consumer. Questo elenco non rappresenta un'approvazione di tali servizi, ma è un invito agli sviluppatori a considerare il fatto che molti utenti dispongono di identità consolidate con questi provider. Le organizzazioni dovrebbero valutare l'integrazione con identità utente esistenti in base al profilo di rischio associato al CSP. Ad esempio, una organizzazione governativa potrebbe non accettare un'identità social come login per sistemi sensibili, poiché è relativamente semplice creare identità fasulle o temporanee, mentre un'azienda di giochi mobile potrebbe essere incentivata a integrarsi con le principali piattaforme di social media per espandere la propria base di utenti attivi.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.1.1	Verificare che le password impostate dagli utenti siano lunghe almeno 12 caratteri (dopo aver unito gli spazi multipli). (C6)	✓	✓	✓	521	5.1.1.2
2.1.2	Verificare che siano consentite password di almeno 64 caratteri e che vengano rifiutate quelle di oltre 128 caratteri. (C6)	✓	✓	✓	521	5.1.1.2
2.1.3	Verificare che non venga effettuato il troncamento della password. Tuttavia, gli spazi multipli consecutivi possono essere sostituiti da un singolo spazio. (C6)	✓	✓	✓	521	5.1.1.2
2.1.4	Verificare che nelle password siano consentiti tutti i caratteri Unicode stampabili, inclusi caratteri neutri come spazi ed emoji.	✓	✓	✓	521	5.1.1.2
2.1.5	Verificare che gli utenti possano cambiare la propria password.	✓	✓	✓	620	5.1.1.2
2.1.6	Verificare che la funzionalità di cambio password richieda all'utente la password corrente e quella nuova.	✓	✓	✓	620	5.1.1.2

#	Descrizione	L1	L2	L3	CWE	NIST §
2.1.7	Verificare che le password inviate durante la registrazione dell'account, l'accesso e il cambio password vengano confrontate con un set di password violate, sia localmente (come le prime 1.000 o 10.000 password più comuni che corrispondono alla policy password del sistema) o utilizzando un'API esterna. Se si utilizza un'API, è necessario utilizzare zero-knowledge proof o un altro meccanismo per garantire che la password non venga inviata o utilizzata in chiaro per verificare il suo stato di violazione. Se la password è violata, l'applicazione deve richiedere all'utente di impostarne una nuova. (C6)	✓	✓	✓	521	5.1.1.2
2.1.8	Verificare che venga fornito un misuratore di robustezza della password per aiutare gli utenti a impostare una password più sicura.	✓	✓	✓	521	5.1.1.2
2.1.9	Verificare che non esistano regole di composizione password che limitino il tipo di caratteri consentiti. Non dovrebbe essere richiesto l'utilizzo di maiuscole, minuscole, numeri o caratteri speciali. (C6)	✓	✓	✓	521	5.1.1.2
2.1.10	Verificare che non vi siano requisiti periodici di rotazione delle credenziali o cronologia delle password.	✓	✓	✓	263	5.1.1.2
2.1.11	Verificare che siano consentite le funzionalità "incolla", gli assistenti password del browser e i gestori di password esterni.	✓	✓	✓	521	5.1.1.2
2.1.12	Verificare che l'utente possa scegliere di visualizzare temporaneamente l'intera password mascherata o l'ultimo carattere digitato su piattaforme che non dispongono nativamente di questa funzionalità.	✓	✓	✓	521	5.1.1.2

Nota: L'obiettivo di consentire all'utente di visualizzare temporaneamente la propria password o l'ultimo carattere digitato è migliorare l'usabilità nell'inserimento delle credenziali, soprattutto quando si utilizzano password lunghe, passphrase o gestori di password. Un ulteriore motivo per includere questo requisito è evitare che, in fase di test, si chieda inutilmente alle organizzazioni di sovrascrivere il comportamento predefinito del campo password della piattaforma, eliminando una moderna funzionalità di sicurezza che risulta user-friendly.

V2.2 Sicurezza Generale degli Authenticator

La flessibilità degli authenticator è essenziale per la sicurezza a lungo termine delle applicazioni. È necessario svolgere un refactor dell'applicazione per consentire l'utilizzo di authenticator aggiuntivi in base alle preferenze dell'utente, nonché per consentire la rimozione graduale di authenticator obsoleti o non sicuri.

Il NIST considera email e SMS come tipi di authenticator "restricted", ed è probabile che vengano rimossi da NIST 800-63 e di conseguenza in futuro anche dall'ASVS. Le applicazioni dovrebbero avere una roadmap che non richieda l'utilizzo di email o SMS.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.2.1	Verificare che i controlli anti-automazione siano efficaci nel mitigare gli attacchi relativi a credenziali compromesse, forza bruta e blocco dell'account. Tali controlli includono il blocco delle password violate più comuni, i blocchi temporanei, la limitazione della velocità, i CAPTCHA, ritardi sempre crescenti tra i tentativi, restrizioni degli indirizzi IP o restrizioni basate sul rischio come posizione, primo accesso su un dispositivo, tentativi recenti di sbloccare l'account o simili. Verificare che non siano possibili più di 100 tentativi falliti all'ora su un singolo account.	✓	✓	✓	307	5.2.2 / 5.1.1.2 / 5.1.4.2 / 5.1.5.2
2.2.2	Verificare che l'uso di authenticator deboli (come SMS ed email) sia limitato alla verifica secondaria o all'approvazione delle transazioni e non come sostituto di metodi di autenticazione più sicuri. Verificare che vengano offerti in primis metodi più robusti, che gli utenti siano consapevoli dei rischi o che siano implementate misure adeguate per limitare i rischi di compromissione dell'account.	✓	✓	✓	304	5.2.10
2.2.3	Verificare che agli utenti vengano inviate notifiche dopo aggiornamenti dei dettagli di autenticazione, come reset delle credenziali, modifiche dell'email o dell'indirizzo, accessi da posizioni sconosciute o rischiose. Si preferisca l'uso di notifiche push, piuttosto che SMS o email, ma in assenza di notifiche push, SMS o email sono accettabili purché non venga divulgata nessuna informazione sensibile.	✓	✓	✓	620	
2.2.4	Verificare la resistenza all'impersonificazione tramite phishing, come l'utilizzo dell'autenticazione a due fattori, dispositivi crittografici (come chiavi connesse con un pulsante per l'autenticazione) o, a livelli AAL più alti, certificati lato client.			✓	308	5.2.5
2.2.5	Verificare che quando un Credential Service Provider (CSP) e l'applicazione che verifica l'autenticazione sono separati, sia presente un mutual TLS tra i due endpoint.			✓	319	5.2.6
2.2.6	Verificare la resistenza al replay mediante l'uso obbligatorio di dispositivi OTP (One-Time Password), autenticatori crittografici o codici di lookup.			✓	308	5.2.8
2.2.7	Verificare l'autenticazione richiedendo l'inserimento di un token OTP o un'azione iniziata dall'utente come la pressione di un pulsante su una chiave hardware FIDO.			✓	308	5.2.9

V2.3 Ciclo di Vita degli Authenticator

Gli *authenticator* includono password, *soft token* (token digitali software), *hardware token* (token digitali fisici) e dispositivi biometrici. Il ciclo di vita degli *authenticator* è cruciale per garantire la sicurezza di un'applicazione. Se chiunque potesse registrarsi autonomamente senza una verifica dell'identità, la fiducia nell'identità sarebbe compromessa. Per piattaforme come Reddit, questo livello di sicurezza può essere accettabile. Tuttavia, per sistemi bancari e altre applicazioni sensibili, è fondamentale prestare particolare attenzione durante la fase di registrazione e l'emissione di credenziali o dispositivi, per assicurare la sicurezza dell'applicazione.

Nota: alle password non deve essere assegnata una scadenza massima, né devono essere soggette a rotazione periodica. Le password devono essere verificate per eventuali compromissioni, non sostituite regolarmente.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.3.1	Verificare che le password iniziali o i codici di attivazione generate dal sistema SIANO casuali e sicuri, DEVONO essere lunghi almeno 6 caratteri e POSSONO contenere lettere e numeri, con scadenza dopo un breve periodo di tempo. Non è consentito l'utilizzo a lungo termine di questi segreti iniziali.	✓	✓	✓	330	5.1.1.2 / A.3
2.3.2	Verificare che sia supportata la registrazione e l'utilizzo di dispositivi di autenticazione forniti dall'utente, come token U2F o FIDO.		✓	✓	308	6.1.3
2.3.3	Verificare che le istruzioni di rinnovo siano inviate con sufficiente anticipo per permettere il rinnovo degli autenticatori con scadenza.		✓	✓	287	6.1.4

V2.4 Memorizzazione delle credenziali

Questa sezione è rivolta ad architetti e sviluppatori impegnati nella creazione o ristrutturazione del codice. La verifica completa può essere effettuata solo attraverso la revisione del codice sorgente o tramite unit test o integration test specifici per la sicurezza. I penetration test non coprono nessuno di questi aspetti.

La lista della funzioni one-way per la derivazione delle chiavi è dettagliata nella sezione 5.1.1.2 del NIST 800-63 B, e nel BSI Kryptographische Verfahren: Empfehlungen und Schlussellängen (2018). In alternativa, si possono utilizzare gli standard nazionali o regionali più aggiornati per algoritmi e lunghezza delle chiavi.

Questa sezione non può essere soggetta a penetration test, pertanto i controlli non sono classificati come L1. Tuttavia, è cruciale per la sicurezza delle credenziali in caso di furto. Se si esegue il fork del ASVS per delineare un'architettura o linee guida di sviluppo sicuro o un elenco di controlli per la revisione del codice sorgente, si consiglia di riportare questi controlli a L1 nella propria versione privata.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.4.1	Verificare che le password siano archiviate in una forma resistente agli attacchi offline. Le password DEVONO essere sottoposte ad hashing e sale usando funzioni approvate di derivazione di chiavi unidirezionale o di hashing delle password. Le funzioni di derivazione di chiavi e di hashing delle password utilizzano come input una password, un sale e un fattore di costo per generare un hash della password. (C6)		✓	✓	916	5.1.1.2
2.4.2	Verificare che il sale sia lungo almeno 32 bit e scelto in modo arbitrario per ridurre al minimo le collisioni relative al sale tra gli hash memorizzati. Per ogni credenziale, DEVONO essere archiviati un valore di sale univoco e l'hash risultante. (C6)		✓	✓	916	5.1.1.2
2.4.3	Verificare che se viene utilizzato PBKDF2, il numero di iterazioni DOVREBBE essere il più alto consentito dalle prestazioni del server, in genere almeno 100.000 iterazioni. (C6)		✓	✓	916	5.1.1.2
2.4.4	Verificare che se viene utilizzato bcrypt, il fattore di lavoro DOVREBBE essere il più alto consentito dalle prestazioni del server, con un minimo di 10. (C6)		✓	✓	916	5.1.1.2

#	Descrizione	L1	L2	L3	CWE	NIST §
2.4.5	Verificare che venga eseguita un'ulteriore iterazione di una funzione di derivazione di chiavi, utilizzando un valore di sale segreto noto solo al verificatore. Generare il valore di sale utilizzando un generatore di bit casuali approvato [SP 800-90Ar1] e fornire almeno il livello minimo di robustezza specificato nell'ultima revisione di SP 800-131A. Il valore segreto del sale DEVE essere archiviato separatamente dalle password con hash (ad esempio, in un dispositivo specializzato come un modulo di sicurezza hardware).		✓	✓	916	5.1.1.2

Laddove sono menzionati gli standard US, possono essere utilizzati, in alternativa, standard regionali o locali.

V2.5 Recupero delle Credenziali

#	Descrizione	L1	L2	L3	CWE	NIST §
2.5.1	Verificare che un segreto di attivazione iniziale o di recupero generato dal sistema non venga inviato all'utente in chiaro. (C6)	✓	✓	✓	640	5.1.1.2
2.5.2	Verificare che i suggerimenti per la password o l'autenticazione basata sulla conoscenza (le cosiddette "domande segrete") non siano presenti.	✓	✓	✓	640	5.1.1.2
2.5.3	Verificare che il recupero delle credenziali tramite password non riveli in alcun modo la password corrente. (C6)	✓	✓	✓	640	5.1.1.2
2.5.4	Verificare che non siano presenti account condivisi o di default (ad esempio "root", "admin" o "sa").	✓	✓	✓	16	5.1.1.2 / A.3
2.5.5	Verificare che se un fattore di autenticazione viene modificato o sostituito, l'utente ne venga informato.	✓	✓	✓	304	6.1.2.3
2.5.6	Verificare che la password dimenticata e gli altri percorsi di recupero utilizzino un meccanismo di recupero sicuro, come OTP basato sul tempo (TOTP) o un altro soft token, push mobile o un altro meccanismo di recupero offline (C6)	✓	✓	✓	640	5.1.1.2
2.5.7	Verificare che in caso di perdita di OTP o di meccanismi di autenticazione a più fattori, venga effettuata una verifica dell'identità comparabile al livello di quella effettuata durante la registrazione.		✓	✓	308	6.1.2.3

V2.6 Verifica con Look-up Secret

I look up secrets sono liste pre-generate di codici segreti, simili ai Numeri di Autorizzazione alle Transazioni (TAN), ai codici di recupero dei social media o a una griglia contenente un set di valori casuali. Questi codici vengono distribuiti in modo sicuro agli utenti. Vengono utilizzati una sola volta e, una volta esauriti, la lista dei look up secret viene scartata. Questo tipo di autenticatore appartiene alla categoria di "qualcosa che possiedi".

#	Descrizione	L1	L2	L3	CWE	NIST §
2.6.1	Verificare che i look up secrets possano essere utilizzati una sola volta.		✓	✓	308	5.1.2.2
2.6.2	Verificare che i lookup secrets abbiano un'entropia sufficiente (112 bit) oppure, se inferiore a 112 bit, siano salati con un valore casuale unico di 32 bit e soggetti ad una funzione hash unidirezionale approvata.		✓	✓	330	5.1.2.2

#	Descrizione	L1	L2	L3	CWE	NIST §
2.6.3	Verificare che i lookup secrets siano resistenti agli attacchi offline, che non siano facilmente prevedibili.		✓	✓	310	5.1.2.2

V2.7 Verifica Out of Band

In passato, un metodo comune per la verifica out of band era inviare un'email o un SMS con un link per reimpostare la password. Tuttavia, questo meccanismo debole è stato sfruttato dagli attaccanti, che possono prendere il controllo dell'account email della vittima e riutilizzare il link di reset per impadronirsi dell'account. Oggi esistono soluzioni più sicure per gestire la verifica out of band.

I meccanismi sicuri di autenticazione out of band includono dispositivi fisici in grado di comunicare con il verificatore attraverso un canale secondario sicuro, come le notifiche push su dispositivi mobili. Questo tipo di autenticatore rientra nella categoria "qualcosa che possiedi". Quando un utente tenta di autenticarsi, l'applicazione verificatrice invia un messaggio al verificatore out of band tramite una connessione sicura diretta o indiretta attraverso un servizio di terze parti. Il messaggio contiene un codice di autenticazione, solitamente un numero casuale a sei cifre o una finestra di dialogo di approvazione. Il verificatore confronta l'hash del codice ricevuto tramite il canale primario con quello del codice originale. Se corrispondono, l'utente è considerato autenticato.

L'ASVS presuppone che raramente gli sviluppatori creeranno nuovi verificatori out of band, come le notifiche push. Pertanto, i seguenti controlli ASVS si applicano a verificatori esistenti, come API di autenticazione, applicazioni e implementazioni di single sign-on. Se si sta sviluppando un nuovo autenticatore out of band, fare riferimento a NIST 800-63B § 5.1.3.1.

Verificatori out of band insicuri, come email e VOIP, non sono consentiti. L'autenticazione PSTN e SMS è attualmente "restricted" dal NIST e dovrebbe essere abbandonata a favore di notifiche push o simili. Se è necessario utilizzare l'autenticazione out of band telefonica o SMS, consultare il § 5.1.3.3.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.7.1	Verificare che gli autenticatori out of band in chiaro (NIST "restricted"), come SMS o PSTN, non siano offerti per impostazione predefinita e che vengano prima offerte alternative più sicure come le notifiche push.	✓	✓	✓	287	5.1.3.2
2.7.2	Verificare che il meccanismo out of band faccia scadere le richieste di autenticazione, i codici o i token dopo 10 minuti.	✓	✓	✓	287	5.1.3.2
2.7.3	Verificare che le richieste di autenticazione, i codici o i token del verificatore fuori banda siano utilizzabili una sola volta e solo per la richiesta di autenticazione originale.	✓	✓	✓	287	5.1.3.2
2.7.4	Verificare che il meccanismo out of band di autenticazione e di verifica comunichino su un canale indipendente sicuro.	✓	✓	✓	523	5.1.3.2
2.7.5	Verificare che il meccanismo out of band conservi solo una versione con hash del codice di autenticazione.		✓	✓	256	5.1.3.2
2.7.6	Verificare che il codice di autenticazione iniziale sia generato da un generatore di numeri casuali sicuro, contenente almeno 20 bit di entropia (in genere è sufficiente un numero casuale a sei cifre).		✓	✓	310	5.1.3.2

V2.8 Verifica Monouso

Le password monouso (OTP) monofattore sono token software o fisici che visualizzano un segreto monouso pseudo-casuale che cambia continuamente. Questi dispositivi rendono il phishing (impersonificazione) difficile, ma non impossibile. Questo tipo di autenticazione appartiene alla categoria "qualcosa che possiedi". I token multifattore, simili agli OTP monofattore, richiedono ulteriori procedure di sicurezza per generare l'OTP finale.

Queste includono l'inserimento di un codice PIN valido, l'autenticazione biometrica, la connessione tramite USB o NFC, o l'aggiunta di un valore supplementare, come quello generato da una transaction signing calculator.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.8.1	Verificare che gli OTP basati sul tempo abbiano una durata definita prima della scadenza.	✓	✓	✓	613	5.1.4.2 / 5.1.5.2
2.8.2	Verificare che le chiavi simmetriche utilizzate per verificare gli OTP inviati siano protette adeguatamente, ad esempio utilizzando un modulo di sicurezza hardware (HSM) o un key storage sicuro fornito dal sistema operativo.		✓	✓	320	5.1.4.2 / 5.1.5.2
2.8.3	Verificare che nella generazione, nell'inizializzazione e nella verifica degli OTP vengano utilizzati algoritmi crittografici approvati.		✓	✓	326	5.1.4.2 / 5.1.5.2
2.8.4	Verificare che gli OTP monouso basati sul tempo possano essere utilizzati una sola volta entro il periodo di validità.		✓	✓	287	5.1.4.2 / 5.1.5.2
2.8.5	Verificare che se un token OTP multi-fattore basato sul tempo viene riutilizzato durante il periodo di validità, venga registrato e rifiutato con l'invio di una notifica al detentore del dispositivo.		✓	✓	287	5.1.5.2
2.8.6	Verificare che il dispositivo hardware generatore di codici OTP possa essere revocato in caso di furto o smarrimento. Garantire che la revoca sia immediata su tutte le sessioni aperte, indipendentemente dalla posizione.		✓	✓	613	5.2.1
2.8.7	Verificare che gli autenticatori biometrici siano limitati all'uso solo come fattori secondari in combinazione con qualcosa che si possiede e qualcosa che si conosce.		o	✓	308	5.2.3

V2.9 Verifica Crittografica

Le chiavi di sicurezza crittografiche, come le smart card o le chiavi FIDO, richiedono che l'utente colleghi o associ il dispositivo crittografico al computer per completare l'autenticazione. Il verificatore invia un *nonce* (numero casuale) al dispositivo o software crittografico, che risponde calcolando una risposta basata su una chiave crittografica memorizzata in modo sicuro.

I requisiti per i dispositivi e i software crittografici, sia monofattore che multifattore, sono gli stessi, poiché la verifica dell'autenticatore crittografico dimostra il possesso dell'autenticatore stesso.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.9.1	Verificare che le chiavi crittografiche utilizzate nella verifica siano archiviate in modo sicuro e protette dalla divulgazione, ad esempio utilizzando un Trusted Platform Module (TPM) o un Hardware Security Module (HSM), o un servizio del sistema operativo in grado di utilizzare questi meccanismi.		✓	✓	320	5.1.7.2
2.9.2	Verificare che il nonce sia di almeno 64 bit e statisticamente unico o unico per l'intera durata di vita del dispositivo crittografico.		✓	✓	330	5.1.7.2
2.9.3	Verificare che vengano utilizzati algoritmi crittografici approvati per la generazione, l'inizializzazione e la verifica.		✓	✓	327	5.1.7.2

V2.10 Autenticazione del Servizio

Questa sezione non può essere verificata tramite penetration test, quindi non include requisiti L1. Tuttavia, se applicata nell'ambito di un'architettura, nella scrittura o nella revisione del codice sicuro, è consigliabile trattare il software di gestione delle chiavi (come Java Key Store) come requisito minimo a livello L1. La memorizzazione in chiaro di segreti non è accettabile in nessun caso.

#	Descrizione	L1	L2	L3	CWE	NIST §
2.10.1	Verificare che i segreti interni al servizio non si basino su credenziali statiche come password, chiavi API o account condivisi con accesso privilegiato.		OS assisted	HSM	287	5.1.1.1
2.10.2	Verificare che, se sono richieste password per l'autenticazione del servizio, l'account di servizio utilizzato non sia uno predefinito. (ad esempio root/root o admin/admin sono predefiniti in alcuni servizi durante l'installazione).		OS assisted	HSM	255	5.1.1.1
2.10.3	Verificare che le password siano archiviate con una protezione sufficiente per prevenire attacchi di recupero offline, incluso l'accesso al sistema locale.		OS assisted	HSM	522	5.1.1.1
2.10.4	Verificare che password, integrazioni con database e sistemi di terze parti, seed e segreti interni e chiavi API siano gestiti in modo sicuro e non siano inclusi nel codice sorgente o archiviati all'interno dei repository del codice sorgente. Tale archiviazione DOVREBBE resistere agli attacchi offline. Si raccomanda l'utilizzo di un archivio software sicuro delle chiavi (L1), hardware TPM o un HSM (L3) per l'archiviazione delle password.		OS assisted	HSM	798	

Requisiti aggiuntivi per le agenzie statunitensi

Le agenzie statunitensi hanno requisiti obbligatori relativi a NIST 800-63. L'Application Security Verification Standard ha sempre riguardato l'80% dei controlli che si applicano a quasi il 100% delle applicazioni, e non l'ultimo 20% dei controlli avanzati o quelli con applicabilità limitata. Pertanto, l'ASVS è un sottoinsieme rigoroso di NIST 800-63, specialmente per le classificazioni IAL1/2 e AAL1/2, ma non è sufficientemente completo, in particolare per le classificazioni IAL3/AAL3.

Esortiamo vivamente le agenzie governative statunitensi a rivedere e implementare NIST 800-63 nella sua interezza.

Glossario

Termine	Significato
CSP	Credential Service Provider also called an Identity Provider, un servizio che gestisce le identità degli utenti e fornisce credenziali di autenticazione.
Authenticator	Componente che verifica le credenziali presentate dall'utente. Può gestire diversi fattori come password, token, autenticazione a due fattori (MFA) o accessi federati.
Verifier	Questa entità verifica la validità delle credenziali dell'utente in base alle informazioni ricevute dall'autenticatore. Potrebbe anche confermare se le credenziali sono associate a un utente valido e se sono attive/non revocate.
OTP	One-time password. Password valida per un solo tentativo di accesso e poi scade

Termine	Significato
SFA	Single-factor authenticators, come qualcosa che conosci (segreti, password, frasi segrete, PINs), qualcosa che sei (biometria, impronte digitali, scansioni facciali), o qualcosa che possiedi (token OTP, un dispositivo crittografico come una smart card).
MFA	Multi-factor authentication, che include almeno due fattori

Riferimenti

Per approfondimenti, consultare:

- NIST 800-63 - Digital Identity Guidelines
- NIST 800-63 A - Enrollment and Identity Proofing
- NIST 800-63 B - Authentication and Lifecycle Management
- NIST 800-63 C - Federation and Assertions
- NIST 800-63 FAQ
- OWASP Testing Guide 4.0: Testing for Authentication
- OWASP Cheat Sheet - Password storage
- OWASP Cheat Sheet - Forgot password
- OWASP Cheat Sheet - Choosing and using security questions

V3 Gestione delle sessioni

Obiettivo del controllo

Uno degli elementi fondamentali di qualsiasi applicazione web o API stateful è il meccanismo che gestisce e mantiene lo stato di un utente o dispositivo durante l'interazione con essa. La gestione delle sessioni consente di trasformare un protocollo senza stato in uno con stato, ed è essenziale per distinguere i vari utenti o dispositivi.

Verificare che un'applicazione soddisfi i seguenti requisiti di alto livello per la gestione delle sessioni:

- Le sessioni sono uniche per ciascun individuo e non possono essere né indovinate né condivise.
- Le sessioni vengono invalidate quando non più necessarie e scadono dopo un periodo di inattività.

Come già osservato, questi requisiti sono stati adattati per essere un sottoinsieme conforme ai controlli selezionati dal NIST 800-63b, concentrandosi su minacce comuni e vulnerabilità di autenticazione frequentemente sfruttate. I requisiti di verifica precedenti sono stati eliminati, deduplicati o, nella maggior parte dei casi, adattati per allinearsi strettamente con le disposizioni obbligatorie del NIST 800-63b .

Requisiti di Verifica della Sicurezza

V3.1 Sicurezza Fondamentale della Gestione Sessione

#	Descrizione	L1	L2	L3	CWE	NIST §
3.1.1	Verificare che l'applicazione non riveli mai token di sessione nei parametri dell'URL.	✓	✓	✓	598	

V3.2 Associazione di Sessione

#	Descrizione	L1	L2	L3	CWE	NIST §
3.2.1	Verificare che l'applicazione generi un nuovo token di sessione all'autenticazione dell'utente. (C6)	✓	✓	✓	384	7.1
3.2.2	Verificare che i token di sessione possiedano almeno 64 bit di entropia. (C6)	✓	✓	✓	331	7.1
3.2.3	Verificare che l'applicazione memorizzi i token di sessione solo nel browser utilizzando metodi sicuri come cookie adeguatamente protetti (vedere sezione 3.4) o il session storage in HTML5	✓	✓	✓	539	7.1
3.2.4	Verificare che i token di sessione siano generati utilizzando algoritmi crittografici approvati. (C6)		✓	✓	331	7.1

L'uso di TLS o di un canale di trasporto sicuro equivalente è obbligatorio per la gestione delle sessioni. Questo aspetto sarà trattato nel capitolo dedicato alla sicurezza delle comunicazioni.

V3.3 Terminazione Sessione

I timeout di sessione sono stati allineati con il NIST 800-63, che consente durate molto più lunghe rispetto a quelle tradizionalmente previste dagli standard di sicurezza. Le organizzazioni dovrebbero consultare la tabella seguente e, se il rischio associato all'applicazione giustifica un timeout più lungo, il valore NIST dovrebbe rappresentare il limite massimo per i timeout di inattività della sessione.

In questo contesto, L1 corrisponde a IAL1/AAL1, L2 a IAL2/AAL2 e L3 a IAL3/AAL3. Per IAL2/AAL2 e IAL3/AAL3, un timeout di inattività più breve implica un limite inferiore per la disconnessione o la necessità di riautenticazione per riprendere la sessione.

#	Descrizione	L1	L2	L3	CWE	NIST §
3.3.1	Verificare che la disconnessione e la scadenza invalidino il token di sessione, in modo che il pulsante Indietro o una relying party a valle non riprendano una sessione autenticata, anche tra relying party. (C6)	✓	✓	✓	613	7.1
3.3.2	Se gli autenticatori consentono agli utenti di rimanere collegati, verificare che la riautenticazione avvenga periodicamente sia quando viene utilizzata attivamente sia dopo un periodo di inattività. (C6)	30 giorni	12 ore o 30 minuti di inattività, 2FA opzionale	12 ore o 15 minuti di inattività, con 2FA	613	7.2
3.3.3	Verificare che l'applicazione consenta di terminare tutte le altre sessioni attive dopo un cambio password riuscito (incluso il cambio tramite reset/recupero password) e che ciò si applichi all'intera l'applicazione, all'accesso federato (se presente) e a qualsiasi relying party.		✓	✓	613	
3.3.4	Verificare che gli utenti possano visualizzare e (dopo aver reinserito le credenziali di accesso) disconnettersi da una o tutte le sessioni e i dispositivi attualmente attivi.		✓	✓	613	7.1

V3.4 Gestione Sessioni Basate su Cookie

#	Descrizione	L1	L2	L3	CWE	NIST §
3.4.1	Verificare che i token di sessione basati su cookie abbiano impostato l'attributo 'Secure'. (C6)	✓	✓	✓	614	7.1.1
3.4.2	Verificare che i token di sessione basati su cookie abbiano impostato l'attributo 'HttpOnly'. (C6)	✓	✓	✓	1004	7.1.1
3.4.3	Verificare che i token di sessione basati su cookie utilizzino l'attributo 'SameSite' per limitare l'esposizione ad attacchi di Cross-Site Request Forgery (CSRF). (C6)	✓	✓	✓	16	7.1.1
3.4.4	Verificare che i token di sessione basati su cookie utilizzino il prefisso "__Host-" in modo che i cookie vengano inviati solo all'host che ha inizialmente impostato il cookie.	✓	✓	✓	16	7.1.1
3.4.5	Verificare che se l'applicazione viene pubblicata su un nome di dominio con altre applicazioni che impostano o utilizzano cookie di sessione che potrebbero rivelare i cookie di sessione, che sia presente l'attributo 'path' nei token di sessione utilizzando il percorso più specifico possibile. (C6)	✓	✓	✓	16	7.1.1

V3.5 Gestione Sessioni Basate su Token

La gestione delle sessioni basate su token include JWT, OAuth, SAML e API key. Tra queste, le API key sono note per essere deboli e non dovrebbero essere utilizzate nelle implementazioni moderne.

#	Descrizione	L1	L2	L3	CWE	NIST §
3.5.1	Verificare che l'applicazione consenta agli utenti di revocare i token OAuth che creano relazioni di fiducia verso le applicazioni collegate.		✓	✓	290	7.1.2
3.5.2	Verificare che l'applicazione utilizzi token di sessione anziché segreti e API key statici, salvo nelle implementazioni legacy.		✓	✓	798	
3.5.3	Verificare che i token di sessione stateless utilizzino firme digitali, crittografia e altre contromisure per proteggersi da manomissioni, enveloping, replay, null cipher, attacchi di key substitution.		✓	✓	345	

V3.6 Riautenticazione Federata

Questa sezione riguarda coloro che scrivono codice per Relying Party (RP) o Credential Service Provider (CSP). Se ci si affida a codice che implementa queste funzionalità, assicurarsi che questi problemi vengano gestiti correttamente.

#	Descrizione	L1	L2	L3	CWE	NIST §
3.6.1	Verificare che i Relying Party (RP) specifichino il tempo massimo di autenticazione ai Credential Service Provider (CSP) e che i CSP richiedano una nuova autenticazione dell'utente se è stata utilizzata una sessione entro tale periodo.			✓	613	7.2.1
3.6.2	Verificare che i Credential Service Provider (CSP) informino i Relying Party (RP) dell'ultimo evento di autenticazione, per consentire ai RP di determinare se è necessario richiedere una nuova autenticazione dell'utente.			✓	613	7.2.1

V3.7 Difese contro gli Exploit di Gestione Sessione

Esistono pochi attacchi noti relativi alla gestione delle sessioni, alcuni dei quali sono legati all'esperienza utente (UX) con le sessioni. In passato, l'ASVS, seguendo i requisiti ISO 27002, imponeva il blocco delle sessioni simultanee multiple. Tuttavia, questo approccio non è più appropriato, sia perché gli utenti moderni utilizzano diversi dispositivi o perché l'applicazione è un'API senza sessione del browser. Inoltre, in molte implementazioni, l'ultimo autenticatore prevale, il che spesso avvantaggia l'attaccante. Questa sezione offre una guida avanzata per scoraggiare, ritardare e rilevare gli attacchi di gestione delle sessioni a livello di codice.

Descrizione dell'Attacco Half-Open

Nel 2018, diversi istituti finanziari sono stati vittime di compromissioni attraverso quello che è noto come "attacco half-open" (semi-aperto), un termine che è rimasto in uso nel settore. Gli attaccanti hanno sfruttato vulnerabilità comuni a diversi sistemi di autenticazione, gestione delle sessioni e controllo degli accessi, anche all'interno delle stesse istituzioni.

L'attacco half-open inizia con il tentativo di bloccare, reimpostare o recuperare una credenziale. Molti sistemi di gestione delle sessioni utilizzano un pattern di progettazione che riutilizza gli oggetti o modelli di sessione del profilo utente tra stati di autenticazione diversi: non autenticato, parzialmente autenticato (ad esempio per il ripristino della password o il recupero dell'username) e completamente autenticato. Questo approccio assegna un token di sessione valido contenente informazioni del profilo della vittima, inclusi hash delle password e ruoli. Se i controlli di accesso nei controller o nei router non verificano correttamente che l'utente sia completamente autenticato, l'attaccante può agire come se fosse l'utente. Le azioni dell'attaccante possono includere la modifica della password dell'utente con un valore noto, l'aggiornamento dell'indirizzo email per effettuare un ripristino password, la disabilitazione dell'autenticazione a due fattori, la registrazione di un nuovo dispositivo MFA, o la visualizzazione e modifica delle API key.

#	Descrizione	L1	L2	L3	CWE	NIST §
3.7.1	Verificare che l'applicazione garantisca una sessione di accesso completa e valida o richieda una riautenticazione o una verifica secondaria prima di consentire qualsiasi transazione sensibile o modifica dell'account.	✓	✓	✓	306	

Riferimenti

Per approfondimenti, consultare:

- OWASP Testing Guide 4.0: Session Management Testing
- OWASP Session Management Cheat Sheet
- Set-Cookie __Host- prefix details

V4 Controllo degli accessi

Obiettivo del controllo

Con il termine autorizzazione si intende il processo di consentire l'accesso alle risorse solo a coloro che sono autorizzati a utilizzarle. È essenziale che un'applicazione verificata soddisfi i seguenti requisiti di alto livello:

- Gli utenti che accedono alle risorse devono disporre di credenziali valide.
- Agli utenti devono essere associati ruoli e privilegi ben definiti.
- I metadati di ruoli e permessi devono essere protetti da attacchi di replay o manomissione (tampering).

Requisiti di verifica della sicurezza

V4.1 Progettazione generale del controllo degli accessi

#	Descrizione	L1	L2	L3	CWE
4.1.1	Verificare che l'applicazione applichi le regole di controllo degli accessi su un livello di servizio trusted, soprattutto se è presente il controllo degli accessi lato client che potrebbe essere bypassato.	✓	✓	✓	602
4.1.2	Verificare che tutti gli attributi utente e dati e le informazioni sulle policy utilizzate dai controlli di accesso non possano essere manipolati dagli utenti finali se non espressamente autorizzati.	✓	✓	✓	639
4.1.3	Verificare che esista il principio del privilegio minimo: gli utenti dovrebbero poter accedere solo a funzioni, file di dati, URL, controller, servizi e altre risorse per le quali possiedono un'autorizzazione specifica. Ciò implica una protezione contro lo spoofing e l'escalation di privilegi (C7)	✓	✓	✓	285
4.1.4	[ELIMINATO, DUPLICATO DI 4.1.3]				
4.1.5	Verificare che i controlli di accesso falliscano in modo sicuro anche in caso di eccezione. (C10)	✓	✓	✓	285

V4.2 Controllo degli accessi a livello di operazione

#	Descrizione	L1	L2	L3	CWE
4.2.1	Verificare che i dati sensibili e le API siano protetti da attacchi IDOR (Insecure Direct Object Reference). Questi attacchi mirano alla creazione, lettura, aggiornamento ed eliminazione di record, come ad esempio quelli appartenenti ad altri utenti, la visualizzazione di tutti i record o l'eliminazione di tutti i dati.	✓	✓	✓	639
4.2.2	Verificare che l'applicazione o il framework implementi un meccanismo anti-CSRF robusto per proteggere le funzionalità autenticate e che un meccanismo anti-automazione o anti-CSRF efficace protegga anche le funzionalità non autenticate.	✓	✓	✓	352

V4.3 Ulteriori considerazioni sul controllo degli accessi

#	Descrizione	L1	L2	L3	CWE
4.3.1	Verificare che le interfacce di amministrazione utilizzino un'adeguata autenticazione a due fattori per prevenire l'utilizzo non autorizzato.	✓	✓	✓	419

#	Descrizione	L1	L2	L3	CWE
4.3.2	Verificare che il directory listing sia disabilitato a meno che non sia espressamente richiesto. Inoltre, le applicazioni non devono consentire la scoperta o la divulgazione di metadati di file o directory, come cartelle Thumbs.db, .DS_Store, .git o .svn.	✓	✓	✓	548
4.3.3	Verificare che l'applicazione implementi un'autorizzazione aggiuntiva (step-up o adattiva) per i sistemi di basso valore e/o separazione dei compiti per le applicazioni di alto valore, al fine di applicare controlli antifrode in base al rischio e alle frodi passate.		✓	✓	732

Riferimenti

Per approfondimenti, consultare:

- OWASP Testing Guide 4.0: Authorization
- OWASP Cheat Sheet: Access Control
- OWASP CSRF Cheat Sheet
- OWASP REST Cheat Sheet

V5 Validazione, Sanificazione e Codifica

Obiettivo del controllo

La problematica di sicurezza più comune nelle applicazioni web è la mancanza di una valida convalida dell'input proveniente dal client o dall'ambiente, e l'assenza di codifica dell'output. Questa carenza è alla base delle vulnerabilità più gravi delle applicazioni web, come Cross-Site Scripting (XSS), SQL injection, injection di interprete, attacchi locale/Unicode, attacchi al file system e buffer overflow.

Per garantire la sicurezza dell'applicazione, è essenziale soddisfare i seguenti requisiti di alto livello:

- L'architettura di convalida dell'input e di codifica dell'output deve seguire procedure concordate per prevenire gli attacchi di injection.
- I dati di input devono essere fortemente tipizzati, validati, controllati per range o lunghezza, sanitizzati o filtrati.
- I dati di output devono essere codificati o "escaped" in base al contesto, il più vicino possibile all'interprete.

Con l'architettura moderna delle applicazioni web, la codifica dell'output è più importante che mai. In alcuni scenari è difficile fornire una convalida dell'input efficace, quindi l'utilizzo di API sicure come query parametrizzate, framework di templating con escaping automatico o una codifica dell'output accuratamente scelta è fondamentale per la sicurezza dell'applicazione.

V5.1 Input Validation

L'implementazione corretta dei controlli di convalida dell'input, utilizzando allow list e tipizzazione forte dei dati, può eliminare oltre il 90% di tutti gli attacchi di injection. I controlli di lunghezza e di range possono ridurre ulteriormente tali attacchi. Integrare una convalida dell'input sicura è fondamentale durante l'architettura dell'applicazione, gli sprint di progettazione, l'implementazione e i test unitari e di integrazione. Sebbene molti di questi aspetti non possano essere verificati durante i penetration test, i risultati della loro mancata implementazione si riflettono generalmente in V5.3 - Requisiti di codifica dell'output e prevenzione delle injection. Si consiglia a sviluppatori e revisori di codice sicuro di trattare questa sezione come se il livello L1 fosse obbligatorio per tutte le voci al fine di prevenire le injection.

#	Descrizione	L1	L2	L3	CWE
5.1.1	Verificare che l'applicazione disponga di difese contro gli attacchi di HTTP Parameter Pollution (HPP), soprattutto se il framework applicativo non distingue tra le sorgenti dei parametri di richiesta (GET, POST, cookie, header o variabili d'ambiente).	✓	✓	✓	235
5.1.2	Verificare che il framework protegga dagli attacchi di assegnazione di massa dei parametri (Mass Assignment) o che l'applicazione implementi contromisure, come la marcatura dei campi come privati, per prevenire l'assegnazione insicura dei parametri. (C5)	✓	✓	✓	915
5.1.3	Verificare che tutti gli input (campi di form HTML, richieste REST, parametri URL, header HTTP, cookie, file batch, feed RSS, ecc.) siano validati utilizzando la convalida positiva (liste positive o allow list). (C5)	✓	✓	✓	20
5.1.4	Verificare che i dati strutturati siano fortemente tipizzati e validati rispetto a uno schema definito, inclusi caratteri consentiti, lunghezza e pattern (ad esempio numeri di carta di credito, indirizzi e-mail, numeri di telefono o convalida di coerenza tra campi correlati, come la verifica della corrispondenza tra città e codice postale). (C5)	✓	✓	✓	20

#	Descrizione	L1	L2	L3	CWE
5.1.5	Verificare che i redirect e i forward degli URL consentano solo destinazioni presenti in una lista consentita, oppure che mostrino un avviso quando si reindirizza verso contenuti potenzialmente non sicuri.	✓	✓	✓	601

V5.2 Sanitization and Sandboxing

#	Descrizione	L1	L2	L3	CWE
5.2.1	Verificare che tutto l'input HTML non fidato proveniente da editor WYSIWYG o simili venga correttamente sanitizzato con una libreria o funzionalità del framework per la sanitizzazione HTML. (C5)	✓	✓	✓	116
5.2.2	Verificare che i dati non strutturati vengano sanitizzati per applicare misure di sicurezza come caratteri consentiti e vincoli sulla lunghezza.	✓	✓	✓	138
5.2.3	Verificare che l'applicazione sanitizzi l'input dell'utente prima di inviarlo ai sistemi di posta elettronica per proteggersi dall'injection di codice SMTP o IMAP.	✓	✓	✓	147
5.2.4	Verificare che l'applicazione eviti l'utilizzo di eval() o altre funzionalità di esecuzione dinamica del codice. Se non ci sono alternative, qualsiasi input utente incluso deve essere sanitizzato o eseguito in sandbox prima dell'esecuzione.	✓	✓	✓	95
5.2.5	Verificare che l'applicazione si protegga dagli attacchi di injection di template assicurando che qualsiasi input utente incluso venga sanitizzato o eseguito in sandbox.	✓	✓	✓	94
5.2.6	Verificare che l'applicazione si protegga dagli attacchi SSRF convalidando o sanificando dati non fidati o metadati di file HTTP, come nomi di file e campi di input URL, e utilizzando liste consentite di protocolli, domini, percorsi e porte.	✓	✓	✓	918
5.2.7	Verificare che l'applicazione sanitizzi, disabiliti o esegua in sandbox contenuti SVG (Scalable Vector Graphics) scriptabili forniti dall'utente, in particolare quelli relativi a XSS derivanti da script inline e foreignObject.	✓	✓	✓	159
5.2.8	Verificare che l'applicazione sanitizzi, disabiliti o esegua in sandbox contenuti forniti dall'utente in linguaggi di scripting o di templating come Markdown, fogli di stile CSS o XSL, BBCode o simili.	✓	✓	✓	94

V5.3 Codifica dell'output e prevenzione dalle injection

L'applicazione della codifica dell'output nelle immediate vicinanze o direttamente all'interno dell'interprete utilizzato è fondamentale per garantire la sicurezza di qualsiasi applicazione. Generalmente, la codifica dell'output non viene memorizzata in modo permanente, ma serve a rendere sicuro l'output nel contesto appropriato per un utilizzo immediato. La mancata codifica dell'output rende l'applicazione vulnerabile a iniezioni e potenzialmente pericolosa.

#	Descrizione	L1	L2	L3	CWE
5.3.1	Verificare che la codifica dell'output sia pertinente all'interprete e al contesto specifico. Ad esempio, utilizzare codificatori specifici per valori HTML, attributi HTML, JavaScript, parametri URL, header HTTP, SMTP e altri a seconda del contesto, soprattutto per input non fidati (ad esempio nomi con caratteri Unicode o apostrofi, come ねこ o O'Hara). (C4)	✓	✓	✓	116

#	Descrizione	L1	L2	L3	CWE
5.3.2	Verificare che la codifica dell'output preservi il set di caratteri e la località scelti dall'utente, in modo tale che qualsiasi punto del carattere Unicode sia valido e gestito in modo sicuro. (C4)	✓	✓	✓	176
5.3.3	Verificare che l'escape dell'output contestuale, preferibilmente automatico o, in casi limite, manuale, protegga da XSS riflessa, memorizzata e basata su DOM. (C4)	✓	✓	✓	79
5.3.4	Verificare che la selezione dei dati o le query del database (ad esempio SQL, HQL, ORM, NoSQL) utilizzino query parametrizzate, ORM, framework di entità o siano altrimenti protette dagli attacchi di injection del database. (C3)	✓	✓	✓	89
5.3.5	Verificare che, quando i meccanismi parametrizzati o più sicuri non siano presenti, venga utilizzata la codifica dell'output specifica del contesto per proteggersi dagli attacchi di injection, come l'utilizzo dell'escaping SQL per proteggersi dall'injection SQL. (C3, C4)	✓	✓	✓	89
5.3.6	Verificare che l'applicazione si protegga dagli attacchi di injection JSON, dagli attacchi eval JSON e dalla valutazione delle espressioni JavaScript. (C4)	✓	✓	✓	830
5.3.7	Verificare che l'applicazione si protegga dalle vulnerabilità di injection LDAP o che siano stati implementati controlli di sicurezza specifici per prevenire l'injection LDAP. (C4)	✓	✓	✓	90
5.3.8	Verificare che l'applicazione si protegga dall'injection di comandi del SO e che le chiamate al sistema operativo utilizzino query del SO parametrizzate o la codifica contestuale dell'output della riga di comando. (C4)	✓	✓	✓	78
5.3.9	Verificare che l'applicazione si protegga dagli attacchi di inclusione locale di file (LFI) o inclusione remota di file (RFI).	✓	✓	✓	829
5.3.10	Verificare che l'applicazione si protegga dagli attacchi di injection XPath o injection XML. (C4)	✓	✓	✓	643

Nota: L'utilizzo di query SQL parametrizzate o l'escaping del codice SQL non è sempre sufficiente a prevenire le injection. Nomi di tabelle, colonne, clausole ORDER BY e altri elementi simili non possono essere soggetti a escaping. L'inclusione di dati forniti dall'utente, anche se sottoposti a escaping, in questi campi può portare a query errate o a vulnerabilità di injection SQL.

Nota: Il formato SVG consente esplicitamente l'esecuzione di script ECMA in quasi tutti i contesti, il che potrebbe rendere impossibile bloccare completamente tutti i vettori XSS basati su SVG. Se il caricamento di file SVG è necessario, si raccomanda fortemente di servire tali file con il tipo MIME testo/plain o di utilizzare un dominio separato per i contenuti forniti dagli utenti, in modo da evitare che un attacco XSS riuscito possa compromettere l'intera applicazione.

V5.4 Memoria, Stringhe e Codice Non Gestito

I seguenti requisiti si applicano solo quando l'applicazione utilizza un linguaggio di sistema o codice non gestito.

#	Descrizione	L1	L2	L3	CWE
5.4.1	Verificare che l'applicazione utilizzi stringhe memory-safe, copie di memoria sicure e operazioni aritmetiche su puntatori per rilevare o prevenire overflow dello stack, del buffer o dell'heap.		✓	✓	120
5.4.2	Verificare che le format string non accettino input potenzialmente ostile e siano immutabili.		✓	✓	134
5.4.3	Verificare che vengano utilizzate tecniche di convalida del segno, dell'intervallo e dell'input per prevenire overflow di interi.		✓	✓	190

V5.5 Prevenzione dalla Deserializzazione

#	Descrizione	L1	L2	L3	CWE
5.5.1	Verificare che gli oggetti serializzati utilizzino controlli di integrità o siano crittografati per prevenire la creazione di oggetti malevoli o la manomissione dei dati. (C5)	✓	✓	✓	502
5.5.2	Verificare che l'applicazione limiti correttamente i parser XML a utilizzare solo la configurazione più restrittiva possibile e che funzionalità non sicure come la risoluzione di entità esterne vengano disabilitate per prevenire attacchi di injection di Entità Esterne XML (XXE).	✓	✓	✓	611
5.5.3	Verificare che la deserializzazione di dati non fidati venga evitata o protetta sia nel codice personalizzato che nelle librerie di terze parti (come parser JSON, XML e YAML).	✓	✓	✓	502
5.5.4	Verificare che durante l'analisi del JSON nei browser o nei backend basati su JavaScript, venga utilizzato JSON.parse per analizzare il documento JSON. Evitare di utilizzare eval() per analizzare il JSON.	✓	✓	✓	95

Riferimenti

Per approfondimenti, consultare:

- OWASP Testing Guide 4.0: Input Validation Testing
- OWASP Cheat Sheet: Input Validation
- OWASP Testing Guide 4.0: Testing for HTTP Parameter Pollution
- OWASP LDAP Injection Cheat Sheet
- OWASP Testing Guide 4.0: Client Side Testing
- OWASP Cross Site Scripting Prevention Cheat Sheet
- OWASP DOM Based Cross Site Scripting Prevention Cheat Sheet
- OWASP Java Encoding Project
- OWASP Mass Assignment Prevention Cheat Sheet
- DOMPurify - Client-side HTML Sanitization Library
- XML External Entity (XXE) Prevention Cheat Sheet

Per informazioni aggiuntive su auto-escaping, consultare:

- Reducing XSS by way of Automatic Context-Aware Escaping in Template Systems
- AngularJS Strict Contextual Escaping
- AngularJS ngBind
- Angular Sanitization
- Angular Security
- ReactJS Escaping
- Improperly Controlled Modification of Dynamically-Determined Object Attributes

Per informazioni aggiuntive su deserialization, consultare:

- OWASP Deserialization Cheat Sheet
- OWASP Deserialization of Untrusted Data Guide

V6 Crittografia per lo storage

Obiettivo del controllo

Verificare che un'applicazione soddisfi i seguenti requisiti di alto livello:

- Tutti i moduli crittografici falliscono in modo sicuro e gli errori vengono gestiti correttamente.
- Viene utilizzato un generatore di numeri casuali appropriato.
- L'accesso alle chiavi è gestito in modo sicuro.

V6.1 Classificazione dei Dati

L'asset più importante è il dato elaborato, memorizzato o trasmesso da un'applicazione. Eseguire sempre una valutazione di impatto sulla privacy per classificare correttamente le esigenze di protezione dei dati memorizzati.

#	Descrizione	L1	L2	L3	CWE
6.1.1	Verificare che i dati personali regolamentati siano memorizzati crittografati a riposo, come le Informazioni di identificazione personale (PII), le informazioni personali sensibili o i dati considerati soggetti al GDPR dell'UE.		✓	✓	311
6.1.2	Verificare che i dati sanitari regolamentati siano memorizzati crittografati a riposo, come cartelle cliniche, dettagli sui dispositivi medici o record di ricerca deanonimizzati.		✓	✓	311
6.1.3	Verificare che i dati finanziari regolamentati siano memorizzati crittografati a riposo, come conti finanziari, inadempienze o cronologie creditizie, registri fiscali, cronologie retributive, beneficiari o record di mercato o di ricerca deanonimizzati.		✓	✓	311

V6.2 Algoritmi

I recenti progressi nella crittografia hanno reso insicuri o insufficienti alcuni algoritmi e lunghezze delle chiavi che in passato erano considerati sicuri per proteggere i dati. Pertanto, dovrebbe essere possibile cambiare facilmente l'algoritmo utilizzato.

Sebbene questa sezione non sia facilmente verificabile tramite penetration testing, gli sviluppatori dovrebbero considerarla obbligatoria, anche se il livello 1 è assente nella maggior parte degli elementi.

#	Descrizione	L1	L2	L3	CWE
6.2.1	Verificare che tutti i moduli crittografici falliscano in modo sicuro e che gli errori vengano gestiti in modo tale da non consentire attacchi Oracle Padding.	✓	✓	✓	310
6.2.2	Verificare che vengano utilizzati algoritmi, modalità e librerie crittografiche comprovate dal settore o approvate dal governo, invece di implementarle da zero. (C8)		✓	✓	327
6.2.3	Verificare che il vettore di inizializzazione, la configurazione e le modalità dei cifrari a blocchi siano impostate in modo sicuro utilizzando le pratiche più recenti.		✓	✓	326
6.2.4	Verificare che numeri casuali, algoritmi di crittografia o hashing, lunghezze delle chiavi, round, cifrari o modalità possano essere riconfigurati, aggiornati o cambiati in qualsiasi momento, per proteggersi da violazioni crittografiche. (C8)		✓	✓	326

#	Descrizione	L1	L2	L3	CWE
6.2.5	Verificare che non vengano utilizzate modalità a blocchi insicure (ad esempio ECB, ecc.), modalità di padding insicure (ad esempio PKCS # 1 v1.5, ecc.), cifrari con dimensioni ridotte del blocco (ad esempio Triple-DES, Blowfish, ecc.) e algoritmi di hashing deboli (ad esempio MD5, SHA1, ecc.) a meno che non siano necessari per la compatibilità con versioni precedenti.		✓	✓	326
6.2.6	Verificare che i nonce, i vettori di inizializzazione e altri valori monouso non vengano riutilizzati con la stessa chiave crittografica. Il metodo di generazione deve essere appropriato per l'algoritmo utilizzato.		✓	✓	326
6.2.7	Verificare che i dati crittografati siano autenticati tramite firme, modalità di cifratura autenticate o HMAC per garantire che il testo cifrato non venga alterato da terze parti non autorizzate.			✓	326
6.2.8	Verificare che tutte le operazioni crittografiche vengano eseguite a tempo costante, senza operazioni di 'cortocircuito' durante i confronti, calcoli o restituzione di valori, per evitare di rivelare informazioni.			✓	385

V6.3 Valori casuali

La generazione di numeri pseudo-casuali (PRNG) veramente casuali è estremamente difficile da ottenere. In generale, le buone fonti di entropia all'interno di un sistema tendono a esaurirsi rapidamente se utilizzate in modo eccessivo, mentre l'uso di fonti con minore casualità può portare alla creazione di chiavi e segreti prevedibili.

#	Descrizione	L1	L2	L3	CWE
6.3.1	Verificare che tutti i numeri casuali, nomi di file, GUID e stringhe siano generati utilizzando un generatore di numeri casuali crittograficamente sicuro approvato dal modulo crittografico quando si desidera che questi valori non siano indovinabili da un attaccante.		✓	✓	338
6.3.2	Verificare che i GUID casuali siano creati utilizzando l'algoritmo GUID v4 e un generatore di numeri pseudo-casuali crittograficamente sicuro (CSPRNG). I GUID creati utilizzando altri generatori di numeri pseudo-casuali potrebbero essere prevedibili.		✓	✓	338
6.3.3	Verificare che i numeri casuali vengano creati con un'entropia adeguata anche quando l'applicazione è sotto carico elevato, oppure che l'applicazione si degradi adeguatamente in tali circostanze.			✓	338

V6.4 Gestione dei Segreti

Sebbene questa sezione non sia facilmente verificabile tramite penetration testing, gli sviluppatori dovrebbero considerarla obbligatoria nella sua interezza, anche se il livello 1 è assente nella maggior parte delle voci.

#	Descrizione	L1	L2	L3	CWE
6.4.1	Verificare che venga utilizzata una soluzione di gestione dei segreti, come un key vault, per creare, archiviare, controllare l'accesso e distruggere i segreti in modo sicuro. (C8)		✓	✓	798
6.4.2	Verificare che il materiale crittografico non venga esposto all'applicazione ma utilizzi invece un modulo di sicurezza isolato, come un vault, per le operazioni crittografiche. (C8)		✓	✓	320

Riferimenti

Per approfondimenti, consultare:

- OWASP Testing Guide 4.0: Testing for weak Cryptography
- OWASP Cheat Sheet: Cryptographic Storage
- FIPS 140-2

V7 Gestione degli errori e logging

Obiettivo del controllo

L'obiettivo principale della gestione degli errori e del logging è fornire informazioni utili all'utente, agli amministratori e ai team di risposta agli incidenti. Lo scopo non è quello di generare enormi quantità di log, ma piuttosto log di alta qualità, con un rapporto più elevato di "segnale" rispetto a "rumore" inutile.

I log di alta qualità spesso contengono dati sensibili e devono essere protetti in conformità con le leggi o direttive locali sulla privacy dei dati. Questo dovrebbe includere:

- Evitare di raccogliere o registrare informazioni sensibili a meno che non siano strettamente necessarie.
- Garantire che tutte le informazioni registrate siano gestite in modo sicuro e protette in base alla loro classificazione dei dati.
- Garantire che i log non vengano conservati indefinitamente, ma abbiano un periodo di conservazione il più breve possibile.

Se i log contengono dati privati o sensibili, la cui definizione può variare da paese a paese, essi diventano alcune delle informazioni più sensibili detenute dall'applicazione e quindi un obiettivo molto appetibile per gli attaccanti.

È inoltre fondamentale garantire che l'applicazione fallisca in modo sicuro e che gli errori non rivelino informazioni sensibili o non necessarie.

V7.1 Contenuto del Log

La registrazione di informazioni sensibili è rischiosa: i log stessi diventano dati riservati, il che implica che devono essere crittografati, soggetti a politiche di conservazione e resi disponibili durante gli audit di sicurezza. È essenziale garantire che nei log siano conservate solo le informazioni strettamente necessarie, evitando categoricamente informazioni di pagamento, credenziali (inclusi token di sessione), informazioni sensibili o dati personali identificabili.

La sezione V7.1 copre la voce A10 dell'OWASP Top 10 del 2017. Poiché né la voce A10 del 2017 né questa sezione sono facilmente verificabili tramite penetration testing, è importante:

- Per gli sviluppatori: garantire la piena conformità a questa sezione, trattando tutte le voci come se fossero contrassegnate come L1.
- Per i penetration tester: convalidare la piena conformità di tutte le voci in V7.1 tramite interviste, screenshot o dichiarazioni.

#	Descrizione	L1	L2	L3	CWE
7.1.1	Verificare che l'applicazione non registri credenziali o dettagli di pagamento. I token di sessione devono essere memorizzati nei log solo in forma di hash irreversibile. (C9, C10)	✓	✓	✓	532
7.1.2	Verificare che l'applicazione non registri altri dati sensibili come definiti dalle leggi sulla privacy locali o dalla relativa politica di sicurezza. (C9)	✓	✓	✓	532
7.1.3	Verificare che l'applicazione registri eventi rilevanti per la sicurezza, inclusi eventi di autenticazione riusciti e non riusciti, errori di controllo dell'accesso, errori di deserializzazione ed errori di convalida dell'input. (C5, C7)		✓	✓	778
7.1.4	Verificare che ogni evento di log includa le informazioni necessarie per consentire un'analisi dettagliata della cronologia degli eventi. (C9)		✓	✓	778

V7.2 Elaborazione dei Log

La registrazione tempestiva è fondamentale per gli eventi di audit, la classificazione e la gestione delle escalation. È necessario assicurarsi che i log dell'applicazione siano chiari e possano essere facilmente monitorati e analizzati localmente oppure inviati ad un sistema di monitoraggio remoto.

La sezione V7.2 copre la voce A10 dell'OWASP Top 10 2017. Poiché né la voce A10 del 2017 né questa sezione sono facilmente verificabili tramite penetration testing, è importante:

- Per gli sviluppatori: garantire la piena conformità a questa sezione, trattando tutte le voci come se fossero contrassegnate come L1.
- Per i penetration tester: convalidare la piena conformità di tutte le voci in V7.1 tramite interviste, screenshot o dichiarazioni.

#	Descrizione	L1	L2	L3	CWE
7.2.1	Verificare che tutti i tentativi di autenticazione vengano registrati, senza memorizzare token di sessione o password sensibili. Questo dovrebbe includere le richieste con metadati necessari per le indagini di sicurezza.		✓	✓	778
7.2.2	Verificare che tutte le decisioni di controllo dell'accesso possano essere registrate e che tutte le autorizzazioni negate vengano registrate. Questo dovrebbe includere le richieste con metadati pertinenti necessari per le indagini di sicurezza.		✓	✓	285

V7.3 Protezione dei log

I log che possono essere facilmente modificati o eliminati risultano inutili per indagini e le azioni legali. La divulgazione dei log può esporre dettagli interni sull'applicazione o sui dati in essa contenuti. È quindi necessario proteggere i log da divulgazione, modifica o eliminazione non autorizzate.

#	Descrizione	L1	L2	L3	CWE
7.3.1	Verificare che tutti i componenti di logging codifichino opportunamente i dati per prevenire l'injection di log. (C9)		✓	✓	117
7.3.2	[ELIMINATO, DUPLICATO DI 7.3.1]				
7.3.3	Verificare che i log di sicurezza siano protetti da accesso e da modifiche non autorizzate. (C9)		✓	✓	200
7.3.4	Verificare che le sorgenti temporali siano sincronizzate con l'ora e il fuso orario corretti. Valutare attentamente la registrazione solo in UTC se i sistemi sono globali per facilitare l'analisi forense post-incidente. (C9)		✓	✓	

Nota: la codifica dei log (7.3.1) è difficile da testare e revisionare utilizzando strumenti dinamici automatizzati e penetration test, ma architetti, sviluppatori e revisori del codice sorgente dovrebbero considerarla un requisito L1.

V7.4 Gestione degli Errori

Lo scopo della gestione degli errori è permettere all'applicazione di generare eventi rilevanti per la sicurezza, utili per monitoraggio, classificazione ed escalation. Non si tratta solo di creare log. Quando si registrano eventi di sicurezza, assicurarsi che i log abbiano uno scopo preciso e possano essere facilmente distinti da SIEM o software di analisi.

#	Descrizione	L1	L2	L3	CWE
7.4.1	Verificare che venga visualizzato un messaggio generico quando si verifica un errore imprevisto o relativo alla sicurezza, potenzialmente con un ID univoco che il personale di supporto può utilizzare per indagare. (C10)	✓	✓	✓	210
7.4.2	Verificare che la gestione delle eccezioni (o un equivalente funzionale) venga utilizzata in tutto il codice per tenere conto di condizioni di errore previste e impreviste. (C10)		✓	✓	544
7.4.3	Verificare che sia definito un gestore di errori "di ultima istanza" in grado di intercettare tutte le eccezioni non gestite (C10)		✓	✓	431

Nota: alcuni linguaggi, come Swift e Go, e molti linguaggi funzionali a causa delle comuni pratiche di progettazione, non supportano eccezioni o gestori di eventi di ultima istanza. In questo caso, architetti e sviluppatori devono utilizzare un modello, un linguaggio o un framework che consenta alle applicazioni di gestire in modo sicuro eventi eccezionali, imprevisti o relativi alla sicurezza.

Riferimenti

Per approfondimenti, consultare:

- OWASP Testing Guide 4.0 content: Testing for Error Handling
- OWASP Authentication Cheat Sheet section about error messages

V8 Protezione dei dati

Obiettivo del controllo

Esistono tre elementi chiave per una solida protezione dei dati: riservatezza, integrità e disponibilità (CIA). Questo standard presuppone che la protezione dei dati venga applicata su un sistema affidabile, come un server, opportunamente protetto e dotato di adeguate misure di sicurezza.

Le applicazioni devono presumere che tutti i dispositivi degli utenti possano essere in qualche modo compromessi. Quando un'applicazione trasmette o archivia informazioni sensibili su dispositivi non sicuri, come computer condivisi, telefoni e tablet, è responsabilità dell'applicazione garantire che i dati archiviati su tali dispositivi siano crittografati e non possano essere facilmente ottenuti, alterati o divulgati in modo illecito.

Verificare che un'applicazione soddisfi i seguenti requisiti di alto livello:

- Riservatezza: I dati devono essere protetti da lettura o divulgazione non autorizzate sia durante il trasferimento che durante l'archiviazione.
- Integrità: I dati devono essere protetti dalla creazione, alterazione o eliminazione malevola da parte di attori malintenzionati.
- Disponibilità: I dati devono essere disponibili all'occorrenza per gli utenti autorizzati.

V8.1 Protezione Generale dei Dati

#	Descrizione	L1	L2	L3	CWE
8.1.1	Verificare che l'applicazione impedisca la memorizzazione nella cache di dati sensibili in componenti del server come bilanciatori di carico e cache applicative.		✓	✓	524
8.1.2	Verificare che tutte le copie temporanee o memorizzate nella cache dei dati sensibili archiviate sul server siano protette dall'accesso non autorizzato o eliminate/invalidate dopo che l'utente autorizzato accede ai dati sensibili.		✓	✓	524
8.1.3	Verificare che l'applicazione riduca al minimo il numero di parametri in una richiesta, come campi nascosti, variabili Ajax, cookie e header.		✓	✓	233
8.1.4	Verificare che l'applicazione possa rilevare e allertare su un numero anomalo di richieste, ad esempio per IP, utente, totale per ora o giorno, o qualsiasi elemento rilevante per l'applicazione.		✓	✓	770
8.1.5	Verificare che vengano eseguiti backup regolari dei dati importanti e che venga eseguito un ripristino di prova dei dati.			✓	19
8.1.6	Verificare che i backup siano archiviati in modo sicuro per impedire il furto o la corruzione dei dati.			✓	19

V8.2 Protezione dei Dati lato Client

#	Descrizione	L1	L2	L3	CWE
8.2.1	Verificare che l'applicazione imposti header anti-caching sufficienti in modo che i dati sensibili non vengano memorizzati nella cache dei browser moderni.	✓	✓	✓	525
8.2.2	Verificare che i dati memorizzati nell'archiviazione del browser (come localStorage, sessionStorage, IndexedDB o cookie) non contengano dati sensibili.	✓	✓	✓	922
8.2.3	Verificare che i dati autenticati vengano cancellati dall'archiviazione client, come il DOM del browser, dopo la chiusura del client o della sessione.	✓	✓	✓	922

V8.3 Protezione dei Dati Sensibili e Privati

Questa sezione aiuta a proteggere i dati sensibili da accessi non autorizzati per creazione, lettura, aggiornamento o eliminazione, specialmente in grandi quantità.

La conformità a questa sezione implica anche la conformità ai controlli di accesso di V4, in particolare a V4.2. Ad esempio, per prevenire aggiornamenti o divulgazioni non autorizzate di informazioni personali sensibili, è necessario rispettare V4.2.1. Per una protezione completa, attenersi sia a questa sezione che a V4.

Nota: regolamenti e leggi sulla privacy, come gli Australian Privacy Principles (APP-11) o il GDPR, influenzano direttamente il modo in cui le applicazioni devono gestire l'archiviazione, l'uso e la trasmissione di informazioni personali sensibili. Questi possono variare da severe sanzioni a semplici linee guida. È consigliabile consultare le leggi e i regolamenti locali e, se necessario, rivolgersi a un esperto di privacy o a un avvocato specializzato.

#	Descrizione	L1	L2	L3	CWE
8.3.1	Verificare che i dati sensibili vengano inviati al server nel corpo del messaggio HTTP o negli header, e che i parametri della stringa di query di qualsiasi metodo HTTP non contengano dati sensibili.	✓	✓	✓	319
8.3.2	Verificare che gli utenti abbiano un metodo per rimuovere o esportare i propri dati su richiesta.	✓	✓	✓	212
8.3.3	Verificare che agli utenti venga fornita una spiegazione chiara in merito alla raccolta e all'utilizzo delle informazioni personali fornite e che gli utenti abbiano fornito il consenso esplicito all'utilizzo di tali dati prima che vengano utilizzati in qualsiasi modo.	✓	✓	✓	285
8.3.4	Verificare che tutti i dati sensibili creati ed elaborati dall'applicazione siano stati identificati e che sia presente una politica su come trattare i dati sensibili. (C8)	✓	✓	✓	200
8.3.5	Verificare che l'accesso ai dati sensibili venga registrato (senza registrare i dati sensibili stessi), se i dati vengono raccolti ai sensi delle direttive pertinenti sulla protezione dei dati o laddove sia richiesta la registrazione degli accessi.		✓	✓	532
8.3.6	Verificare che le informazioni sensibili contenute nella memoria vengano sovrascritte non appena non sono più necessarie per mitigare gli attacchi di dump della memoria, utilizzando zeri o dati casuali.		✓	✓	226
8.3.7	Verificare che le informazioni sensibili o private che necessitano confidenzialità vengano crittografate utilizzando algoritmi approvati che forniscono sia riservatezza che integrità. (C8)		✓	✓	327
8.3.8	Verificare che le informazioni personali sensibili siano soggette a classificazione della conservazione dei dati, in modo tale che i dati vecchi o obsoleti vengano eliminati automaticamente, secondo una pianificazione specifica o in base alle necessità.		✓	✓	285

Quando si considera la protezione dei dati, una priorità dovrebbe essere la prevenzione dell'estrazione o modifica in blocco, nonché dell'uso eccessivo. Ad esempio, molti sistemi di social media limitano gli utenti a 100 nuove amicizie al giorno, indipendentemente da quale sistema provengano le richieste. Allo stesso modo, una piattaforma bancaria potrebbe bloccare più di 5 transazioni all'ora per importi superiori a 1000 euro verso istituzioni esterne. I requisiti di ogni sistema variano, quindi la definizione di ciò che è "anormale" deve riflettere il modello di minaccia e il rischio aziendale. È fondamentale poter rilevare, scoraggiare o, preferibilmente, bloccare azioni anomale di massa.

Riferimenti

Per approfondimenti, consultare:

- Consider using Security Headers website to check security and anti-caching headers
- OWASP Secure Headers project
- OWASP Privacy Risks Project
- OWASP User Privacy Protection Cheat Sheet
- European Union General Data Protection Regulation (GDPR) overview
- European Union Data Protection Supervisor - Internet Privacy Engineering Network

V9 Comunicazione

Obiettivo del controllo

Assicurarsi che l'applicazione rispetti i seguenti requisiti di alto livello:

- È necessaria la crittografia TLS o una crittografia equivalente, indipendentemente dalla riservatezza del contenuto.
- Seguire le linee guida più recenti, comprese:
  - Consigli sulla configurazione
  - Cifrari e algoritmi consigliati
- Evitare l'uso di algoritmi e cifrature deboli o prossimi alla deprecazione, salvo necessità specifiche.
- Disabilitare algoritmi e cifrature deprecati o noti per essere insicuri.

All'interno di questi requisiti:

- Seguire costantemente i consigli del settore sulla configurazione sicura di TLS, poiché tali indicazioni cambiano frequentemente a causa di vulnerabilità critiche rilevate in algoritmi e cifrature esistenti.
- Utilizzare le versioni più aggiornate degli strumenti di revisione della configurazione TLS per definire l'ordine preferito e la selezione degli algoritmi.
- Verificare la validità e l'attendibilità dei certificati utilizzati per l'autenticazione TLS.

V9.1 Sicurezza della Comunicazione lato Client

Garantire che tutti i messaggi client vengano inviati su reti crittografate, utilizzando TLS 1.2 o versioni successive. Utilizzare strumenti aggiornati per esaminare regolarmente la configurazione del client.

#	Descrizione	L1	L2	L3	CWE
9.1.1	Verificare che TLS venga utilizzato per tutte le connessioni client e che non si ripieghi su comunicazioni non sicure o non crittografate. (C8)	✓	✓	✓	319
9.1.2	Utilizzare strumenti l'ultima versione dei test TLS per verificare che siano abilitate solo suite crittografiche robuste, con le suite più sicure impostate come preferite.	✓	✓	✓	326
9.1.3	Verificare che siano abilitate solo le versioni più recenti consigliate del protocollo TLS, come TLS 1.2 e TLS 1.3. La versione più recente del protocollo TLS dovrebbe essere l'opzione preferita.	✓	✓	✓	326

V9.2 Sicurezza della Comunicazione lato Server

La comunicazione server va oltre il semplice HTTP. Devono essere implementate connessioni sicure da e verso altri sistemi, come sistemi di monitoraggio, strumenti di gestione, accesso remoto e SSH, middleware, database, mainframe, sistemi partner o sorgenti esterne. Tutte queste connessioni devono essere crittografate per evitare situazioni in cui le difese perimetrali siano robuste, ma l'intercettazione interna risulti banale.

#	Descrizione	L1	L2	L3	CWE
9.2.1	Verificare che le connessioni da e verso il server utilizzino certificati TLS affidabili. Se vengono utilizzati certificati auto-generati o self-signed, il server deve essere configurato per fidarsi solo di CA interne specifiche e di certificati self-signed specifici. Tutti gli altri devono essere rifiutati.		✓	✓	295

#	Descrizione	L1	L2	L3	CWE
9.2.2	Verificare che la comunicazione crittografata, come TLS, venga utilizzata per tutte le connessioni in ingresso e in uscita, incluse quelle per porte di gestione, monitoraggio, autenticazione, chiamate API o di servizi web, database, cloud, serverless, mainframe, connessioni esterne e con partner. Il server non deve ripiegare su protocolli non sicuri o non crittografati.		✓	✓	319
9.2.3	Verificare che tutte le connessioni crittografate a sistemi esterni che coinvolgono informazioni o funzioni sensibili siano autenticate.		✓	✓	287
9.2.4	Verificare che la revoca corretta dei certificati, come lo "Online Certificate Status Protocol (OCSP) Stapling", sia abilitata e configurata.		✓	✓	299
9.2.5	Verificare che gli errori di connessione TLS con il backend vengano loggati.			✓	544

Riferimenti

Per approfondimenti, consultare:

- OWASP – TLS Cheat Sheet
- OWASP - Pinning Guide
- Note su "Modalità di TSL approvate":
  - In passato, l'ASVS faceva riferimento allo standard statunitense FIPS 140-2, ma applicare standard americani a livello globale può essere difficile, contraddittorio o confusionario.
  - Un metodo migliore per raggiungere la conformità alla sezione 9.1 sarebbe quello di consultare guide come TLS lato server di Mozilla, generare configurazioni note e sicure, e utilizzare strumenti di valutazione TLS aggiornati e riconosciuti per ottenere il livello di sicurezza desiderato.

V10 Codice malevolo

Obiettivo del controllo

Assicurarsi che l'applicazione rispetti i seguenti requisiti di alto livello:

- La gestione delle attività malevole deve essere sicura e appropriata per non compromettere il resto dell'applicazione.
- Il codice non deve contenere bombe a tempo o altri attacchi temporizzati.
- Il codice non deve "chiamare casa" verso destinazioni malevole o non autorizzate.
- Il codice deve essere privo di backdoor, Easter egg, attacchi salami, rootkit o codice non autorizzato controllabile da un aggressore.

È impossibile verificare completamente l'assenza di codice malevolo. Tuttavia, si devono compiere tutti gli sforzi possibili per garantire che il codice non contenga funzionalità dannose o indesiderate.

V10.1 Code Integrity

La migliore difesa contro il codice malevolo è "fidarsi, ma verificare". L'introduzione di codice non autorizzato o malevolo nel sorgente è spesso considerata un reato in molte giurisdizioni. Le politiche e le procedure dovrebbero chiarire le sanzioni relative al codice malevolo.

I lead developer dovrebbero revisionare regolarmente i commit del codice, in particolare quelli che potrebbero accedere alle funzioni di time, I/O o di rete.

#	Descrizione	L1	L2	L3	CWE
10.1.1	Verificare che sia in uso uno strumento di analisi del codice in grado di rilevare codice potenzialmente malevolo, come funzioni temporali, operazioni insicure su file e connessioni di rete.			✓	749

V10.2 Ricerca di codice malevolo

Il codice malevolo è estremamente raro e difficile da individuare. La revisione manuale riga per riga del codice può essere utile per cercare bombe logiche, ma anche il revisore più esperto potrebbe avere difficoltà a individuare codice malevolo, anche sapendo della sua presenza.

La conformità a questa sezione non è possibile senza un accesso completo al codice sorgente, incluse le librerie di terze parti.

#	Descrizione	L1	L2	L3	CWE
10.2.1	Verificare che il codice sorgente dell'applicazione e le librerie di terze parti non contengano funzionalità non autorizzate di "chiamata a casa" o raccolta dati. Se tale funzionalità esiste, è necessario ottenere il permesso dell'utente prima di raccogliere qualsiasi dato.		✓	✓	359
10.2.2	Verificare che l'applicazione non richieda autorizzazioni non necessarie o eccessive per funzionalità o sensori legati alla privacy, come contatti, fotocamere, microfoni o posizione.		✓	✓	272
10.2.3	Verificare che il codice sorgente dell'applicazione e le librerie di terze parti non contengano backdoor, come account o chiavi aggiuntive non documentate e codificate, offuscamento del codice, blob binari non documentati, rootkit, funzionalità anti-debug, funzionalità di debug non sicure o funzionalità obsolete, non sicure o nascoste che potrebbero essere utilizzate in modo malevolo se scoperte.			✓	507

#	Descrizione	L1	L2	L3	CWE
10.2.4	Verificare che il codice sorgente dell'applicazione e le librerie di terze parti non contengano bombe a tempo cercando funzioni relative a data e ora.			✓	511
10.2.5	Verificare che il codice sorgente dell'applicazione e le librerie di terze parti non contengano codice malevolo, come attacchi salami, bypass logici o bombe logiche.			✓	511
10.2.6	Verificare che il codice sorgente dell'applicazione e le librerie di terze parti non contengano Easter egg o altre funzionalità potenzialmente indesiderate.			✓	507

V10.3 Integrità dell'applicazione

Anche dopo la distribuzione, un'applicazione può essere vulnerabile all'inserimento di codice malevolo. Le applicazioni devono proteggersi da attacchi comuni, come l'esecuzione di codice non firmato proveniente da fonti non fidate e il takeover di sottodomini.

La conformità a questa sezione richiede probabilmente l'implementazione di controlli operativi e monitoraggi continui.

#	Descrizione	L1	L2	L3	CWE
10.3.1	Verificare che, se l'applicazione dispone di una funzione di aggiornamento automatico client o server, gli aggiornamenti vengano ottenuti su canali sicuri e firmati digitalmente. Il codice di aggiornamento deve convalidare la firma digitale dell'aggiornamento prima di installarlo o eseguirlo.	✓	✓	✓	16
10.3.2	Verificare che l'applicazione impieghi protezioni dell'integrità, come la firma del codice o l'integrità delle sotto-risorse. L'applicazione non deve caricare o eseguire codice da fonti non fidate, come inclusioni, moduli, plugin, codice o librerie provenienti da fonti non fidate o da Internet.	✓	✓	✓	353
10.3.3	Verificare che l'applicazione disponga di protezioni contro takeover di sottodomini se si affida a voci DNS o sottodomini DNS, come nomi di dominio scaduti, puntatori DNS o CNAME obsoleti, progetti scaduti in repository di codice sorgente pubblici, o API cloud temporanee, funzioni serverless o bucket di archiviazione (autogen-bucket-id.cloud.example.com) o simili. Le protezioni possono includere la verifica periodica della scadenza o della modifica dei nomi DNS utilizzati dalle applicazioni.	✓	✓	✓	350

Riferimenti

Per approfondimenti, consultare:

- Hostile Subdomain Takeover, Detectify Labs
- Hijacking of abandoned subdomains part 2, Detectify Labs

V11 Logica di business

Obiettivo del controllo

Verificare che un'applicazione soddisfi i seguenti requisiti di alto livello:

- Il flusso della logica di business deve essere sequenziale, elaborato nell'ordine corretto e non deve essere aggirabile.
- La logica di business deve includere limiti per rilevare e prevenire attacchi automatizzati, come trasferimenti di fondi ripetuti di piccolo importo o l'aggiunta massiva di utenti (ad esempio, un milione di amici contemporaneamente).
- I flussi della logica di business ad alto valore devono considerare possibili abusi e attori malintenzionati, e devono includere protezioni contro attacchi di spoofing, manomissione, divulgazione di informazioni e escalation di privilegi.

V11.1 Sicurezza della logica di business

La sicurezza della logica di business è così specifica per ogni applicazione che non esiste una lista di controllo universale. Deve essere progettata per proteggersi da minacce esterne probabili e non può essere implementata successivamente tramite firewall per applicazioni web o comunicazioni sicure. Si raccomanda di utilizzare il threat modeling durante le fasi di progettazione, ad esempio attraverso strumenti come OWASP Cornucopia o simili.

#	Descrizione	L1	L2	L3	CWE
11.1.1	Verificare che l'applicazione elabori i flussi della logica di business per lo stesso utente solo in sequenza e senza saltare passaggi.	✓	✓	✓	841
11.1.2	Verificare che l'applicazione elabori i flussi della logica di business solo con tutti i passaggi completati in un lasso di tempo umano realistico, ovvero le transazioni non vengano eseguite troppo rapidamente.	✓	✓	✓	799
11.1.3	Verificare che l'applicazione abbia limiti appropriati per azioni o transazioni aziendali specifiche, applicati correttamente su base per utente.	✓	✓	✓	770
11.1.4	Verificare che l'applicazione disponga di controlli anti-automazione per proteggersi da chiamate eccessive, come esfiltrazione di massa di dati, richieste di logica di business, caricamenti di file o attacchi denial of service.	✓	✓	✓	770
11.1.5	Verificare che l'applicazione disponga di limiti o convalide della logica di business per proteggersi da probabili rischi o minacce aziendali, identificati mediante threat modeling o metodologie simili.	✓	✓	✓	841
11.1.6	Verificare che l'applicazione non soffra di problemi "Time Of Check to Time Of Use" (TOCTOU) o altre condizioni di race condition per operazioni sensibili.		✓	✓	367
11.1.7	Verificare che l'applicazione monitori eventi o attività inusuali da una prospettiva di logica di business. Ad esempio, tentativi di eseguire azioni fuori ordine o azioni che un utente normale non proverebbe mai ad eseguire. (C9)		✓	✓	754
11.1.8	Verificare che l'applicazione abbia un sistema di allerta configurabile in caso di rilevamento di attacchi automatizzati o attività inusuali.		✓	✓	390

Riferimenti

Per approfondimenti, consultare:

- OWASP Web Security Testing Guide 4.1: Business Logic Testing
- Anti-automation can be achieved in many ways, including the use of OWASP AppSensor and OWASP Automated Threats to Web Applications
- OWASP AppSensor can also help with Attack Detection and Response.
- OWASP Cornucopia

V12 File e risorse

Obiettivo del controllo

Verificare che un'applicazione soddisfi i seguenti requisiti di alto livello:

- I file non considerati attendibili devono essere gestiti in modo appropriato e sicuro.
- I file non considerati attendibili ottenuti da fonti non affidabili devono essere archiviati al di fuori della root web e con permessi limitati.

V12.1 Caricamento di file

Sebbene gli attacchi "Zip Bomb" siano facilmente testabili tramite tecniche di penetration testing, sono classificati come di livello L2 e superiore per incoraggiare una progettazione e uno sviluppo accurati, con una revisione manuale attenta. Questo evita il ricorso a test di penetration testing automatizzati o manuali non qualificati per individuare condizioni di denial-of-service.

#	Descrizione	L1	L2	L3	CWE
12.1.1	Verificare che l'applicazione non accetti file di grandi dimensioni che potrebbero saturare lo storage o causare un denial-of-service.	✓	✓	✓	400
12.1.2	Verificare che l'applicazione controlli i file compressi (es. zip, gz, docx, odt) confrontandoli con la dimensione massima consentita allo stato non compresso e con il numero massimo di file prima di procedere alla decompressione.		✓	✓	409
12.1.3	Verificare che venga applicata una quota per la dimensione dei file e un numero massimo di file per utente, per garantire che un singolo utente non possa saturare lo storage con troppi file o file eccessivamente grandi.		✓	✓	770

V12.2 Integrità dei file

#	Descrizione	L1	L2	L3	CWE
12.2.1	Verificare che i file provenienti da fonti non affidabili siano effettivamente del tipo dichiarato, analizzandone il contenuto.		✓	✓	434

V12.3 Esecuzione dei file

#	Descrizione	L1	L2	L3	CWE
12.3.1	Verificare che il nome del file inviato dall'utente non venga utilizzato direttamente dai filesystem di sistema o del framework e che venga utilizzata una API URL per proteggere da path traversal.	✓	✓	✓	22
12.3.2	Verificare o ignorare il nome del file inviato dall'utente per impedire la divulgazione, la creazione, l'aggiornamento o l'eliminazione di file locali (LFI).	✓	✓	✓	73
12.3.3	Verificare o ignorare il nome del file inviato dall'utente per impedire la divulgazione o l'esecuzione di file remoti tramite attacchi di Remote File Inclusion (RFI) o Server-Side Request Forgery (SSRF).	✓	✓	✓	98
12.3.4	Verificare che l'applicazione protegga dal Reflective File Download (RFD) convalidando o ignorando i nomi dei file inviati dall'utente in un parametro JSON, JSONP o URL. L'header Content-Type della risposta deve essere impostato su "text/plain" e l'header Content-Disposition deve avere un nome file immutabile.	✓	✓	✓	641

#	Descrizione	L1	L2	L3	CWE
12.3.5	Verificare che i metadati di file non affidabili non vengano utilizzati direttamente con le API o le librerie di sistema per proteggere da OS command injection.	✓	✓	✓	78
12.3.6	Verificare che l'applicazione non includa ed esegua funzionalità da fonti non affidabili, come content delivery network non verificate, librerie JavaScript, librerie npm di Node.js o DLL lato server.		✓	✓	829

V12.4 Archiviazione File

#	Descrizione	L1	L2	L3	CWE
12.4.1	Verificare che i file ottenuti da fonti non affidabili siano archiviati al di fuori della root web, con permessi limitati.	✓	✓	✓	552
12.4.2	Verificare che i file ottenuti da fonti non affidabili vengano scansionati da antivirus per prevenire il caricamento e la distribuzione di contenuti dannosi noti.	✓	✓	✓	509

V12.5 Download file

#	Descrizione	L1	L2	L3	CWE
12.5.1	Verificare che il livello web sia configurato per servire solo file con estensioni specifiche per prevenire divulgazione involontaria di informazioni e codice sorgente. Ad esempio, è necessario bloccare il download di file di backup (es. .bak), file di lavoro temporanei (es. .swp), file compressi (.zip, .tar.gz, ecc.) e altre estensioni comunemente utilizzate dagli editor, a meno che non siano espressamente richiesti.	✓	✓	✓	552
12.5.2	Verificare che le richieste dirette a file caricati non vengano mai eseguite come contenuto HTML/JavaScript.	✓	✓	✓	434

V12.6 Protezione da SSRF

#	Descrizione	L1	L2	L3	CWE
12.6.1	Verificare che il server web o applicativo sia configurato con una allow-list di risorse o sistemi verso i quali il server può inviare richieste o caricare dati/file.	✓	✓	✓	918

Riferimenti

Per approfondimenti, consultare:

- File Extension Handling for Sensitive Information
- Reflective file download by Oren Hafif
- OWASP Third Party JavaScript Management Cheat Sheet

V13 API and Web Service

Obiettivo del controllo

Assicurarsi che un'applicazione verificata che utilizza API del livello di servizio trusted (comunemente JSON, XML o GraphQL) disponga di:

- Autenticazione, gestione delle sessioni e autorizzazione adeguate per tutti i servizi web.
- Validazione dell'input di tutti i parametri che transitano da un livello di trust inferiore a uno superiore.
- Controlli di sicurezza efficaci per tutti i tipi di API, incluse quelle cloud e serverless.

Si raccomanda di leggere questo capitolo insieme a tutti gli altri capitoli dello stesso livello; le problematiche relative all'autenticazione o alla gestione delle sessioni API non verranno ripetute qui.

V13.1 Sicurezza generica dei servizi web

#	Descrizione	L1	L2	L3	CWE
13.1.1	Verificare che tutti i componenti dell'applicazione utilizzino le stesse codifiche e parser per evitare attacchi di parsing che sfruttano comportamenti diversi di parsing URI o file, che potrebbero essere utilizzati in attacchi SSRF e RFI.	✓	✓	✓	116
13.1.2	[ELIMINATO, DUPLICATO DI 4.3.1]				
13.1.3	Verificare che gli URL delle API non espongano informazioni sensibili, come chiavi API, token di sessione, ecc.	✓	✓	✓	598
13.1.4	Verificare che le decisioni di autorizzazione vengano prese sia a livello di URI, applicate da controlli di sicurezza programmatici o dichiarativi a livello di controller o router, sia a livello di risorsa, applicate da permessi basati sul modello.		✓	✓	285
13.1.5	Verificare che le richieste contenenti tipi di contenuto inaspettati o mancanti vengano rifiutate con header appropriati (status 406 Unacceptable della risposta HTTP o 415 Unsupported Media Type).		✓	✓	434

V13.2 Servizi web RESTful

La validazione dello schema JSON è ancora in fase di standardizzazione (vedi riferimenti). Quando si valuta l'uso della validazione dello schema JSON, considerata una best practice per i servizi web RESTful, è consigliabile adottare le seguenti strategie di validazione dei dati in combinazione con la convalida dello schema JSON:

- Validazione del parsing dell'oggetto JSON, per verificare la presenza di elementi mancanti o aggiuntivi.
- Validazione dei valori dell'oggetto JSON utilizzando metodi standard di convalida dell'input, come tipo di dati, formato dei dati, lunghezza, ecc.
- Convalida formale dello schema JSON.

Una volta che lo standard di convalida dello schema JSON sarà formalizzato, ASVS aggiornerà le sue raccomandazioni in questa sezione. È importante monitorare regolarmente le librerie di validazione dello schema JSON in uso, poiché richiederanno aggiornamenti continui fino alla formalizzazione dello standard e alla correzione dei bug nelle implementazioni di riferimento.

#	Descrizione	L1	L2	L3	CWE
13.2.1	Verificare che i metodi HTTP RESTful abilitati siano una scelta valida per l'utente o l'azione, ad esempio impedendo a utenti normali di utilizzare DELETE o PUT su API o risorse protette.	✓	✓	✓	650
13.2.2	Verificare che la convalida dello schema JSON sia implementata e verificata prima di accettare l'input.	✓	✓	✓	20
13.2.3	Verificare che i servizi web RESTful che utilizzano cookie siano protetti da Cross-Site Request Forgery utilizzando almeno uno o più dei seguenti metodi: pattern double submit per i cookie, nonce CSRF, o controlli sull'header Origin.	✓	✓	✓	352
13.2.4	[ELIMINATO, DUPLICATO DI 11.1.4]				
13.2.5	Verificare che i servizi REST controllino esplicitamente che il Content-Type in ingresso sia quello previsto, come application/xml o application/json.		✓	✓	436
13.2.6	Verificare che gli header e il payload del messaggio siano affidabili e non vengano modificati durante il transito. In molti casi, potrebbe essere sufficiente richiedere una crittografia forte per il trasporto (solo TLS) in quanto fornisce protezione sia della riservatezza che dell'integrità. Le firme digitali per messaggio possono fornire un'ulteriore garanzia in aggiunta alle protezioni di trasporto per applicazioni ad alta sicurezza, ma introducono una maggiore complessità e rischi che devono essere valutati rispetto ai benefici.		✓	✓	345

V13.3 Servizi web SOAP

#	Descrizione	L1	L2	L3	CWE
13.3.1	Verificare che la convalida dello schema XSD avvenga per garantire un documento XML ben formato, seguita dalla convalida di ciascun campo di input prima di qualsiasi elaborazione dei dati.	✓	✓	✓	20
13.3.2	Verificare che il payload del messaggio sia firmato utilizzando WS-Security per garantire un trasporto affidabile tra client e servizio.		✓	✓	345

Nota: A causa di problemi con attacchi XXE contro DTD, la convalida DTD non deve essere utilizzata e la valutazione DTD del framework deve essere disabilitata secondo i requisiti stabiliti in V14 Configurazione.

V13.4 GraphQL

#	Descrizione	L1	L2	L3	CWE
13.4.1	Verificare che per le query venga utilizzata una allow-list o una combinazione di limitazione della profondità e della quantità dei risultati restituiti per prevenire Denial of Service (DoS) a livello di GraphQL o dello strato di persistenza a causa di query costose e annidate. Per scenari più avanzati, è consigliabile utilizzare l'analisi del costo delle query.		✓	✓	770
13.4.2	Verificare che la logica di autorizzazione GraphQL o di altro livello dati venga implementata a livello di logica di business anziché a livello GraphQL.		✓	✓	285

Riferimenti

Per approfondimenti, consultare:

- OWASP Serverless Top 10
- OWASP Serverless Project
- OWASP Testing Guide 4.0: Configuration and Deployment Management Testing
- OWASP Cross-Site Request Forgery cheat sheet
- OWASP XML External Entity Prevention Cheat Sheet - General Guidance
- JSON Web Tokens (and Signing)
- REST Security Cheat Sheet
- JSON Schema
- XML DTD Entity Attacks
- Orange Tsai - A new era of SSRF Exploiting URL Parser In Trending Programming Languages

V14 Configurazioni

Obiettivo del controllo

Assicurarsi che un'applicazione verificata disponga di:

- Un ambiente di build sicuro, ripetibile e automatizzabile.
- Una gestione rigorosa di librerie, dipendenze e configurazione di terze parti in modo che l'applicazione non includa componenti obsoleti o non sicuri.

La configurazione predefinita dell'applicazione deve essere sicura per l'utilizzo su Internet.

V14.1 Build e distribuzione

Le pipeline di build sono fondamentali per garantire una sicurezza ripetibile: ogni volta che viene rilevata una vulnerabilità, è possibile correggerla nel codice sorgente, negli script di build o di distribuzione, e testare automaticamente la correzione. Si raccomanda vivamente di utilizzare pipeline di build con controlli automatici di sicurezza e gestione delle dipendenze, che avvisino o interrompano la build per prevenire la distribuzione in produzione di problemi di sicurezza noti. L'esecuzione irregolare di passaggi manuali può facilmente portare a errori di sicurezza evitabili.

Con il modello DevSecOps, è essenziale garantire la disponibilità e l'integrità continua della distribuzione e configurazione, mantenendo uno stato noto e funzionante. In passato, in caso di compromissione di un sistema, potevano servire giorni o mesi per dimostrare l'assenza di ulteriori intrusioni. Oggi, con infrastrutture software-defined, distribuzioni A/B a zero downtime e build automatizzate in container, è possibile creare, rafforzare e distribuire automaticamente una versione "nota e funzionante" di qualsiasi sistema compromesso.

Se si utilizzano modelli tradizionali, è necessario implementare procedure manuali per rafforzare e fare il backup delle configurazioni, consentendo la rapida sostituzione di sistemi compromessi con sistemi integri in modo tempestivo.

La conformità a questa sezione richiede un sistema di build automatizzato e l'accesso agli script di build e distribuzione.

#	Descrizione	L1	L2	L3	CWE
14.1.1	Verificare che i processi di build e distribuzione dell'applicazione vengano eseguiti in modo sicuro e ripetibile, come automazione CI/CD, gestione automatizzata della configurazione e script di distribuzione automatizzati.		✓	✓	
14.1.2	Verificare che le flag del compilatore siano configurate per abilitare tutte le protezioni disponibili e gli avvisi per buffer overflow, inclusa la randomizzazione dello stack, la data execution prevention e l'interruzione della build se vengono trovate operazioni non sicure su puntatori, memoria, format string, numeri interi o stringhe.		✓	✓	120
14.1.3	Verificare che la configurazione del server sia rafforzata secondo le raccomandazioni del server applicativo e dei framework utilizzati.		✓	✓	16
14.1.4	Verificare che l'applicazione, la configurazione e tutte le dipendenze possano essere ridistribuite in un lasso di tempo ragionevole utilizzando script di distribuzione automatizzati, creati da un runbook documentato e testato, o ripristinate da backup in modo tempestivo.		✓	✓	
14.1.5	che gli amministratori autorizzati possano verificare l'integrità di tutte le configurazioni relative alla sicurezza, al fine di rilevare eventuali manomissioni.			✓	

V14.2 Dipendenze

La gestione delle dipendenze è cruciale per il funzionamento sicuro di qualsiasi applicazione. Il mancato aggiornamento di dipendenze obsolete o non sicure è una delle principali cause di attacchi gravi e costosi.

Nota: Al Livello 1, la conformità a 14.2.1 si basa sull'osservazione o il rilevamento di librerie e componenti lato client e di altro tipo, piuttosto che su un'analisi statica più approfondita del codice in fase di build o sull'analisi delle dipendenze. L'uso di queste tecniche più affidabili può essere documentato attraverso colloqui, se necessario.

#	Descrizione	L1	L2	L3	CWE
14.2.1	Verificare che tutti i componenti siano aggiornati, preferibilmente utilizzando un dependency checker durante la fase di build o compilazione. (C2)	✓	✓	✓	1026
14.2.2	Verificare che tutte le funzionalità, documentazione, applicazioni di esempio e configurazioni non necessarie vengano rimosse.	✓	✓	✓	1002
14.2.3	Verificare che se le risorse dell'applicazione, come librerie JavaScript, CSS o web font, sono ospitate esternamente su una Content Delivery Network (CDN) o un provider esterno, venga utilizzata la Subresource Integrity (SRI) per assicurare l'integrità della risorsa.	✓	✓	✓	829
14.2.4	Verificare che i componenti di terze parti provengano da repository predefiniti, affidabili e mantenuti frequentemente. (C2)		✓	✓	829
14.2.5	Verificare che venga mantenuto un Software Bill of Materials (SBOM) di tutte le librerie di terze parti utilizzate. (C2)		✓	✓	
14.2.6	Verificare che la superficie di attacco sia ridotta tramite sandbox o incapsulamento delle librerie di terze parti per esporre all'applicazione solo le funzionalità richieste. (C2)		✓	✓	265

V14.3 Divulgazione accidentale di informazioni di sicurezza

Le configurazioni di produzione devono essere rafforzate per proteggersi da attacchi comuni, come la modalità di debug attiva, Cross-Site Scripting (XSS), Remote File Inclusion (RFI) ed eliminare le vulnerabilità legate alla scoperta di informazioni, spesso segnalate nei report di penetration testing. Sebbene molti di questi problemi non siano classificati come rischi significativi singolarmente, possono essere sfruttati in combinazione con altre vulnerabilità. Assicurandosi che tali problemi siano assenti per impostazione predefinita, si aumenta la difficoltà per il successo della maggior parte degli attacchi.

#	Descrizione	L1	L2	L3	CWE
14.3.1	[ELIMINATO, DUPLICATO DI 7.4.1]				
14.3.2	Verificare che le modalità di debug del server web o dell'applicazione e del framework dell'applicazione siano disabilitate in produzione per eliminare funzionalità di debug, console degli sviluppatori e divulgazioni di sicurezza accidentali.	✓	✓	✓	497
14.3.3	Verificare che gli header HTTP o qualsiasi parte della risposta HTTP non espongano informazioni dettagliate sulla versione dei componenti di sistema.	✓	✓	✓	200

V14.4 Header di sicurezza HTTP

#	Descrizione	L1	L2	L3	CWE
14.4.1	Verificare che ogni risposta HTTP contenga un header Content-Type. Specificare inoltre un set di caratteri sicuro (ad esempio, UTF-8, ISO-8859-1) se i tipi di contenuto sono text/*, /+xml e application/xml. Il contenuto deve corrispondere all'header Content-Type fornito.	✓	✓	✓	173
14.4.2	Verificare che tutte le risposte dell'API contengano un header Content-Disposition: attachment; filename="api.json" (o altro nome file appropriato per il tipo di contenuto).	✓	✓	✓	116
14.4.3	Verificare che sia presente un header di risposta Content Security Policy (CSP) che contribuisca a mitigare l'impatto degli attacchi XSS come vulnerabilità di HTML, DOM, JSON e JavaScript injection.	✓	✓	✓	1021
14.4.4	Verificare che tutte le risposte contengano un header X-Content-Type-Options: nosniff.	✓	✓	✓	116
14.4.5	Verificare che un header Strict-Transport-Security sia incluso su tutte le risposte e per tutti i sottodomini, come Strict-Transport-Security: max-age=15724800; includeSubdomains.	✓	✓	✓	523
14.4.6	Verificare che sia incluso un header Referrer-Policy adatto per evitare di esporre informazioni sensibili nell'URL tramite l'header Referer a terze parti non fidate.	✓	✓	✓	116
14.4.7	Verificare che il contenuto di un'applicazione web non possa essere incorporato per impostazione predefinita in un sito di terze parti e che l'incorporamento delle risorse sia consentito solo se necessario utilizzando header di risposta Content-Security-Policy: frame-ancestors e X-Frame-Options appropriati.	✓	✓	✓	1021

V14.5 Validazione dell'header di richiesta HTTP

#	Descrizione	L1	L2	L3	CWE
14.5.1	Verificare che il server dell'applicazione accetti solo i metodi HTTP utilizzati dall'applicazione/API, incluse le opzioni di pre-flight (OPTIONS), e registri o generi alert per qualsiasi richiesta non valida per il contesto dell'applicazione.	✓	✓	✓	749
14.5.2	Verificare che l'header Origin fornito non venga utilizzato per l'autenticazione o il controllo degli accessi, in quanto può essere facilmente modificato da un attaccante.	✓	✓	✓	346
14.5.3	Verificare che l'header CORS (Cross-Origin Resource Sharing) Access-Control-Allow-Origin utilizzi una allow-list di domini e sottodomini trusted per il confronto e non supporti l'origine "null".	✓	✓	✓	346
14.5.4	Verificare che gli header HTTP aggiunti da un proxy trusted o da dispositivi SSO, come un token bearer, siano autenticati dall'applicazione.		✓	✓	306

Riferimenti

Per approfondimenti, consultare:

- OWASP Web Security Testing Guide 4.1: Testing for HTTP Verb Tampering
- Adding Content-Disposition to API responses helps prevent many attacks based on misunderstanding on the MIME type between client and server, and the "filename" option specifically helps prevent Reflected File Download attacks.
- Content Security Policy Cheat Sheet
- Exploiting CORS misconfiguration for BitCoins and Bounties
- OWASP Web Security Testing Guide 4.1: Configuration and Deployment Management Testing
- Sandboxing third party components

Appendice A: Glossario

- **Address Space Layout Randomization** (ASLR) – Una tecnica per rendere più complicato lo sfruttamento di bug di corruzione della memoria.
- **Allow list** – Un elenco di dati o operazioni consentite, ad esempio un elenco di caratteri consentiti per la convalida degli input.
- **Application Security** – La sicurezza a livello di applicazione si concentra sull'analisi dei componenti che compongono il livello applicativo del modello di riferimento per l'interconnessione di sistemi aperti (modello OSI), piuttosto che focalizzarsi, ad esempio, sul sistema operativo sottostante o sulle reti collegate.
- **Application Security Verification** – La valutazione tecnica di un'applicazione rispetto all'OWASP ASVS.
- **Application Security Verification Report** – Un report che documenta i risultati complessivi e le analisi di supporto prodotte dal verificatore per una particolare applicazione.
- **Authentication** – La verifica dell'identità dichiarata di un utente dell'applicazione.
- **Automated Verification** – L'utilizzo di strumenti automatizzati (strumenti di analisi dinamica, strumenti di analisi statica o entrambi) che utilizzano firme di vulnerabilità per individuare problemi.
- **Black box testing** – È un metodo di test del software che esamina le funzionalità di un'applicazione senza analizzare le sue strutture o il suo funzionamento interno.
- **Component** – Un'unità di codice autoconsistente, con interfacce disco e di rete associate che comunica con altri componenti.
- **Cross-Site Scripting** (XSS) – Una vulnerabilità di sicurezza che si trova tipicamente nelle applicazioni web e consente l'injection di script lato client nel contenuto.
- **Cryptographic module** – Hardware, software e/o firmware che implementa algoritmi crittografici e/o genera chiavi crittografiche.
- **Common Weakness Enumeration** (CWE) - Un elenco sviluppato dalla comunità delle comuni debolezze di sicurezza del software. Serve come linguaggio comune, come metro di misura per gli strumenti di sicurezza software e come base per gli sforzi di identificazione, mitigazione e prevenzione delle debolezze.
- **Design Verification** – La valutazione tecnica dell'architettura di sicurezza di un'applicazione.
- **Dynamic Application Security Testing** (DAST) - Tecnologie progettate per rilevare condizioni indicative di una vulnerabilità di sicurezza in un'applicazione durante il suo stato di esecuzione.
- **Dynamic Verification** – L'utilizzo di strumenti automatizzati che utilizzano firme di vulnerabilità per individuare problemi durante l'esecuzione di un'applicazione.
- **Fast IDentity Online** (FIDO) - Un insieme di standard di autenticazione che consentono l'utilizzo di diversi metodi di autenticazione, tra cui dati biometrici, Trusted Platform Module (TPM), token di sicurezza USB, ecc.
- **Globally Unique Identifier** (GUID) – Un numero di riferimento univoco utilizzato come identificatore nei software.
- **Hyper Text Transfer Protocol** (HTTPS) – Un protocollo applicativo per sistemi informativi ipermediali distribuiti, collaborativi. È la base della comunicazione dati per il World Wide Web.
- **Hardcoded keys** – Chiavi crittografiche memorizzate sul file system, sia nel codice, nei commenti o nei file.
- **Hardware Security Module** (HSM) - Componente hardware in grado di memorizzare chiavi crittografiche e altri segreti in modo protetto.
- **Hibernate Query Language** (HQL) - Un linguaggio di query simile a SQL utilizzato dalla libreria ORM Hibernate.

- **Input Validation** – La normalizzazione e la convalida dell'input utente non fidato.
- **Malicious Code** – Codice introdotto in un'applicazione durante il suo sviluppo a insaputa del proprietario dell'applicazione, che elude la politica di sicurezza prevista dall'applicazione. Non è la stessa cosa del malware come un virus o un worm!
- **Malware** – Codice eseguibile introdotto in un'applicazione durante il runtime a insaputa dell'utente o dell'amministratore dell'applicazione.
- **Open Web Application Security Project** (OWASP) – L'Open Web Application Security Project (OWASP) è una comunità mondiale gratuita e aperta che si concentra sul miglioramento della sicurezza del software applicativo. La nostra missione è rendere la sicurezza delle applicazioni "visibile", in modo che le persone e le organizzazioni possano prendere decisioni informate sui rischi per la sicurezza delle applicazioni. Vedi: https://www.owasp.org/
- **One-time Password** (OTP) - Una password univoca generata per essere utilizzata una sola volta.
- **Object-relational Mapping** (ORM) - Un sistema utilizzato per consentire a un database relazionale/basato su tabelle di essere referenziato e interrogato all'interno di un programma applicativo utilizzando un modello a oggetti compatibile con l'applicazione.
- **Password-Based Key Derivation Function 2** (PBKDF2) - Un algoritmo speciale unidirezionale utilizzato per creare una chiave crittografica forte da un testo di input (come una password) e un valore di salt casuale aggiuntivo e può quindi essere utilizzato per rendere più difficile crackare una password offline se il valore risultante viene archiviato invece della password originale.
- **Personally Identifiable Information** (PII) - Sono informazioni che possono essere utilizzate da sole o con altre informazioni per identificare, contattare o localizzare una singola persona, o per identificare un individuo in un contesto.
- **Position-independent executable** (PIE) - Un eseguibile in codice macchina che, posizionato da qualche parte nella memoria primaria, viene eseguito correttamente indipendentemente dal suo indirizzo assoluto.
- **Public Key Infrastructure** (PKI) - Un accordo che lega le chiavi pubbliche con le rispettive identità delle entità. Il legame viene stabilito attraverso un processo di registrazione e rilascio di certificati presso e da parte di un'autorità di certificazione (CA).
- **Public Switched Telephone Network** (PSTN) - La rete telefonica tradizionale che include sia i telefoni fissi che i telefoni cellulari.
- **Relying Party** (RP) - Generalmente un'applicazione che si basa sull'autenticazione di un utente presso un provider di autenticazione separato. L'applicazione si basa su un qualche tipo di token o insieme di asserzioni firmate fornite da quel provider di autenticazione per fidarsi che l'utente sia chi dice di essere.
- **Static application security testing** (SAST) - Un insieme di tecnologie progettate per analizzare il codice sorgente dell'applicazione, il bytecode e i binari per condizioni di codifica e progettazione che indicano vulnerabilità di sicurezza. Le soluzioni SAST analizzano un'applicazione dall'interno verso l'esterno senza eseguire il codice.
- **Software development lifecycle** (SDLC) - Il processo passo-passo mediante il quale viene sviluppato il software, passando dai requisiti iniziali all'implementazione e alla manutenzione.
- **Security Architecture** – Un'astrazione della progettazione di un'applicazione che identifica e descrive dove e come vengono utilizzati i controlli di sicurezza, e identifica e descrive anche la posizione e la riservatezza sia dei dati utente che applicativi.
- **Security Configuration** – La configurazione runtime di un'applicazione che influenza il modo in cui vengono utilizzati i controlli di sicurezza.
- **Security Control** – Una funzione o componente che esegue un controllo di sicurezza (ad esempio, un controllo di accesso) o, se chiamato, produce un effetto di sicurezza (ad esempio, la generazione di un record di audit).

- **Server-side Request Forgery** (SSRF) - Un attacco che abusa della funzionalità sul server per leggere o aggiornare risorse interne fornendo o modificando un URL attraverso cui il codice in esecuzione sul server leggerà o invierà dati.
- **Single Sign-on Authentication** (SSO) - Ciò si verifica quando un utente accede a un'applicazione e viene quindi automaticamente collegato ad altre applicazioni senza dover effettuare nuovamente l'autenticazione. Ad esempio, quando accedi a Google, quando utilizzi ad altri servizi Google come YouTube, Google Documenti e Gmail, non dovrai svolgere nuovamente il login.
- **SQL Injection** (SQLi) – Una tecnica di injection utilizzata per attaccare applicazioni basate su dati, in cui istruzioni SQL dannose vengono inserite in input e trattate come codice.
- **SVG** - Scalable Vector Graphics
- **Time-based OTP** - Un metodo per generare un OTP in cui l'ora corrente funge da parte dell'algoritmo per generare la password.
- **Threat Modeling** - Una tecnica che consiste nello sviluppare architetture di sicurezza sempre più accurate per identificare agenti di minaccia, zone di sicurezza, controlli di sicurezza e importanti asset tecnici e aziendali.
- **Transport Layer Security** (TLS) – Protocolli crittografici che forniscono la sicurezza della comunicazione su una connessione di rete
- **Trusted Platform Module** (TPM) - È un tipo di HSM (Hardware Security Module) che viene solitamente integrato in un componente hardware più grande, come una scheda madre, e funge da "root of trust" per quel sistema.
- **Two-factor authentication** (2FA) - Aggiunge un secondo livello di verifica per l'accesso ad un account.
- **Universal 2nd Factor** (U2F) - Uno degli standard creati da FIDO (Fast Identity Online) specificamente per consentire l'utilizzo di una chiave di sicurezza USB o NFC come secondo fattore di autenticazione.
- **URI/URL/URL fragments** – Un Uniform Resource Identifier (URI) è una stringa di caratteri utilizzata per identificare un nome o una risorsa web. Spesso viene utilizzato un Uniform Resource Locator (URL) come riferimento a una risorsa web.
- **Verifier** – La persona o il team che sta analizzando un'applicazione rispetto ai requisiti OWASP ASVS (Open Web Application Security Project Application Security Verification Standard).
- **What You See Is What You Get** (WYSIWYG) - Un tipo di editor di contenuti avanzato che mostra l'aspetto effettivo del contenuto al momento della visualizzazione anziché mostrare il codice utilizzato per controllarne la resa grafica.
- **X.509 Certificate** – Un certificato digitale che utilizza lo standard internazionale X.509 PKI (Public Key Infrastructure) ampiamente riconosciuto per verificare che una chiave pubblica appartenga all'identità di utente, computer o servizio contenuta nel certificato.
- **XML eXternal Entity** (XXE) - Un tipo di entità XML che può accedere a contenuti locali o remoti utilizzando un identificatore di sistema. Questo può causare vari tipi di attacchi di injection.

Appendice B: Riferimenti

I seguenti progetti OWASP sono probabilmente i più utili per gli utenti/adottatori di questo standard:

Progetti principali OWASP

1. OWASP Top 10 Project: https://owasp.org/www-project-top-ten/
2. OWASP Web Security Testing Guide: https://owasp.org/www-project-web-security-testing-guide/
3. OWASP Proactive Controls: https://owasp.org/www-project-proactive-controls/
4. OWASP Security Knowledge Framework: https://owasp.org/www-project-security-knowledge-framework/
5. OWASP Software Assurance Maturity Model (SAMM): https://owasp.org/www-project-samm/

Progetti OWASP Cheat Sheet

Questo progetto contiene una serie di cheat sheet rilevanti per diversi argomenti del ASVS.

È disponibile una mappatura con l'ASVS che può essere trovata qui:
https://cheatsheetseries.owasp.org/cheatsheets/IndexASVS.html

Progetti realitivi alla sicurezza Mobile

1. OWASP Mobile Security Project: https://owasp.org/www-project-mobile-security/
2. OWASP Mobile Top 10 Risks: https://owasp.org/www-project-mobile-top-10/
3. OWASP Mobile Security Testing Guide and Mobile Application Security Verification Standard: https://owasp.org/www-project-mobile-security-testing-guide/

Progetti OWASP relativi all'Internet of Things

1. OWASP Internet of Things Project: https://owasp.org/www-project-internet-of-things/

Progetti OWASP relativi all'ambiente Serverless

1. OWASP Serverless Project: https://owasp.org/www-project-serverless-top-10/

Altri

Analogamente, i seguenti siti web sono probabilmente i più utili per gli utenti/adottatori di questo standard

1. SecLists Github: https://github.com/danielmiessler/SecLists
2. MITRE Common Weakness Enumeration: https://cwe.mitre.org/
3. PCI Security Standards Council: https://www.pcisecuritystandards.org
4. PCI Data Security Standard (DSS) v3.2.1 Requirements and Security Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf
5. PCI Software Security Framework - Secure Software Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-Software-Standard-v1_0.pdf
6. PCI Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures: https://www.pcisecuritystandards.org/documents/PCI-Secure-SLC-Standard-v1_0.pdf

Appendice C: Requisiti di Verifica per l'Internet delle Cose (IoT)

Questo capitolo era originariamente nella sezione principale, ma con il lavoro svolto dal team OWASP IoT, non ha senso mantenere due filoni differenti sull'argomento. Per la versione 4.0, stiamo spostando questo contenuto nell'Appendice e invitiamo tutti coloro che ne hanno bisogno a consultare il progetto OWASP IoT

Obiettivo del controllo

I dispositivi Embedded/IoT devono:

- Avere lo stesso livello di controlli di sicurezza presenti nel server, applicando i controlli di sicurezza in un ambiente trusted.
- I dati sensibili memorizzati sul dispositivo devono essere protetti in modo sicuro utilizzando meccanismi supportati dall'hardware, come secure element.
- Tutti i dati sensibili trasmessi dal dispositivo devono utilizzare la sicurezza a livello di trasporto.

Requisiti di Verifica della Sicurezza

#	Descrizione	L1	L2	L3	Da
C.1	Verificare che le interfacce di debug a livello di applicazione come USB, UART e altre varianti seriali siano disabilitate o protette da una password complessa.	✓	✓	✓	4.0
C.2	Verificare che le chiavi crittografiche e i certificati siano unici per ogni singolo dispositivo.	✓	✓	✓	4.0
C.3	Verificare che, se applicabili, i controlli di protezione della memoria come ASLR e DEP siano abilitati dal sistema operativo embedded/IoT.	✓	✓	✓	4.0
C.4	Verificare che le interfacce di debug integrate come JTAG o SWD siano disabilitate o che il meccanismo di protezione disponibile sia abilitato e configurato adeguatamente.	✓	✓	✓	4.0
C.5	Verificare che se disponibile, la trusted execution sia implementata e abilitata sul SoC o CPU del dispositivo.	✓	✓	✓	4.0
C.6	Verificare che i dati sensibili, le chiavi private e i certificati siano memorizzati in modo sicuro in un Secure Element, TPM, TEE (Trusted Execution Environment), o protetti mediante crittografia robusta.	✓	✓	✓	4.0
C.7	Verificare che le app del firmware proteggano i dati in transito utilizzando la sicurezza a livello di trasporto.	✓	✓	✓	4.0
C.8	Verificare che le app del firmware convalidino la firma digitale delle connessioni server.	✓	✓	✓	4.0
C.9	Verificare che le comunicazioni wireless siano autenticate reciprocamente.	✓	✓	✓	4.0
C.10	Verificare che le comunicazioni wireless siano inviate su un canale criptato.	✓	✓	✓	4.0
C.11	Verificare che qualsiasi utilizzo di funzioni C vietate sia sostituito con le versioni equivalenti sicure.	✓	✓	✓	4.0
C.12	Verificare che ogni firmware mantenga un catalogo delle terze parti (software bill of materials), con versioni e vulnerabilità pubblicate.	✓	✓	✓	4.0
C.13	Verificare che tutto il codice, inclusi i binari di terze parti, le librerie e i framework, sia esaminato per le credenziali hardcoded (backdoor).	✓	✓	✓	4.0

#	Descrizione	L1	L2	L3	Da
C.14	Verificare che i componenti applicativi e firmware non siano suscettibili a OS Command Injection tramite l'invocazione di wrapper di comandi shell, script, o che i controlli di sicurezza impediscano l'OS Command Injection.	✓	✓	✓	4.0
C.15	Verificare che le app del firmware associno la firma digitale ai server trusted.		✓	✓	4.0
C.16	Verificare la presenza di funzionalità di resistenza e/o rilevamento di manomissioni.		✓	✓	4.0
C.17	Verificare che le tecnologie di protezione della proprietà intellettuale fornite dal produttore del chip siano abilitate.		✓	✓	4.0
C.18	Verificare che siano presenti controlli di sicurezza per ostacolare il reverse engineering del firmware (ad esempio, rimozione di simboli di debug).		✓	✓	4.0
C.19	Verificare che il dispositivo convalidi la firma dell'immagine di boot prima di caricarla.		✓	✓	4.0
C.20	Verificare che il processo di aggiornamento del firmware non sia vulnerabile agli attacchi di time-of-check vs time-of-use.		✓	✓	4.0
C.21	Verificare che il dispositivo utilizzi la firma del codice e convalidi i file di aggiornamento del firmware prima dell'installazione.		✓	✓	4.0
C.22	Verificare che sul dispositivo non possano essere installate versioni precedenti (anti-rollback) del firmware.		✓	✓	4.0
C.23	Verificare l'uso di un generatore di numeri pseudo-casuali crittograficamente sicuro sul dispositivo embedded (ad esempio, utilizzando generatori di numeri casuali forniti dal chip).		✓	✓	4.0
C.24	Verificare che il firmware possa eseguire aggiornamenti automatici secondo una pianificazione predefinita.		✓	✓	4.0
C.25	Verificare che il dispositivo cancelli il firmware e i dati sensibili in caso di rilevamento di manomissioni o ricezione di messaggi non validi.			✓	4.0
C.26	Verificare che vengano utilizzati solo microcontrollori che supportano la disabilitazione delle interfacce di debug (ad esempio, JTAG, SWD).			✓	4.0
C.27	Verificare che vengano utilizzati solo microcontrollori che forniscono una protezione efficace contro gli attacchi di decapping e side channel.			✓	4.0
C.28	Verificare che le piste sensibili non siano esposte verso gli strati esterni del circuito stampato.			✓	4.0
C.29	Verificare che la comunicazione inter-chip sia criptata (ad esempio, comunicazione tra la scheda principale e la scheda figlia).			✓	4.0
C.30	Verificare che il dispositivo utilizzi la firma del codice e convalidi il codice prima dell'esecuzione.			✓	4.0
C.31	Verificare che le informazioni sensibili mantenute in memoria vengano sovrascritte con zeri non appena non sono più necessarie.			✓	4.0
C.32	Verificare Verifica che le app del firmware utilizzino container a livello kernel per l'isolamento tra le app.			✓	4.0

#	Descrizione	L1	L2	L3	Da
C.33	Verificare che per le build del firmware siano configurate le flag di compilazione sicura come -fPIE, -fstack-protector-all, -Wl,-z,noexecstack, -Wl,-z,noexecheap.			✓	4.0
C.34	Verificare che i microcontrollori siano configurati con la protezione del codice (se applicabile).			✓	4.0

Riferimenti

Per approfondimenti, consultare:

- OWASP Internet of Things Top 10
- OWASP Embedded Application Security Project
- OWASP Internet of Things Project
- Trudy TCP Proxy Tool