OWASP
Open Web Application
Security Project

FOR DEVELOPERS

2018 v 3.0

10 Critical Security Areas That Software Developers Must Be Aware Of

PROJECT LEADERS

KATY ANTON
JIM MANICO
JIM BIRD

Riguardo a OWASP

L'Open Web Application Security Project (OWASP) è una organizzazione, senza scopo di lucro, impegnata per agevolare la cultura della sicurezza sulle Web Application, ovvero aiutare le organizzazioni affinché possano progettare, sviluppare, acquisire, gestire e mantenere software in maniera sicura. Tutti gli strumenti, documenti, forum e capitoli internazionali OWASP sono gratuiti e aperti a chiunque sia interessato a migliorare la sicurezza delle applicazioni. Potete trovarci all'indirizzo www.owasp.org.

OWASP è un nuovo tipo di organizzazione. La nostra indipendenza da pressioni commerciali ci permette di fornire informazioni imparziali, pratiche e coerenti in termini di costi sulla sicurezza delle applicazioni.

OWASP non è affiliato con alcuna azienda tecnologica. Simile a molti progetti software open source, OWASP produce molti tipi di materiale in modo collaborativo e aperto. La Fondazione OWASP è un'entità senza scopo di lucro che garantisce il successo a lungo termine dei progetti.

PREMESSA

Il software non sicuro sta mettendo a rischio le nostre infrastrutture finanziarie, sanitarie, di difesa, energetiche e altre infrastrutture critiche in tutto il mondo. Mentre la nostra infrastruttura digitale globale diventa sempre più complessa e interconnessa, la difficoltà nel raggiungere la sicurezza delle applicazioni aumenta in modo esponenziale. Non possiamo più permetterci di tollerare problemi di sicurezza relativamente semplici.

SCOPO E OBIETTIVO

L'obiettivo del progetto OWASP Top 10 Proactive Controls (OPC) è aumentare la consapevolezza sulla sicurezza delle applicazioni, descrivendo le aree di interesse più importanti delle quali gli sviluppatori software devono essere consapevoli. Ti incoraggiamo a utilizzare i Controlli Proattivi OWASP come punto di partenza sulla sicurezza per i tuoi sviluppatori. Questi possono così imparare dagli errori di altre organizzazioni. Ci auguriamo che i controlli proattivi OWASP vi siano utili nello sviluppo di software sicuro.

CALL TO ACTION

Non esitate a contattare il progetto OWASP Proactive Control con le vostre domande, commenti e idee, sia pubblicamente alla nostra mailing list, che privatamente a jim@owasp.org.

COPYRIGHT E LICENZA

RESPONSABILI DEL PROGETTO

Katy Anton Jim Bird Jim Manico

COLLABORATORI

Chris Romeo	Dan Anderson	David Cybuck
Dave Ferguson	Josh Grossman	Osama Elnaggar
Colin Watson	Rick Mitchell	And many more…

TRADUZIONE

Massimiliano Graziani lifemember OWASP e co-fondatore del Capitolo Italiano OWASP

STRUTTURA DEL DOCUMENTO

Questo documento è strutturato sotto forma di elenco. Ogni controllo è descritto come segue:

Descrizione

Descrizione dettagliata del controllo, comprese alcune best practice da prendere in considerazione.

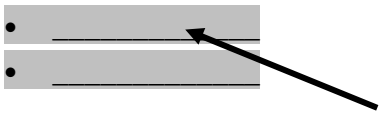
Implementazione

Implementazione delle best practice ed esempi che spiegano come attuare ciascun controllo.

Vulnerabilità Prevenute

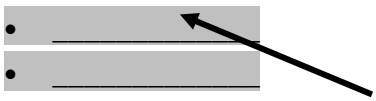
Referenze

Elenco delle vulnerabilità evitate o dei rischi affrontati (OWASP TOP 10 Risk, CWE, etc.)



Elenco dei riferimenti per ulteriori studi (OWASP Cheat sheet, Security Hardening Guidelines, etc.)

Strumenti



Set di strumenti/progetti per introdurre/integrare facilmente i controlli di sicurezza nel tuo software.

INTRODUZIONE

OWASP Top Ten Proactive Controls 2018 è un elenco di tecniche di sicurezza che dovrebbero essere prese in considerazione per ogni progetto di sviluppo software. Questo documento è stato scritto per sviluppatori, per assistere quelli che, riguardo lo sviluppo di software sicuro, sono alle prime armi.

Uno degli obiettivi principali di questo documento è fornire una guida pratica e concreta che aiuti gli sviluppatori a creare software sicuro. Tali tecniche dovrebbero essere applicate in modo proattivo nelle prime fasi di sviluppo del software per garantirne la massima efficacia.

I 10 migliori controlli proattivi

Questa lista é in ordine di importanza, partendo con il primo elemento come il più importante:

C1: Definire i Requisiti di Sicurezza

C2: Sfruttare Framework e Librerie

C3: Accesso Sicuro ai Database

C4: Codifica ed Escape dei Dati

C5: Convalidare Tutti gli Inputs

C6: Implementare la Digital Identity

C7: Imporre il Controllo degli Accessi

C8: Proteggere Dati Ovunque

C9: Implementare il Logging e il Monitoraggio della Sicurezza

C10: Gestire Tutti gli Errori e le Eccezioni

Come è stata creata questa lista

Questa lista è stata originariamente creata dagli attuali responsabili del progetto con il contributo di diversi volontari. Il documento è stato quindi condiviso a livello globale, in modo da poter prendere in considerazione anche suggerimenti anonimi. Centinaia di cambiamenti sono stati accettati con questo metodo di comunità aperta.

Destinatari del documento

Questo documento è stato scritto principalmente per sviluppatori. Tuttavia, anche responsabili allo sviluppo, proprietari di prodotti, professionisti del settore, manager dei programmi e chiunque sia coinvolto nella creazione di software possono trarre vantaggio da questo documento.

Come Usare Questo Documento

Questo documento ha lo scopo di fornire una consapevolezza iniziale sulla creazione di software sicuro. Questo documento fornirà anche una buona base di argomenti per aiutare a guidare la formazione introduttiva per sviluppatori di software sicuro. Questi controlli dovrebbero essere utilizzati costantemente ed accuratamente in tutte le applicazioni. Tuttavia, questo documento dovrebbe essere visto come un punto di partenza piuttosto che un insieme completo di tecniche e pratiche. Un processo di sviluppo completamente sicuro dovrebbe includere requisiti estesi da standard come OWASP ASVS oltre ad includere una gamma di attività di sviluppo software descritte in modelli maturi come OWASP SAMM e BSIMM.

Link to the OWASP Top 10 Project

I controlli top 10 proattivi di OWASP sono simili alle top 10 di OWASP, ma si concentrano su tecniche e controlli difensivi anziché sui rischi. Ciascuna tecnica o controllo in questo documento verrà associata ad uno o più elementi nella Top 10 OWASP basata sul rischio. Le informazioni riguardo questi collegamenti sono incluse nella descrizione alla fine di ogni controllo.

OWASP Proactive Controls v 3.0

C1: Determinare i Requisiti di Sicurezza

Descrizione

Un requisito di sicurezza è una dichiarazione della funzionalità di sicurezza necessaria che garantisce che una delle molte diverse proprietà di sicurezza del software sia soddisfatta. I requisiti di sicurezza derivano dagli standard del settore, dalle leggi applicabili e dallo storico delle vulnerabilità passate. I requisiti di sicurezza definiscono nuove funzionalità o integrazioni a funzionalità esistenti, per risolvere un problema di sicurezza specifico o eliminare una potenziale vulnerabilità.

I requisiti di sicurezza forniscono una base alle funzionalità di sicurezza per un'applicazione. Invece di creare un approccio personalizzato per ogni applicazione, i requisiti di sicurezza standard consentono agli sviluppatori di riutilizzare la definizione di controlli di sicurezza e di best practice. Gli stessi requisiti di sicurezza controllati forniscono soluzioni per i problemi di sicurezza che si sono verificati in passato. Questi requisiti esistono con lo scopo di prevenire il ripetersi di errori di sicurezza precedentemente rilevati.

OWASP ASVS

L' OWASP Application Security Verification Standard (ASVS) è un catalogo di requisiti di sicurezza disponibili e di criteri di verifica. OWASP ASVS può essere una fonte di dettagliati requisiti di sicurezza per i team di sviluppo.

I requisiti di sicurezza sono classificati in diversi contenitori definiti "bucket", in base a una funzione di sicurezza di alto livello. Ad esempio, ASVS contiene categorie sull'autenticazione, sul controllo degli accessi, sulla gestione/registrazione degli errori e sui servizi web. Ogni categoria contiene una raccolta di requisiti che rappresentano le best practice per quella categoria.

Migliorare i requisiti con User Story e Casi di Uso Improprio

I requisiti ASVS sono dichiarazioni verificabili di base che possono essere migliorate con User Story e Casi di Abuso. Il vantaggio di una User Story o di un Caso di Uso Improprio è che mette a confronto l'applicazione esattamente a ciò che l'utente o l'aggressore fa al sistema, invece di descrivere ciò che il sistema offre all'utente.

Ecco un esempio di espansione su un requisito ASVS 3.0.1. Dalla sezione "Authentication Verification Requirements" di ASVS 3.0.1, il requisito 2.19 si concentra sulle password predefinite.

> 2.19 Verificare che non vi siano in uso password predefinite per il framework dell'applicazione o per qualsiasi componente utilizzato dall'applicazione (come "admin/password").

Questo requisito contiene sia un'azione per verificare che non esistano password predefinite, sia l'istruzione che nessuna password predefinita debba essere utilizzata all'interno dell'applicazione.

Una User Story si concentra sulla prospettiva dell'utente, amministratore o aggressore del sistema e descrive le funzionalità in base a ciò che un utente desidera che il sistema faccia per loro. Una User Story prende la forma di "Come utente, posso fare x, y e z".

```
In qualità di utente, posso inserire il mio nome utente e la password per
accedere all'applicazione.

In qualità di utente, posso inserire una password lunga che abbia al massimo
1023 caratteri.
```

Quando la Story è incentrata sull'aggressore e sulle sue azioni, si parla di caso di uso improprio.

```
In qualità di aggressore, posso inserire un nome utente e una password
predefiniti per ottenere l'accesso.
```

Questa Story contiene lo stesso messaggio del requisito tradizionale di ASVS, con ulteriori dettagli sull'utente o sull'aggressore per contribuire a rendere il requisito più dimostrabile.

Implementazione

L'utilizzo corretto dei requisiti di sicurezza prevede quattro passaggi. Il processo include la scoperta/selezione, la documentazione, l'implementazione e quindi la conferma della corretta implementazione delle nuove caratteristiche e funzionalità di sicurezza all'interno di un'applicazione.

Scoperta e selezione

Il processo inizia con la scoperta e la selezione dei requisiti di sicurezza. In questa fase, lo sviluppatore sta comprendendo i requisiti di sicurezza da una fonte standard come ASVS e sta scegliendo quali requisiti includere per una determinata versione di un'applicazione. Lo scopo

di analisi e selezione è quello di scegliere un numero gestibile di requisiti di sicurezza per versione corrente o successiva, da ripetere quindi ad ogni nuova release, aggiungendo ulteriori funzionalità di sicurezza nel tempo.

Revisione del Codice

Durante l'analisi migliorativa, lo sviluppatore riesamina l'applicazione esistente confrontandola con il nuovo set di requisiti di sicurezza per determinare se l'applicazione attualmente soddisfa il requisito o se è necessario un ulteriore sviluppo. Questa indagine culmina con la documentazione dei risultati della revisione.

Implementazione e Verifica

Dopo aver determinato cosa è necessario sviluppare, lo sviluppatore deve quindi modificare l'applicazione in qualche modo per aggiungere le nuove funzionalità o eliminare un'opzione non sicura. In questa fase lo sviluppatore determina prima quale progetto sia richiesto per far fronte al requisito, quindi completa le modifiche al codice per soddisfarlo. Casi di test dovrebbero essere creati per confermare l'esistenza della nuova funzionalità o confutare l'esistenza di un'opzione precedentemente non sicura.

Prevenzione delle Vulnerabilità

I requisiti di sicurezza definiscono le funzionalità di sicurezza di un'applicazione. Una migliore sicurezza integrata dall'inizio del ciclo di vita delle applicazioni ha come risultato la prevenzione di molti tipi di vulnerabilità.

Riferimenti

- OWASP Application Security Verification Standard (ASVS)
- OWASP Mobile Application Security Verification Standard (MASVS)
- OWASP Top Ten

OWASP Proactive Controls v 3.0

C2: Sfruttare Framework e Librerie

Descrizione

Le Librerie ed i Framework con protezione integrata aiutano gli sviluppatori software a proteggersi dai difetti di progettazione o implementazione legati alla sicurezza. Uno sviluppatore che scrive un'applicazione da zero potrebbe non avere conoscenze, tempo o budget sufficienti per implementare o mantenere correttamente le funzionalità di sicurezza. L'utilizzo dei framework con protezione integrata aiuta a raggiungere gli obiettivi di sicurezza in modo più efficiente e accurato.

Implementazione delle Best Practice

Quando si incorporano librerie o framework di terze parti nel software, è importante considerare le seguenti Best Practice:

1. Utilizzare librerie e framework da fonti attendibili che siano attivamente mantenute e ampiamente utilizzate da molte applicazioni.
2. Creare e mantenere un catalogo di tutte le librerie di terze parti.
3. Mantenere aggiornate le librerie e i componenti in modo proattivo. Utilizzare strumenti come OWASP Dependency Check e Retire.JS per identificare le “dependencies” del progetto e verificare se siano presenti vulnerabilità note e divulgate pubblicamente per tutto il codice di terze parti.
4. Ridurre la superficie di attacco compilando le librerie utilizzate ed esporre solo il le funzionalità necessarie al tuo software.

Prevenzione delle Vulnerabilità

Framework e Librerie con protezione integrata possono aiutare a prevenire un'ampia gamma di vulnerabilità delle applicazioni web. È fondamentale mantenere questi framework e librerie aggiornati come descritto in using components with known vulnerabilities Top Ten 2017 risk.

OWASP Proactive Controls v 3.0

C3: Accesso Sicuro ai Database

Tools

- OWASP Dependency Check - identifica le “dependencies” del progetto e controlla le vulnerabilità divulgate pubblicamente
- Retire.JS scanner per Librerie JavaScript

Descrizione

Questa sezione descrive l'accesso sicuro a tutti i dati archiviati, inclusi i database relazionali e i database NoSQL. Alcune aree da considerare:

1. Mettere in Sicurezza le Query
2. Mettere in Sicurezza le Configurazioni
3. Mettere in Sicurezza le Autenticazioni
4. Mettere in Sicurezza le Comunicazioni

Mettere in Sicurezza le Queries

Una SQL Injection si verifica quando l'input di un utente non attendibile viene aggiunto dinamicamente a una query SQL in modo non sicuro, spesso tramite concatenazione di stringhe. La SQL Injection è uno dei rischi più pericolosi per la sicurezza delle applicazioni. Le SQL Injection sono facili da sfruttare e potrebbero portare al furto, alla cancellazione o alla modifica dell'intero Database. L'applicazione può anche essere utilizzata per eseguire comandi pericolosi contro il sistema operativo che ospita il Database, dando così ad un aggressore un punto d'ingresso dalla rete.

Per mitigare l’SQL Injection, è necessario impedire che l'input non attendibile venga interpretato come parte di un comando SQL. Il modo migliore per farlo è con la tecnica di programmazione nota come "Parametrizzazione della Query". Questo meccanismo di difesa dovrebbe essere applicato a SQL, OQL e comandi di store procedure.

Un buon elenco di esempi di parametrizzazione delle query in ASP, ColdFusion, C#, Delphi, .NET, Go, Java, Perl, PHP, PL/SQL, PostgreSQL, Python, R, Ruby e Scheme può essere trovato qui http://bobby-tables.com e qui OWASP Cheat Sheet on Query Parameterization.

Prestare attenzione durante la Parametrizzazione

In alcuni casi alcune Query in determinate condizioni non sono parametrizzabili. Questo vale più o meno in modo differente per i diversi tipi di Database. Assicurarsi di eseguire una convalida della corrispondenza esatta o un “escape manuale” molto accurato quando si confrontano parametri di query del database che non possono essere associati a una query parametrizzata. Inoltre, mentre l'uso di query parametrizzate ha un impatto positivo sulle prestazioni, alcune query parametrizzate in specifiche implementazioni di database influenzeranno negativamente le prestazioni dell’accesso ai dati. Assicurarsi di verificare le prestazioni delle query; specialmente per query complesse con ampie capacità di ricerca di testo o clausole simili.

Mettere in Sicurezza le Configurazioni

Sfortunatamente, i sistemi di gestione dei database non vengono sempre forniti con una configurazione "secure by default". È necessario prestare attenzione per garantire che i controlli di sicurezza disponibili dal Database Management System (DBMS) e dalla piattaforma che lo ospita siano abilitati e configurati correttamente. Esistono standard, guide e benchmark, disponibili per i DBMS più comuni.

Autenticazioni Sicure

Tutti gli accessi ai database devono essere autenticati correttamente. L'autenticazione al DBMS deve essere effettuata in modo sicuro. L'autenticazione dovrebbe avvenire solo attraverso un canale sicuro. Le credenziali devono essere adeguatamente protette e disponibili per l'uso.

Comunicazioni Sicure

La maggior parte dei DBMS supporta molteplici metodi di comunicazione (servizi, API, ecc.) : sicuri (autenticati, crittografati) e non sicuri (non autenticati o non crittografati). È buona norma utilizzare solo le opzioni di comunicazione protetta trattata nella sezione “Proteggere i Dati Ovunque”

Prevenzione delle Vulnerabilità

- OWASP Top 10 2017- A1: Injection
- OWASP Mobile Top 10 2014-M1 Weak Server-Side Controls

OWASP Proactive Controls v 3.0

C4: Encode ed Escape dei Dati

Referenze

- OWASP Cheat Sheet: Query Parameterization
- Bobby Tables: A guide to preventing SQL injection
- CIS Database Hardening Standards

Descrizione

“Encode” ed "Escape" sono tecniche difensive pensate per bloccare i tentativi di injection. **Encode** (comunemente chiamato "Output Encoding") implica la traduzione di caratteri speciali in una forma diversa ma equivalente che non è più pericolosa per l’interprete di destinazione, ad esempio traducendo il carattere " **<** " nella stringa " **<** " quando si scrive su una **pagina HTML**. **Escape** implica l'aggiunta di un carattere speciale prima del carattere/stringa per evitare che venga interpretato male, ad esempio l'aggiunta di un carattere "****" prima di un carattere " **"** " (doppie virgolette) in modo che venga interpretato come testo e non come chiusura di una stringa.

L'**output encoding** viene applicato al meglio **subito prima** che il contenuto venga passato all'interprete di destinazione. Se questa difesa venisse eseguita troppo presto nell'elaborazione di una richiesta, l'Encode o l'Escape potrebbero interferire con l'uso del contenuto in altre parti del programma. Ad esempio, se esegui l'Escape HTML del contenuto prima di memorizzare tali dati nel database e l'interfaccia utente esegue automaticamente l'Escape di quei dati una seconda volta, il contenuto non verrà visualizzato correttamente a causa del doppio Escape.

Codifica Contestuale dell'Output

La Codifica Contestuale dell'output è una tecnica cruciale di programmazione sicura necessaria per arrestare gli XSS. Questa difesa viene eseguita sull'output, quando si crea un'interfaccia utente, all'ultimo momento prima che i dati non attendibili vengano aggiunti dinamicamente all'HTML. Il tipo di codifica dipenderà dalla posizione (o dal contesto) nel documento in cui i dati vengono visualizzati o memorizzati. I diversi tipi di codifica che verrebbero utilizzati per la creazione di interfacce utente sicure includono HTML Entity Encoding, HTML Attribute Encoding, JavaScript Encoding e URL Encoding.

Esempi di Java Encoding

Per esempi di OWASP Java Encoder che forniscono la codifica di output contestuale, vedere: OWASP Java Encoder Project Examples.

Esempi di .NET Encoding

A partire da .NET 4.5, la libreria Anti-Cross Site Scripting fa parte del framework, ma non è abilitata di default. È possibile specificare di utilizzare AntiXssEncoder da questa libreria come encoder predefinito per l'intera applicazione utilizzando le impostazioni web.conf.

Quando applicato è importante per la codifica contestualmente dell'output - ciò significa utilizzare la funzione corretta dalla libreria AntiXSSEncoder per la posizione appropriata dei dati nel documento.

Esempi di PHP Encoding

Zend Framework 2

Zend Framework 2 (ZF2), Zend\Escaper possono essere utilizzati per codificare l'output. Per esempi di codifica contestuale vedere Context-specific escaping with zend-escaper.

Altri tipi di difesa da Encoding ed Injection

Encoding/Escaping possono essere utlizzati per neutralizzare altre forme di injection. Ad esempio, è possibile neutralizzare alcuni metacaratteri speciali quando si aggiunge input ad un comando del sistema operativo. Questo è chiamato "OS Command Escaping", o "Shell Escaping". Questo meccanismo di difesa può essere usato per fermare la vulnerabilità di tipo "Command Injection".

Esistono altre forme di escaping che possono essere utilizzate per interrompere le injection, come l'escaping dell'attributo XML che interrompe varie forme di XML e XML Path Injection, nonché l'escaping di comandi LDAP che può essere utilizzato per interrompere varie forme di LDAP Injection

Encoding dei Caratteri e Canonizzazione

Unicode Encoding è un metodo per memorizzare caratteri con byte multipli. Ovunque siano consentiti dati in input, i dati possono essere inseriti utilizzando Unicode per mascherare il

codice dannoso e consentire una varietà di attacchi. RFC 2279 fa riferimento a molti modi in cui il testo può essere codificato.

La Canonizzazione è un metodo con il quale i sistemi convertono i dati in una forma più semplice o standard. Le applicazioni Web utilizzano comunemente la Canonizzazione dei caratteri per garantire che tutto il contenuto abbia lo stesso tipo di carattere quando viene memorizzato o visualizzato.

Essere protetti dagli attacchi correlati alla Canonizzazione significa che un'applicazione dovrebbe essere al sicuro quando vengono immessi Unicode e altre rappresentazioni di caratteri non corrette.

Prevenzione delle Vulnerabilità

- OWASP Top 10 2017 - A1: Injection
- OWASP Top 10 2017 - A7: Cross Site Scripting (XSS)
- OWASP Mobile_Top_10_2014-M7 Client-Side Injection

Riferimenti

- XSS - Informazioni Generali
- OWASP Cheat Sheet: XSS Prevention – Fermare XSS nelle web app
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: Injection Prevention

Tools

- OWASP Java Encoder Project
- AntiXSSEncoder
- Zend\Escaper - esempi di codifica contestuale

OWASP Proactive Controls v 3.0

C5: Convalidare tutti gli input

Descrizione

La convalida dell'input è una tecnica di programmazione che garantisce solo ai dati formattati correttamente l'ingresso nei componenti del software dell'applicazione.

Validità Sintattica e Semantica

Un'applicazione dovrebbe verificare che i dati siano validi sia sintatticamente che semanticamente (in questo ordine) prima di utilizzarli in qualsiasi modo (inclusa la visualizzazione all'utente).

Validità della sintassi significa che i dati sono nella forma prevista. Ad esempio, un'applicazione può consentire a un utente di selezionare un "ID account" di quattro cifre per eseguire qualche tipo di operazione. L'applicazione dovrebbe presumere che l'utente potrebbe star inserendo un payload SQL injection e dovrebbe verificare che i dati inseriti dall'utente siano esattamente di quattro cifre e siano costituiti solo da numeri (oltre a utilizzare una corretta parametrizzazione delle query).

Validità semantica include solo l'accettazione di input entro un intervallo accettabile per la funzionalità e il contesto dell'applicazione. Ad esempio, una data di inizio deve essere precedente a una data di fine quando si scelgono gli intervalli di date.

Whitelisting vs Blacklisting

Esistono due approcci generali per eseguire la convalida della sintassi degli input, comunemente noti come Blacklisting e Whitelisting:

- La *Blacklisting* o la *Convalida della Blacklist* tenta di verificare che i dati non contengano contenuti riconosciuti come dannosi. Ad esempio, un'applicazione web potrebbe bloccare l'input che contenente la dicitura <SCRIPT> per aiutare a prevenire XSS. Tuttavia, questa difesa potrebbe essere aggirata con un tag script minuscolo o misto maiuscolo/minuscolo.
- *La Whitelisting o Convalida della Whitelist tenta di verificare che un dato ricevuto corrisponda a una serie di regole "conosciute"*. Ad esempio, una regola di convalida

della Whitelist per uno Stato degli USA sarebbe un codice di due lettere corrispondente unicamente ad un solo Stato.

Quando si crea un software sicuro, la Whitelisting è l'approccio minimo consigliato. La lista nera è soggetta a errori e può essere aggirata con varie tecniche di evasione, quindi può essere pericoloso fare affidamento esclusivamente su di essa. Anche se la lista nera può spesso essere aggirata, talvolta può essere utile per aiutare a rilevare attacchi evidenti. Concludendo, mentre la Whitelist aiuta a limitare la superficie di attacco garantendo che i dati abbiano la giusta validità sintattica e semantica, la Blacklist aiuta a rilevare e potenzialmente bloccare attacchi evidenti.

Convalida lato Client e lato Server

La convalida dell'input deve essere sempre eseguita lato server per motivi di sicurezza. Sebbene la convalida lato client possa essere utile sia per scopi funzionali che per alcuni scopi di sicurezza, spesso può essere facilmente aggirata. Ciò rende la convalida lato server ancora più fondamentale per la sicurezza. Ad esempio, la convalida JavaScript può avvisare l'utente che un particolare campo deve essere composto da numeri, ma l'applicazione lato server deve convalidare che tutti i dati inviati siano composti solo da cifre comprese nell'intervallo numerico appropriato per quella funzione.

Espressioni Regolari

Le Espressioni Regolari offrono un modo per verificare che i dati corrispondano ad un modello specifico. Cominciamo con un esempio di base.

La seguente Espressione Regolare viene utilizzata per definire una regola Whitelist per la convalida dei nomi utente.

```
^[a-z0-9_]{3,16}$
```

Questa Espressione Regolare consente solo lettere minuscole, numeri e caratteri e linea di underscore. La lunghezza del nome utente è inoltre limitata dai 3 ai 16 caratteri.

Attenzione: Pericolo di Denial of Service

È necessario prestare attenzione durante la creazione delle Espressioni Regolari. Espressioni realizzate male possono causare potenziali condizioni di Denial of Service (come ReDoS). Con svariati strumenti é possibile verificare che le espressioni regolari non siano vulnerabili al ReDoS.

Attenzione alla Complessità

Le espressioni regolari sono solo un modo per eseguire la convalida. Le espressioni regolari possono essere difficili da gestire o essere comprese da alcuni sviluppatori. Altre alternative implicano la scrittura di metodi di convalida a livello di codice, che possono essere più facili da gestire per alcuni sviluppatori.

Limiti della Convalida dell'Input

La convalida dell'input non sempre rende i dati "sicuri", poiché alcune forme di input complessi possono essere "valide" ma comunque pericolose. Per esempio un indirizzo email valido può contenere un attacco SQL injection o un URL valido può contenere un attacco Cross Site Scripting. Oltre alla convalida dell'input, dovrebbero sempre essere applicati ulteriori meccanismi di difesa per i dati, come la parametrizzazione delle query o l'escaping.

Challenge do Covalida dei Dati Serializzati

Alcune forme di input sono così complesse che la convalida può proteggere solo in minima parte l'applicazione. Ad esempio, è pericoloso deserializzare dati non attendibili o dati che possono essere manipolati da un utente malintenzionato. L'unico pattern architetturale sicuro è quello di non accettare oggetti serializzati da fonti non attendibili, oppure deserializzare solo con capacità limitata e solo per tipi di dati semplici. Si dovrebbe evitare di elaborare formati di dati serializzati, ed utilizzare, quando possibile, formati più facili da difendere, come JSON.

Se ciò non fosse possibile, prendere in considerazione una serie di difese di convalida durante l'elaborazione dei dati serializzati.

- Implementare controlli di integrità o crittografare gli oggetti serializzati, per impedire la creazione di oggetti ostili o la manomissione dei dati.
- Applicare rigidi vincoli di “type constraints” durante la deserializzazione prima della creazione dell'oggetto; in genere il codice si aspetta un insieme di classi definite. Sono stati dimostrati dei bypass a questa tecnica di controllo.

- Isolare il codice che deserializza, tipo quello eseguito in ambienti con privilegi molto bassi, come contenitori temporanei.
- Registrare le eccezioni e gli errori di deserializzazione della sicurezza, ad esempio quando il tipo in ingresso non è il tipo previsto o la deserializzazione genera eccezioni.
- Limitare o monitorare la connettività di rete in entrata e in uscita da contenitori o da server che deserializzano.
- Monitorare la deserializzazione, avvisando se un utente deserializza costantemente.

Input Utente Inaspettato (Mass Assignment)

Alcuni framework supportano l'associazione automatica dei parametri delle richieste HTTP agli oggetti lato server utilizzati dall'applicazione. Questa funzione di associazione automatica può consentire ad un utente malintenzionato di aggiornare gli oggetti lato server che non dovevano essere modificati. L'attaccante può eventualmente modificare il proprio livello di controllo d'accesso o aggirare la logica aziendale prevista dell'applicazione con questa funzionalità.

Questo attacco ha svariati nomi, tra i quali: mass assignment, autobinding ed object injection.

Prendendo un esempio semplice, se l'oggetto utente dotato di specifici privilegi, usa un campo dove è possibile rilevare il tipo di privilegio, un utente malintenzionato può cercare le pagine in cui i dati dell'utente vengono modificati e aggiungere "privilege=admin" ai parametri HTTP inviati. Se l'associazione automatica è abilitata in modo non sicuro, l'oggetto lato server che rappresenta l'utente verrà modificato di conseguenza.

È possibile utilizzare due approcci per gestire questo problema:

- Evitare di associare direttamente l'input e utilizzare invece un DTO (Data Transfer Object).
- Abilitare l'associazione automatica ma impostare le regole della Whitelist per ogni pagina o funzionalità, per definire quali campi possono essere associati automaticamente.
- Altri esempi sono disponibili nell' OWASP Mass Assignment Cheat Sheet.

Convalida e Sanitizzazione HTML

Consideriamo un'applicazione che deve accettare HTML dagli utenti (tramite un editor WYSIWYG che rappresenta il contenuto come HTML o funzionalità che accettano direttamente HTML in input). In questa situazione la convalida o l'escaping non aiuteranno.

- Le espressioni regolari non sono abbastanza esaustive per comprendere la complessità di HTML5.
- La codifica o l'escaping dell'HTML non sarà di aiuto poiché causerà un rendering dell'HTML non corretto.

Pertanto, è necessaria una libreria in grado di analizzare e pulire il testo formattato in HTML. Si prega di consultare l'XSS Prevention Cheat Sheet on HTML Sanitization per ulteriori informazioni sulla sanitizzazione HTML.

Funzionalità di Convalida nelle Librerie e nei Framework

Tutti i linguaggi e la maggior parte dei framework forniscono librerie o funzioni di convalida che dovrebbero essere sfruttate per convalidare i dati. Le librerie di convalida in genere coprono tipi di dati comuni, requisiti di lunghezza, intervalli di numeri interi, controlli "is null" e altro ancora. Molte Librerie e Framework consentono di definire la propria espressione regolare o logica per la convalida personalizzata in un modo che consente al programmatore di sfruttare tale funzionalità in tutta l'applicazione. Esempi di funzionalità di convalida includono PHP's filter functions per PHP o Hibernate Validator per Java. Esempi di HTML Sanitizers includono Ruby on Rails sanitize method, OWASP Java HTML Sanitizer or DOMPurify.

Prevenzione delle Vulnerabilità

- La convalida dell'input riduce la superficie di attacco delle applicazioni e talvolta può rendere gli attacchi contro un'applicazione più difficili.
- La convalida dell'input è una tecnica che fornisce sicurezza a determinate forme di dati, specifiche per determinati attacchi e non può essere applicata in modo affidabile come regola generale di sicurezza.
- La convalida dell'input non dovrebbe essere utilizzata come metodo principale di prevenzione da XSS, SQL Injection e altri attacchi

Riferimenti

- OWASP Cheat Sheet: Input Validation
- OWASP Cheat Sheet: iOS - Security Decisions via Untrusted Inputs
- OWASP Testing Guide: Testing for Input Validation

Tools

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate Validator
- JEP-290 Filter Incoming Serialization Data
- Apache Commons Validator
- PHP's filter functions

OWASP Proactive Controls v 3.0

C6: Implementare la Digital Identity

Descrizione

Digital Identity è la rappresentazione univoca di un utente (o di un altro soggetto) mentre effettua una transazione online. L'autenticazione è il processo con cui si verifica che un individuo o un'entità siano chi dichiarano di essere. La gestione delle sessioni è un processo mediante il quale un server mantiene lo stato dell'autenticazione degli utenti in modo che l'utente possa continuare a utilizzare il sistema senza eseguire nuovamente l'autenticazione. Il NIST Special Publication 800-63B: Digital Identity Guidelines (Authentication and Lifecycle Management fornisce una solida guida sull'implementazione delle identità digitali, delle autenticazioni e dei controlli di gestione delle sessioni.

Di seguito sono riportati alcuni consigli per un'implementazione sicura.

Livelli di autenticazione

NIST 800-63b descrive tre livelli di affidabilità di un'autenticazione chiamata Authentication Assurance Level (AAL). AAL livello 1 è riservato per applicazioni a basso rischio che non contengono dati personali (Personally Identifiable Information o PII) o altri dati privati. A livello 1 AAL è richiesta solo l'autenticazione a fattore singolo, in genere tramite l'uso di una password.

Livello 1: password

Le password sono davvero molto importanti. Abbiamo bisogno di policy, dobbiamo salvarle in modo sicuro, e a volte dobbiamo consentire agli utenti di ripristinarle.

Requisiti per le password

Le password devono soddisfare perlomeno i seguenti requisiti:

- contenere almeno 8 caratteri quando utilizzati anche l'autenticazione a più fattori (MFA) o altri controlli. Se MFA non è possibile, la lunghezza dovrebbe essere aumentata ad almeno 10 caratteri
- tutti i caratteri ASCII stampati così come il carattere spazio dovrebbero essere accettabili nelle domande segrete per il ripristino
- incoraggiare l'uso di password e passphrase lunghe

- rimuovere i requisiti di complessità poiché si è riscontrato siano di efficacia limitata. Si consiglia invece l'adozione di Multi-factor authentication (MFA) o di password di lunghezza maggiore
- assicurarsi che le password utilizzate non siano password di uso comune o che siano già trapelate in precedenti data breach. Puoi scegliere di bloccare le prime 1000 o 10000 password più comuni che soddisfano i requisiti di lunghezza sopra indicati e che si trovano negli elenchi pubblici di password compromesse. Il seguente link contiene le password più comuni: https://github.com/danielmiessler/SecLists/tree/master/Passwords

Implementare meccanismi sicuri di recupero password

È comune che un'applicazione abbia un meccanismo atto a consentire ad un utente di accedere al proprio account nel caso in cui costui si dimentichi la password. Un buon design del workflow per una procedura di recupero della password utilizzerà elementi di autenticazione a più fattori. Ad esempio, potrebbe porre una domanda di sicurezza (qualcosa che conosco) oppure inviare un token generato a un dispositivo (qualcosa che possiedo).

Si prega di consultare Forgot_Password_Cheat_Sheet e Choosing_and_Using_Security_Questions_Cheat_Sheet per ulteriori dettagli.

Implementare l'archiviazione sicura delle password

Per fornire forti controlli di autenticazione, un'applicazione deve archiviare in modo sicuro le credenziali utente. Inoltre, dovrebbero essere predisposti controlli crittografici in modo tale che se una credenziale (ad esempio una password) è compromessa, l'aggressore non abbia accesso immediato a queste informazioni.

Esempio in PHP per l'archiviazione delle password

Di seguito è riportato un esempio per l'hashing sicuro della password in PHP utilizzando la funzione password_hash () (disponibile dalla versione 5.5.0) che utilizza come impostazione predefinita l'algoritmo bcrypt. L'esempio utilizza un work factor di 15.

```
<?php
  $cost = 15;
  $password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" =>
$cost] );
?>
```

Si prega di consultare OWASP Password Storage Cheat Sheet per ulteriori dettagli.

Livello 2: autenticazione a più fattori

NIST 800-63b AAL livello 2 è riservato per applicazioni ad alto rischio che contengono "self-asserted PII o altre informazioni personali rese disponibili online." A livello AAL 2 è richiesta l'autenticazione a più fattori, inclusa OTP o altre forme di implementazione a più fattori.

L'autenticazione a più fattori (MFA) garantisce che gli utenti siano chi dichiarano di essere, richiedendo loro di identificarsi con una combinazione di:

- Qualcosa che conosci: password o PIN
- Qualcosa che possiedi: token o telefono
- Qualcosa che sei: biometrie, come un'impronta digitale

L'utilizzo delle password come unico fattore fornisce una protezione debole. Le soluzioni multi-fattore forniscono una soluzione più robusta, poiché richiedono ad un attaccante di acquisire più di un elemento per autenticarsi al servizio.

Vale la pena notare che anche i dati biometrici, se utilizzati come singolo fattore di autenticazione, non sono considerati abbastanza sicuri per l'autenticazione digitale. Infatti possono essere ottenuti online o scattando una foto di qualcuno con una fotocamera del telefono (immagini del viso), prelevati da oggetti che qualcuno tocca (impronte digitali latenti) o catturati con immagini ad alta risoluzione (reticolo dell'iride). I dati biometrici devono essere utilizzati solo come parte dell'autenticazione a più fattori con un autenticatore fisico (qualcosa che possiedi). Ad esempio, accedere ad un dispositivo OTP (Multi-Factor One-Time Password) per generare una password monouso che l'utente potrà inserire manualmente.

Livello 3: autenticazione basata sulla crittografia

NIST 800-63b Authentication Assurance Level 3 (AAL3) è richiesto quando l'impatto ai sistemi compromessi potrebbe portare a danni personali, perdite finanziarie significative, danneggiare l'interesse pubblico o comportare violazioni civili o penali. AAL3 richiede un'autenticazione "basata sulla prova del possesso di una chiave attraverso un protocollo crittografico." Questo tipo di autenticazione viene utilizzato per ottenere il massimo livello di affidabilità dell'autenticazione. Questa operazione viene in genere eseguita tramite moduli crittografici hardware.

Gestione delle sessioni

Una volta eseguita con successo l'autenticazione iniziale dell'utente, un'applicazione può scegliere di tenere traccia e mantenere questo stato di autenticazione per un periodo di tempo limitato. Ciò consentirà all'utente di continuare a utilizzare l'applicazione senza dover autenticarsi nuovamente ad ogni richiesta. Il tracciamento di questo stato utente è chiamato Gestione delle Sessioni.

Generazione e scadenza della sessione

Lo stato dell'utente viene tracciato durante la sessione. Questa sessione viene in genere memorizzata sul server per la classica gestione della sessione basata sul Web. Viene quindi fornito un identificatore di sessione all'utente in modo che possa identificare quale sessione lato server contiene i dati utente corretti. Il client deve solo conservare questo identificatore di sessione, il quale custodisce anche i dati sensibili della sessione lato server.

Di seguito sono riportati alcuni controlli da tenere in considerazione durante la creazione o l'implementazione di soluzioni per la gestione delle sessioni:

- Essere sicuri che l'ID di sessione sia lungo, unico e casuale.
- L'applicazione dovrebbe generare una nuova sessione o almeno aggiornare l'id di sessione durante l'autenticazione e la nuova autenticazione.
- L'applicazione dovrebbe implementare un timeout di inattività dopo un periodo di inutilizzo ed una durata massima assoluta per ciascuna sessione, dopodiché gli utenti devono autenticarsi nuovamente. La durata dei timeout dovrebbe essere inversamente proporzionale al valore dei dati protetti.

Si prega di consultare il Session Management Cheat Sheet per maggiori dettagli. ASVS Section 3 copre requisiti aggiuntivi per la gestione delle sessioni.

Browser Cookies

I cookie del browser vengono comunemente usati per memorizzare gli identificatori di sessione per le applicazioni Web che implementano tecniche di gestione delle sessioni standard. Ecco alcuni meccanismi di difesa da considerare quando si utilizzano i cookie del browser.

- Quando i cookie del browser vengono utilizzati come meccanismo per tracciare la sessione di un utente autenticato, questi dovrebbero essere accessibili ad un insieme

minimo di domini o percorsi e dovrebbero essere contrassegnati per scadere subito dopo il periodo di validità della sessione.

- Il flag "secure" dovrebbe essere impostato per garantire che il trasferimento avvenga solo tramite un canale protetto (TLS).
- Il flag HttpOnly dovrebbe essere impostato per impedire l'accesso al cookie tramite JavaScript.
- L'aggiunta dell'attributo "samesite" ai cookie impedisce ad alcuni browser moderni di inviare cookie con richieste cross-site e fornisce protezione contro la contraffazione di richieste cross-site e contro la fuga di informazioni.

Tokens

Le sessioni lato server possono essere limitanti per alcune forme di autenticazione. Gli "Stateless Services" consentono la gestione lato client dei dati di sessione al fine di migliorare le prestazioni, in modo che il server abbia meno oneri nell'archiviare e verificare la sessione utente. Queste applicazioni "Stateless" generano un token di accesso di breve durata che può essere utilizzato per autenticare una richiesta client senza inviare le credenziali dell'utente dopo l'autenticazione iniziale.

JWT (JSON Web Tokens)

JSON Web Token (JWT) è uno standard aperto (RFC 7519) che definisce un modo compatto e autonomo di trasmettere in modo sicuro le informazioni tra le parti sotto forma di oggetto JSON. Queste informazioni possono essere verificate e ritenute attendibili poiché sono firmate digitalmente. Un token JWT viene creato durante l'autenticazione e viene verificato dal server (o dai server) prima di qualsiasi elaborazione.

Tuttavia, i JWT spesso non vengono salvati dal server dopo la creazione iniziale. I JWT vengono generalmente creati e quindi assegnati ad un client senza essere salvati in alcun modo dal server. L'integrità del token viene mantenuta attraverso l'utilizzo di firme digitali, in modo che un server possa successivamente verificare che il JWT sia ancora valido e non sia stato manomesso da quando é stato creato.

Questo approccio è allo stesso tempo Stateless e portabile, in quanto le tecnologie client e server possono essere diverse ma possono comunque interagire.

Attenzione

L'identità digitale, l'autenticazione e la gestione delle sessioni sono argomenti molto importanti. Questa è solo la punta dell'iceberg di questi temi. Assicurarsi che queste funzionalità siano progettate dai migliori ingegneri del team di sviluppo, i quali devono assumersi la responsabilità anche della successiva manutenzione.

Prevenzione delle Vulnerabilità

- OWASP Top 10 2017 A2- Broken Authentication and Session Management
- OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication

Riferimenti

- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Choosing and Using Security Questions
- OWASP Cheat Sheet: Session Management
- OWASP Cheat Sheet: IOS Developer
- OWASP Testing Guide: Testing for Authentication
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines

Tools

- Daniel Miessler: Most commonly found passwords

OWASP Proactive Controls v 3.0

C7: Applicare il Controllo degli Accessi

Descrizione

Il Controllo degli Accessi (o Autorizzazione) è il processo che autorizza o nega le richieste specifiche di un utente, programma o processo. Il Controllo degli Accessi implica anche l'atto di concedere e revocare tali privilegi.

É opportuno sottolineare che l'autorizzazione (verifica degli accessi a funzionalità o risorse specifiche) non è equivalente all'autenticazione (verifica dell'identità).

La funzionalità del controllo degli accessi spesso copre molte aree del software, a seconda della complessità del sistema di controllo degli accessi. Ad esempio, la gestione dei metadati di controllo degli accessi o la creazione di cache per scopi di scalabilità sono spesso componenti aggiuntivi in un sistema di controllo degli accessi, che devono essere creati o gestiti.

Esistono diversi tipi di design per Controllo degli Accessi che dovrebbero essere considerati.

- Il Discretionary Access Control (DAC) è un mezzo per limitare l'accesso agli oggetti (ad esempio file o Data Entities) in base alle identità e alle necessità dei soggetti (es. utenti, processi) e / o dei gruppi a cui appartiene l'oggetto.
- Il Mandatory Access Control (MAC) è un mezzo per limitare l'accesso alle risorse di sistema in base alla sensibilità (rappresentata da un'etichetta) delle informazioni contenute in tale risorsa e all'autorizzazione formale (cioè l'autorizzazione) degli utenti ad accedere alle informazioni con tale sensibilità.
- Il Role Based Access Control (RBAC) è un modello per il controllo degli accessi alle risorse, dove le azioni consentite sulle risorse sono identificate con dei ruoli piuttosto che con delle identità di soggetti individuali.
- L'Attribute Based Access Control (ABAC) concederà o negherà le richieste dell'utente in base ad attributi arbitrari dell'utente, ad attributi arbitrari dell'oggetto e a condizioni ambientali, che possono essere riconosciute a livello globale e più rilevanti per le politiche in questione.

Principi di progettazione del controllo degli accessi

I seguenti requisiti "positivi" per la progettazione del controllo degli accessi dovrebbero essere considerati nelle fasi iniziali dello sviluppo dell'applicazione.

1) Progettare in anticipo il controllo degli accessi in modo accurato

Dopo aver scelto uno specifico modello di progettazione del controllo degli accessi, è spesso difficile e richiede tempo riprogettarlo nella propria applicazione con un nuovo modello. Il controllo degli accessi è una delle aree principali della progettazione della sicurezza delle applicazioni che deve essere accuratamente progettata in anticipo, soprattutto quando si affrontano requisiti come il multi-tenancy ed il controllo degli accessi orizzontale (dipendente dai dati).

La progettazione del controllo degli accessi può iniziare in modo elementare ma spesso può trasformarsi in un controllo di sicurezza complesso e ricco di funzionalità. Quando si valuta la capacità di controllo degli accessi dei framework, assicurarsi che sia consentita la personalizzazione per eventuali esigenze specifiche.

2) Costringere tutte le richieste a passare tramite la verifica del controllo degli accessi

Assicurarsi che tutte le richieste passino attraverso una sorta di livello di verifica del controllo degli accessi. Tecnologie come i filtri Java o altri meccanismi di elaborazione automatica delle richieste sono artefatti di programmazione ideali che aiuteranno a garantire che tutte le richieste passino attraverso una sorta di controllo degli accessi.

3) Deny by Default

Deny by default è il principio per il quale una richiesta non espressamente consentita, viene rifiutata. Questa regola può essere presente nel codice dell'applicazione in vari modi. Alcuni esempi sono:

1. Il codice dell'applicazione può generare un errore o un'eccezione durante l'elaborazione delle richieste al controllo dell'accesso. In questi casi l'accesso dovrebbe essere sempre negato.
2. Quando un amministratore crea un nuovo utente o un utente si registra per un nuovo account, quell'account dovrebbe avere un accesso minimo o nessun accesso per impostazione predefinita fino a quando tale accesso non viene verificato e configurato.

3. Quando una nuova funzione viene aggiunta ad un'applicazione, a tutti gli utenti dovrebbe essere negato l'utilizzo di tale funzione fino a quando non viene configurata correttamente.

4) Principio del Privilegio Minimo

Assicurarsi che a tutti gli utenti, programmi o processi venga concesso solo il minimo accesso necessario possibile. Diffidare dei sistemi che non forniscono funzionalità granulari per le configurazioni del controllo degli accessi.

5) Non inserire ruoli nel codice

Per impostazione predefinita, molti framework utilizzano il controllo degli accessi basato sui ruoli. È comune trovare il codice dell'applicazione controlli di questa natura.

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
      deleteAccount();
}
```

Prestare attenzione nel codice a questo tipo di programmazione basata sui ruoli. Può presentare le seguenti limitazioni o pericoli:

- La programmazione basata sui ruoli di questa natura è fragile. È facile creare nel codice controlli di ruolo errati o mancanti.
- La programmazione basata sui ruoli non consente il multi-tenancy. Saranno necessarie misure estreme come il fork del codice o controlli aggiuntivi per ogni cliente, atti a consentire ai sistemi basati sui ruoli di avere regole diverse per clienti diversi.
- La programmazione basata sui ruoli non consente regole orizzontali o data-specific per il controllo degli accessi.
- Database di grandi dimensioni con molti check sul controllo degli accessi, possono risultare difficili sia da revisionare che da verificarne le policy generali.

Si prega invece di considerare la seguente metodologia di programmazione per controllo degli accessi:

```
if (user.hasAccess("DELETE_ACCOUNT")) {
      deleteAccount();
}
```

Le verifiche degli accessi basate su attributi o funzionalità di questa natura sono il punto di partenza per la costruzione dei sistemi di controllo degli accessi ben progettati e ricchi di funzionalità. Questo tipo di programmazione consente anche una maggiore capacità di personalizzazione del controllo degli accessi nel tempo.

6) Tracciare tutti gli eventi sui controlli di accesso

Tutti gli errori di controllo dell'accesso devono essere registrati in quanto potrebbero essere indicativi di un utente malintenzionato che sta sondando l'applicazione per individuare eventuali vulnerabilità.

Prevenzione delle Vulnerabilità

- OWASP Top 10 2017-A5-Broken Access Control
- OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication

Referenze

- OWASP Cheat Sheet: Access Control
- OWASP Cheat Sheet: iOS Developer - Poor Authorization and Authentication
- OWASP Testing Guide: Testing for Authorization

Tools

- OWASP ZAP with the optional Access Control Testing add-on

OWASP Proactive Controls v 3.0

C8: Proteggere i Dati Ovunque

Descrizione

Dati sensibili come password, numeri di carte di credito, cartelle cliniche, informazioni personali e segreti industriali richiedono una protezione aggiuntiva, in particolare se tali dati rientrano nelle leggi sulla privacy (Regolamento generale sulla protezione dei dati dell'UE, il GDPR), regole sulla protezione dei dati finanziari come PCI Data Security Standard (PCI DSS) o altre normative.

Gli aggressori possono sottrarre dati dalle applicazioni web e dai servizi web in diversi modi. Ad esempio, se le informazioni sensibili vengono inviate (attraverso Internet) in chiaro, un utente malintenzionato su una connessione wireless condivisa potrebbe vedere e rubare i dati di un altro utente. Inoltre, un utente malintenzionato potrebbe utilizzare una SQL Injection per rubare password e altre credenziali dal database di un'applicazione ed esporre tali informazioni al pubblico.

Classificazione dei dati

È fondamentale classificare i dati nel sistema e determinare il livello di sensibilità a cui appartiene ciascun dato. Ogni categoria di dati può quindi essere associata a regole di protezione necessarie per ogni livello di sensibilità. Ad esempio, informazioni di marketing non sensibili possono essere classificate come dati pubblici, e possono quindi essere inseriti in sito web pubblico. I numeri di carta di credito possono essere classificati come dati utente privati, e dovrebbero quindi essere crittografati durante l'archiviazione e durante il transito.

Crittografare i Dati in Transito

Quando si trasmettono dati sensibili attraverso qualsiasi rete, è necessario considerare una qualche forma di sicurezza delle comunicazioni end-to-end (o crittografia in transito). TLS è di gran lunga il protocollo crittografico più comune e ampiamente supportato per la sicurezza delle comunicazioni. Viene utilizzato da molti tipi di applicazioni (web, webservice, mobile) per comunicare attraverso la rete in modo sicuro. TLS deve essere configurato correttamente in vari modi per difendere adeguatamente le comunicazioni protette.

Il vantaggio principale della sicurezza del layer di trasporto è la protezione dei dati delle applicazioni Web dalla divulgazione e dalla modifica non autorizzate quando vengono trasmessi tra i client (browser Web) e il server dell'applicazione Web o tra il server di applicazioni Web e il back-end di altre componenti aziendali.

Crittografia dei dati inattivi

La prima regola per la gestione dei dati sensibili è evitare di archiviarli quando possibile. Se fosse necessario archiviare dati sensibili, assicurarsi che siano protetti crittograficamente in qualche modo, per evitare divulgazioni e modifiche non autorizzate.

La crittografia è uno degli argomenti più avanzati riguardante la sicurezza delle informazioni, la cui comprensione richiede la massima conoscenza ed esperienza. È difficile farla bene perché esistono molti approcci alla crittografia, ciascuno con vantaggi e svantaggi che devono essere compresi a fondo dagli architetti e dagli sviluppatori di soluzioni web. Inoltre, uno studio approfondito sulla crittografia è tipicamente basato sulla matematica avanzata e sulla teoria dei numeri, e ciò crea una notevole barriera iniziale.

Invece di creare funzionalità crittografiche da zero, si consiglia vivamente di utilizzare soluzioni open e sottoposte a revisione pubblica periodica, come il progetto Google Tink, Libsodium, e funzionalitá di archiviazione sicura incorporate in molti framework e servizi in cloud.

Applicazioni per dispositivi mobili: archiviazione locale sicura

Le applicazioni per dispositivi mobili sono particolarmente a rischio di fuga dati, poiché i dispositivi mobili vengono regolarmente persi o rubati pur contenendo dati sensibili.

Come regola generale, solo la quantità minima di dati necessari dovrebbero essere memorizzata su un dispositivo mobile. Tuttavia, se fosse necessario archiviare dati sensibili su un dispositivo mobile, questi devono essere archiviati all'interno di ciascuna directory di archiviazione dati specifica del sistema operativo mobile. Su Android questa directory sarà il keystore e su iOS sarà il keychain.

Ciclo di vita delle chiavi

Le chiavi di cifratura vengono utilizzate nelle applicazioni per le funzioni che trattano dati sensibili. Ad esempio, le chiavi segrete possono essere utilizzate per firme JWT, per proteggere carte di credito, fornire varie forme di autenticazione e facilitare altre funzioni di

sicurezza sensibili. Nella gestione delle chiavi, è necessario seguire una serie di regole, tra le quali:

- Assicurarsi che ogni chiave segreta sia protetta da accessi non autorizzati
- Conservare le chiavi in un vault segreto appropriato come descritto di seguito
- Utilizzare chiavi indipendenti quando sono necessarie più chiavi
- Crea supporto per modificare algoritmi e chiavi quando necessario
- Creare funzionalità dell'applicazione per gestire una rotazione delle chiavi

Gestione dei segreti dell'applicazione

Le applicazioni contengono numerosi "segreti" necessari per le operazioni di sicurezza. Questi includono certificati, password di connessione SQL, credenziali di account di servizi di terze parti, password, chiavi SSH, chiavi di crittografia e altro. La divulgazione o la modifica non autorizzata di questi segreti potrebbe portare alla completa compromissione del sistema. Nella gestione dei segreti dell'applicazione, considerare quanto segue.

- Non archiviare segreti nel codice, nei file di configurazione o passarli attraverso variabili ambientali. Usare strumenti come GitRob o TruffleHog per scansionare i repository del codice per cercare di trovarne.
- Conservare le chiavi e gli altri segreti a livello applicativo in un vault segreto come KeyWhiz o Hashicorp's Vault project o Amazon KMS per fornire archiviazione sicura e accesso ai segreti a livello applicativo in fase di esecuzione.

Prevenzione delle Vulnerabilità

- OWASP Top 10 2017 - A3: Sensitive Data Exposure
- OWASP Mobile Top 10 2014-M2 Insecure Data Storage

Referenze

- OWASP Cheat Sheet: Transport Layer Protection
- Ivan Ristic: SSL/TLS Deployment Best Practices
- OWASP Cheat Sheet: HSTS
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: IOS Developer - Insecure Data Storage
- OWASP Testing Guide: Testing for TLS

Tools

- SSLyze - SSL configuration scanning library and CLI tool
- SSLLabs - Free service for scanning and checking TLS/SSL configuration

- OWASP O-Saft TLS Tool - TLS connection testing tool
- GitRob - Command line tool to find sensitive information in publicly available files on GitHub
- TruffleHog - Searches for secrets accidentally committed
- KeyWhiz - Secrets manager
- Hashicorp Vault - Secrets manager
- Amazon KMS - Manage keys on Amazon AWS

OWASP Proactive Controls v 3.0

C9: Implementare il Logging e il Monitoraggio della sicurezza

Descrizione

Il logging è un concetto che la maggior parte degli sviluppatori utilizza già per scopi di debug e diagnostica. Il security logging è un concetto altrettanto basilare: fare il log di informazioni riguardanti la sicurezza durante l'operazione di runtime di un'applicazione. Il monitoraggio è la revisione in tempo reale delle applicazioni e dei security log utilizzando varie forme di automazione. Gli stessi strumenti e modelli possono essere utilizzati per operazioni, debug e scopi di sicurezza.

Vantaggi del Security Logging

Security logging può essere utilizzato per:

1) Alimentare sistemi di rilevamento delle intrusioni
2) Analisi e indagini dim digital forensics
3) Soddisfare i requisiti di conformità di eventuali normative

Implementazione del Security Logging

La seguente è una lista di best practices per i log di sicurezza.

- Seguire uno standard per quanto riguarda il format dei log e approcciarsi tra il sistema e i sistemi dell' organizzazione.
- Non "loggare" troppo o troppo poco. Nei log devono essere presenti tutte le informazioni utili ma non devono esserci presenti i dati sensibili o confidenziali.
- Prestare attenzione ai fusi orari dei log e la sincronizzazione degli orologi

Di seguito è riportato un elenco di best practice per l'implementazione della registrazione della sicurezza.

- Segui un formato e un approccio di registrazione comuni all'interno del sistema e tra i sistemi di un'organizzazione. Un esempio di un framework di registrazione comune è Apache Logging Services che aiuta a fornire coerenza di registrazione tra le applicazioni Java, PHP, .NET e C ++.

- Non registrare troppo o troppo poco. Ad esempio, assicurati di registrare sempre il timestamp e le informazioni di identificazione, inclusi l'IP di origine e l'ID utente, ma fai attenzione a non registrare dati privati o riservati.
- Presta molta attenzione alla sincronizzazione dell'ora tra i nodi per assicurarti che i timestamp siano coerenti.

Tracciamento di Intrusioni Rilevazioni e Risposte

Usare la procedura di logging per identificare l'attività che rileva un comportamento sospetto di un utente. L'attività da segnalare nei log include:

- Ingresso di dati che supera un range numerico predefinito.
- Dati inviati che comportano modifiche a dati che non dovrebbero essere modificabili (selezionare elenco, casella di controllo o altri componenti a immissione limitata).
- Richieste che violano le regole di controllo dell'accesso lato server.
- È disponibile un elenco più completo dei possibili punti di rilevamento qui.

Quando l'applicazione rileva tali attività anomale, l'applicazione dovrebbe almeno registrare l'attività e contrassegnarla come un problema di gravità elevata. Idealmente, l'applicazione dovrebbe anche rispondere a un possibile attacco identificato, ad esempio invalidando la sessione dell'utente e bloccando l'account dell'utente. I meccanismi di risposta consentono al software di reagire in tempo reale a possibili attacchi identificati.

Progettare un Design Sicuro del Tracciamento

Le soluzioni di logging devono essere costruite e gestite in modo sicuro. Il logging sicuro dovrebbe rispettare le seguenti regole:

- Codificare e validare qualsiasi carattere pericoloso prima del loging per prevenire attacchi di log injection o di log forging
- Non salvare in log informazioni sensibili. Per esempio non salvare log di password, session ID o carte di credito.
- Proteggere l'integrità dei log. Un attaccante potrebbe provare un attacco tamper, per cui andrebbero gestiti anche i permessi di lettura dei log.
- Trasmettere I log dalla macchina ad un sistema centralizzato, onde evitare la compromissione dei log e facilitare il monitoring esterno.

Referenze

- OWASP AppSensor Detection Points - Punti usati per identificare un utente malevolo che cerca di individuare vulnerabilità o debolezze in un' applicazione.
- OWASP Log injection
- OWASP Log forging
- OWASP Cheat Sheet: Logging Come implementare correttamente il logging in un'applicazione
- OWASP Development Guide: Logging
- OWASP Code Review Guide: Reviewing Code for Logging Issues

Tools

- OWASP Security Logging Project
- Apache Logging Services

OWASP Proactive Controls v 3.0

C10: Handle all Errors and Exceptions

Descrizione

La gestione delle eccezioni è un concetto della programmazione che permette ad una applicazione di rispondere a diversi tipi di errori (come la rete non raggiungibile, la connessione interrotta al database, ecc.) in diverse maniere.

Gestire le eccezioni e gli errori correttamente è essenziale per rendere il tuo codice affidabile e sicuro.

La gestione delle eccezioni si può applicare in ogni area dell' applicazione includendo la logica nel business come la sicurezza del codice.

La gestione degli errori è importante allo stesso modo da parte di un'attività di intrusion detection. Alcuni attacchi alla tua applicazione possono triggerare errori che possono aiutare ad identificare l'attacco.

Gestione degli errori sbagliata

I ricercatori all'università di Toronto hanno scoperto che anche dei piccoli errori di configurazione della "Error Handling" possono portare a fallimenti catastrofici nei sistemi in distribuzione.

Queste configurazioni errate possono portare a diverse vulnerabilità quali:

- Information leakage: La fuoriuscita di informazioni sensibili nei messaggi di errori può permettere agli attaccanti di ottenere informazioni sul sistema o su quale servizio è possibile effettuare un determinato attacco (es. Il ritorno di "utente non trovato" o "password errata") procurandosi indizi su come il sistema è stato costruito.
- TLS bypass: Il "fail bug" di Apple era un messaggio di errore gestito malamente che portava la compromissione delle connessioni TLS nei sistemi Apple.
- DOS: Una mancanza di configurazione dell'error handling può causare un crash dei sistemi, oltre che la possibilità che un attaccante appesantisca volontareamente la CPU o il disco per rallentarli o bloccarli.

Consigli

- Gestire le eccezioni in un metodo centralizzato per evitare di avere nel codice try/catch ridondanti. Assicurarsi che il comportamento anomalo è gestito correttamente dall'applicazione.
- Assicurarsi che i messaggi di errore apparsi agli utenti non espongano dati sensibili o pezzi di codice, ma siano comunque esplicativi in modo da far capire all'utente che l'applicazione non sta rispondendo nel modo richiesto.
- Assicurarsi che le eccezioni sono salvate nei log con tutte le informazioni necessarie per i teams di supporto, forensics, QA o incident response per capire a pieno il problema.
- Testare e verificare gli errori manualmente

Referenze

- OWASP Code Review Guide: Error Handling
- OWASP Testing Guide: Testing for Error Handling
- OWASP Improper Error Handling
- CWE 209: Information Exposure Through an Error Message
- CWE 391: Unchecked Error Condition

Tools

- Error Prone - Un tool di analisi statica da Google per identificare gli errori di programmazione più comuni di Java
- Uno dei tool automatici più famosi per identificare errori durante la runtime è Netflix's Chaos Monkey, che disabilita intenzionalmente le istanze di sistema per assicurare che il servizio nel suo complesso venga ripristinato correttamente.

Conclusioni

Questo documento dovrebbe essere interpretato come un punto di inizio piuttosto che una raccolta di tecniche completa. Vogliamo enfatizzare ancora una volta che l'intento di questo documento è di trasmettere una consapevolezza dei pericoli che comporta affrontare durante la costruzione di un software sicuro.

I passi successivi per aiutare a costruire un programma di sicurezza applicativo includono:

1) Per capire i rischi dell'ambiente web consultando le OWASP Top Ten e le OWASP Mobile Top Ten

2) Per il controllo proattivo, un programma sicuro dovrebbe includere le condizioni necessarie di sicurezza consultando lo standard OWASP (Web) ASVS oppure OWASP (Mobile) MASVS

3) Per capire i punti cardine di blocco di un programma sicuro da un punto di vista più macroscopico, consultare OWASP OpenSAMM project.

Per domande e chiarimenti consultare la mailing list all'indirizzo https://lists.owasp.org/mailman/listinfo/owasp_proactive_controls.

OWASP
Open Web Application
Security Project

OWASP PROActive CONTROLS

FOR DEVELOPERS