
UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Scienze e Tecnologie dell'Informazione

# Tecniche di analisi statica per la sicurezza di applicazioni web: problematiche ed implementazioni

RELATORE:
Prof. Marco Cremonini

TESI DI LAUREA DI:
Dario Battista Ghilardi
753708

Anno Accademico 2010/2011

# Prefazione

Testo della prefazione.

# Indice

# Capitolo 1

# Introduzione

Testo dell'introduzione.

# Capitolo 2

# Motivazione

Testo della motivation.

# Capitolo 3

# Sicurezza di applicazioni web

> If you think technology can solve your security problems, then you don't understand the problems and you don't understand the technology.
>
> — Bruce Schneier, 'Applied Cryptography' author

La diffusione globale di internet è un fenomeno in costante crescita, determinato dalle sempre più agevoli condizioni di accesso e coadiuvato dall'interesse per i servizi che la rete offre.
Le applicazioni web sono parte fondamentale di questo processo, la loro evoluzione nel corso degli anni è stata un fattore determinante per la crescita della rete. Servizi sempre più complessi hanno favorito l'interazione con gli utenti e la crescita di nuove opportunità di business. Ciò ha catturato l'interesse di individui malintenzionati, per tale motivo è nata l'esigenza di meccanismi che potessero garantire la sicurezza dei dati.

Creare sistemi sicuri comporta la soluzione di numerosi e complessi problemi: dallo sviluppo di un'architettura sicura alla creazione di robusti sistemi crittografici fino alla definizione di policy di sicurezza. Nonostante l'esistenza di queste problematiche, grossa parte degli attacchi alla security di un'applicazione sono rivolti all'errata implementazione del software oppure a vulnerabilità create dalla negligenza dello sviluppatore. Nel corso degli anni il problema della sicurezza dei dati ha assunto dimensioni rilevanti tanto che sono state proposte soluzioni per integrare la security nel processo di sviluppo software.

## 3.1 Fondamenti di sicurezza

La sicurezza nel software è basata sui principi di *confidentiality*, *integrity* ed *availability*, solitamente caratterizzata dell'acronimo *CIA*.

- Confidentiality: indica la misura che vieta la diffusione di informazioni a soggetti o sistemi non autorizzati. E' condizione necessaria (ma non sufficiente) per garantire la privacy.
- Integrity: indica la certezza che un dato non venga modificato in modo imprevisto da chi non ne possiede l'autorizzazione.
- Availability: indica la disponibilità del dato per chi ne ha l'autorizzazione quando richiesto.

Negli ultimi anni tuttavia è stata messa in discussione la definizione di sicurezza attraverso questi tre termini, con la proposta di termini aggiuntivi. Ad esempio nel 2002 Donn Parker[**?**] propose un'alternativa composta da sei termini, aggiungendo ai tre classici principi le nozioni di possession, authenticity e utility. Possession indica la proprietà dei diritti di controllo dei dati, authenticity indica la capacità di accertare la validità del dato, utility indica la capacità di un dato di essere utile per un determinato scopo.

## 3.2 Vulnerabilità nelle applicazioni web

Le vulnerabilità sono presenti nel software per vari motivi: per errate decisioni architetturali ed implementative, per mancata conoscenza da parte dello sviluppatore delle problematiche di security e per negligenza. Queste ultime due motivazioni sono ancora più veritiere nel mondo degli applicativi web: linguaggi come PHP non hanno una curva di apprendimento ripida e consentono a chiunque di realizzare applicazioni web.
Molti sviluppatori non conoscono o non si rendono conto delle problematiche di security a cui vanno incontro se vengono inseriti dati malevoli nelle loro applicazioni. E' un problema principalmente di educazione, i vari libri di programmazione difficilmente si soffermano sull'importanza di scrivere codice sicuro. Allo stesso modo il lavoro di sviluppatore non sempre richiede determinate certificazioni per essere praticato.
Un'altra motivazione che esclude la sicurezza dal processo di sviluppo software è costituita dalle condizioni economiche, le quali possono incidere sulle tempistiche e quindi restringere il tempo da dedicare al testing ed al controllo del codice. Solitamente infatti raggiungere una release stabile del progetto è la massima priorità, mentre la sicurezza non lo è.

Il controllo qualità nei progetti software è altamente focalizzato sull'adesione ai requisiti imposti in fase di progettazione, molto meno sulle implicazioni che può avere una errata implementazione delle specifiche. Solitamente anche in presenza di vulnerabilità non si violano le specifiche imposte dai requisiti.
Tecnicamente, al fine di rendere un software sicuro, tutte le parti di quel software devono essere sicure, non solo le parti sensibili dal punto di vista della sicurezza. E' proprio in questo codice che statisticamente si concentrano le vulnerabilità, quelle in cui la sicurezza non è un requisito.
Un esempio di tale situazione è la procedura di acquisto prodotti su un e-commerce: non

è necessario che solo la parte di acquisto tramite carta di credito sia sicura, un attaccante può sfruttare una vulnerabilità in qualunque punto dell'applicazione per accedere ai dati delle carte di credito degli utenti.

OWASP (Open Web Application Security Project) è un gruppo composto da volontari che produce tools, standard e documentazione open-source gratuita inerente la web security. Gli obiettivi di OWASP sono i seguenti:

- Diffondere la cultura dello sviluppo di applicativi web sicuri.
- Contribuire alla sensibilizzazione sia dei professionisti che delle aziende verso le problematiche di Web Security, attraverso la circolazione di idee, articoli, best-practice e tool.
- Promuovere l'uso di metodologie e tecnologie che consentano di migliorare il livello di sicurezza della realta‘ web.

OWASP Top Ten è un progetto, rilasciato da OWASP con cadenza triennale, che raccoglie le tipologie di vulnerabilità più rischiose nelle applicazioni web. E' una classificazione accettata a livello globale, che fornisce anche le contromisure per mitigare il rischio. Di seguito si riportano le vulnerabilità citate dalla OWASP Top Ten 2010 per introdurre le vulnerabilità nella applicazioni web.

### 3.2.1 Injection

Questa categoria di vulnerabilità raccoglie tutte le casistiche in cui dati non fidati vengono inviati ad un interprete come parte di un comando o di una query. Tali dati possono essere eseguiti dall'interprete e possono condurre all'esecuzione di comandi non voluti oppure all'accesso a dati non autorizzati. Fanno parte di questa categoria SQL injection, LDAP injection e OS injection.
E' molto comune trovare questa tipologia di vulnerabilità in codice legacy, ovvero non più supportato dal produttore, ed è una vulnerabilità che ha un severo impatto sull'applicazione poichè può portare alla corruzione del sistema, ad un *denial of service* oppure alla perdita di dati.
Un esempio di tale vulnerabilità può essere il seguente:

```
1 $query = "SELECT * FROM accounts WHERE custID ='" . $_GET["id
    "] ."'";
2 mysql_query($query);
```

L'applicazione esegue la query sul database MySql sottostante, utilizzando come parametro un valore preso direttamente dall'URL. Tale query espone però l'applicazione ad un possibile attacco di tipo SQL Injection. Infatti inserendo nell'URL una stringa come la seguente

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

la query viene interpretata in modo diverso, ritornando tutti i record di quella tabella dal database. Nel caso peggiore un attaccante può utilizzare questa vulnerabilità per eseguire query che alterano i dati nel database, riuscendo ad ottenere il completo controllo.
Per evitare di incorrere in questa tipologia di vulnerabilità è opportuno utilizzare un API per il dialogo con il database, che si occupa di filtrare i parametri in ingresso alle query. Una soluzione alternativa può essere quella di effettuare l'escape dei caratteri speciali usando specifiche sintassi per ogni interprete.

### 3.2.2 Cross site scripting

Vulnerabilità di tipo Cross site scripting (denominate spesso con l'acronimo XSS, da non confondere con CSS di cascading style sheets) si verificano quando un'applicazione riceve dati di input non fidati e li invia ad un browser senza un'appropriata validazione o escaping.
XSS consente ad un attaccante di eseguire scripts sul browser della vittima, i quali possono effettuare l'hijacking della sessione utente, possono recuperare cookie di sessione, possono redirezionare l'utente su siti web malevoli o possono effettuare defacing del sito web.
Un esempio di cross site scripting può essere il seguente:

```
1 $page += "<input name='creditcard' type="text" value='" +
    $_GET["CC"] + "'>";
```

Supponendo di avere una pagina che mostra a video il numero di carta di credito di un individuo, ottenuto attraverso i parametri in input dall'URL, l'attaccante potrà semplicemente costruire un URL con un valore del parametro CC modificato e fare in modo che l'utente visiti tale URL per effettuare l'hijacking della sessione utente, come nell'esempio seguente:

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

Esistono tre tipologie di cross site scripting:

- Stored: Il codice viene iniettato nel server in modo permanente, in un database, in un forum, in un commento, ecc. La vittima ottiene lo script malevolo ad ogni visita della pagina.
- Reflected: Il codice malevolo non viene iniettato nel server ma viene inviato alla vittima attraverso mezzi alternativi, come un email contenente un link. Quando l'utente viene convinto a cliccare su tale link il codice viene eseguito.
- DOM based: Un payload malevolo viene eseguito come risultato della modifica del DOM[1] del browser dell'utente. La risposta HTTP in questo caso non cambia ma il codice contenuto nella pagina viene eseguito in modo diverso a causa delle modifiche effettuate al DOM.

[1] Document Object Model

E' diverso da stored e reflected XSS poichè in questo caso il payload melevolo non è nella pagina di risposta del server ad una richiesta.

Vulnerabilità di tipo XSS hanno impatto significativo sull'utente, meno significativo sull'applicazione (ad eccezione del caso stored, in cui l'applicazione è direttamente coinvolta). Prevenire vulnerabilità di tipo XSS comporta la separazione tra dati non fidati ed il contenuto attivo del browser. E' quindi necessario effettuare l'escape dei contenuti in input basati su codice HTML, a seconda del contesto in cui tali dati verranno poi utilizzati.

### 3.2.3 Broken authentication and session management

Funzionalità come l'autenticazione e la gestione delle sessioni utente sono spesso implementate non correttamente, consentendo all'attaccante di ottenere passwords, chiavi, tokens di sessione o di impersonificare altri utenti.
Il classico esempio di questa vulnerabilità si verifica quando il timeout della sessione utente non è settato correttamente. Supponendo che l'utente sia loggato nell'applicazione attraverso un browser su un computer e al termine dell'uso si dimentichi di cliccare su logout. Un attaccante potrebbe collegarsi al sito attraverso lo stesso computer e ritrovarsi già loggato nell'applicazione, con il profilo del vecchio utente.
Questa tipologia di vulnerabilità ha radici architetturali oltre che implementative, per tale motivo le contromisure consistono nel seguire raccomandazioni e specifiche per il management delle sessioni e dell'autenticazione utente.

### 3.2.4 Insecure direct object references

Una direct object reference si verifica quando uno sviluppatore espone un collegamento ad un oggetto interno, come un file, una directory, ad una chiave per accedere al database, ecc. Senza le opportune protezioni gli attaccanti possono manipolare tali collegamenti per accedere a dati in modo non autorizzato.

## 3.3 Una proposta di development cycle che include la security

# Capitolo 4

# Analisi Statica

> As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.
>
> — Maurice Wilkes, Inventore di EDSAC, 1949

Tutti i progetti software condividono una caratteristica fondamentale: possiedono un codice sorgente che ne definisce il funzionamento. Tale codice sorgente è costituito da una serie di istruzioni scritte in un linguaggio di programmazione che vengono interpretate da un compilatore e successivamente eseguite. Il codice sorgente di un software risiede tipicamente su uno o più files di testo.
Il codice sorgente non è esente da errori bensì ha l'intrinseca proprietà di possedere difetti. Sin dagli albori della programmazione software gli sviluppatori hanno avuto a che fare con tali difetti, individuando un rapporto di proporzionalità diretta tra il numero di questi ultimi ed il numero di righe di codice scritte per un determinato software. L'aumentare della complessità dei programmi e la necessità di affidabilità hanno reso il controllo dei difetti fondamentale nell'industria del software, tanto che è opportuno che prima di un rilascio determinati standard di qualità siano rispettati.
Al fine di ridurre il quantitativo di difetti nel software e di aderire agli standard i programmatori hanno pensato di sviluppare altro software in grado di analizzare il codice sorgente di un programma durante la fase di sviluppo.
Tale analisi è detta *analisi statica* ed è una tecnica che consiste nell'ispezionare automaticamente il codice sorgente di un software senza però eseguirlo. Il grosso vantaggio di tale tecnica consiste nella sua applicazione alla radice del processo di sviluppo, in contrasto alle esistenti tecniche di testing che vedevano posticipata la correzione dei difetti alla fase di pre-rilascio. Anticipare l'identificazione del difetto software comporta minori costi di correzione; è proprio questo il motivo del successo dell'analisi statica.

## 4.1 Storia

La nascita delle tecniche di analisi statica viene solitamente attribuita al tool Lint, sviluppato da Stephen C. Johnson e rilasciato alla fine degli anni '70. Lint fu realizzato allo scopo di segnalare come sospetti alcuni costrutti nel sorgente in linguaggio C, come la mancanza di punti e virgola, parentesi, cast impliciti, ecc. Lint era integrato con il processo di compilazione, soluzione che sembrava essere la migliore per riportare segnalazioni relative al codice e che ne contribuì alla diffusione.
Purtroppo le limitate capacità di analisi, quali ad esempio l'obbligo di eseguire la scansione un file per volta, fecero si che Lint riportasse un elevata percentuale di rumore tra i risultati, ovvero valori corretti dal punto di vista dell'analisi ma irrilevanti per lo sviluppatore al fine di correggere difetti. Ciò si tradusse nella necessità di eseguire dei controlli manuali sui risultati di Lint, esattamente la situazione che Lint si era proposto di eliminare. Per tale motivo Lint non fu mai adottato globalmente come tool per l'individuazione di difetti.
Nei primi anni 2000 una seconda generazione di tools emerse, che si è evoluta fino ad oggi. Gli sviluppatori intuirono che era necessario comprendere attraverso il software di analisi maggiori dettagli relativi al funzionamento del programma. Produssero tools in grado di analizzare più files contemporaneamente e di identificare i percorsi di flusso dei dati, ma si scontrarono con la problematica che da sempre caratterizza l'analisi statica: il necessario compromesso da attuare tra performance ed accuratezza. L'efficacia delle tecniche di analisi statica è altamente condizionata dal fatto che devono essere gli sviluppatori ad utilizzarle, poichè prima si è in grado di identificare il difetto e minore costo ha la sua correzione.

## 4.2 Applicazioni

Sebbene le tecniche di analisi statica nacquero allo scopo di individuare difetti e di aderire a standard nella stesura del codice, molteplici successivi utilizzi vennero identificati ed implementati.
Attualmente l'analisi statica è utilizzata negli IDE[1] per evidenziare e per mostrare eventuali errori alla sintassi, per riportare segnalazioni in caso di non adesione agli standard di scrittura del codice in molteplici linguaggi.
In caso di software che possiede requisiti di sicurezza, ed ultimamente sempre più spesso visto il diffondersi di applicazioni critiche sotto questo punto di vista, l'analisi statica consente di individuare codice potenzialmente vulnerabile.

[1] Integrated Development Environment

## 4.3 Analisi statica vs. analisi dinamica vs. code review

Le tecniche di analisi statica analizzano il sorgente di un programma in modo automatico senza eseguirlo. Prima della nascita di tali tecniche gli sviluppatori effettuavano un controllo manuale sul codice chiamato *code review*.

La code review può essere eseguita da più sviluppatori (*peer review*) oppure da un solo sviluppatore. E' una procedura complessa, che ha come requisito fondamentale la piena conoscenza delle decisioni architetturali prese durante la progettazione e la scrittura del codice oltre ad un'ottima padronanza del linguaggio in analisi.
Esistono due categorie di code review: *formal code review* e *lightweight code review*. La prima categoria richiede un dettagliato processo di analisi suddiviso in molteplici fasi. Tale metodologia comporta l'analisi di copie stampate del materiale ed è svolta da più partecipanti che contemporaneamente analizzano il codice.
La seconda categoria richiede meno formalismi rispetto alla precedente e viene svolta solitamente durante il normale processo di sviluppo. All'interno di essa si possono individuare le seguenti pratiche:

- Over-the-shoulder: uno sviluppatore osserva il codice che l'altro sta scrivendo per segnalare eventuali problemi
- Email pass around: un SCM[2] invia tramite email il nuovo codice inserito nella codebase ad un soggetto che si occupa di effettuare la review.
- Pair programming: Due sviluppatori scrivono codice contemporaneamente sulla stessa workstation.
- Tool-assisted code review: Sviluppatori e reviewers usano tools in grado di effettuare code review collaborativa.

La problematica di questa tipologia di analisi consiste nel tempo che richiede; i dati raccolti dai maggiori operatori del settore stimano che in media si possa effettuare code review su 150 linee di codice per ogni ora, fino a rimuovere l'85% dei difetti presenti nel software.

L'analisi dinamica è una tecnica che consiste nell'osservare il comportamento del software durante la sua esecuzione. Al fine di rendere tale tecnica effettiva è necessario che il programma venga eseguito con diversi input. L'analisi dinamica è una tecnica precisa, che non comporta approssimazione poichè osserva l'esatto comportamento runtime dell'applicazione. Lo svantaggio dell'analisi dinamica è la sua specificità: i risultati proposti riguardano solo ed esclusivamente quell'esecuzione, non c'è garanzia che la test suite utilizzata esegua effettivamente tutti i possibili data flow all'interno del software e che quindi esegua ogni porzione di codice. L'approccio definito dall'analisi dinamica è particolarmente adatto per il testing ed il debugging.

[2]Source Code Management System

Analisi statica ed analisi dinamica sono due approcci complementari che possono essere applicati allo stesso problema, i risultati hanno però diverse proprietà e l'esecuzione di ognuno ha diversi costi. Per tale ragione esistono soluzioni in grado di combinare analisi statica ed analisi dinamica al fine di ridurre i difetti tipici delle due tecniche e fornire risultati più attendibili.

# Capitolo 5

# Applicazione dell'analisi statica alla sicurezza di applicazioni web

Cosa posso risolvere usando analisi statica delle vulnerabilità precedenti?

# Capitolo 6

# Analisi statica di codice PHP

# Capitolo 7

# Comparazione dei principali tool esistenti

## 7.1 Pixy

## 7.2 Saner

## 7.3 RIPS

# Capitolo 8

# Vulture

## 8.1 Problematiche

## 8.2 Sviluppi futuri

# Capitolo 9

# Discussione

# Capitolo 10

# Conclusioni

# Capitolo 11

# Conclusioni

In questo documento sono state analizzate diverse tecniche per violare la privacy degli utenti sfruttando due tipologie di applicativi molto utilizzati: i social networks ed i file hosting services.
Per ognuna sono state evidenziate le modalità tramite le quali un attaccante può prendere possesso di dati sensibili ed è stato dimostrato come la fiducia degli utenti non venga poi ripagata dall'effettivo grado di protezione dei loro dati sensibili. Infatti si è dimostrato che i social networks ed i file hosting services, pur conoscendo tali problematiche, non sempre si sono dimostrati affidabili nel fornire l'adeguata protezione, per le ragioni più varie: commerciali, relative alla user interface, di convenienza.
Si è dimostrato poi come talvolta la fiducia e la non consapevolezza delle potenziali problematiche porti l'utente a compiere azioni in grado di mettere a repentaglio i propri dati personali.

# Bibliografia


[1] Marco Balduzzi, Christian Platzer, Thorsten Holz, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. Abusing Social Networks for Automated User Profiling. In *Preceeding of the 13th international conference on Recent advances in intrusion detection*, pages 422–441, Berlin, Heidelberg, 2010. Springer-Verlag.

[2] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. All Your Contacts Are Belong to Us : Automated Identity Theft Attacks on Social Networks. In *Preceedings of the 19th international conference on World Wide Web (WWW)*, 2009.

[3] B. Bowen, S. Hershkop, A. Keromytis, and S. Stolfo. Baiting inside attackers using decoy documents. *Security and Privacy in Communication Networks*, pages 51–70, 2009.

[4] Monica Chew and Ben Laurie. ( Under ) mining Privacy in Social Networks. In *Preecedings of Web 2.0 Security and Privacy Workshop (W2SP)*, 2008.

[5] Catherine Dwyer and Starr Roxanne Hiltz. Trust and privacy concern within social networking sites : A comparison of Facebook and MySpace. In *Preceedings of the Thirteenth Americas Conference on Information Systems (AMCIS)*, 2007.

[6] Brad Fitzpatrick and David Recordon. Thoughts on the Social Graph. `http://bradfitz.com/social-graph-problem`, 2007.

[7] Ralph Gross, Alessandro Acquisti, and H John Heinz Iii. Information Revelation and Privacy in Online Social Networks ( The Facebook case ). *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2005.

[8] Miguel Helft. Google thinks it knows your friends. `http://bits.blogs.nytimes.com/2007/12/26/google-thinks-it-knows-your-friends/`.

[9] Facebook Inc. Facebook privacy guidelines. `http://www.facebook.com/policy.php`.

[10] Jesper M. Johansson and Roger Grimes. Security-through-obscurity. `http://technet.microsoft.com/en-us/magazine/2008.06.obscurity.aspx`.

[11] Aleksandra Korolova. Privacy Violations Using Microtargeted Ads : A Case Study. 2010.

[12] W. Mackay. Triggers and barriers to customizing software. In *Preceedings of CHI'91*, pages 153–160. ACM Press, 1991.

[13] Kevin D. Mitnick. *The art of deception.* John Wiley & Sons, Inc., 2002.

[14] Nick Nikiforakis, Marco Balduzzi, Steven Van Acker, Wouter Joosen, Davide Balzarotti, and Sophia Antipolis. Exposing the Lack of Privacy in File Hosting Services. *Sophia*, 2010.

[15] J. Yuill, M. Zappe, D. Denning, and F. Feer. Honey- files: deceptive files for intrusion detection. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop*, pages 116–122, 2004.

[16] Mark Zuckerberg. Making control simple. `http://blog.facebook.com/blog.php?post=391922327130`.