



Politecnico di Milano

Facoltà di Ingegneria dell'Informazione
Corso di Laurea Specialistica in Ingegneria Informatica

Analisi statica per l'identificazione di vulnerabilità in ambiente J2EE

Relatore:

Prof. Giuseppe Serazzi

Correlatore:

Ing. Stefano Zanero

Laureando:

Luca Caretoni – Matr. 667031



Introduzione: sicurezza nelle applicazioni web

- Le applicazioni web sono diventate lo strumento principale per fornire informazioni e servizi a clienti e fornitori (Dal 2000 al 2006: +200.9%)
- La sicurezza: un requisito importante, al pari di quelli funzionali, che purtroppo viene spesso trascurato...
- Applicazioni web sempre più aperte e flessibili; garantire il paradigma C.I.A. non è un compito semplice:
 - Un web server, per sua stessa natura, fornisce un servizio non autenticato e su un canale tendenzialmente insicuro
 - Crescente complessità del software online
 - Tecnologie Internet sviluppate per un concetto di Rete diverso
 - Cambio di paradigma: da attacchi di rete ad attacchi verso interfacce applicative pubbliche



Problematiche di Input Validation

- Oggi, il problema principale di molte applicazioni web
- Secondo OWASP-Italy, il 75% degli auditing hanno dato esito positivo rispetto a questa categoria di vulnerabilità
- Parliamo di *Input Validation flaw* quando un potenziale aggressore può modificare parte delle richieste, eludendo eventuali checkpoint, al fine di deviare il flusso di esecuzione delle istruzioni

Qualche esempio: SQL Injection, XSS, Path Traversal,...

```
$query = sprintf("SELECT * FROM %s WHERE owner='%s' AND nickname='%s'",  
    $this->table, $this->owner, $alias);  
$res = $this->dbh->query($query);
```

E se `$alias` contenesse `' UNION ALL SELECT * FROM address WHERE '1'='1 '` ?



Strumenti per la verifica di sicurezza

- Strumenti diversi, con finalità diverse:
 - Source Code Analyzer
 - Web Application Scanner
 - Database Scanner
 - Binary Analysis Tool
 - Runtime Analysis Tool
 - Configuration Scanner
 - HTTP Proxy
- L'esigenza di identificare vulnerabilità software, sin dai primi stadi dello sviluppo applicativo, ha generato un interesse crescente verso gli strumenti di revisione automatica del codice
- Ricerca di pattern, dataflow analysis

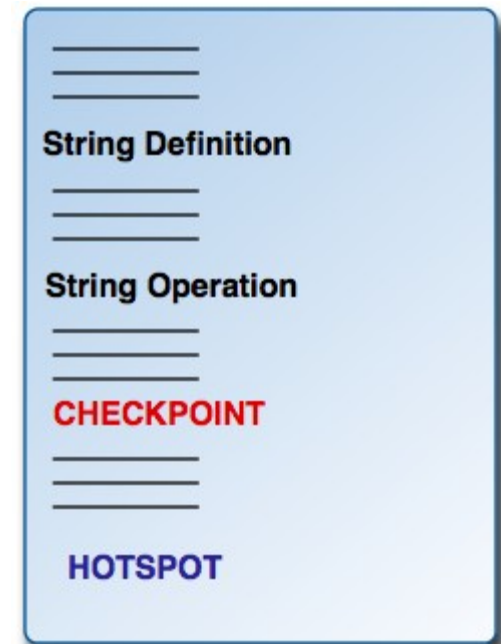


Checkpoint: approccio blacklist,whitelist

- L'elaborazione delle informazioni nelle applicazioni web avviene prevalentemente attraverso lo scambio di **dati testuali**, generalizzabile con il concetto di **stringa** ed **operazioni su stringa**.

Checkpoint:

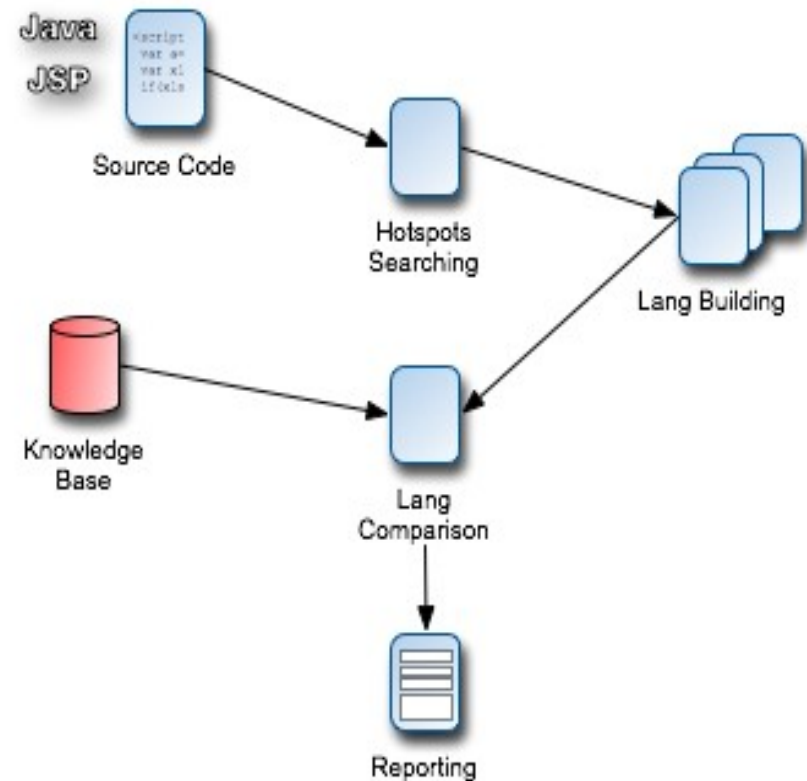
- **Blacklist**: si definiscono i valori non conformi alle specifiche; si adottano politiche di *escaping*, sostituzione di caratteri, etc.
- **Whitelist**: si definiscono i valori ammissibili e si verifica l'appartenenza dell'input a questo insieme.





Panoramica sul metodo proposto

- Ricerca dei metodi potenzialmente pericolosi (hotspot)
- Ricostruzione statica del linguaggio associato ai parametri di tipo stringa
- Comparazione di tale linguaggio con quello considerato “safe” dalla base di conoscenza.
- Segnalazione di vulnerabilità





Ricerca degli hotspot

- **Hotspot**: qualsiasi operazione di input/output che, se mal parametrizzata, può generare una vulnerabilità.

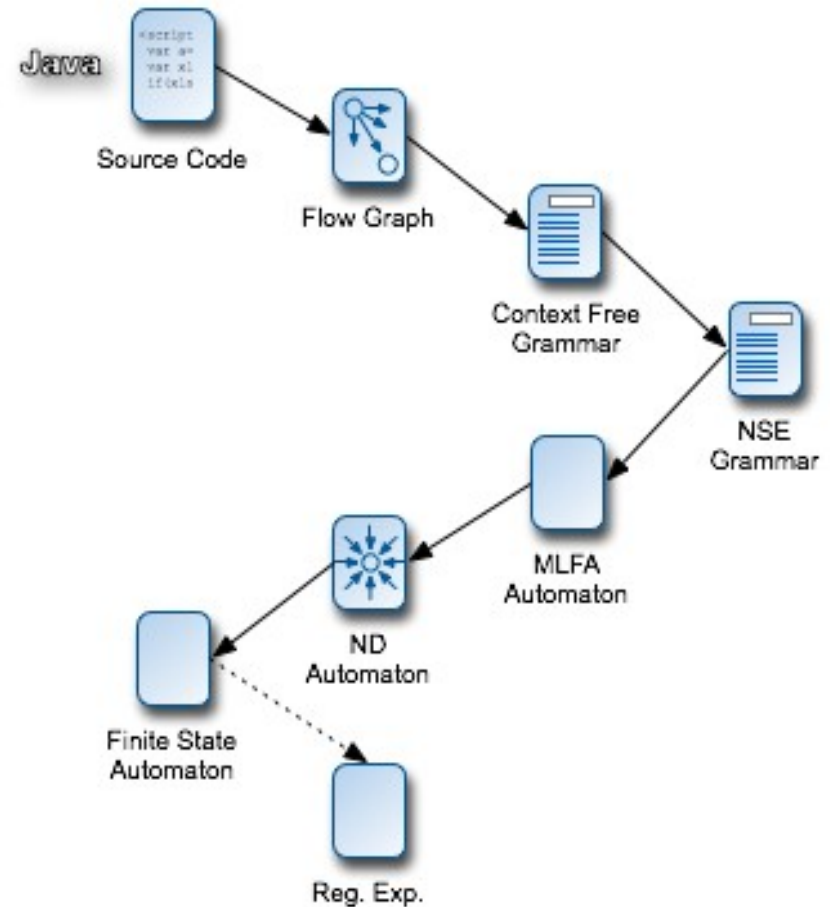
Qualche esempio in ambiente J2EE:

- Lettura/Scrittura sul filesystem: **Path Traversal**
`java.io.FileReader(java.lang.String)`
 - Connettività e query verso dbms: **SQL Injection**
`java.sql.Statement.executeQuery(java.lang.String)`
 - Caricamento dinamico delle classi, esecuzione di comandi di sistema: **Command Injection**
`java.lang.Runtime.exec(java.lang.String, ...)`
- Ad ogni *hotspot* associamo una *signature*
 - Valutazione sistematica su ogni parametro *String* e *StringBuffer*



Costruzione dei linguaggi (1/2)

- Analisi statica e valutazione delle espressioni stringa per la determinazione del linguaggio associato ad ogni parametro
- Approssimazione in eccesso sui valori ottenibili a run-time
- Rappresentazione finale tramite automi a stati finiti ed espressioni regolari





Costruzione dei linguaggi (2/2)

- Costruzione del *flow graph* associato al programma in analisi
- Conversione della descrizione astratta mediante *flow graph* in una grammatica *Context-free*.
Ogni non terminale della grammatica identifica i possibili valori delle stringhe nei corrispondenti nodi del grafo
- Approssimazione della grammatica *Context-free* in una forma *Non-Self-Embedding (NSE)* utilizzando una variante dell'algoritmo Mohri-Nederhof (*Precise Analysis of String Expressions*, BRICS)
- Trasformazione della grammatica regolare in forma di MLFA (multi level finite automaton), un grafo diretto aciclico di automi a stati finiti non deterministici.
- Per ogni singolo hotspot, costruzione di un automa deterministico minimo.



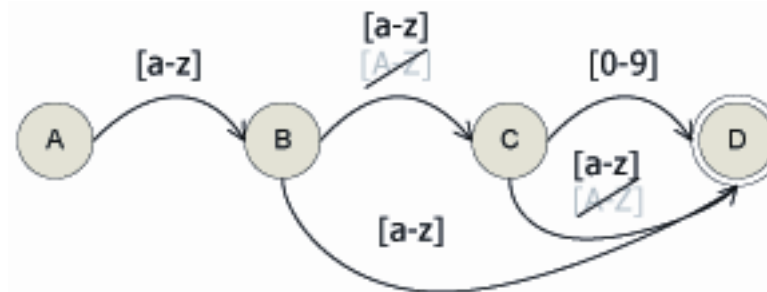
Da operazioni su stringhe ad operazioni su automi

- Ogni singola operazione su stringa f non è trattata a livello di flow graph o grammatica ma come operazione $T(f)$ nello spazio degli automi:

$$A(L) \xrightarrow{T(f)} A(L')$$

- Un esempio: il metodo *toLowerCase()*

$$L_L = \{X_1X_2\dots X_n \mid X_1, X_2, \dots, X_n \in L_i \wedge X_1, X_2, \dots, X_n \notin L_U\}$$





Confronto tra linguaggi

- Nel caso di validazione assente e/o errata, attraverso vettori di input risulta possibile modificare i parametri delle invocazioni
- Oltre a sfruttare la vulnerabilità risulta possibile creare invocazioni sintatticamente errate per il linguaggio specifico della chiamata
- Usiamo questi “controesempi” per determinare l’effettiva vulnerabilità

```
import java.servlet.*;
...
public class Servlet extends HttpServlet{

public void doGet(...){
    String str1 = request.getParameter("par1");
    String qry = "SELECT pass FROM table WHERE
                myRow='";
    qry = qry.concat(str1);
    qry = qry.concat("'");
    ...
    Connection cn = ... ;
    Statement cmd = cn.createStatement();
    ResultSet res = cmd.executeQuery(qry);
    ...
}}
```

$$(L_b \cap \neg L_d) = \emptyset$$



JSEC – Java.String Eclipse Checker

- Plugin per l'ambiente di sviluppo *Eclipse*
- Approccio “*Sviluppa, verifica, correggi*”
- Analisi statica su applicazioni J2EE, svolta in maniera completamente automatica
- Personalizzazione delle signature e della modalità di scansione
- Architettura a plugin per il supporto a nuove vulnerabilità

```
<detector id="jsec.sqlDetector">
  <category>SQL Injection</category>
  <regexp>sql.reg</regexp>
  <name>SQL Injection Detector</name>
  <description>
    Plugin for the detection of
    J2EE webapp SQL Injection flaws
  </description>
</detector>
```



JSEC – Java.String Eclipse Checker

The screenshot shows the Eclipse IDE interface. The main editor displays the code for `SimpleServlet.java`. A security vulnerability is highlighted in the code, specifically the line `File userFile= new File(user);`. The Problems view at the bottom shows the results of the JSEC checker, listing two issues: SQL injection and Directory traversal.

```
if(res.getString("password").equals(str2)){
    //Login OK!

    //Show the txt to the user
    String thisLine;
    try {
        File userFile= new File(user);
        FileInputStream fin = new FileInputStream(userFile);
        BufferedReader myInput = new BufferedReader
            (new InputStreamReader(fin));
        while ((thisLine = myInput.readLine()) != null) {
            PrintWriter out = response.getWriter();
            out.println(thisLine);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    res.close();
}
```

Gravity	Category	Project	File	Method Call	Par	In Method	Line
High	SQL injection	simpleSer	SimpleServlet.jav	ResultSet res = cmd.executeQuery(qry);	0	doGet	59
Medium	Directory traversal	simpleSer	SimpleServlet.jav	File userFile= new File(user);	0	doGet	67



Valutazione sperimentale

In mancanza di *Security Benchmark* per la valutazione oggettiva degli strumenti di revisione del codice:

- Revisione manuale per eliminare i *falsi positivi*
 - Comparazione con altri strumenti (LAPSE) per eliminare i *falsi negativi*
- Valutazione sul software dimostrativo OWASP *WebGoat* (35 classi, 8474 righe di codice)

	Analysis time	Vulnerabilities
JSEC	201.366 sec.	13 (9) SQL Injection 5 (3) Path Traversal
LAPSE	21.54 sec.	9 (8) SQL Injection 0 Path Traversal



Conclusioni. Sviluppi futuri.

- L'identificazione di vulnerabilità attraverso la ricostruzione/comparazione statica dei parametri di tipo stringa risulta essere una tecnica di analisi efficace
- Lo strumento software sviluppato ha mostrato l'applicabilità della metodologia anche in contesti applicativi reali
- I risultati sperimentali evidenziano le caratteristiche distintive dell'analisi che, in alcuni contesti applicativi, potrebbero farla preferire ad altre soluzioni
- L'evoluzione dello strumento software in termini di usabilità, lo sviluppo di nuovi plugin e l'integrazione con altri paradigmi di analisi potrebbero migliorare ulteriormente i risultati sperimentali, diminuendo il numero di falsi positivi e negativi.