security analysis with WALA

Omer Tripp
IBM TJ Watson
collaborators

• Marco Pistoia  • Patrick Cousot  • Radhia Cousot
• Julian Dolby  • Manu Sridharan  • Pietro Ferrrara
• Steve Fink  • Sal Guarnieri  • Takaaki Tateishi
• Omri Weisman  • Steve Teilhet  • Ryan Berg
• Shay Artzi  • Lotem Guy  • Roee Hay
motivation
threat evolution

1960: Program installation only by experts
1970: Program sharing
1980: Program downloading
1990: Mobile code
2000: Web applications
2010: Mobile applications
threats: XSS

Web Application

<SCRIPT>...</SCRIPT>

Attacker

Attacker’s evil script

Attacker’s evil script executed using victim’s credentials

Victim
threats: SQLi

String query = "SELECT * FROM users WHERE name='" + userName + "' AND pwd='" + pwd + "'";

SELECT * FROM users WHERE name='jsmith' AND pwd='Demo1234'

SELECT * FROM users WHERE name='foo';drop table custid;--' AND pwd='"
threats: info leakage
threats: JS

According to an IBM study performed in 2010, 15% of Fortune 500 websites have exploitable security issues in JavaScript.

DOM-based XSS
```
document.write(document.URL.substring(pos,document.URL.length));
```

Open Redirect
```
var pos = document.location.href.indexOf("name=");
var val = document.location.href.substring(pos);
document.location.href = "http://" + val;
```
threats: mobile

<table>
<thead>
<tr>
<th></th>
<th>Android</th>
<th>iOS</th>
<th>Java ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRLF Injection</td>
<td>37%</td>
<td>Information Leakage</td>
<td>62% Cryptographic Issues</td>
</tr>
<tr>
<td>Cryptographic Issues</td>
<td>33%</td>
<td>Error Handling</td>
<td>20% Information Leakage</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>10%</td>
<td>Cryptographic Issues</td>
<td>7% Directory Traversal</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>9%</td>
<td>Directory Traversal</td>
<td>6% Insufficient Input Validation</td>
</tr>
<tr>
<td>Time and State</td>
<td>4%</td>
<td>Buffer Management Errors</td>
<td>3% Credentials Management</td>
</tr>
</tbody>
</table>

owasp top ten

1. Cross-site scripting (XSS)
2. Injection flaws
3. Malicious file executions
4. Insecure direct object reference
5. Cross site request forgery (CSRF)
6. Information leakage and improper error handling
7. Broken authentication and improper session management
8. Unsecure cryptographic storage
9. Unsecure communications
10. Failure to restrict URL accesses
outline of our work
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
the high-level progression
server-side security verification
TAJ (PLDI’09): Java taint analysis
TAJ: design choices

- pointer analysis: variant of Andersen’s analysis
- custom context-sensitivity policy:
  - unlimited-depth object sensitivity for Java collections (up to recursion)
  - one level of call-string context for factory methods
  - one level of call-string context for taint APIs
  - one-level receiver-object context sensitivity as default
- field sensitivity
- flow insensitive, though intraprocedurally flow sensitive thanks to SSA
TAJ: WALA support

- pointer analysis / context sensitivity
- field sensitivity
- synthetic IR for taint APIs
- SSA representation
- IFDS enhancements [contributed to WALA]
- ICFGSupergraph [contributed to WALA]
- .NET support [contributed to WALA]
Andromeda (FASE’13): demand-driven taint analysis

\[
\begin{align*}
\text{\textbf{[2]} } y.f.h & \quad x = y.f \\
& \quad [2] y.f.h \\
& \quad x.h = z.g \\
\text{\textbf{[1]} } x.h & \quad T = \{z.g\} \\
\text{\textbf{[3]} } w & \quad w = y.f.h \\
\text{\textbf{[3]} } w & \quad o.sinkfld = w \\
& \quad T = \{z.g, x.h, y.f.h, w\} \\
\end{align*}
\]
Andromeda: WALA support

- PA-free CG construction algs for Java (CHA, RTA, …)
- efficient infrastructure for fixpoint iteration
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
framework modeling
F4F (OOPSLA’13): modeling Java web frameworks
effect of modeling
F4F: WALA support

- synthetic IR for call-graph methods
- customization of call-graph target resolution (CustomContextSelector)
- various knobs to tweak in pointer analysis
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
sanitizer verification
PISA (ISSTA’11, TOSEM’13): encoding into M2L

\[
\begin{align*}
\text{“a}_1 \cdots \text{a}_n\-Bold{(P)} & \equiv \exists t_1, \cdots, t_n \cdot \\
& \quad \text{\texttt{‘a}_1\texttt{’}(t_1) \land \cdots \land \text{\texttt{‘a}_n\texttt{’}(t_n)} \\
& \quad \land t_1 < t_2 \land t_2 < t_3 \land \cdots \land t_{n-1} < t_n \\
& \quad \land P = \{t_1, \cdots, t_n\} \\
\text{concat}(R, P, Q) & \equiv R = P \cup Q \\
\text{strr}(R, P, p, q) & \equiv p \leq q \land R \subseteq P \land (\forall r . r \in P \\
& \quad \Rightarrow (r \in R \iff p \leq r \land r < q)) \\
\text{substr}(R, P, p, q) & \equiv \exists p', q' . p \leq p' \land p' \leq q' \land q' \leq q \\
& \quad \land \text{strr}(R, P, p', q') \\
\text{substr}(R, P) & \equiv \text{substr}(R, P, \text{min}(P), \text{max}(P) + 1) \\
\text{consecutive}(p, q, R) & \equiv p < q \land p \in R \land q \in R \\
& \land (\forall r . p < r \land r < q \Rightarrow r \not\in R)
\end{align*}
\]
PISA: sanitizers from the wild

```java
static final String PUNCTUATION_CHARS_ALLOWED = "()&+,-_.";
static String cleanLink(String link) {
    return cleanLink(link, PUNCTUATION_CHARS_ALLOWED);
}
static String cleanLink(String link, String allowedChars) {
    if (link == null) return null;
    link = link.trim();
    StringBuffer clean = new StringBuffer(link.length());
    boolean isWord = true; boolean wasSpace = false;
    for (int i = 0; i < link.length(); i++) {
        char ch = link.charAt(i);
        if (Character.isWhitespace(ch)) {
            if (wasSpace) continue;
            wasSpace = true;
        } else { wasSpace = false; }
        if (Character.isLetterOrDigit(ch) ||
            allowedChars.indexOf(ch) != -1) {
            if (isWord) ch = Character.toUpperCase(ch);
            clean.append(ch); isWord = false;
        } else { isWord = true; }
    }
    return clean.toString();
}
```

```java
private static String getFileName(String s) {
    String fileName = new File(s).getName();
    if (fileName.indexOf(\"/\") != -1) {
        fileName =
            fileName.substring(fileName.lastIndexOf(\"/\"),
            fileName.length());
    } else if (fileName.indexOf(\".\") != -1) {
        fileName =
            fileName.substring(0, fileName.indexOf(\".\") );
    }
    return fileName;
}
```

```java
public String getCcnParameter(String name) {
    return getRegexParameter(name, "\d{16}");
}

public String getPhoneParameter(String str) {
    return getRegexParameter(str, "\d\s+");
}
private String getRegexParameter(String name, String regexp) {
    String param = getRegexParameter(name, regexp);
    if (Pattern.matches(str, regexp)) return str; 
else return "";
}
```
PISA: WALA support

- uniform IR: JS, Java, …
- pointer analysis: aliasing
- SSA

IBM AppScan Source Edition
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
client-side security verification
Actarus (ISSTA’11): JS taint analysis
Actarus: handling aliases

1: function id(x) {
2:     return x;
3: }
4: function set(y, z) {
5:     var x = z.g;
6:     x.f = y;
7: }
8: var p = document.URL;
9: var q = { g: { } }
10: var r = id(p);
11: var s = id(q);
12: set(r, s);
Actarus: WALA support

- JS IR, including
  - property accesses
  - arguments array
  - lexical scoping
  - function pointers
- novel CG construction algs for JS [contributed to WALA]
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
the high-level pieces

dynamically loaded client-side code
statically loaded client-side code
server-side code
security defenses: sanitizers and validators
libraries and frameworks
handling dynamic aspects of JS
JSA (ISSTA’14):

- black-box scanner
- HTML/JavaScript, concrete URLs, …
- DOM modeling
- taint analysis
- string analysis

- reduce scope
- find issues
- eliminate false positives
JSA (ISSTA’14):

- specialized string analysis using dynamic pieces of information (e.g., concrete URL)

```javascript
var str = document.URL;
var url_check = str.indexOf('login.html');
if (url_check > -1) {
    result = str.substring(0, url_check);
    result = result + 'login.jsp' +
             str.substring((url_check+search_term.length),
                          str.length);
    document.URL = result;
}
```

- part controlled by attacker is unknown, but known prefix modeled precisely

**URL as Source**

http://www.mysite.com/folder/page?a=1&b=2#anchor
JSA: WALA support

- call-graph construction
- IFDS / RHS (proved general enough!)
  - ability to leverage domain knowledge
  - CallNoneToReturnFlow as a means to plug in models
the high-level pieces

- dynamically loaded client-side code
- statically loaded client-side code
- server-side code
- security defenses: sanitizers and validators
- libraries and frameworks
the high-level pieces

- dynamically loaded client-side code
- statically loaded client-side code
- server-side code
- security defenses: sanitizers and validators
- libraries and frameworks
nowadays
PL / ML
PL / ML
Android Application Malware Analysis

Please select an Android mobile application to analyze

Upload an Android application (.apk file) to have it scanned for suspected malware.

Upload Mobile Application

Application file:
Choose File | No file chosen

Result Type:

- HTML web page
- JSON

Upload and Analyze Android Application

Note: the uploaded application and result are not retained but rather are discarded as soon as the analysis is complete. If you wish to retain the analysis for future reference you should make a local copy of the result for yourself.
PL / ML

IBM Research: MobileFirst Security Team

Android Application Malware Analysis result
Application "malicious.jar" analyzed result is:

**Determination: malware**
Confidence level is 77%

<table>
<thead>
<tr>
<th>Evidence</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ibm.wala.FakeRootClass.fakeRootMethod()V</td>
<td>commit-exec:true</td>
</tr>
<tr>
<td>→ onCreate()V</td>
<td>getDeviceId-getLineNumber:true</td>
</tr>
<tr>
<td>com.google.update.UpdateService.getPermission2()V</td>
<td>commit-getDeviceId:true</td>
</tr>
<tr>
<td>→ commit():Z</td>
<td>getOutputStream-getOutputStream:true</td>
</tr>
<tr>
<td>com.google.update.RU.U6(Ljava/lang/String;Ljava/lang/String;)V</td>
<td>commit-getLineNumber:true</td>
</tr>
<tr>
<td>→ exec(Ljava/lang/String;Ljava/lang/Process;)</td>
<td>commit-getOutputStream:true</td>
</tr>
<tr>
<td>com.google.update.RU.UI.U2(Landroid/content/Context;Ljava/lang/String;)</td>
<td>exec-getDeviceId:true</td>
</tr>
<tr>
<td>→ getDeviceId():Ljava/lang/String;</td>
<td>getLineNumber-getOutputStream:true</td>
</tr>
<tr>
<td>com.google.update.RU.UI.UI3(Landroid/content/Context;Ljava/lang/String;)</td>
<td>exec-getLineNumber:true</td>
</tr>
<tr>
<td>→ getLineNumber():Ljava/lang/String;</td>
<td>exec-getOutputStream:true</td>
</tr>
<tr>
<td>com.google.update.RU.UI.UI2(Ljava/lang/String;)Z</td>
<td>getOutputStream:Many</td>
</tr>
<tr>
<td>→ getOutputStream():Ljava/io/OutputStream;</td>
<td></td>
</tr>
<tr>
<td>com.ibm.wala.FakeRootClass.fakeRootMethod()V</td>
<td>d-e:true</td>
</tr>
<tr>
<td>→ onCreate(Landroid/os/Bundle;)V</td>
<td>getSimSerialNumber-getLineNumber:true</td>
</tr>
<tr>
<td>com.adw.o.adsdk.i.b(Ljava/lang/String;)V</td>
<td>e-getDeclaredConstructor:true</td>
</tr>
<tr>
<td>→ d(Ljava/lang/String;Ljava/lang/String;)I</td>
<td>getSimSerialNumber-getDeclaredConstructor:true</td>
</tr>
<tr>
<td>com.adw.o.adsdk.i.a(Ljava/lang/String;)V</td>
<td>e-getSimSerialNumber:true</td>
</tr>
<tr>
<td>→ e(Ljava/lang/String;Ljava/lang/String;)I</td>
<td>e-getLineNumber:true</td>
</tr>
<tr>
<td>com.adw.o.adsdk.i.b(Landroid/content/Context;)V</td>
<td>getLineNumber-getDeclaredConstructor:true</td>
</tr>
<tr>
<td>→ getSimSerialNumber():Ljava/lang/String;</td>
<td>d-getSimSerialNumber:Many</td>
</tr>
<tr>
<td>com.adw.o.adsdk.i.b(Landroid/content/Context;)V</td>
<td>d-getSimSerialNumber:true</td>
</tr>
<tr>
<td>→ getSimSerialNumber():Ljava/lang/String;</td>
<td>d-getSimSerialNumber:Ljava/lang/String;</td>
</tr>
</tbody>
</table>

http://malwaredetection.mybluemix.net/analyzer/
thoughts into the future
• query-based analyses

• more WALA support around IFDS / IDE (already underway!!)

• integration with machine-learning engines like WEKA

• integration with dynamic analyses