Security in Hop Web Applications

T. Rezk
INDES team
INRIA Sophia Antipolis –Méditerranée
INRIA-MSR Joint Lab

JOINT WORK (in progress):
G. Boudol, Z. Luo, M. Serrano

29/01/2010
PARSEC meeting
Reports on web vulnerabilities

Cenzic company report (2009)

- Others (web browser, request forgery, web server)
- Information Leak/Disclosure
- Authentication and Authorization
- Code Injection

Tuesday, 2 February 2010
Code Injection

An untrusted source introduces code to change the course of “normal” execution
A code injection example

```html
<script>hop_innerHTML_set
  ((document.getElementById("allentries")),"You
  can only see this entry!!")</script>
```
A code injection example

client browser 1

client browser 1

client browser 1

guestbook server

entries database

guestbook service request
A code injection example

client browser 1

client browser 1

client browser 1

guestbook server

guestbook service request

guestbook server

get all the entries

entries database

Tuesday, 2 February 2010
A code injection example

client browser 1

post dynamic web page

guestbook server

get all the entries

entries database

client browser 1

client browser 1

client browser 1
A code injection example

- Client browser 1
- Client browser 1
- Client browser 1

- Guestbook server
- Entries database

- Add an entry to the database
- Store entry in the database
Cross Site Scripting (XSS)

Untrusted code in injected in a dynamic generated web page that is executed in a trusted environment
XSS example

- client browser 1
- guestbook server
- add an entry to the database
- store entry in the database
- entries database

This is a script to execute service delete-all-entries in database
**XSS example**

- The script to execute service delete-all-entries is stored in the database (but it cannot be executed without admin rights)

- The next user to log in: the admin user
XSS example

client browser 1

client browser 1

client browser 1

guestbook server

request + login as admin

post dynamic web page

guestbook server

get all the entries

entries database
XSS example

client browser 1

request + login as admin
post dynamic web page

call to service delete-all-entries as admin user!
(XSS attack)

client browser 1

get all the entries

guestbook server

entries database

Tuesday, 2 February 2010
Code injection: more formally

The dynamically generated web page should behave as the specification.

Security against code injection definition:

- a formal specification of the expected behaviour of dynamically generated web pages
- an application is secure if it always generates web pages that are observational equivalent to their specifications
A web application

http protocol

guestbook server

get all the entries

entries database

client browser 1

post dynamic web page

service request

client browser 1

client browser 1

Tuesday, 2 February 2010
A web application

client browser 1

http protocol

html pages, javascript

post dynamic web page

get all the entries

entries database

guestbook server

client browser 1

client browser 1

Tuesday, 2 February 2010
A web application

client browser 1

client browser 1

client continuations

client browser 1

http protocol

html pages, javascript

post dynamic web page

Scheme, Java

guestbook server

SQL

entries database

http protocol
How to formally define code injection?

Unless there is a prior, generally-accepted mathematical definition of a language at hand, who is to say whether a proposed implementation is correct? (Dana Scott 1969)

Formal specification of a web application might be complex since it involves several heterogeneous languages and technologies.

A solution

Multi-tiers compilers
Multi-tiers compilers

Input: a web application written in a single homogenous language

A multi-tiers compiler

- Scheme code and protocols over HTML (server code)
- Javascript (client code)
- SQL (server)
Multi-tiers compilers

Examples:

Links: Philip Wadler et al
Hop: Manuel Serrano and Florian Loitsch
Google Web Toolkit: Google
Swift: Andrew Myers et al
Hop (simplification)

Syntax

\[
\begin{align*}
e^s &::= e \\
&\quad | (\text{tag } e^s \ldots) \\
&\quad | \sim e^{cs} \\
\text{tag} &::= \text{html} | \text{head} | \text{div} | \text{canvas} \ldots \\
v^s &::= v | \sim e^c | \@p \\
\end{align*}
\]

Configuration

\[(id, m, h, e)\]

- id for client or server
- environment
- heap
- expression
Hop (simplification)

small step relation

\[ p \text{ is a fresh pointer} \]

\[
( S, m, [], (C[(html v)]) ) \to (S, m, [p \rightarrow (html v), nil, C[@p]])
\]

\[
(m,e) \to (m',e')
\]

\[
( S, m, [], (C[(html e)]) ) \to (S, m, [], C[(html e')])
\]
Hop execution

\[(S, m_0, h_0, e_0) \rightarrow (S, m_1, h_1, e_1) \ldots (S, m_k, h_k, e_k) \rightarrow (C, m_{k+1}, h_{k+1}, e_{k+1}) \rightarrow \ldots\]
Hop execution

$(S, m_0, h_0, e_0) \rightarrow (S, m_1, h_1, e_1) \ldots (S, m_k, h_k, e_k) \rightarrow (C, m_{k+1}, h_{k+1}, e_{k+1}) \rightarrow \ldots$

HTML + Javascript configuration

compilation of $e_{k+1}$ and its environment to javascript

$(m_{k+1}, TREE, eJS) \rightarrow \ldots$
Hop execution

The compiler is NOT correct if a string in the hop expression

\[ \langle \text{script} \rangle \text{ do sth evil } \langle /\text{script} \rangle \]

is interpreted as a javascript node in the target expression

\[ ( C, mk+1, hk+1, ek+1 ) \rightarrow \ldots \]

compilation of \( ek+1 \) and its environment to javascript

\[ ( mk+1, \text{TREE}, eJS ) \rightarrow \ldots \]
## How to detect strings that will change the behaviour?

### standard practice

<table>
<thead>
<tr>
<th>Standard practice: filter or validate tainted input</th>
</tr>
</thead>
<tbody>
<tr>
<td>- advantage: efficiency</td>
</tr>
<tr>
<td>- disadvantage: each language requires a different filter, and this kind of problems: <code>&lt;script&gt;</code> tainted string here <code>&lt;/script&gt;</code> is not detected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantage: the same technique applies to different languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>- disadvantage: it does not help against the insecure javascript practice (document.write, eval)</td>
</tr>
</tbody>
</table>
A tree comparison technique

\(( C, m_{k+1}, h_{k+1}, e_{k+1}) \to \ldots\)

Does the abstract syntax tree of \(e_{k+1}\) correspond to the AST of eJS?

\(( m_{k+1}, \text{TREE}, e_{JS}) \to \ldots\)

compilation of \(e_{k+1}\) and its environment to javascript+html

\(\text{Tuesday, 2 February 2010}\)
A tree comparison technique

\((C, mk+1, hk+1, ek+1) \rightarrow \ldots\)

compilation of \(ek+1\) and its environment to javascript+html

Does the abstract syntax tree of \(ek+1\) correspond to the AST of eJS?

\((mk+1, \text{TREE}, eJS) \rightarrow \ldots\)
Conclusions

- We have:
  - a theorem for a toy Hop language that does not include any DOM operations such as “dom-append-child(node, tree)”
  - a Hop implementation of the technique (-s2 flag) that disallows insecure javascript (eval, (SCRIPT ..) hop constructor, document.write, call to unknown javascript functions)

- Our wish list:
  - understand how to prove compiler correctness with DOM operations (innerHTML)
  - understand whether we are being too restrictive for the disallowed JS