A Collection of Real World (JavaScript) Security Problems

Examples from $2^{1/2}$ Applications Areas of JavaScript

Achim D. Brucker

achim.brucker@sap.com

SAP AG, Central Code Analysis Team

July 2, 2014

A Collection of Real World (JavaScript) Security Problems

Abstract

JavaScript is gaining more and more popularity as an implementation language for various applications types such as Web applications (client-side), mobile applications, or server-side applications.
We outline a few security challenges that need to be prevented in such applications and, thus, for which there is a demand for analysis methods that help to detect them during during development.

Agenda

1 Motivation and Basics

2 SAP UI5: Client-side JavaScript

3 Apache Cordova: JavaScript on Mobile

4 HANA XS Engine: Server-side JavaScript

Agenda

1 Motivation and Basics

2 SAP UI5: Client-side JavaScript

3 Apache Cordova: JavaScript on Mobile

4 HANA XS Engine: Server-side JavaScript

What We Want to Find

Programming Patterns That May Cause Security Vulnerabilities

Mainly two patterns

Local issues (no data-flow dependency), e. g.,

- Insecure functions

```
1   var x = Math.random();
```

- Secrets stored in the source code

```
1   var password = 'secret';
```

Data-flow related issues, e. g.,

- Cross-site Scripting (XSS)

```
1   var docref  = document.location.href;
2   var input = docref.substring(
3               docref.indexOf("default=")+8);
4   var fake = function (x) {return x;}
5   var cleanse = function (x) {
6               return 'hello_world';}
7   document.write(fake(input));
8   document.write(cleanse(uinput));
```

- Secrets stored in the source code

```
1       var foo = 'secret';
2       var x = decrypt(foo,data);
```

Functions as First-Class Objects

```
1   var href = document.location.href;
2   var unsafeInput = href.substring(href.indexOf("default=")+8) // unsafe input
3   var safeInput = "1+2";                                       // safe input
4
5   // aliasing eval
6   var exec = eval;
7   var doit = exec;
8
9   var func_eval1     = function (x) {eval(x);};
10  var func_eval2     = function (x,y) {eVaL(y);};
11
12  var func_eval_eval = function (x) {func_eval1(x);};
13  var func_doit      = function (x) {doit(x);};
14  var func_exec      = function (x) {exec(y);};
15  var run            = func_eval1;
16  var inject_code    = func_exec;
17
18  doit(safeInput);   // secure
19  doit(unsafeInput); // code injection
```

Where is The Code of my Application?

```
1   var input  = document.location.href.substring(document.location..indexOf("default=")+8);
2   var fake = function (x) {return x;}
3   var cleanse = function (x) {return 'hello_world';}
4
5   var uinput = unknown(input); // unknown is nowhere defined
6   document.write(uinput); // secure!?
7
8   var finput = fake(input);
9   document.write(finput);  // not secure
10
11  var cinput = cleanse(input);
12  document.write(cinput); // secure
13
14  var extfinput = extfake(input);  // defined externally (part of scan)
15  document.write(extfinput);  // not secure
16
17  var extcinput = extcleanse(input); defined externally (part of scan)
18  document.write(extcinput); // secure
19
20  var nobodyKnows = toCleanOrNotToCleanse(input); multiply defined (underspecified)
21  document.write(nobodyKnows); // not secure!?
```

Agenda

1 Motivation and Basics

2 SAP UI5: Client-side JavaScript

3 Apache Cordova: JavaScript on Mobile

4 HANA XS Engine: Server-side JavaScript

The SAP UI5 Architecture
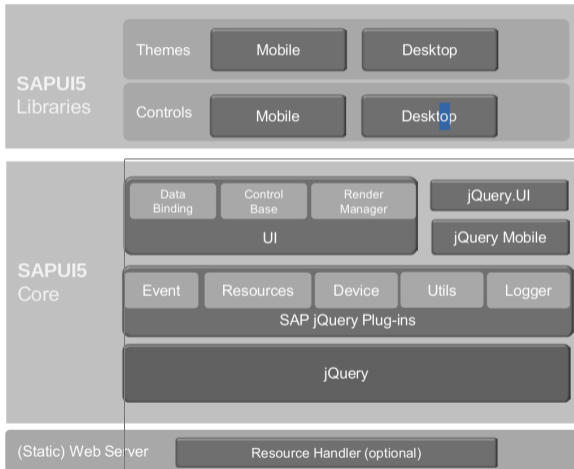
Prototype-based Inheritance

```
1    var vl = new sap.ui.commons.layout.VerticalLayout();
2    sap.ui.core.Control.extend("foo.Label", {
3        metadata : {
4            properties : {
5                "text" : "string"
6            }
7        },
8        renderer : function(oRm, oControl) {
9            oRm.write("<span>XSSLabel:_");
10           oRm.write(oControl.getText());
11           oRm.write("</span>");
12       }
13   });
14   var p = jQuery.sap.getUriParameters().get("xss");
15   vl.addContent(new foo.Label({text:p}));
16   return vl;
```

CSRF Prevention

You need to know your frameworks

```
1   var request = {
2           headers : {
3                   "X-Requested-With" : "XMLHttpRequest",
4                   "Content-Type" : "application/atom+xml",
5                   "X-CSRF-Token" : "Fetch"
6           },
7   };
8   if (Appcc.CSRFToken)
9           var request = {
10                  headers : {
11                          "X-Requested-With" : "XMLHttpRequest",
12                          "Content-Type" : "application/atom+xml",
13                          "X-CSRF-Token" : Appcc.CSRFToken
14                  },
15          };
16  else  var request = {
17                  headers : {
18                          "X-Requested-With" : "XMLHttpRequest",
19                          "Content-Type" : "application/atom+xml",
20                          "X-CSRF-Token" : "etch"  // secure?
21                  },
22          };
```

Agenda

1 Motivation and Basics

2 SAP UI5: Client-side JavaScript

3 Apache Cordova: JavaScript on Mobile

4 HANA XS Engine: Server-side JavaScript

Apache Cordova (SAP Kapsel): Overall Idea

An integrated platform for developing hybrid mobile apps

- Apache Cordova plus
  - App management
  - Encrypted Storage
  - Authentication
  - Logging
  - . . .
- Application management (SMP)
- Can be used with device management solutions
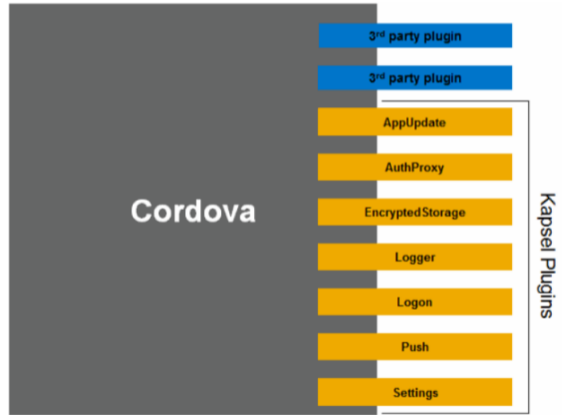
Exploiting the JavaScript to Java Bridge

- We can expose Java methods in JavaScript

```
foo.addJavascriptInterface(new FileUtils(), "FUtil");
```

- And use them in JavaScript easily

```
1  <script type="text/javascript">// <![CDATA[
2  filename = '/data/data/com.livingsocial.www/' + id +'_cache.txt';
3  FUtil.write(filename, data, false);
4  // ]]></script>
```

- Which might expose much more than expected

```
1  function execute(cmd){
2      return
3      window._cordovaNative.getClass().forName('java.lang.Runtime').
4                   getMethod('getRuntime',null).invoke(null,null).exec(cmd);
5  }
```

Agenda

1 Motivation and Basics

2 SAP UI5: Client-side JavaScript

3 Apache Cordova: JavaScript on Mobile

4 HANA XS Engine: Server-side JavaScript

The HANA XS Engine Architecture

Overview

Front-end Technologies
- http/s
- HTML5 / SAPUI5
- Client-side JavaScript

Control Flow Technologies
- OData
- Server-Side JavaScript
- XMLA

Data Processing Technologies
- SQL / SqlScript
- Calculation Engine Functions
- Application Function Library (AFL)

Client: Browser or Mobile

Presentation logic

SAP HANA

XS — **Control flow logic**

Data — **Calculation logic**

History Repeats: SQL Injection

```
1   $.response.contentType = "text/html";
2   var userInput = $.request.parameters.get('userStuff');
3
4   // We assume
5   // - $.db.getConnection().prepareStatement(x0, ..., xn) is secure iff x0 is *not*
6   //   influenced by user input
7   // - sql_sanitize() safeguards us against SQL injections.
8   // - any other preparedStatement call is evil regardless if it is influenced by
9   //   user input or not
10
11  if (userInput) {
12
13          var sql      = "select_*_from_SFLIGHT.SNVOICE_where_CustomID_='"
14                         + userInput + "'";
15          var safe_sql = "select_*_from_SFLIGHT.SNVOICE_where_CustomID_='"
16                         + sql_sanitize(userInput) + "'";
17
18          var db_object = $.db;
19          var conn      = db_object.getConnection();
20
21          var pstmt00 = $.db.getConnection().prepareStatement(sql);        // SQL injection
22          var pstmt01 = $.db.getConnection().prepareStatement(safe_sql);   // secure
```

History Repeats: SQL Injection

```
1  var sql      = "select_*_from_SFLIGHT.SNVOICE_where_CustomID_='"
2               + userInput + "'";
3  var safe_sql = "select_*_from_SFLIGHT.SNVOICE_where_CustomID_='"
4               + sql_sanitize(userInput) + "'";
5
6  var db_object = $.db;
7  var conn      = db_object.getConnection();
8
9  var pstmt00 = $.db.getConnection().prepareStatement(sql);        // SQL injection
10 var pstmt01 = $.db.getConnection().prepareStatement(safe_sql);   // secure
11
12 var pstmt02 = db_object.getConnection().prepareStatement(sql);      // SQL injection
13 var pstmt03 = db_object.getConnection().prepareStatement(safe_sql); // secure
14
15 var pstmt04 = conn.prepareStatement(sql);                         // SQL injection
16 var pstmt05 = conn.prepareStatement(safe_sql);                    // secure
17
18 var pstmt06 = conn.prepareStatement("..._where_ID_=_'$1'",userInput);   // secure
19 var pstmt07 = myconn.prepareStatement("..._where_ID_=_'$1'",userInput); // SQL injection
20
21 var pstmt08 = $.mydb.getConnection().prepareStatement(sql);       // SQL injection
22 var pstmt09 = $.mydb.getConnection().prepareStatement(safe_sql);  // SQL injection
```