中软国际公司内部技术规范

JavaScript语言安全编程规范

中软国际科技服务有限公司

修订声明

参考:华为 Javascript 语言安全编码规范。

0.

日期	修订版本	修订描述	作者
2017-07-28	1.0	整理创建初稿	陈丽佳
2017-08-02	1.1	统一整理格式，并修正部分错误	陈丽佳

目录

1. 前言

背景

JavaScript 是动态的、隐式的、解析型的高级编程语言。是 Web 程序的三种基本技术之一。大多数网站都会采用 JavaScript，并且所有现代的浏览器都无需插件支持 JavaScript。JavaScript 也用于非 Web 环境，如 PDF 文档、特定站点浏览器和桌面组件。新出现的快速 JavaScript 虚拟编译器与在其之上构建的平台增加了 JavaScript 用于服务器端应用的普及。在客户端，JavaScript 传统被实现为解释型语言，但越来越多的浏览器执行即时编译。JavaScript 被开发人员广泛采用，但是，同时它也有着其长期且严峻的安全问题，例如 XSS，内存泄露，跨域访问等。此本档为 JavaScript 安全编码提供指南。

使用对象

本规范的读者及使用对象主要为使用 JavaScript 语言的研发人员和测试人员等。

适用范围

该规范适用于基于 JavaScript 语言的产品开发。

术语定义

规则：编程时必须遵守的约定
建议：编程时可以作为参考的约定
说明：某个规则的具体解释
错误示例：违背某条规则的例子
正确示例：遵循某条规则的例子
例外情况：相应的规则不适用的场景

2. 通用规范

规则 2.1 访问外部对象时，需先判断该对象是否为空

说明：在访问某个外部对象或外部对象内的方法及属性时，必须先判断该对象是否为空，或该对象是否包含调用的方法及属性，如果未进行以上判断，很有可能会访问了 undefined 的对象或不存在的方法及属性。

错误示例：

```
function print(error) {
   addToSysLog(error.msg);  // err 可能是 undefined
}
```

推荐做法：

```
function print(error) {
    if (error && error.msg) {
      addToSysLog(error.msg);
    }
}
```

规则 2.2 功能失效时必须彻底删除对应的功能代码

说明：JavaScript 为解释性语言，若代码中的功能失效时必须彻底删除相应的代码，因为使用注释行等形式并不能实现彻底的清理功能，恶意人员可通过修改注释轻易恢复待清理的功能，影响产品使用。

规则 2.3 禁止注释中含有员工个人信息

说明：带有员工个人信息的注释有可能会泄露具体的开发人员信息，从而引入社会工程学方面的风险，因此要从脚本注释中删除员工个人信息，如 JavaScript 代码中的注释不能够体现工号信息、姓名、部门、邮箱等。

规则 2.4 禁止硬编码用户口令

说明：硬编码的口令将口令敏感信息泄露。

源代码示例

下面的代码片段使用硬编码，编码用户名和口令以链接到数据库：

```
...
var db = openDatabase('mydb', '1.0', 'TestDB', 2 * 1024 *
1024,'Tom','tom1990');
...
```

解决方案

- 对于出站身份验证：在代码外存储口令，存储在一个强保护的加密的配置文件或数据库中，免受所有外部访问，包括在同一系统上的其他本地用户。妥善保护密匙。如果你不能使用加密以保护文件，那么请确保尽可能严格的限制权限。
- 比起对第一次登录的默认用户名和口令进行硬编码，利用"first login"（第一次登录）模式，要求用户输入唯一的强口令是更好的方法。
- 对于入站身份验证：对口令采用强大的单向 hashes（散列）并且在有适当的访问控制的配置文件或数据库中存储 hashes。这样一来，盗取该文件/数据库，仍然要求攻击者设法破解口令。当认证过程中接收到进入的口令，提取口令的 hash 值，并将其与已保存的 hash 值对比。

另外，使用随机分配 salts（盐）为您生成每个单独的 hash。这增加了攻击者需要进行暴力攻击的计算量，可能会限制彩虹表方法的有效性。

建议 2.5 在客户端对外部输入进行校验

说明：虽然提交的参数在后台也会进行校验，但是如果在前端做适当的校验，就能在前端阻止一些不符合规则的输入，从而减少后台的校验压力，当然这样的校验也是从纵深防御的角度考虑，这样的校验可以防御 SQL 注入，XSS 注入等。

建议 2.6 使用 JavaScript 混淆器

说明：Javascript 混淆器转换 JavaScript 源代码为加扰和完全不可读形式，以防止代码被分析和盗窃。JavaScript 混淆器还可以使 HTML 标记的可读性变差，通过去除空格、行和注释，以及编码一些字符，如字符转义，丑化标签名和属性名。

JavaScript 源代码的混淆包括：

- 使用无意义符号更换符号名称。
  例如，使用 zcadaa4fc81 取代 list_of_customers
- 使用表达式替换数字常量。
  例如，使用（0x14b6+2119-0x1c15）更换 232
- 使用十六进制转义，更换字符串。例如"cust"转换成"\x63\x75\x73\x74"
- 删除或混淆意见
- 删除代码行中的空格和制表符

- 连接所有代码行
- 对前期阶段的结果进行编码

3. 输入校验

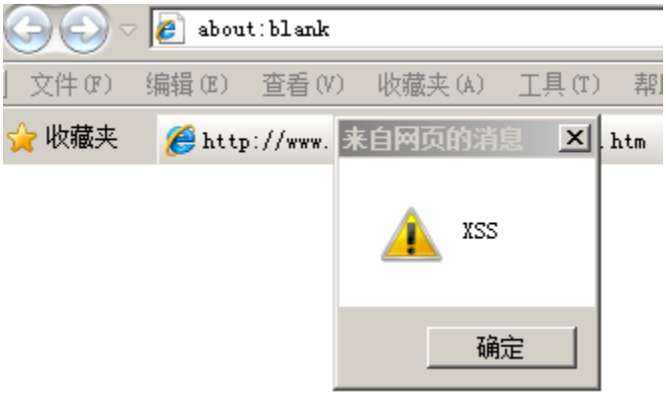
规则 3.1 禁止使用 eval()函数来处理来自外部的不可信数据

说明：eval()函数存在安全隐患，该函数可以把输入的字符串当作 JavaScript 表达式执行，容易被恶意用户利用，且 IE9 下使用 eval()存在内存泄露问题（在微软的 MSDN 中（http://support.microsoft.com/kb/2572253/zh-cn）已经确认了 IE9 浏览器下，使用 eval()函数反复来构造对象从大 JavaScript 对象符号(JSON) 字符串时，浏览器占用的内存使用量意外增加）。

错误示例：将字符串直接作为 JavaScript 执行，会引起跨站脚本攻击。

```
<script language="JavaScript">
  <!--
    var test = 'alert ("XSS")';
    eval(test)
  //-->
</script>
```

运行结果如下：



例外情况：

如果用户输入的部分作为 eval 函数参数进行拼接的一部分，则需要对这部分输入进行 JavaScript 转义：

```
var input=Encoder.encodeForJS(untrustedData);
```

规则 3.2 禁止直接对不可信的 JS 对象进行序列化

描述: JavaScript支持面向对象编程（OOP）技术。它有很多不同的内置对象，并允许用户创建对象。一个JS对象可以同时拥有数据和方法。如果这些数据或者方法来自用户输入，那么这个对象的序列化将产生可以被注入代码利用的安全漏洞。

源代码示例

可以使用 new object()或简单的如下所示的内联代码来创建新的对象。

```
//创建 JS 对象
message = {
    from : document.getElementById(“emailFrom”).value,
    to : document.getElementById(“emailTo”).value,
    subject : "I am fine",
    body : "Long message here",
    showsubject : function(){document.write(this.subject)}
};
//将 JS 对象进行序列化
var target = JSON.stringify(message);
```

这是一个简单的消息对象，其中有 2 个字段需要从页面输入框获得电子邮件地址。我们可以将该对象序列化并发送到另外的页面 A。程序员可以将它赋值到变量或者使用 eval()。如果攻击者在 from 邮件输入框中输入的是攻击的脚本，那么访问 A 页面的用户就将成为跨站点脚本攻击的受害者。

解决方案

- 安全敏感对象不应被序列化。
- 验证反序列化对象的数据，过滤附录 A 中的特殊字符。

规则 3.3 禁止直接将不可信数据组装成 JSON 对象

描述: JSON是一种简单有效的轻量级的数据交换格式，并且包含对象、数组、hash表、向量和列表等数据结构。JSON可用于JavaScript、Python、C、C++、C#和Perl等语言。在Web2.0应用程序上，序列化的JSON是一种非常有效的交换机制。开发人员经常利用Ajax通过JSON方式来传递必要的信息给DOM，但是如果没有对用于构建JSON结构的来自外部的数据进行校验，有可能发生JSON注入。

错误示例

下面是一个简单的 JSON 对象 “bookmarks”，它 使用不同的 name-value 对.

```
{"bookmarks":[{"Link":"www.example.com","Desc":"Interesting
link"}]}
```

可以在链接或 Desc 中注入恶意脚本。

```
{"bookmarks":[{"Link":"www.example.com","Desc":"<script>alert(1)</script>"}]}
```

如果它被注入到 DOM 并且执行，它将成为 XSS。这是另一种序列化恶意内容到终端用户的方法。

解决方案

验证 JSON 对象数据，过滤附录 A 中的特殊字符。

规则 3.4 禁止将不可信数据直接组装成 JS 数组

描述：JS数组是另一个比较普遍的序列化对象。人们可以很容易地跨平台移植它，并且它在使用不同语言的结构中也很有效。感染一个JS数组可以扰乱DOM上下文内容。可以在浏览器中使用简单的跨站脚本攻击Web页面。

错误示例：

下面是一个 JS 数组的示例：

```
new Array(“Laptop”, “Lenovo”, “T60”, “Used”, “900$”, “I have used it for 2 years”)
```

该数组是从一个拍卖二手笔记本的网站传出来的。如果这个数组对象在服务器端没有被仔细处理，用户就可以在最后字段中注入脚本。这种注入将危及浏览器安全并被攻击者利用。

```
new Array(“Laptop”, “Lenovo”, “T60”, “Used”, “900$”, “<script>alert(1)</script>”)
```

解决方案

验证 JS 数组元素，过滤附录 A 中的特殊字符。

规则 3.5 禁止未验证的输入作为重定向 URL

描述：

通过重定向，Web 应用程序能够引导用户访问同一应用程序内的不同网页或访问外部站点。应用程序利用重定向来帮助进行站点导航，有时还跟踪用户退出站点的方式。当 Web 应用程序将客户端重定向到攻击者可以控制的任意 URL 时，就会发生 Open redirect 漏洞，这种重定向可能是通过后台的重定向也可能是仅仅发生在前端重定向。攻击者可能利用 Open redirect 漏洞诱骗用户访问某个可信赖站点的 URL，并将他们重定向到恶意站点。攻击者通过对 URL 进行编码，使最终用户很难注意到重定向的恶意目标，即使将这一目标作为 URL 参数传递给可信赖的站点时也会发生这种情况。因此，Open redirect 常被作为钓鱼手段的一种而滥用，攻击者通过这种方式来获取最终用户的敏感数据。

解决方案

对重定向 URL 进行合法性校验，比如校验 URL 白名单校验。

规则 3.6 禁止将未验证的输入作为客户端系统路径

描述：通过用户输入控制 file system 操作目标路径，借此攻击者可以访问或修改其他受保护的系统资源。

当满足以下两个条件时，就会产生 path manipulation 错误：

1. 攻击者能够指定某一 file system 操作中所使用的路径。

2. 攻击者可以通过指定特定资源来获取某种权限，而这种权限在一般情况下是不可能获得的。

例如，在某一程序中，攻击者可以获得特定的权限，以重写指定的文件或是在其控制的配置环境下运行程序。

例 1：下面的代码使用来自于 HTTP 请求的输入来创建一个文件名。程序员没有考虑到攻击者可能使用像 "../../tomcat/conf/server.xml" 一样的文件名，从而导致应用程序删除它自己的配置文件。

```
var rName = document.URL.indexOf("reportName=")+10;
window.requestFileSystem(window.TEMPORARY, 1024*1024, function(fs) {
  fs.root.getFile('/usr/local/apfr/reports/' + rName, {create:
false}, function(fileEntry) {
    fileEntry.remove(function() {
      console.log('File removed.');
    }, errorHandler);
  }, errorHandler);
}, errorHandler);
```

例 2：以下代码使用来自于本地存储的输入来决定该打开哪个文件，并返回到用户。如果恶意用户能够更改本地存储的内容，就可以使用该程序来读取系统中扩展名为 .txt 的任何文件。

```
...
var filename = localStorage.sub + '.txt';
function oninit(fs) {
  fs.root.getFile(filename, {}, function(fileEntry) {
    fileEntry.file(function(file) {
       var reader = new FileReader();
       reader.onloadend = function(e) {
         var txtArea = document.createElement('textarea');
         txtArea.value = this.result;
         document.body.appendChild(txtArea);
       };
       reader.readAsText(file);
    }, errorHandler);
  }, errorHandler);
}

window.requestFileSystem(window.TEMPORARY, 1024*1024, oninit,
errorHandler);
...
```

建议 3.7 禁止直接将不可信数据插入到 WEB 页面中

描述：跨站脚本攻击的发生是由于WEB应用程序没能正确地过滤用户提交的内容而把它们呈现在HTML中。这允许攻击者插入任何的HTML元素进入WEB页面中，通

常是<script>标签的形式。攻击者经常使用XSS攻击盗取cookie和会话信息，或者诱骗用户把隐私信息发送给错误的人。

错误示例

假设下面的代码是用于创建一个表单，让用户选择他/她的首选语言。在查询字符串中还提供了一个默认的语言，作为参数“default”（默认）。

```
…
Select your language:
<select><script>
document.write("<OPTION
value=1>"+document.location.href.substring(document.location.href.i
ndexOf("default=")+8)+"</OPTION>");
document.write("<OPTION value=2>English</OPTION>");
</script></select>
…
```

如果URL调用页面，如下：

```
http://www.some.site/page.html?default=<script>alert(document.cookie)
</script>
```

当受害者点击这个链接时，浏览器执行攻击者的脚本:

```
alert(document.cookie)
```

解决方案

- 在把 HTML 和 JavaScript 插入执行环境上下文的 HTML 当中之前，先把它们进行转义。
  为了让 DOM 安全地动态更新 HTML，我们建议先进行 HTML 编码，然后再进行 JavaScript 编码所有不可信的输入。如下示例表示将 HTML 内容插入到当前 HTML 中：

```
element.innerHTML =
 “<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrustedData))%>”;
```

如下示例表示将 JavaScript 内容插入到当前 HTML 中：

```
document.write(“<%=Encoder.encodeForJS(Encoder.encodeForHTML(untrust
edData))%>”);
```

- 在插入不可信数据到执行环境中的 HTML 属性上下文之前，先转义 JavaScript。
  示例如下：

```
var x = document.createElement(“input”);
x.setAttribute(“name”, “company_name”);
x.setAttribute(“value”,
 ‘<%=Encoder.encodeForJS(companyName)%>’);
var form1 = document.forms[0];
form1.appendChild(x);
```

- 当插入不可信数据到事件处理程序和执行环境中 javaScript 上下文中的时，要先转义 JavaScript。示例如下：

```
var x =<%=Encoder.encodeForJS(companyName)%>;
```

- 在插入不可信数据到执行环境的 CSS 属性上下文之前，转义 CSS。
  示例如下：

```
document.body.style.backgroundImage =
"url(<%=Encoder.encodeForCSS(Encoder.encodeForURL(companyName))%>)"
;
```

4. 其他

规则 4.1 禁止在 localStorage 存储敏感信息

说明：HTML5 提供 localStorage 和 sessionStorage 映射，以支持开发人员保留程序值。sessionStorage 映射仅在页面实例和即时浏览器会话期间为调用页面提供存储。但是，localStorage 映射会提供可供多个页面实例和浏览器实例访问的存储。此功能允许应用程序在多个浏览器选项卡或窗口中保留和使用同一信息。

例如，开发人员可能希望在旅游应用程序中使用多个浏览器选项卡或实例，以支持用户打开多个选项卡来比较住宿选择，同时保留用户最初的搜索条件。在传统的 HTTP 存储方法中，用户会面临在一个选项卡中执行的购买和决策（并存储在会话或 cookies 中）与另一个选项卡中的购买相干扰的风险。

借助跨多个浏览器选项卡使用用户值的功能，开发人员必须多加小心，以免将敏感信息从 sessionStorage 范围移至 localStorage，反之亦然。

示例：

以下示例将信用卡 CCV 信息存储在会话中，表明用户已授权该站点收取文件中卡的购买费用。对于在浏览器选项卡环境中的每个购买尝试，都需要信用卡许可。为避免重新输入 CCV，此信息被存储在 sessionStorage 对象中。但是，开发人员还将信息存储在 localStorage 对象中。

```
...
try {
   sessionStorage.setItem("userCCV", currentCCV);
} catch (e) {
   if (e == QUOTA_EXCEEDED_ERR) {
           alert('Quota exceeded.');
   }
}
    ...
 var retrieveObject = sessionStorage.getItem("userCCV");
 try {
    localStorage.setItem("userCCV",retrieveObject);
  } catch (e) {
     if (e == QUOTA_EXCEEDED_ERR) {
         alert('Quota exceeded.');
     }
     ...
   var userCCV = localStorage.getItem("userCCV");
...
}
```

...

通过将信息放回 localStorage 对象中，此 CCV 信息在其他浏览器选项卡和新调用的浏览器中可用。这样可以绕开预期工作流的应用程序逻辑。

附录 A

特殊字符	描述	ASCII
&	与字符	&amp;
<	角括弧	&lt;
>	角括弧	&gt;
“	双引号	&quot;
‘	单引号	&#x27;
/	斜杠	&#x2F;

附录 B

同源策略

同源策略阻止从另外一个网站上加载脚本或者发送信息给另外一个网站。这一政策阻止被另外一个站点的恶意代码接管或操作另一个站点的文档。没有它，来自一个恶意网站的 JavaScript 可以做任意数量的不良的操作，比如监听按键，当你在不同的窗口登录网站并等待你去网上银行网站时，将被插入虚假交易，偷取来自其他域的登录 cookies 等。

当一个脚本试图访问在不同窗口的属性或方法时，例如，使用window.open()返回的句柄 ，在浏览器上执行有关文件的URL的同源检查。如果文件的URL通过此检查，则该文档可以被访问。如果他们没有通过检查，则会抛出一个错误。同源检查包括验证在目标窗口中的文档的URL具有相同的“源”。如果两个文件使用相同的协议和端口在同一台服务器加载，那么两个文件同源。例如：从http://www.example.com/dir/page.html加载的脚本，可以访问使用HTTP从www.example.com加载的任何对象。下表显示试图访问包含不同URL窗口的结果，假设访问脚本来自http://www.example.com/dir/page.html。

同源检查结果清单

目标窗口的 URL	同源检查结果(使用 www.example.com)	原因
http://www.example.com/index.html	通过	同一域和协议
http://www.example.com/other1/other2/index.html	通过	同一域和协议
http://www.example.com:8080/dir/page.html	未通过	不同端口

http://www2.example.com/dir/page.html	未通过	不同服务器
http://otherdomain.com/	未通过	不同域
ftp://www.example.com/	未通过	不同协议