Computer Science and Application 计算机科学与应用, 2015, 5(11), 403-409
Published Online November 2015 in Hans. http://www.hanspub.org/journal/csa
http://dx.doi.org/10.12677/csa.2015.511051

Hans汉斯

Research of UIWebView Component Vulnerability on iOS Platform

Fuyi Li, Shaozhang Niu, Wen Zhang

Beijing University of Posts and Telecommunications, Beijing
Email: lfy_5248@163.com, szniu@bupt.edu.cn, 14311648@qq.com

Received: Nov. 4th, 2015; accepted: Nov. 20th, 2015; published: Nov. 27th, 2015

Abstract

Now more and more iOS applications adopt the Hybrid model, which not only brings the advantage of cross-platform development, but also brings some security problems. This paper summarizes the current security problems of using UIWebView components on the iOS platform and we do a very detailed analysis about these security problems. Finally, for each security problem, we put forward the corresponding solution to ensure the security of the application.

Keywords

Hybrid Mode, Uiwebview Component, Security Problem

iOS平台下基于UIWebView漏洞的研究

李伏一，牛少彰，张　文

北京邮电大学，北京
Email: lfy_5248@163.com, szniu@bupt.edu.cn, 14311648@qq.com

收稿日期：2015年11月4日；录用日期：2015年11月20日；发布日期：2015年11月27日

摘　要

现在越来越多的iOS应用程序在进行系统设计时，采用了Hybrid混合架构模式，这种模式虽然带来了跨平台开发的优势，但是也带来了一些安全问题，本文针对这一问题展开研究，总结了目前iOS平台在使用

文章引用: 李伏一, 牛少彰, 张文. iOS平台下基于UIWebView漏洞的研究[J]. 计算机科学与应用, 2015, 5(11): 403-409.
http://dx.doi.org/10.12677/csa.2015.511051

UIWebView组件时所带来的一些安全问题，并对这些安全问题做了很详尽的分析，最后针对每一个安全问题，分别提出了一个解决方案，确保在享受Hybrid模式的优势时，也保证了应用程序的安全性。

关键词

Hybrid模式，UIWebView组件，安全性

1. 引言

UIWebView 是 iOS 平台上很重要的组件，它能够使移动应用内置一个浏览器，能够解析和展示服务器端的网页[1]，是移动应用进行 Hybrid 模式架构的一个基础组件，如著名的 PhoneGap 框架就是 native 和网页之间的中间件，它的实现原理就是基于 UIWebView 组件。现在有很多主流应用都采用这种 Hybrid 模式，如国外的 Facebook，国内的百度搜索等。但目前来说，在使用 UIWebView 组件的过程中也出现了一些安全性问题，如 CVE-2013-6893 报出的猛犸浏览器 iOS 版 UXSS 漏洞[2]，如乌云平台上报出的 WooYun-2015-146717 Webview 拒绝服务漏洞[3]，可见 UIWebView 安全性问题还是很严重的。

本文首先简单介绍了 UIWebView 的使用方法，包括 Javascript 代码和 Objective-C 代码相互调用的过程，然后总结了目前在使用 UIWebView 时存在的安全问题，并对每个漏洞进行详细的分析，最后分别对每种不同的漏洞提出了相应的解决方案，为开发者提供一份很详尽的安全开发指南。

2. UIWebView 简介

UIWebView 是 iOS 开发中最常用的控件，相当于在 iOS App 中内置了一个浏览器，它能使 App 应用浏览网页，打开文件等操作，使 iOS 应用和网络应用直接交互，使得两者紧密的结合在一起，这是一个跨平台的部署方案，开发一次既可以部署在 iOS 平台又可以部署在 Android 平台，而且更新升级会更加的容易，只要在服务端更新升级，客户端就可以实时的更新展示，不需要通过 AppStore 的审核，大大缩短更新周期。下面来简单介绍一下 UIWebView 的使用方法。

UIWebView 有关键的 API：

- loadRequest：可以用来加载一个 url 地址，它需要一个 NSURLRequest 参数。
- loadHTMLString:baseURL：这个方法用于直接加载 html 代码，也可以用这个方法从本地 html 读取代码，然后加载。
- loadData：MIMEType:textEncodingName:baseURL：加载带有 MIME 类型的数据,包含视频、图像、文本、音频、应用程序等数据。这个方法不常用，一般使用前两个方法。
- goBack/goForward/stopLoading/reload。导航的几个方法，向前，向后，停止加载，重新加载等。

下述代码描述了在 WebView 里加载 google 网页的过程，如下图所示：

```Objective C code
UIWebView *myWebView=[[UIWebViewalloc] initWithFrame:CGRectMake(0, 20, 320, 300)];
NSURL *url=[NSURLURLWithString:@"http://www.google.com.hk"];
NSURLRequest *request=[[NSURLRequestalloc] initWithURL:url];
[myWebView loadRequest:request];
```

UIWebView 的委托 UIWebViewDelegate 主要有下面几个关键的委托 API [4]：

-(BOOL)webView:(UIWebView*)webViewshouldStartLoadWithRequest:(NSURLRequest*)request navigationType:(UIWebViewNavigationType)navigationType

加载前执行该方法。

-(void)webViewDidStartLoad:(UIWebView *)webView;

开始加载的时候执行该方法。

-(void)webViewDidFinishLoad:(UIWebView *)webView;

加载完成的时候执行该方法。

-(void)webView:(UIWebView*)webViewdidFailLoadWithError:(NSError *)error;

加载出错的时候执行该方法。

UIWebView 更常见的用法是 Objective-C 与 JavaScript 之间的相互调用，使得页面展示，程序交互更加的灵活，其中 Objective-C 调用 Javascript 的 API：通过该 API：stringByEvaluatingJavaScriptFromString: 可以在当前域里插入 Javascript 代码，具体使用方法如下图：

```
Objective C code
[webView stringByEvaluatingJavaScriptFromString:@"var script = document.createElement(
        "script.type = 'text/javascript';"
        "script.text = \"function myFunction() { "
        "var field = document.getElementsByName('q')[0];"
        "field.value='MyValue';"
        "document.forms[0].submit();"
        "}\";"
        "document.getElementsByTagName('head')[0].appendChild(script);"];
[webView stringByEvaluatingJavaScriptFromString:@"myFunction();"];
```

上述代码，首先通过 js 创建一个 script 的标签，type 为'text/javascript'。然后在这个标签中插入一段字符串，这段字符串就是一个函数：myFunction，这个函数实现 google 自动搜索关键字的功能。然后 stringByEvaluatingJavaScriptFromString 执行 myFunction 函数。

相反使用 Javascript 调用 Objective-C 的方法就有很多种实现方法，下面介绍一种最常见的调用方法，该方法是利用 UIWebViewDelegate 委托的方法重定向请求实现的，具体实现过程如下图所示：

首先实现一个 JavaScript 函数，来传递所请求的 Objective-C 的函数名和参数，自己定义请求的协议的类型：testapp://"+cmd+":"+param。

testapp 是自己定义的一个协议，cmd 是要调用的 oc 的方法，param 是调用的参数，然后利用 document.location 来达到目的，上面的"testapp://"+cmd+":"+param 是自定义的一种请求格式，其请求格式符合如下范式都可以：

document.location="MyCustomProtocolName:functionName?param1=value1&param2=value2.

请求传递后，可以通过上述函数 shouldStartLoadWithRequest 里捕获这次请求，然后对请求做对应的解析，就可以调用的对应的函数，函数名和参数分别是请求里解析出来的值，具体的实现方法如下图所示：

```
Objective C code
- (BOOL)webView:(UIWebView *)webView shouldStartLoadWithRequest:(NSURLRequest *)requ
 {
NSString *requestString = [[request URL] absoluteString];
NSArray *components = [requestString componentsSeparatedByString:@":"];
if ([components count] > 1 && [(NSString *)[components objectAtIndex:0] isEqualToStr
     if([(NSString *)[components objectAtIndex:1] isEqualToString:@"alert"])
       {
            UIAlertView *alert = [[UIAlertView alloc]
                                  initWithTitle:@"Alert from Cocoa Touch" message:[c
                                  delegate:self cancelButtonTitle:nil
                                  otherButtonTitles:@"OK", nil];
            [alert show];
       }
        return NO;
    }
    return YES;
}
```

上述方法通过对请求的 URL 解析，先判断传入所请求的函数是什么，如果匹配上”alert”则继续执行该逻辑下代码，即调用了本地的 Objective-C 代码。

3. UIWebView 存在的安全漏洞

虽然 UIWebView 的使用给程序开发带来很多便利，但是同时也带来一些安全性问题，目前 UIWebView 存在如下几点安全问题：

(1) UIWebView XSS 漏洞

漏洞详情：

UIWebVIew 跨站漏洞分两种情况，第一种是普通的跨站，第二种是浏览器通用性 XSS。UIWebView 普通 XSS 漏洞，该漏洞是由于网站的漏洞造成的，如下两种情况有可能触发[5]：

第一种情况是：未知的来源的进入了 web 页面，如对用户输入框输入的数据没有进行过滤，比如反射性的 XSS，恶意数据通过用户组件，URL scheme handlers 或者 notifications 传递给了 web 页面；又或者如存储性的 XSS，恶意数据通过数据库或者其他数据源传递给了 web 页面。第二种情况是：恶意的数据动态的发送到 UIWebView 组件，并且 UIWebView 组件没有验证该数据的合法性。

第二种种类型是：浏览器通用性 XSS [6]，浏览器的同源策略在 iOS 端和桌面端是有些许不同的，比如说 file 域 applewevdata 域的 URL 是完全可以跨域访问浏览器里的任何资源的，除了访问本地其他文件。这就意味着如果你将一个不受信任的未知源的 HTML 的文件保存到了本地，并通过 UIWebView 的上述三个函数加载了该路径的文件，这就是全局的 XSS 漏洞，这个恶意的本地 HTML 可以读取所有的保存在该应用里的 cookie，绕过同源策略做一些非法的操作，该跨域漏洞是利用的本地浏览器的漏洞，而不是网站的漏洞。

漏洞危害：

基于 XSS 漏洞的攻击种类有很多形式，一般是通过将用户的像 cookie，session 等敏感数据传递给攻击者，或者重定向用户页面到攻击者控制的服务器上，或者诱导用户输入一些敏感数据，然后传递到攻击者的服务器上，实现钓鱼攻击。

(2) UIWebView 接口使用不当风险。

漏洞详情：

UIWebView 使用了敏感 API 的接口，如发短信，发邮件等，导致恶意网页可以调用这些接口完成一些恶意的行为，通过过前面介绍的 Objectivr-C 和 Javascript 之间的 bridge 来实现 Javascript 去调用 native 的代码，如果在调用 native 代码之前没有验证调用者的合法性的话，就有可能导致恶意网页调用本地 native 方法去完成一些恶意的行为。

漏洞危害：有可能对用户财产造成一定的影响。

(3) UIWebView https 中间人攻击漏洞

漏洞详情：

UIWebView 一般会使用 SSL 安全链接来连接服务器端，来保证客户端与服务器之间传输数据的安全性默认 iOSUIWebView https 请求无效证书的话，如果用户不点击信任此站点时，此会话会被拒绝的，但这个设置可以被更改的，比如允许用户接受自签名的证书等，国外有一个测试表明 14%的测试浏览器会静默接受自签名的 SSL 证书[7]。所以 UIWebView 中间人漏洞有如下两种情况，第一种情况是直接没有没有校验证书合法性，如校验根证书是否为自签名，是否可信，证书是否过期。第二种情况是校验了证书的合法性但没有校验访问的主机域名与证书上的域名是否匹配。如之前 AFNetWorking 爆出的 https 校验漏洞，该漏洞原因就是因为默认情况下不去校验证书的域名和所请求的域名是否一致，导致任何一个

只要是可信任 CA 机构签发的证书，都可以校验成功[8]。

到目前为止，iOS 系统已经更新到 iOS9，在这个版本的 iOS 系统下新增加了一个安全机制-App Transport Security，简称 ATS。这个安全机制将 app 内的非 https 连接都阻塞掉了，强制服务器端与客户端用户 https 来连接通信，并且 TLS 的版本不得低于 v1.2，而且规定了加密套件和证书签名的哈希算法，该机制仅仅是要求在 Xcode7 和 iOS9 SDK 的，现有的应用是不会受影响，但是还是建议服务器端升级网络配置使用 https 链接，并且做好检验证书的工作，保证使用 https 的安全性[9]。

漏洞危害：

通过中间人攻击可以查看客户端与服务器端传输的数据，有可能导致用户隐私数据的泄露，通过中间人可以修改客户端与服务器之间传递的数据，有可能导致用户财产的损失。

当用户输入了帐号密码等信息后，钓鱼应用就可以通过网络数据或发送短信等形式将获取的数据发送至指定服务器或号码。一些钓鱼应用的伪造界面做的比较逼真，很多用户在账号密码等信息被窃取了都未能察觉，从而给用户的个人隐私和财产安全带来了极大的危害。

4. UIWebView 安全漏洞的解决方案

(1) UIWebView XSS 漏洞的解决方案[10]：

要很好解决 UIWebView XSS 漏洞需要做到如下几点：

a) 如果没有特殊需要，可以使用 Safari 来打开一个链接，而没有必要通过 UIWebView 去实现。

b) 加载远程内容时，应确保使用了安全的机密的链路，比如说使用 https 连接。

c) 在使用 SSL/TLS 连接时确保使用了安全性强的加密算法

d) 在使用 SSL/TLS 连接时确保正确的校验了证书，保证客户端不能接受自签名的证书或者非法证书。

e) 通过 NSData 类的 dataWithContentsOfURL 方法去检测加载到 UIWebVIew 的内容，确保 UIWebView 没有加载一些非法脚本。比如如下代码段：

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
webView.delegate = self;
NSString *urlAddress = @"http://www.google.com";
NSURL *url = [NSURL URLWithString:urlAddress];
NSData* HTMLData = [NSData dataWithContentsOfURL:url];
NSString* HTMLStr;
HTMLStr = [[NSString alloc] initWithData:HTMLData encoding:NSASCIIStringEncoding];
HTMLStr = [HTMLStr stringByReplacingOccurrencesOfString:@"script"
withString:@"REMOVED"];
[webView loadHTMLString:HTMLStr baseURL:nil];
[window makeKeyAndVisible];
}
```

这段代码在加载网页的内容之前，会用 dataWithContentsOfURL 方法将加载的网页的内容提取出来放入 NSData 中，在使用 loadHTMLString 之前可以对 NSData 的内容过滤，编码等等，上述代码中是将 script 字符串都替换掉，保证了 UIWebView 不去加载脚本，从而也避免了加载恶意脚本的可能。

f) 最好不要使用 UIWebView 加载第三方的内容，如果非加载不可时，确保使用 UIWebViewDelegate 协议来实现一个白名单策略，来确保内容来源都是可信的。比如如下代码段：

```
- (BOOL)webView:(UIWebView*)webView shouldStartLoadWithRequest:(NSURLRequest*)request
  navigationType:(UIWebViewNavigationType)navigationType {
    NSURL *url = request.URL;
    NSString *urlString = url.absoluteString;
    NSLog(@"urlString: %@",urlString);
    return YES;
}
```

上述代码通过通过上述函数 shouldStartLoadWithRequest 里捕获这次请求，将 URL 拿出来存入 NSURL 中，这样就这样通过这个 NSURL 来实现白名单黑名单策略，NSURL 为提供如下六个可以截断 URL 的字段：

absoluteString：返回完整 URL

standardizedURL：返回一个完整标准的 NSURL 对象

host：主域名

scheme：返回统一资源定位符的类型如 http，https，ftp.file 等等

pathExtension：返回文件 URL 的扩展名

lastPathComponent：返回 URL 路径的最后一部分内容

可以通过这些截断的 URL 来做一些白名单、黑名单策略，如禁用 file 域，不让它加载本地文件，这样就避免了浏览器的 UXSS 问题。

(2) UIWebView 接口使用不当风险解决方案：

a) 尽量不要在 UIWebView 上开发一些敏感接口，如发短信，发邮件等，发微博等，

b) 如果非要开放一些敏感接口，需要对调用该接口来源的合法性进行校验，通过 UIWebViewDelegate 协议来实现一个白名单，确保来源的合法性或者自定义 NSURLProtocol handler 来处理由 UIWebView 发送的每个请求，这样也可以对每个请求用白名单和黑名单策略来过滤，确保不加载攻击者的恶意脚本。这个防御的具体的方法跟上条 f)中讲述的方法基本一致，这里不再赘述。

(3) UIWebView https 中间人攻击漏洞解决方案：

a) 完整正确的检验证书，检验签名的 CA 机构是否合法，证书是否是自签名的，证书是否过期，校验整个证书链等。

b) 校验证书的主机域名是否一致。如使用 AFNetworking 来支持 https 的话，它的整个安全验证证书的流程应该如下代码：

```
NSURL * url = [NSURL URLWithString:@"https://www.google.com"];
AFHTTPRequestOperationManager * requestOperationManager = [[AFHTTPRequestOperationMa
dispatch_queue_t requestQueue = dispatch_create_serial_queue_for_name("kRequestCompl
requestOperationManager.completionQueue = requestQueue;

AFSecurityPolicy * securityPolicy = [AFSecurityPolicy policyWithPinningMode:AFSSLPin

//allowInvalidCertificates 是否允许无效证书（也就是自建的证书），默认为NO
securityPolicy.allowInvalidCertificates = NO;

//validatesDomainName 是否需要验证域名，默认为YES
securityPolicy.validatesDomainName = YES;

//validatesCertificateChain 是否验证整个证书链，默认为YES
securityPolicy.validatesCertificateChain = YES;

requestOperationManager.securityPolicy = securityPolicy;
```

上述代码配置了 securityPolicy，其中有三个比较重要的配置点，即 allowInvalidCertificates 代表是否允许无效证书，该选项一定是 NO，确保不接受自签名的证书。

validatesDomainName 代表是否需要验证域名，默认为 YES，一定要验证主机域名和证书上的域名是否是一致的，假如证书的域名与你请求的域名不一致，需把该项设置为 NO；如设成 NO 的话，即服务器使用其他可信任机构颁发的证书，也可以建立连接，这个非常危险。置为 NO，主要用于这种情况：客户端请求的是子域名，而证书上的是另外一个域名。因为 SSL 证书上的域名是独立的，假如证书上注册的域名是 www.google.com，那么 mail.google.com 是无法验证通过的，如置为 NO，建议自己添加对应域名

的校验逻辑。

validatesCertificateChain 代表是否验证整个证书链，默认为 YES，如是自建证书的时候，可以设置为 YES，增强安全性；假如是信任的 CA 所签发的证书，则建议关闭该验证，因为整个证书链一一比对是完全没有必要。

5. 总结

本文总结了在 iOS 平台上使用 UIWebView 过程中所带来的一些安全问题，并对每个安全隐患做了详细的分析，如漏洞触发的条件，漏洞产生的位置，漏洞产生的危害等方面进行了很详尽的阐述，最后对针对每个安全隐患，也给出了相应的解决方案，为开发者提供一份很详尽的安全开发指导书。

致　谢

本文的研究得到国家自然科学基金的资助，为研究的顺利进行提供了资金保证，在此对国家自然科学基金表示衷心的感谢。

基金项目

国家自然科学基金项目(61070207, 61370195)。

参考文献 (References)

[1] UIWebViewClassReference. https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWebView_Class/index.html

[2] CVE-2013-6893. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2013-6893

[3] WooYun-2015-146717. http://www.wooyun.org/bugs/wooyun-2015-0146717

[4] Phonegap. http://www.phonegap.com

[5] Extracting html from a webview. http://lexandera.com/2009/01/extracting-html-from-a-webview/

[6] Pilorz, L. and Wylecial, P. (2014) 探讨 iOS 浏览器的安全问题. SyScan360.

[7] Intercepting Page Loads in webview. (2009). http://lexandera.com/2009/02/intercepting-page-loads-in-webview/

[8] iOS 安全系列之二: HTTPS 进阶[EB/OL]. http://oncenote.com/2015/09/16/Security-2-HTTPS2/

[9] Apple iOS 9: Security & Privacy Features. https://medium.com/@FredericJacobs/apple-ios-9-security-privacy-features-8d82d9da10eb#.7b7zakeqe

[10] Adventures with iOSUIWebviews. https://labs.mwrinfosecurity.com/blog/2012/04/16/adventures-with-ios-uiwebviews/