透视黑客技术发展焦点，把握黑客攻防技术跳动脉搏，全面收录流行黑客技术

黑客防线

《黑客防线》编辑部 编

2011

精华奉献本 下册

- 黑客编程实战大演练
- 黑器免杀与入侵进阶
- 加密与破解经典实例
- 网络安全与加固精讲

2CD-ROM

人民邮电出版社
POSTS & TELECOM PRESS

黑客防线

2011

精华奉献本 下册

《黑客防线》编辑部 编

人民邮电出版社
北京

内 容 提 要

《黑客防线》是国内最早创刊的网络安全技术媒体之一。本书收录了《黑客防线》总第109期至第120期的精华文章。

《黑客防线》一直秉承“在攻与防的对立统一中寻求突破”的核心理念，关注网络安全技术的相关发展，并一直跟踪国内网络安全技术的发展，经过2001年创刊至今，已经成为国内网络安全技术的顶尖媒体。本书选取了编程解析、工具与免杀、网络安全顾问、密界寻踪、首发漏洞、特别专题、漏洞攻防、脚本攻防、溢出研究以及渗透与提权等方面的精华文章，并配有两张CD光盘，其中包含安全技术工具、代码和录像，为读者阅读、理解提供了非常便捷的途径。

本书分为上、下两册，适合高校在校生、网络管理员、网络安全公司从业人员、黑客技术爱好者阅读。

《黑客防线》总编：孙彬
《黑客防线》总编邮箱：binsun20000@gmail.com
《黑客防线》联系电话：010-62145877
《黑客防线》客服QQ：200993886　200993865

前　言

2010年匆匆过去了。对于网络安全的相关技术而言,这一年其实很不平凡。在这一年里,纷乱的网络环境得到治理。在技术层面上,抑制了浮躁,同时也引发了深入研究底层核心技术的思考。这一年,恰逢《黑客防线》创刊十周年。十年磨一剑,作为一本技术月刊,基本上守住了纯技术这样一块净土,坚持了核心技术成就未来的理念,同时在方法上,一直倡导在攻与防的对立统一中寻求突破的方法论,因而过去的一年也可以说是本刊丰收的一年。

这本精华奉献本,就是2010年全年技术月刊精华文章的集合。

需要说明的是:由于多种原因,技术月刊从去年第7期开始停止了纸张版本的出版,只有网站提供的电子版半月刊,所以,这本精华本就更显得弥足珍贵。特别是对于没有按期订阅电子版月刊的读者,更需要购买的就是2011精华本了。

如果需要及时跟踪和捕捉新的技术,还是希望读者随时到黑客防线网站订阅电子半月刊。

本书附带的光盘,是为了方便读者进行源程序测试。精华本中的每篇文章基本上都提供了源程序。如果光盘中找不到某一篇文章涉及的源程序,那就是当时作者投稿时未提供,读者只要研究领会文章所述的技术要义,然后自己完成程序并不是很难。

在这里,要感谢十年来坚持阅读《黑客防线》并且一直践行技术创新和突破的读者朋友们,更要感谢作者朋友们一直进行着艰苦卓绝的技术研究和探索,不断将新技术突破,并整理成文章与大家分享。

在此,吁请具备一定技术能力的读者,积极参与到作者队伍中来,我们的作者群其实就是一支虚拟的技术团队——黑客防线技术团队,这是网络信息安全方面的一支精英团队。这个团队希望不断有新生力量参与进来。

让我们从现在做起、从我做起,一起关注网络安全技术的快速发展势头,抓住网络安全技术的新亮点,一起开创中国网络安全技术的新篇章!

郑重声明:本书所讲述的内容仅供学习参考,切勿用于非法用途,由此引起的后果自己负责,出版社不负任何责任。通过阅读本书,希望读者能树立良好的网络安全意识,提高网络安全防御水平。同时提醒读者,应在受控的环境(比如单独用来作为测试的计算机)里分析、使用书中的代码,切勿在企业的业务或生产网络中使用这些程序,以免造成不必要的损失。

《黑客防线》编辑部

2011年3月

《黑客防线》2011精华奉献本目录(下册)

密界寻踪

首发漏洞

特别专题

漏洞攻防

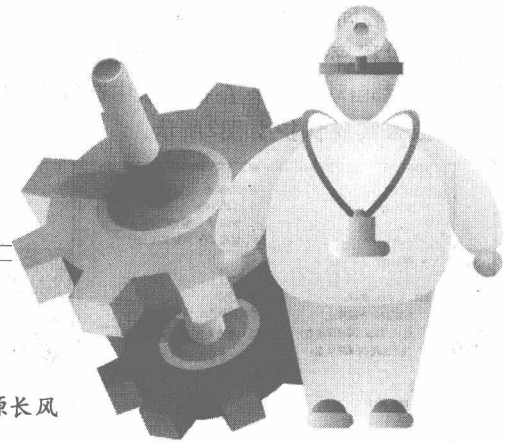
脚本攻防

溢出研究

渗透与提权

前置知识：Delphi
关键词：破解、Shell、SDK

利用Shell SDK保护程序



文/图 清原长风

加密与破解永远是对立的，加壳与脱壳也永远是势不两立的。对于我们平时使用的壳来说，脱壳已经不再是难事，各种各样的教程在网上随处可见。当然，大牛们有自己未公开的壳可以用，但对于技术有限的大多数人来说，开发壳是不太可能的事情，那我们还有其他的选择吗？当然有，虽然如今的公开壳已经被大家分析得一清二楚，但这些Shell的开发者的想法也并不局限于壳的简单实用，而是提供很大的灵活性供使用者对抗破解，这就是本文的主角——Shell SDK。

Shell的SDK是一套可以在源代码基础上中使用的加壳包，并不是我们平时开发使用的完整意义上的SDK，它的使用没有过高的技术门槛，使我们在不需要了解壳的编写及原理的情况下灵活地使用壳。常见的公布了SDK的Shell有EncryptPE、ASProtect、VMProtect等。这些Shell SDK的使用都十分类似，只需要在我们程序的源码中需要保护的部分按照SDK中提供的规则添加语句行即可实现保护，添加语句行的目的就是形成固定标记，这让保护壳能够识别源码中需要保护的代码段。下面以VMProtect SDK来说明其具体用法。

VMProtect出自俄罗斯Polytech公司，使用伪指令虚拟机来保护程序，其兼容性也是非常不错的，支持的编译程序很多，常见的Delphi、BCB、VC、VB等均支持；格式上支持exe、src、dll、ocx、sys和bpl。VMProtect SDK的保护方式有两种：MAP映射文件法和语句标记法。

首先我们来说MAP映射文件法。什么是MAP文件呢？援引微软官方的说法："映射文件是一个文本文件，包含有关被链接程序的下列信息：模块名称，为文件的基名称；时间戳，来自程序文件头（不是来自文件系统）；程序中的组列表，包括每个组的起始地址（节：偏移量的形式）、长度、组名和类；公共符号的列表，包括每个地址（节：偏移量的形式）、符号名称、平直地址和包含符号定义的.obj文件；入口点（节：偏移量的形式）"。在保护我们的程序前，在生成程序的时候需要生成这个程序的MAP文件，VMProtect就是通过MAP文件得知编译出来的程序信息的。以VMProtect V1.21和Delphi7为例，新建一个exe工程，在FormCreate过程中写入以下代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var i,x,y:integer;
function add(a,b:integer):integer;
begin
result:=a+b;
end;
begin
x:=1;y:=10;
showmessage(inttostr(add(x,y)));
end;
```

这里新建了一个Add函数，实现两个变量的相加。一个简单的程序就写好了，现在来生成map文件，单击Delphi菜单栏的"工程"→"选项"→"映像文件"命令，在弹出对话框中将默认选择的"关闭"改成"详细"，如图1所示。

然后编译工程，在工程目录下就会生成一个map后缀的文件。运行VMProtect V1.21，打开生成的Project1.exe（Project1.exe和Project1.

map需放在同一目录),在右键菜单中选择"添加地址",此时程序中所使用的和内部自定义的函数与过程就都自动列出来了,在列表中选择我们的Add函数来实现保护,如图2所示。

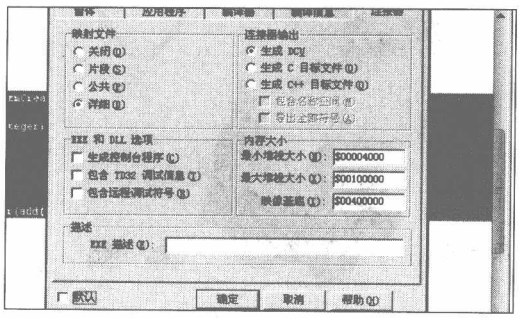


图1

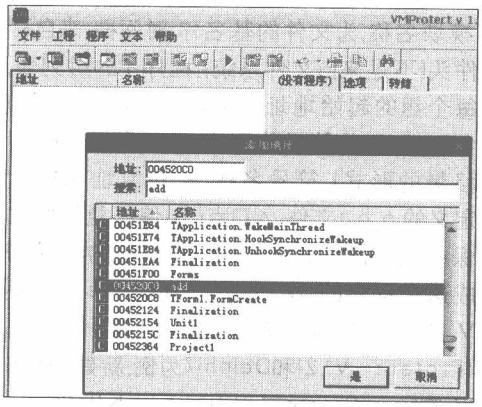


图2

再选择"工程"→"编辑"命令,或者按F9键,开始保护Add函数,完成后会生成Project1.vmp信息文件和保护后的文件Project1.vmp.exe,如图3所示。

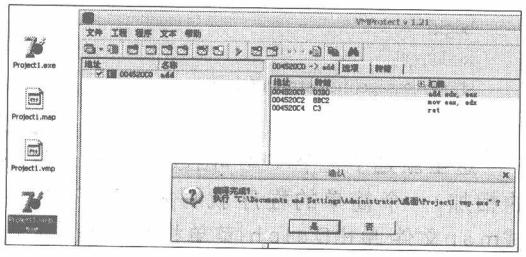


图3

使用map文件保护程序后,使用PEID查壳依然是Delphi,这是因为VMProtect在对程序函数进行保护时并不对程序的其他资源进行处理,但是多了一个区段,如图4所示。

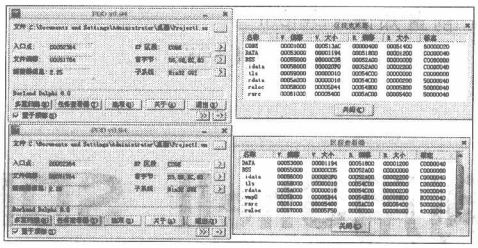


图4

接下来我们再说说语句标记法保护。先关闭Delphi的生成map文件的功能,依然以保护Add函数为目的,在Add函数begin开始的地方添加标记:

```
asm
db  $EB,$10,'VMProtect begin',0
end;
```

在Add函数end之前添加标记结束语句:

```
asm
db  $EB,$0E,'VMProtect end',0
end;
```

完整的代码如下:

```
function add(a,b:integer):integer;
begin
asm
db  $EB,$10,'VMProtect begin',0
end;
result:=a+b;
asm
db  $EB,$0E,'VMProtect end',0
end;
end;
```

完成后编译工程即可,将生成的Project1.exe用VMProtect打开,添加地址,程序会自动识别出在源码中添加的标识,选择要保护的标记段,如图5所示。按F9键保护标记中间的代码,生成的Project1.vmp.exe就是我们的保护后的文件。

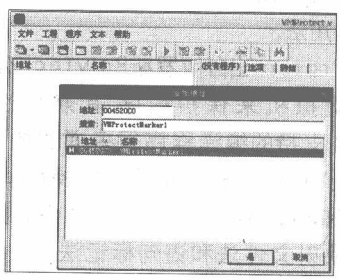
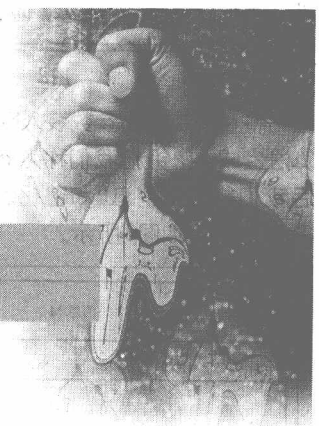


图5

两种方法就说完了,只要代码段或者函数及过程保护得当,是会给软件的破解带来很大麻烦的。这一点我们应用到木马的源码免杀上,加密那些被杀软重点关注的函数,能达到很有效的免杀效果。另外,Shell SDK保护技术与普通加壳结合起来,形成两层保护,这样就能大大减小程序被破解的可能,作为木马免杀效果也会相当不错!本文只讨论了VMProtect在Delphi中的应用,在其他编辑器如Visual C++等中应用如出一辙。如今开放SDK的Shell已经有不少了,如果利用多个SDK分别保护不同的代码,我想破解者一定会头疼的。

前置知识:无
关键词:破解、WEP、数据加密



WEP加密算法的实现原理与破解

文/图 邪公子(evilboy)

WEP(Wired Equivalent Privacy)的中文名为有线等效协议,是一种可选的链路层安全机制,用来提供访问控制、数据加密和安全性检验等。802.11定义了WEP算法对数据进行加密。

WEP最初设计的是40位密钥,后来WEP2的出现将密钥增加到104位。它们通常在市场上被称为64位加密和128位加密,真正的还是40位或104位加密,其余24位是初始化向量(IV)。

关注无线安全的朋友都知道,WEP加密是存在很严重安全问题的,已经存在很成熟的理论和破解工具。但由于国内整体上对该方面关注比较晚,所以导致很多朋友只会按照工具一步步破解,却根本不知道其原理,所以本文将详谈WEP的实现原理、存在的安全问题,以及流行的几种破解理论和详细实现。

WEP加密的实现原理

WEP所有加密都是以每个数据包为基础的,因此每个数据包基本上是一个单独发送的明文,我们将这个数据包称为M,下面详谈如何一步步实现加密。

第一步,我们需要计算消息M的校验值,以便对方接收后能够检查该包的完整性。校验值的计算使用了一个名为CRC32的32位循环冗余校验码函数实现。我们称该校验码为CS,因此CS=CRC32(M),这样就形成了以下包,我们称该包为明文P,如图1所示。

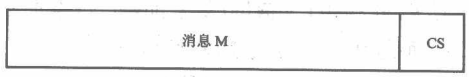


图1

小知识

CRC校验的基本思想是利用线性编码理论,在发送端根据要传送的K位二进制码序列,以一定的规则产生一个校验用的监督码(即CRC码)r位,并附在信息后边,构成一个新的二进制码序列数共(K+r)位,最后发送出去。在接收端,根据信息码和CRC码之间所遵循的规则进行检验,以确定传送中是否出错。

第二步,我们需要将明文包P进行加密,该处将运用RC4算法对明文包P进行加密。

RC4是1987年提出的加密算法,作为一种分组

密码体系,被广泛使用于计算机保密通信领域。它是一种相当简单的算法,该算法用一个种子值初始化,然后生成一个密钥流。该种子值由24位IV(确保最终产生密钥流不同)和40位或104位密钥组成,我们称为种子值Z,如图2所示。关于IV值,在一些旧版程序中简单地使用顺序值,而其他一些使用某种伪随机数生成程序。

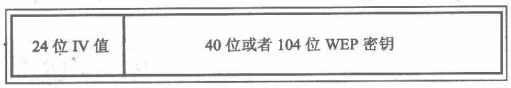


图2

将种子值作为参数装入RC4算法,将生成由RC4算法生成的密钥流,我们称为密钥流R,R=R C4(Z),然后我们将明文消息P和密钥流R进行XOR运算,将得到密文C。如图3所示。

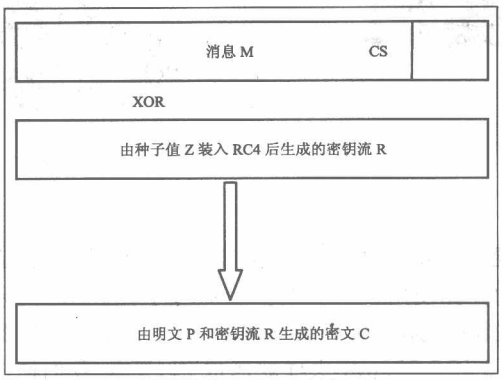


图3

最后将24位I V值(明文未加密I V)添加到密文C前面,就生成了整个加密包W。

解密过程正好与此过程相反。首先接收方从该加密包W中取出24位I V值,然后将I V和自己的密钥连接起来形成种子Z',如果发送方和接收方密钥相同,那么种子值也应该是相同的,将该种子值Z'装入RC4中运算形成密钥流R'。如果种子值相同,生成的密钥流也应该是相同的,然后将该密钥流同加密消息C进行X O R异或,这将产生原始的消息P,由明文M和CS校验值组成,接着,接收方使用相同的C R C32算法对明文M进行运算,生成C S',如果C S和C S'相同,那么数据包就会通过,否则,就会丢弃,原理如图4所示。

针对WEP的几种破解方式

1. 离线暴力攻击

针对任何的密码系统,暴力破解密码总是一种可能的方式,惟一需要考虑的是该种攻击是否切合实际。假如破解一个密码要一万年,那肯定是不切合实际的做法。

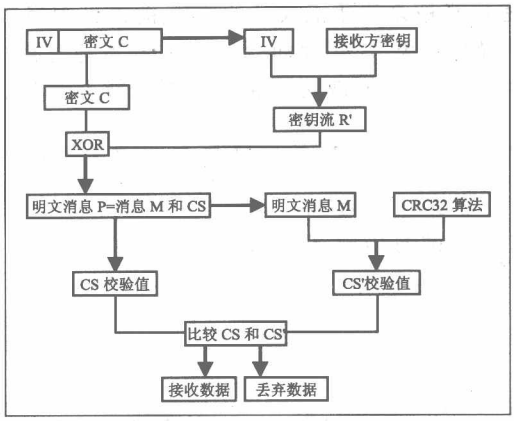


图4

针对W E P的离线暴力破解,原理就是我们捕获一些数据包,然后用我们所有可能的密钥同I V结合,产生种子值,将种子值代入R C4算法,产生密钥流,将密钥流同密文进行异或,获取明文(明文M和校验码C S),然后去不断对比C S值是否相同来确定是否是正确密钥。原理如图5所示。

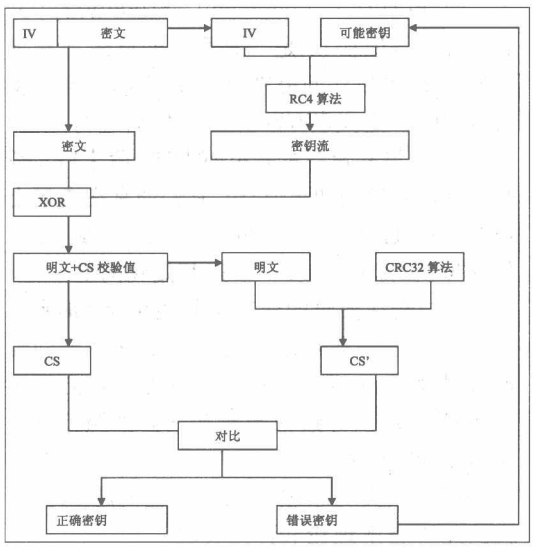


图5

该种攻击至少需要破译两个数据包,因为破译的单一数据包很可能有一个无效密钥,而其校验值CS可能是正确的。

然而,在假设每秒尝试10000次的情况下,暴力破解40位的字符空间大约需要3年时间。实际

上,现在处理器的速度可以达到每秒10000次,但即使是20000次每秒,整体上仍然是不确定的。

Tim Newsham提供了一种更有效的方法,该方法基于密码的密钥生成算法中的弱点。他的方法可以将40位的字符空间有效地缩减为21秒,在每秒10000次的情况下,大约仅需几秒或者几分。而针对104位WEP网络,暴力破解是不可行的。

2. 密钥流重用攻击

如果两个明文A和B运用同样的密钥流加密生成A'和B',那么将A'和B'进行异或,将消除密钥流,等于两个明文的异或,生成A XOR B=C,如果知道A或者B,用A或者B同C进行异或,将获得另一段明文。

IV是被用来进行阻止此类攻击的,没有IV,那么每个数据包将被使用同样的密钥流进行加密。但是,如果重用了相同的IV,两个数据包就会使用相同的密钥流加密。WEP使用中,IV由24位构成,这可以说不可避免将会重用IV,而且IV在数据包中是以明文方式存在的。按照统计理论,5000个数据包将会有一个密钥流重用。

所以,如果发现一个IV冲突之后,一些训练有素者即可猜测明文的结构,将两段密文进行异或,将获得两段明文的异或值,如果知道其中一个明文,将可以获得另一段明文。获得已知明文的方法之一是通过攻击者发送垃圾邮件。

3. 基于IV的解密字典表

密钥流的不同取决于IV的值,IV的值的可能是2的24次方分之一,如果为每个IV值存储1500字节的密钥流,整个表值需要24G B的存储空间。一旦创建了这样一个IV索引表,那么只需依IV查询其密钥流,然后异或,即可轻松解密任何加密包。原理如图6所示。

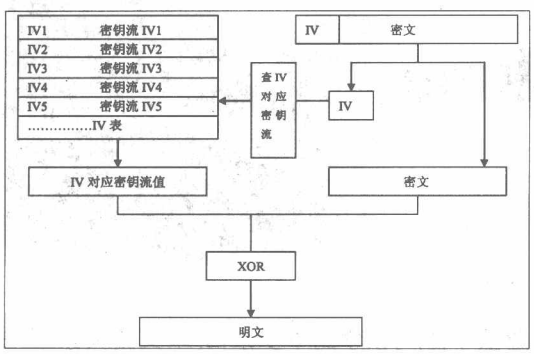


图6

4. IP重定向破解加密数据包(基于CRC32算法缺陷)

IP重定向攻击是一种比较巧妙的攻击方式,基于CRC32算法是一个线性不加密函数,所以使得修改数据包成为可能,这意味着我们修改数据包后算出的结果仍然会出现相同的校验值。

该攻击假设源IP地址和目的IP地址是已知的。例如假设源地址是192.168.2.57,目的地址是192.168.2.1,攻击者控制的地址为123.45.67.89,且想要将流量重定向到这个地址。这些IP地址在数据包中以二进制的高低位16位字形式存在,其转换相当简单:

```
Src IP=192.168.2.57
S高=192*256+168=50344
S低=2*256+57=569
Dst IP=192.168.2.1
D高=192*256+168=50344
D低=2*256+1=513
New IP=123.45.67.89
N高=123*256+45=31533
N低=67*256+89=17241
N高+N低-D高-D低=31533+17241-50344-513=-2083
```

"$N_{高}+N_{低}-D_{高}-D_{低}$"将会改变校验值,因此必须从数据包中其他某个地方减去这个值,因为源地址已知且无关紧要,所以源IP地址的低16位字是一个好目标:$S'=S_{低}-(N_{高}+N_{低}-D_{高}-D_{低})=569-(-2083)=2652$。

因此,新的源IP地址应当为192.168.10.92。可以使用相同的异或技巧在加密数据包中修改源IP地址,这时校验值将会匹配。当数据包发送到无线接入点,数据包将会被解密并且发送到我们指定的IP地址123.45.67.89,攻击者将接受到它。

5. FMS攻击

FMS攻击是由Scott Fluhrer、Itsik Mantin和Adi Shamir3位学者提出的,利用未加密的IV值及RC4设计上的缺陷来达到破解密钥的目的。

RC4算法由KSA密钥排列算法和PRGA伪随机生成算法组成。这两个算法都使用8×8的S盒,该盒恰好是一个256个数字的数组,每个数字都是惟一的且数字变化范围为0~255。

KSA算法的伪代码如下:

```
j=0;
for i=0 to 255
{
```

```
j=(j+S[i]+K[i]) mod 256;
swap S[i] and S[j];
}
```

K·S A算法将先用0～255的顺序值填充S盒数组,将该数组命名为S,然后用种子值填充数组K,如果不足,重复填充。K S A算法主要目的就是打散S盒的顺序(即S和K)。

PRGA算法伪代码如下:

```
i=(i+1) mod 256;
j=(j+S[i]) mod 256;
swap S[i] and S[j];
t=(S[i]+S[j]) mod 256;
Output the value of S[t];
```

输出的S[t]就是密钥流的第一个字节。

下面给出完整的RC4实现代码:

```
int RC4(int *IV,int *key){
int  K[256];
int  S[256];
int  seed[16];
int  i,j,k,t;
//seed=IV+key;
for(k=0;k<3;k++)
seed[k]=IV[k];
for(k=0;k<13;k++)
seed[k+3]=key[k];
//KSA
//初始化数组S和K
for(k=0;k<256;k++){
S[k]=k;
K[k]=seed[k%16];
}
j=0;
for(i=0;i<256;i++){ //KSA
j=(j+S[i]+K[i])%256;
t=S[i];S[i]=S[j];S[j]=t;
}
//PRGA
i=0;
j=0;
i=i+1;
j=j+S[i];
t=S[i];S[i]=S[j];S[j]=t;
k=(S[i]+S[j])%256;
return S[k];
}
```

基于802.11b数据包第一个字节总是0xAA,并且存在弱IV值,我们可以按照一定的算法,如果收集足够的弱IV,就可以推出密钥。该算法被广泛应用到现在大多的无线破解工具,如airsnort和aircrack-ng等,具体对此算法感兴趣的朋友,我提供了一个源代码,详细揭示该破解算法的实现。

小结

无线WEP破解在2001年就已经变得可行,它当时的设计更多地没有考虑安全因素,随着WPA及WPA2的推出,WEP可以说应该退出无线舞台了,因为它确实是可以被真正破解的。所以,建议现在还在应用无线WEP加密的,马上升级为WPA。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:汇编

关键词:破解、算法、Anti-debug

Anti-debug Crackme 算法分析

文/图 木木

对于想学习逆向的朋友来说,www.crackmes.de确实是一个不错的网站,里面的资源很丰富,可以学到不少东西。本文给大家带来的Crackme的注册算法与注册表有关,其中还有

Anti-debug,个人觉得实现得还是蛮精彩的,有必要与大家分享一下。

算法分析

算法的实现是与注册表相关的,我们先在操作注册表的函数上下断,用OllyDbg载入,单击右键→搜索→当前模块中的名称,看看它都调用了哪些函数,如图1所示。

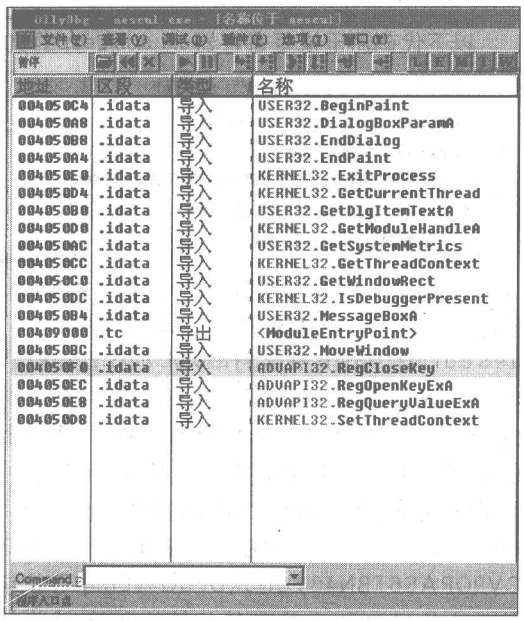


图1

单击鼠标右键→切换断点,在函数上下断。成功断下后,取消断点,执行到用户代码,会得到以下代码:

```
00403286  push aescul.00404B0E
//pHandle = aescul.00404B0E
0040328B  push 1
// |Access = KEY_QUERY_VALUE
0040328D  push 0    //|Reserved = 0
0040328F  push aescul.00404000
//|Subkey = "Software\Microsoft\
//Windows\CurrentVersion"
00403294  push 80000002
// |hKey = HKEY_LOCAL_MACHINE
00403299  call <jmp.&ADVAPI32.RegOpenKeyExA>
0040329E  push aescul.00404B12
//pBufSize = aescul.00404B12
004032A3  push aescul.004043BC
// |Buffer = aescul.004043BC
004032A8  push 0  // |pValueType = NULL
004032AA  push 0    //|Reserved = NULL
004032AC  push aescul.00404033
//|ValueName = "ProductId"
004032B1  push dword ptr ds:[404B0E]
// |hKey = 194
004032B7  call <jmp.&ADVAPI32.RegQueryValueExA>
```

程序通过读取注册表HKEY_LOCAL_MACHINE Software\Microsoft\Windows\CurrentVersion\ ProductId的键值获得了Windows的注册码,向上翻阅,得到如下代码:

```
00403224  push 40// \Count = 40 (64.)
00403226  push aescul.00404IBA
// |Buffer = aescul.00404IBA
0040322B  push 3E8// |ControlID = 3E8 (1000.)
00403230  push dword ptr ss:[ebp+8]
// |hWnd = 0132029E ('Y2K +HCU Strainer-
// (c) Aescu...',class='#32770')
00403233  call <jmp.&USER32.GetDlgItemTextA>
00403255  push 40// \Count = 40 (64.)
00403257  push aescul.004042BA
// |Buffer = aescul.004042BA
0040325C  push 3E9// |ControlID = 3E9 (1001.)
00403261  push dword ptr ss:[ebp+8]
// |hWnd = 0132029E ('Y2K +HCU Strainer-
// (c) Aescu...',class='#32770')
00403264  call <jmp.&USER32.GetDlgItemTextA>
```

程序在这里调用函数GetDlgItemTextA,得到我们输入的注册名和注册码。算法的精髓部分从下面代码处开始:

```
00403061  push ebp
00403062  push edi
00403063  push esi
```

将ebp,edi,esi中的值压入堆栈。这里ebp储存的是循环的总次数,edi和esi的初始值为0,esi在这里充当循环计数器。

```
00403064  mov ebp,aescul.004044C0
//取了一个字符串 ASCII "0I5LZ7G123
//RXCV9OPAS6TBN48YUHJKDF0QWEM"
00403069  mov ebx,aescul.004043BA
//取得到的Windows注册码 ASCII
//"WS76481-640-8834005-23375"
0040306E  mov al,byte ptr ds:[ebx+esi]
//取Windows注册码的第一个字符的十六进制
00403071  sar eax,4
//右移4位,原来eax中的值是357,右移后为35
00403074  and eax,0F
//这里的与运算相当于取得算数字的个位值,35经过
```

```
//运算后为5
00403077  call aescul.0040313B//  步入这个call
```

之后继续得到如下代码：

```
0040313B  mov dword ptr ds:[4044EE],esi
//将循环计数器的值保存到内存ds:[4044EE]
00403141  mov edx,dword ptr ds:[4044EA]
//edx充当这个call的循环计数器,初始edx为0
00403147  mov ecx,dword ptr ds:[4044E6]
//将循环总次数送入ecx
0040314D  cmp edx,ecx
//比较循环是否相等
0040314F  jb short aescul.00403153
//小于则跳转
00403151  xor edx,edx
00403153 movsx esi,byte ptr ss:[ebp+edx]
//取字符串"0I5LZ7GI23RXCV9OPAS6TBN48YU
//HJKDF0QWEM"第一个字符的十六进制
00403158  and esi,8000000F
//这里的与运算相当于取待算数字的个位值
0040315E  jns short aescul.00403165 //非负则跳转
00403160  dec esi
00403161  or esi,FFFFFFF0
00403164  inc esi
00403165  cmp esi,eax
//判断从两个字符串中取得的字符的十六进制的个位
//是否相等
00403167  je short aescul.00403172 //相等则跳转
00403169  inc edx
0040316A  cmp edx,ecx
0040316C  jl short aescul.00403153
0040316E  xor edx,edx
00403170  jmp short aescul.00403153
00403172  mov dword ptr ds:[4044EA],edx
//将循环计数器的值保存到内存ds:[4044EA]
00403178  mov esi,dword ptr ds:[4044EE]
//将寄存器esi清空
0040317E  movsx eax,byte ptr ds:[edx+ebp]
//将找出的字符送入寄存器eax
00403182  inc edx //将循环计数器增1
00403183  retn    //返回
```

这个call的作用是在字符串0I5L Z7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中找与Windows注册码中的字符的十六进制的个位数相等的字符。分析完这个call后，我们继续往下分析。

```
0040307C  mov byte ptr ds:[edi],al
//将找到的字符保存到从00404502开始的内存中
0040307E  mov cl,byte ptr ds:[ebx+esi]
//取Windows注册码的第一个字符的十六进制
00403081  and ecx,0F
//取待算数字的个位值,这次是57,运算后为7
00403084  mov eax,ecx //送入寄存器eax
00403086 call aescul.0040313B
//在字符串"0I5LZ7GI23RXCV9OPAS6TBN48YU
//HJKDF0QWEM"中找十六进制的个位是7的字符
0040308B  mov byte ptr ds:[edi+1],al
//将找到的字符保存到内存中
0040308E  pop esi
0040308F  pop edi
00403090  pop ebp
```

以下代码用于恢复寄存器esi、edi和ebp的值。

```
00403091  inc esi //循环计数器增1
00403092  add edi,2 //edi增2
00403095  cmp ebp,esi
//比较循环计数器的值是否与循环总次数相等
00403097  jnz short aescul.00403061
//不等则进入下一轮循环
```

总结一下，算法先取得Windows注册码中的字符的十六进制数的10位，然后调用call aescul.0040313B，在字符串0I5LZ7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中寻找字符的十六进制数个位与其相等的字符，并将找到的字符保存到内存。之后算法又取Windows注册码中的字符的十六进制数的个位，调用call aescul.0040313B在字符串0I5LZ7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中寻找字符的十六进制数个位与其相等的字符，并将找到的字符保存到内存。这些从字符串0I5L Z7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中找到的字符组成了真正的注册码。

接下来的算法就是与我们输入的假注册码分段比较，决定程序是跳向注册成功还是注册失败，代码如下：

```
00403099  xor esi,esi      //清空esi
0040309B  mov eax,dword ptr ds:[esi+4042BA]
//取输入的注册码的前四个字符
004030A1  mov ebx,dword ptr ds:[esi+404502]
//取生成的真注册码的前四个字符
004030A7  cmp eax,ebx   //比较是否相等
004030A9  jnz short aescul.004030F0
//不相等则跳向注册失败
004030AB  add esi,4       //esi增4
004030AE  mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030B4  mov ebx,dword ptr ds:[esi+404502]
//接着向下取
```

```
004030BA   cmp eax,ebx   //比较是否相等
004030BC   jnz short aescul.004030F0
//不相等则跳向注册失败
004030BE   add esi,4 //esi增4
004030C1   mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030C7   mov ebx,dword ptr ds:[esi+404502]
//接着向下取
004030CD   cmp eax,ebx //比较是否相等
004030CF   jnz short aescul.004030F0
//不相等则跳向注册失败
004030D1   add esi,4 //esi增4
004030D4   mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030DA   mov ebx,dword ptr ds:[esi+404502]
//接着向下取
004030E0   cmp eax,ebx //比较是否相等
004030E2   jnz short aescul.004030F0
//不相等则跳向注册失败
```

到这里,算法部分的分析就告一段落了,接下来我们看看Anti-debug是如何操作的。

Anti-debug分析

Anti-debug部分是设置在上面算法部分之上的,从地址00403465处开始,代码如下:

```
00403038   xor eax,eax //清空eax
0040303A   push aescul.00403393
//offset of SEH handler
0040303F   push dword ptr fs:[eax]
00403042   mov dword ptr fs:[eax],esp
//install handler
```

这里设置了异常处理,一旦发生异常,程序就会调用地址00403393处的代码来处理异常。

```
00403045   pushfd //将EFL寄存器的值保存到堆栈
00403046   pushfd //将EFL寄存器的值保存到堆栈
00403047   pop eax //将EFL寄存器的值弹出到eax
00403048   or eax,100
//进行异或运算,原值为246,异或后为346
0040304D   push eax //将异或得到的值保存到堆栈
0040304E   popfd //恢复EFL寄存器的值
```

这里的代码通过修改EFL寄存器中的值,设置了陷阱标志。

```
0040304F   nop
//因为设置了陷阱标志,这里的nop将发送一个
//EXCEPTION_SINGLE_STEP(单步异常),程序调
//用前面设置的异常处理,前往地址00403393
00403393   push ebp
00403394   mov ebp,esp
00403396   call <jmp.&KERNEL32.GetCurrentThread>
//调用函数GetCurrentThread
0040339B   mov dword ptr ds:[40418F],eax
//保存得到的当前线程的句柄到内存ds:[40418F]
```

这里通过调用函数GetCurrentThread得到了当前线程的句柄。

```
004033A0   mov dword ptr ds:[4040BB],7C
004033AA   push aescul.004040BB
004033AF   push dword ptr ds:[40418F]
004033B5   call <jmp.&KERNEL32.GetThreadContext>
//调用函数GetThreadContext
```

这里通过调用函数GetThreadContext得到了调试寄存器。

```
004033BA   mov dword ptr ds:[4040BF],eax //清零
004033BF   mov dword ptr ds:[4040C3],eax //清零
004033C4   mov dword ptr ds:[4040C7],eax //清零
004033C9   mov dword ptr ds:[4040CB],eax //清零
004033CE   mov dword ptr ds:[4040D3],24FF
004033D8   mov dword ptr ds:[4040BB],18
004033E2   push aescul.004040BB
004033E7   push dword ptr ds:[40418F]
004033ED   call <jmp.&KERNEL32.SetThreadContext>
//调用函数SetThreadContext
```

这里先将调试寄存器的值清零,然后调用函数SetThreadContext更新调试器寄存器。

```
004033F2   mov eax,dword ptr ss:[ebp+10]
004033F5   mov edi,eax
004033F7   mov ebx,dword ptr ds:[edi+B8]
004033FD   cmp byte ptr ds:[ebx],0CC
00403400   je short aescul.0040341E
```

这里会读取从地址00403050处开始的代码(算法部分从这个地址开始),判断有没有0CC(Int3),也就是检查是不是由int3引发了异常,变相地检查了我们是否在那里下断,若是就会跳转到一个读取内存错误的异常处。

```
00403402   cmp byte ptr ds:[ebx],9D
00403405   je short aescul.00403411
```

判断有没有9D(popfd),即检查是不是由于堆栈的不平衡而引发的异常,若是则跳到00403411。

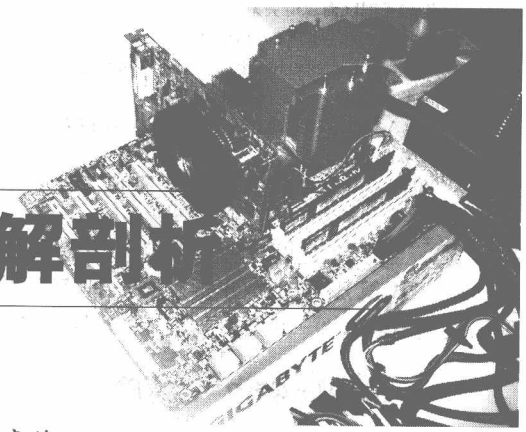
```
00403407   or dword ptr ds:[edi+C0],100
00403411   mov eax,dword ptr ss:[ebp+8]
00403414   cmp dword ptr ds:[eax],80000003
//这里的异常代码80000003就相当于int3,这里继续
//检查异常是不是由int3引起的
```

```
0040341A    je short aescul.0040341E
//跳转到一个读取内存错误的异常处
```

程序通过修改EFL寄存器的值设置陷阱标志,nop引发单步异常,在异常处理中检测调试。先调用GetCurrentThread、GetThreadContext和SetThreadContext清空调试寄存器,然后检测在算法代码部分是否有断点,若有就引发内存异常。

到此,我们就将这个Anti-debug Crackme分析完了。因为自己也是刚刚开始学习破解,虽然部分内容不是独立完成的,不过还是从中学到了不少新东西,希望大家也能从中学有所得。

前置知 识:无
关键词: Web、三国、外挂破解



游戏外挂软件破解剖析

文/图 孙嘉骏

傲视天地是一款由上海鑫韵网络科技有限公司耗时2年时间精心打造的新一代战争策略类网页游戏。在玩游戏的过程中不免重复地建筑升级、武将升级,很麻烦,于是网上找了许久,总算找到了游戏的一款外挂软件来辅助使用,可没想到,我仅仅使用了10分钟左右,软件就弹出了"使用时间过期,请联系作者"的对话框,然后外挂软件就关闭了。哎,此时可谓相当不快啊,但转眼一看这种软件对反破解设计应该不是很复杂,于是就尝试进行了一下破解,记下破解思路与大家分享。

先看看软件,运行图如1所示。

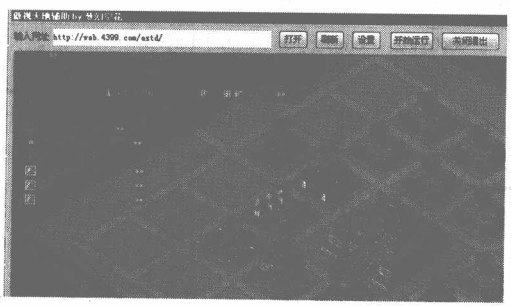


图1

我们进行游戏菜单设置完以后,单击"开始运行",运行过一会就会弹出提示过期对话框,如图2所示。

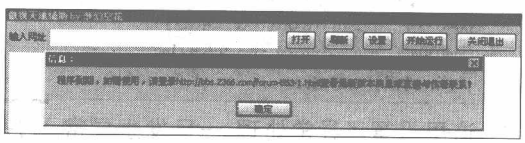


图2

既然是单击"开始运行"后程序弹出一个对话框,那么我们就应该可以找到一个断点,使程序断在弹出对话框之前,然后再对是否弹出对话框的条件进行分析,如图2所报,程序使用到期,那么就应该和系统的时间有直接的关系了。我们先运行程序,再用OllyDbg加载程序,如我所料,程序还是比较简单,并没有加反调试之类的东西(省去了不少麻烦)。因为上面提示对话框弹出时会调用到CreateWindow函数(或者CreateWindowExCreate WindowExA,都是一样的,实则上是CreateWindow->CreateWindowEx),于是我们在此类函数上下一个断点bp CreateWindow(有的OllyDbg