

文章编号:1001-9081(2011)11-2975-04

doi:10.3724/SP.J.1087.2011.02975

# 基于逆向技术的恶意程序分析方法

罗文华

(中国刑事警察学院 计算机犯罪侦查系, 沈阳 110854)

(luowenhua770404@126.com)

**摘要:** 逆向分析是恶意程序分析的常用方法之一, 在揭示恶意程序意图及行为方面发挥着其他方法无法比拟的作用。着重从启动函数、函数参数传递、数据结构、控制语句、Windows API 等方面归纳总结恶意程序反汇编代码一般规律, 并结合一起利用恶意程序窃取 QQ 账号与密码的真实案例说明快速准确定位关键信息的具体方法。

**关键词:** 逆向技术; 启动函数; 参数传递; 数据结构; 控制语句; Windows API

中图分类号: TP309.2 文献标志码:A

## Malware analysis method based on reverse technology

LUO Wen-hua

(Department of Computer Crime Investigation, China Criminal Police University, Liaoning Shenyang 110854)

**Abstract:** Reverse analysis is the most common method in analyzing malware. The reverse analysis process is an advanced and efficient method that exposes the intention and processes of malware. The focus of this paper was to show the general patterns ascertained using reverse analysis applied to the aspects of start function, parameter transfer of function, data structure, control statement and Windows API. A case study of malware, used to obtain account information, login names, and passwords for the popular Chinese social networking program "QQ", was presented to illustrate how the reverse analysis quickly and accurately locates key information used to determine general patterns.

**Key words:** reverse technology; start function; parameter transfer; data structure; control statement; Windows API

## 0 引言

进行电脑安全事件调查时, 极有可能会遇到黑客所部署的恶意程序, 此时就有必要鉴别出这些文件, 并对其进行分析。常规方法主要是通过激活恶意程序样本, 抓取分析其网络传输信息, 获知被窃取的信息内容以及信息传输的目的地址; 或是通过架设系统监控工具, 收集恶意程序引发的相关应用程序及其环境变化的信息, 从而判断其对操作系统所造成的影响<sup>[1]</sup>。但在实践中发现, 多数情况下由于运行条件已不满足或是其内部所设置的运行有效期限已经过期, 恶意程序样本往往无法被激活, 或是改变自身行为, 掩盖恶意程序特征, 应用上述方法往往无法发现有价值的信息。此种情况下, 便有必要针对恶意程序进行逆向分析鉴定, 从而发现相关证据或线索。

针对恶意程序进行逆向分析, 是指将可执行恶意程序反汇编, 通过反汇编代码来理解其代码功能, 如各接口的数据结构等, 逆向分析原程序思路, 全面或重点掌握恶意程序行为, 从中分析出其他方法无法发现的证据或线索<sup>[2]</sup>。本文结合恶意程序取证工作实际, 针对查壳、脱壳、断点设置、程序跟踪、信息获取等环节所涉及的关键技术, 归纳总结了逆向取证分析的一般性规律方法, 并结合恶意程序实例, 说明了讨论方法的有效性。

## 1 针对恶意程序的逆向分析方法

### 1.1 启动函数

编写软件时, 通常需要在源代码里实现 WinMain 函数,

即主函数。但 Windows 程序在运行时并不是从 WinMain 函数开始, 而是执行启动函数相关代码, 这段代码由编译器自动生成, 启动函数被执行后再调用 WinMain 函数<sup>[3]</sup>。逆向分析时, 可以通过启动函数判断恶意程序实际的编写语言。实践中, 分析人员一般使用工具软件判断具体编写语言, 如 PEiD。然而工具软件有时也会被嫌疑人设计的陷阱所欺骗, 如被 PEiD 判断为由“Microsoft Visual C++ 6.0 [ Overlay ]”编写的程序, 可能是由 Delphi 编写; 被其判断为“Not a valid PE file”的文件, 却极可能是经过花指令伪装的可执行程序。因此通过启动函数判断恶意程序真正的编写语言十分必要。

使用 OllyDBG 载入恶意程序样本, 由 Visual C++ 编写的程序, 其汇编窗口的反汇编指令序列中通常出现有“push-1”(或“push FFFFFFFF”)、“call dword ptr [ < &kernel32.GetVersion > ]”(获取 Windows 系统版本信息)、“call dword ptr [ < &kernel32.GetCommandLineA > ]”(指向进程命令行的指针)、“call dword ptr [ < &kernel32.GetStartupInfoA > ]”(获取进程启动信息)等字样。另外, 如果该程序的运行模块窗口中出现有 MFC 字样, 也说明其由 Visual C++ 编写。

Delphi 语言编写的恶意程序, 其反汇编指令规则整齐, 通常是 call 语句与 mov 语句间隔出现; Borland C 程序的启动函数中则会出现有两个相邻的 JMP 语句; VB 程序的反汇编指令则很不规则, 起始部分会出现有数量较多的 JMP 语句; 汇编语言编写的语句一般比较规整, 通常由多个 push、call、mov 等语句组成。恶意程序讲求效率, 程序体积也比较小巧, 以上五种语言是恶意程序最常选择的编写语言。了解不同语言启

收稿日期:2011-05-23;修回日期:2011-06-27。 基金项目:公安部应用创新计划项目(2011YYCXXJXY121)。

作者简介:罗文华(1977-),男,辽宁沈阳人,副教授,主要研究方向:计算机犯罪侦查、电子数据取证。

动函数反汇编代码特征,可以快速准确判断恶意程序真实的编写语言,从而有的放矢地寻找突破口。

## 1.2 函数参数的传递

恶意程序自身所包含的函数,其所传递参数往往是用户名、用户口令、邮箱地址、网页地址等重要信息。本节重点讨论如何在反汇编指令中快速定位用于函数参数传递的语句,进而获取参数的具体信息。下面以 stdcall 约定为例,说明调用函数 Trojan(Par1,Par2)时堆栈的变化过程。首先依次将 Par2 与 Par1 入栈;接下来调用子程序,将原 EBP 指针保存起来,以保护现场;然后设置新的 EBP 指针,指向 ESP(即栈顶);然后通过距离 EBP 的偏移地址调用参数 2 和 1;同时通过改变栈顶指针为局部变量分配存放空间。具体代码与注释如下所示:

Push par2	参数 2 入栈
Push par1	参数 1 入栈
Call Trojan	调用子程序
(	
Push ebp	把原 EBP 的指针保存起来,保护现场
Mov ebp, esp	设置新的 EBP 指针,指向 ESP(即栈顶)
Mov eax, dword ptr[ebp + 0c]	调用参数 2
Mov ebx, dword ptr[ebp + 08]	调用参数 1
Sub esp, 8	esp - 8, 局部变量的存放空间
...	
)	

分析得出,[ebp + 08] 所存放的是第一个参数,[ebp + 0C] 存放的是第二个参数,依此类推。而函数内部的局部变量则使用[ebp - 04]、[ebp - 08]进行读写。若编译器按优化方式编译程序,则为节省扩展基址指针寄存器(Extended Base Pointer,EBP),会直接通过扩展栈顶指针寄存器(Extended Stack Pointer,ESP)对参数寻址,此时[esp + 04] 与[esp + 08] 存放的分别是参数 1 与参数 2。而对于函数的返回值,在利用 return 操作符实现返回的情况下,一般使用 eax 与 edx 寄存器存放返回值;或使用指针实现返回。

## 1.3 数据结构

了解典型数据结构的反汇编代码,对于分析恶意程序行为会起到非常重要的帮助作用<sup>[4]</sup>。在 1.2 节中已讨论过对局部变量的存取方式,因此本节重点研究全局变量、数组与链表等典型数据结构的存取。全局变量作用于整个程序,存放在内存的某一块区域中,一般存放在数据区块(. data),因此通常使用固定的内存地址来进行存取。反汇编代码中如多次出现某固定内存地址时,该地址存放的极有可能为全局变量,需加以特殊关注。

图 1 所示为数组实例语句的反汇编代码。分析可知,编译器通过“xor eax,eax”语句将 eax 置 0, 使用“mov edi,dword ptr [eax + 407030]”与“add eax,4”两条语句循环实现依次访问数组中各元素的目的;其中“cmp eax,0C”是循环控制语句,以保证只访问数组中指定数目元素。因此,如果反汇编代码中出现寄存器与内存地址相加进行访问的语句,同时配以比较与跳转语句,便极有可能是数组访问语句。

链表也是一种广泛使用的内存管理数据方法。从逆向角度来看,链表与数组最大的不同在于链表中数据项分散存放

于内存中,并且每个数据项除了需要包含有效载荷,还要包含有指向下一个数据项的指针(双向链表中甚至还需要有指向前面一个数据项的指针)。因此,在使用链表的恶意程序反汇编代码中,通常获取有效载荷信息与获取指针信息的语句会相邻出现,如“mov eax,dword ptr[esi + 68]; mov ecx,dword ptr[esi + 64]”语句即是将有效载荷与指针信息分别送 eax 与 ecx 寄存器。

```

83EC 0C      sub    esp, 0C
33C9         xor    ecx, ecx
33C0         xor    eax, eax
56             push   esi
57             push   edi
8B88 30704001 mov    edi, dword ptr [eax+407030]
83C0 04       add    eax, 4
03CF         add    ecx, edi
83F8 0C       cmp    eax, 0C

```

图 1 数组示例语句反汇编代码

## 1.4 控制语句

分析控制语句,进而识别关键跳转或关键调用是恶意程序行为分析的一个重要方面。对于 IF-THEN-ELSE 语句,其反汇编语句一般使用 cmp,fcmp,fcom 等命令进行比较,然后使用 jz 或 jnz 命令实现跳转;在某些情况下编译器还可能使用 test 或 or 之类的较短逻辑指令替换 cmp 指令,如执行“test eax, eax”语句(如 eax 为零,则 ZF 置为 1,否则 ZF 置为 0),然后再根据 ZF 的具体值进行跳转。

而对于 SWITCH-CASE 语句,其编译后实质是多个 IF-THEN 语句的嵌套组合。在编译器未优化的情况下,反汇编代码中会出现多个 cmp 命令与 je 等跳转指令的组合;而在编译器优化情况下,一般不使用开销较大的 cmp 命令,而是使用数学运算命令处理后,再进行跳转。需要指出的是,一些编译器在优化时,使用数学技巧把源代码中的逻辑分支语句转化成算术操作,以消除或减少程序中出现的条件转移指令,进而提高 CPU 的性能,此时的控制语句无明显特征可供识别(有时甚至没有跳转指令),这种情况下需结合上下文全面分析后才可能做出准确判断<sup>[5]</sup>。

## 1.5 Windows API

分析恶意程序常用方法之一就是在调试器中设置断点,使恶意程序运行到断点后,调试器将控制权交由工作人员继续分析。设置正确恰当的断点对于快速准确分析恶意程序非常重要,好的断点设置可以使分析人员迅速找到关键程序段,而不恰当的断点则会对分析工作造成不必要的精力消耗,有的甚至根本起不到拦截程序运行的作用<sup>[6]</sup>。执行恶意功能的程序往往会调用特殊 API 函数或是自行编制的函数以完成相应功能,个别情况下还可能会有相应提示信息出现。本节即从 API 函数入手讨论恶意程序的断点设置。

Windows API 包含有数百个供应用程序调用的函数,这些函数执行那些必须与操作系统相关的操作,如内存分配、屏幕输出和创建窗口等,用户程序通过调用 API 接口同 Windows 打交道,无论什么样的应用程序,其底层最终都是通过调用各种 API 函数来实现各种功能的。恶意程序若要完成其功能(如生成文件、修改注册表、信息传输等),也必须通过 API 函数完成<sup>[7]</sup>,表 1 所示即为常用的特定功能与 Windows API 函数对应关系。

表 1 特定功能与 Windows API 函数的对应关系

特定功能	Windows API 函数名称
字符串操作	GetDlgItemTextA( W ), GetDlgItemInt GetWindowTextA( W ), GetWindowWord
对话框操作	MessageBeep MessageBoxA( W ), MessageBoxExA( W ), DialogBoxParamA( W )
	CreateWindowExA( W ), ShowWindow UpdateWindow
进程(线程)操作	CreateRemoteThread, CreateThread LoadLibrary, NtCreateThread GetWindowThreadProcessId, GetProcAddress, CreateProcess LdrLoadDll LdrGetProcedureAddress, EnumProcessModules() GetProcAddress(), CreateProcess()
网络传输	FtpGetFile, FtpPutFile, InternetOpen, InternetConnect, HttpOpenRequest InternetOpenUrl, URLDownloadToFile, HttpSendRequest, HttpQueryInfo
时间处理	GetLocalTime, GetFileTime, GetSystemtime
注册表操作	RegOpenKeyA( W ), RegOpenKeyExA( W ), RegCreateKeyA( W ) RegCreateKeyExA( W ), RegDeleteKeyA( W ), RegDeleteValueA( W ) RegQueryValueA( W ), RegQueryValueExA( W ), RegSetValueA( W ), RegSetValueExA( W )
文件操作	OpenFile ReadFile, WriteFile, CreateFileA, SetFilePointer, GetSystemDirectory

## 2 实际应用

### 2.1 案例背景

目前,由于 QQ 账号所涉及的 Q 币与游戏装备等虚拟财产可以转化为现实世界中的货币,因此偷盗 QQ 账号及密码的现象日益猖獗,情节严重者甚至触犯了相关法律。“QQ 密码大划拉”即是一款用于偷盗 QQ 账号及密码的恶意程序,该软件属于木马生成器范畴,能够根据使用者的设置(如收信邮箱、发信邮箱、发信邮箱密码等)生成相应木马(配置界面如图 2 所示),使用者可以利用生成的木马来盗取他人 QQ 账号和密码。但该生成器本身就是一款能够偷盗 QQ 账号及密码的木马,使用者在生成自己盗号木马的同时,其自身的账号与密码也极可能被该生成器窃取。本节即以木马生成器“QQ 密码大划拉”及其默认生成的木马 QQ\_DYP 为例,说明逆向分析在电子数据取证中的实际应用。



图 2 “QQ 密码大划拉”木马生成器配置界面

### 3.2 针对木马生成器“QQ 密码大划拉”的分析

#### 3.2.1 查壳与脱壳

使用 PEiD 对木马生成器“QQ 密码大划拉”查壳。查壳结果表明该软件未加壳,由 Visual C++ 语言编写;使用 OllyDBG 加载该软件,却提示该软件为“self-extracting 或 self-modifying”文件,其启动函数中也并未发现 2.1 节所述 Visual C++ 启动函数典型特征,由此怀疑该软件使用了伪装壳。

利用 PEiD 的“Deep Scan”功能查得该恶意程序系使用 UPX 壳。直接利用 PEiD 自带插件“PEiD Generic Unpacker”对其进行脱壳。针对脱壳后的程序执行查壳操作,“Deep Scan”显示该程序由“Borland Delphi 6.0-7.0”编写,其启动函数中 call 语句与 mov 语句间隔出现(如图 3 所示),满足 Delphi 程序

启动函数的典型特征,从而验证了查壳结果的正确。

```

EB 00408FFF EB33 60405C68
A1 64114600 mov eax, dword ptr [461164]
BB88 mov ebx, dword ptr [eax]
EB 840EFFFF EB33 00450040
BB80 mov ecx, dword ptr [460040]
A1 64114600 mov eax, dword ptr [461164]
BB88 mov ebx, dword ptr [eax]
BB81 mov edx, dword ptr [450BD4]
EB 840EFFFF EB33 0045029C
A1 64114600 mov eax, dword ptr [461164]
BB80 mov ebx, dword ptr [eax]

```

图 3 脱壳后 QQ 密码大划拉的启动函数反汇编代码

#### 2.2.2 通过逆向分析发现隐秘的网页地址

脱壳成功后,使用 OllyDBG 载入,对恶意程序进行逆向分析。在 OllyDBG 中运行该程序,主程序窗口出现后,点击“生成木马”按钮,程序会弹出保存窗口以供选择生成木马的存放位置;点击 OllyDBG 工具栏“暂停”按钮,执行 ALT + K 操作打开函数窗口,查询得到保存窗口对应的 API 函数 DialogBoxIndirectParamW(图 4),可在此处设置断点,以追踪被窃信息的发送地址。

```

77D19418 includes nt!NtFastSystemCall0
77D27700 USER32.WaitMessage
77D290C4 USER32.77D29758
77D29400 USER32.77D29400
77D29400 USER32.77D29400
7632355F 763232_81DialogBoxIndirectParamW
76329008 fncall 76329000
00150659 g�opsize = 00150659
001009C0 59pop = 001009C0 “qq密码大划拉2”, class=“TApplication”
76322615: 919999 76322615: cond1g32.76322615
0012E39C 779999 ~ 0012E39C
763404B7: cond1g32.7632336F
76323394: cond1g32.763406C8
763231B7: cond1g32.76323295
76387C22: 7 cond1g32.76326668

```

图 4 保存窗口对应的 API 函数

在关键字符串信息上设置断点,也可达到调查取证的目的。载入恶意程序后,点击“插件”→“Ultra 字符串参考”→“查找 ASCII”,得到该程序的字符串信息。从字符串信息中发现有“QQ2009\_Hooker\_Head”字样出现,其实际为该生成器设计者编制的实现窃取功能的函数名称,故可在此设置断点。设置断点后,运行该恶意程序,在弹出的配置界面中随意设置“收信邮箱”、“发信邮箱”、“发信邮箱密码”等信息;然后点击“生成木马”按钮,在弹出的保存窗口中选择木马存放位置,然后点击“保存”按钮;由于事先已设置断点,木马并未生成,而是自动中断在了“QQ2009\_Hooker\_Head”的位置上(图 5)。

恶意程序中断在断点处(0045E310),点击 F8 单步运行以分析程序。当到达地址 0045E339 时,堆栈窗口中出现“SS: [0012F620] = 009D3724, ( ASCII " http://www. XXX. com /QQ456/XXX.asp" )”字样(图 6),此即该生成器窃取 QQ 账号及密码后发往的网页地址。分析人员获得该隐秘地址后,可

通过调查相关网站获取犯罪分子用来访问的 IP 地址,进而找到其真实藏身之所。

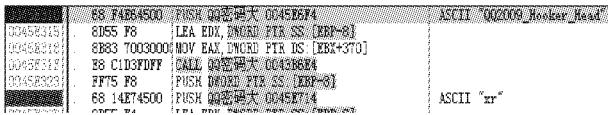


图 5 恶意程序自动中断在事先设置的断点上

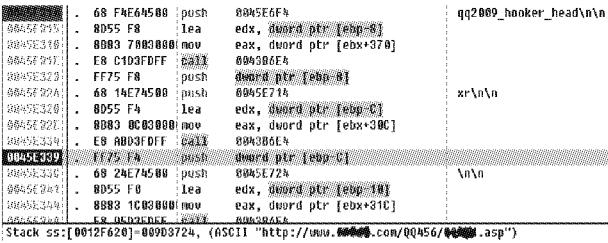


图 6 堆栈窗口中出现被窃信息发往的网页地址

### 2.3 针对生成木马 QQ\_DYP 的逆向分析

“QQ 密码大划拉”所生成木马的默认名称为“QQ\_DYP”,该木马能够根据配置的“收信邮箱”、“发信邮箱”、“发信邮箱密码”等信息,将窃取的 QQ 账号与密码发送到指定地址。本节重点针对 QQ\_DYP 的反查杀技术予以分析。

将 QQ\_DYP 脱壳后,使用 OllyDbg 载入。其反汇编代码中出现有如图 7 所示的语句。分析发现,语句中出现的“4B”、“65”、“72”、“6E”、“65”、“6C”、“33”、“32”、“2E”、“64”、“6C”、“6C”分别是字符“K”、“e”、“r”、“n”、“e”、“l”、“3”、“2”、“.”、“d”、“l”、“1”的 ASCII 编码,该木马通过对字符串“Kernel32.dll”进行拆解,以避免整体字符串出现导致被查杀。

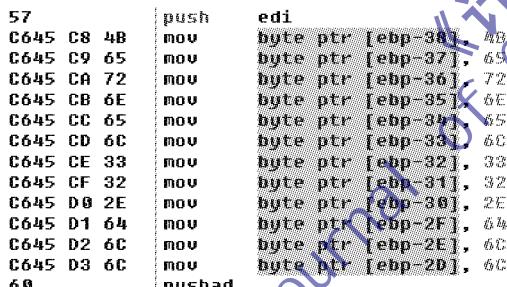


图 7 QQ\_DYP 通过拆解字符串以避免被查杀

反汇编代码中,还出现有图 8 所示的语句,如 2.2 节所述,该木马通过“lea eax, dword ptr[ebp-38]”与“mov dword ptr[ebp-4], eax”等关键语句将 Kernel32.dll 的 HMODULE 作为局部变量存放在了[ebp-4]中,以供后续调用使用。

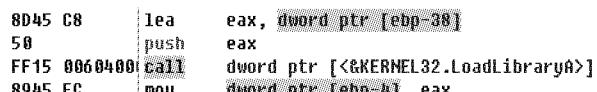


图 8 利用 EBP 寄存器访问局部变量

另外,QQ\_DYP 的反汇编代码中还出现有为数组赋值的循环语句(图 9)。结合 2.3 节的讨论,分析得出该循环语句负责为 char 数组的 3~6 位填充随机内容(范围是 0~9 十个数字),从而使其释放的文件无固定名称,以逃避反病毒软件的查杀。

## 3 结语

恶意程序编写者一般具有较高的专业知识水平,他们会

千方百计地模糊恶意程序行为,隐藏其真实意图。因此,针对恶意程序进行逆向取证分析,通常需要查壳、脱壳、字符串查找、断点设置、程序跟踪等多个环节<sup>[8]</sup>。特别是在关键信息获取方面,本文虽然介绍了一些典型断点的设置方法,但实际上整个恶意程序的分析花费了作者大量的时间与精力,因此针对快速定位关键信息的研究依然有很长的路要走。另外,本文的研究工作在某些方面还属于探索与尝试阶段,如在恶意程序编写语言方面,讨论了 Visual C++、Borland C++、Delphi、VB、汇编等常用语言,未涉及.NET;典型数据结构部分讨论了数组与链表,未对结构更为复杂的树加以分析说明;如何将逆向分析方法与其他分析方法更好地结合,更加全面地揭示恶意程序隐藏的秘密,也将是未来的主要研究方向之一。

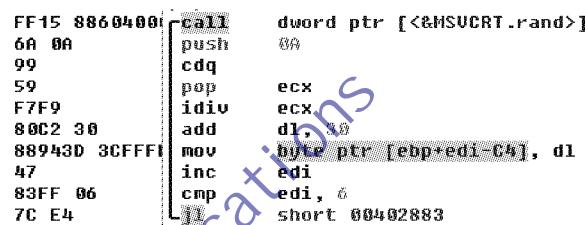


图 9 为数组赋值的循环语句

恶意程序既危险又复杂,电子数据取证人员需要借助逆向工具分析各个寄存器与内存中各个字节间的数据关系,逆向工具的智能化程度极大影响着分析工作的效率,因此研制开发功能更为强大的反汇编器、调试器及其集成工具,依然是恶意程序分析领域的主要工作之一;另外,作为证据出示的恶意程序分析结果,其处理、固定以及分析过程必须正规化,以避免造成证据污染<sup>[9]</sup>,因此如何规范分析工作行为,使分析结果更易于被法庭接受与采纳,同样也是电子数据取证领域亟待解决的重点问题。

### 参考文献:

- CARVEY H. Windows forensic analysis[M]. 2nd ed. Waltham: Syngress, 2007: 157.
- (美) EILAM E. 逆向工程揭秘[M]. 韩琪,译. 北京:电子工业出版社,2007: 4~16.
- 段钢. 加密与解密[M]. 北京:电子工业出版社,2008: 71~99.
- COHEN M, GARFINKEL S, SCHATZ B. Extending the advanced forensic format to accommodate multiple data sources, logical evidence, arbitrary information and forensic workflow[EB/OL].[2011-01-01]. www.pyflag.net/papers/dfrws\_2009.pdf.
- FREDERIC B, SOLAL J. Digital forensics framework[EB/OL].[2010-01-08]. http://www.digital-forensic.org.
- SZEWCZYK P, BRAND M. Malware detection and removal: An examination of personal anti-virus software[EB/OL].[2008-05-09]. http://scissec.scis.ecu.edu.au/proceedings/2008/forensics/Szewczyk%20%20Malware%20detection.pdf.
- TWCERT/CC. Spware forensic with reversing and static analysis[EB/OL].[2010-03-17]. http://www.hitcon.org/Download/2010/Spyware%20Forensic%20With%20Reversing%20and%20Static%20Analysis.pdf.
- OVERTON M. Malware forensics: detecting the unknown[EB/OL].[2009-07-23]. http://momusings.co.uk/Documents/VB2008-Malware-Forensics-1.01.pdf.
- 罗洁,张国臣.谨防电子物证提取和检验中的“污染”[J].刑事技术,2007(2):43.