Windows 内核变量定位与应用研究

车生兵 易 文
(中南林业科技大学 计算机与信息工程学院 长沙 410004)

摘 要：Windows 内核变量是内存分析过程中经常需要使用到的数据，但是由于 Windows 操作系统的封闭性，定位到 Windows 内核变量的位置非常困难。前人提出了一些内核变量定位的方法，但是在实验后发现，结果并不尽人意。针对这一现状，在前人的算法上进行了改进，提出一种基于虚拟地址转换的算法，使得可以准确定位内核变量位置。另外，也提出了一个基于 Windows XP 全新的内核变量快速定位方法。最后，以内核变量 MmPhysicalMemoryBlock 的应用为例，提出了基于 MmPhysicalMemoryBlock 的内存数据快速导出算法。实验结果表明，2 个内核变量定位算法能准确的定位内核变量，内存数据快速导出算法也能准确完整的导出需要的内存数据。

关键词：内核变量定位；内存取证；MmPhysicalMemoryBlock；内存分析
中图分类号：TP2 **文献标识码**：A **国家标准学科分类代码**：510.1050

Researches on Windows kernel variable locating and application

Che Shengbing Yi Wen
(College of Computer and Information Engineering of Central South University of Forestry & Technology, Changsha 410004, China)

Abstract: The data of Windows kernel variables was used frequently on the analysis of memory. But locating these kernel variables was limited by the operating system. Former scholars have proposed some algorithms of Windows kernel variables locating. But after experiments, the result was not satisfactory. With an improvement on precedent algorithms, an algorithm based on virtual address translation was proposed for accurately locating kernel variables. It could improve the accuracy of locating the kernel variables. And, an innovative fast locating algorithm based on the Windows XP kernel variables was proposed. At last, a fast memory data export algorithm based on MmPhysicalMemoryBlock was suggested with the example of kernel variable MmPhysicalMemoryBlock application. The experiments showed that, these two kernel variables locating algorithm are able to locate kernel variables preciesly, the fast memory data export algorithm is able to export wanted memory data with accuracy and integrity.

Keywords: locate Windows kernel variables; memory forensic; MmPhysicalMemoryBlock; memory analyze

1 引 言

科技不断的发展，计算机犯罪已经成为一个非常严峻的问题。自 20 世纪 60 年代以来，越来越多的人使用计算机从事一些违法犯罪的活动，不仅，有各种黑客藏匿在互联网之中，伺机对个人或企业的计算机进行攻击，更有不法分子传播盗版，情色等非法资料，或者通过互联网传输非法信息等等。据赛门铁克公布的数据显示，2011 年 7 月到 2012 年 7 月间，全球由于计算机犯罪造成的损失共计 1 100 亿美元[1]。这些违法犯罪活动，由于技术含量较高，使得侦破过程中的取证变得非常的困难。

随着计算机犯罪的日益增加，计算机取证变得尤为的重要，它的目的就是把犯罪者留在计算机中的“痕迹”作为有效的诉讼证据提取出来，交给法庭，以便将犯罪分子绳之于法。

作为计算机取证中不可或缺的一部分，内存取证的实施在以前主要都是对内存的数据进行简单的字符串搜索，不能自动化的定位所需要的数据，从而不能快速的分析用户行为。直到 2005 年，电子取证工作研究组（Digital Forensic Research Workshop，DFRWS）举办了内存分析挑战大赛，以 Windows 2000 的物理内存镜像为载体，对其进行分析，提取有用的数据。从那以后，越来越多的研究者投入到内存的取证研究当中。Betz 开发了工具 Mempaser，可以从 Windows 2000 的内存镜像中获取进程和线程的相关数据。也有文章提出过分析内核中的数据结构，以在 Windows 2000 到 Windows 2003 中搜索进程和线程的方法。Okolica 和 Peterson 做了许多的研究工作，设计了可以

收稿日期：2014-09

从 Windows NT 内核的操作系统镜像文件中提取进程和线程的工具[2]。2009 年，郭牧与王连海提出了一种基于 KPCR 结构的 Windows 物理内存分析方法[3-4]，可以定位内核变量位置，达到分析取证的效果。有人提出了从新版操作系统 Windows 7/8 中获取内存数据并分析的方法[5-6]。有人提出从内存中获得 Command Line 数据，注册表信息的方法[7-8]。本文将以 Windows XP 为例，阐述前人的相关研究与缺陷，并提出自己的内核变量定位算法。并且根据内核变量数据，说明的内核变量在计算机取证当中的应用。最后将以 MmPhysicalMemoryBlock 为例，提出内存数据导出算法。

2　基于 KPCR 虚拟地址的内核变量定位算法

内核变量是计算机操作系统内核程序所使用的一系列变量，对于内存取证有非常大的作用，以上提到的最近几年的内存分析方法，都是基于这种内核变量来分析内存数据，取证用户的行为。郭牧提出的基于 KPCR 虚拟地址的 Windows 内核变量定位算法[3]，是后人进行内存取证研究最常用的内核变量定位方法。

2.1　算法原理

随着计算机的发展，有部分的机器开始支持多个 CPU 同时协同工作，Windows 内核中为此定义了一套以处理器控制区（Processor Control Region）为枢纽的数据结构 KPCR[9]，使得每个 CPU 都有一个 KPCR 结构，用来保存与线程切换相关的全局数据信息。

KPCR 结构包含有 KPRCB、MajorVersion、MinorVersion、KdVersionBlock、TTS 等信息，这些信息将会在内存分析的过程中起到非常重要的作用。比如通过 KPRCB 内的 KTHREAD 指针可以找到当前运行的线程是哪一个，而通过寻找线程所属的进程可以找到运行的进程链表。而 KdVersionBlock 就是需要的内核控制块，通过该变量就可以找到一系列的内核变量的位置。因此，定位 KPCR 显得尤为重要。

该文经过大量的研究之后发现，KPCR 结构位于虚拟地址 0xFFDFF000 处，而相隔 0x120 处的 0xFFDFF120 处为 KPRCB 结构。通过研究 KPCR 前部分结构，可以发现，分别有 2 个指针 SelfPcr 与 Prcb 指向了 KPCR 本身与 KPRCB。

```
typedef struct _KPCR // 27 elements, 0xD70 bytes (sizeof)
    {
    /* 0x000 */ struct _NT_TIB NtTib; // 8 elements, 0x1C bytes (sizeof)
    /* 0x01C */ struct _KPCR * SelfPcr;
    /* 0x020 */ struct _KPRCB * Prcb;
    /* 0x024 */ UINT8 Irql;
    /* 0x025 */ UINT8 _PADDING0_[0x3];
    /* 0x028 */ ULONG32 IRR;
    /* 0x02C */ ULONG32 IrrActive;
    /* 0x030 */ ULONG32 IDR;
    /* 0x034 */ VOID * KdVersionBlock;
      ……
    }
```

基于这样的原因，可以推断出，在物理内存中，0xFFDFF000 与 0xFFDFF120 2 个虚拟地址是连续存放的。因此，郭牧等人[3]提出了，在内存镜像中直接搜索二进制串“00F0DFFF20F1DFFF”，就可以定位 SelfPcr 与 Prcb 指针，从而在该串物理地址减去 0x1C 的位置，就可以直接定位 KPCR 结构。

2.2　算法缺陷

这个算法在大部分的机器上能够做到准确的定位 KPCR 结构，但是在经过大量的实验之后发现，在某些 Windows XP 机器上并不能准确定位。经过研究之后发现，其原因在于，这些机器在真正的 KPCR 结构之前，出现了“00F0DFFF20F1DFFF”的二进制数值串。从而导致了，算法没能准确定位到 KPCR 结构。

事实上，在真正的 KPCR 结构之前出现同样的二进制串的原因，可能来源于很多方面。系统可能对 KPCR 数据进行备份，可能在某些程序中使用到了 KPCR 与 KPRCB 的指针，也有可能是其他的应用程序偶发性的使用到了同样的二进制串，甚至有可能受到恶意程序攻击向内存中注入了同样的二进制数值串等。

因此，郭牧提出的这种方法，有非常大的研究意义，但是在取证各种不同计算机的内存的时候，还需要进行一定的改进。

3　改进的内核变量定位算法

以上的算法之所以有一定的缺陷，主要原因就在于，搜索到指定的二进制数值串之后直接认为该结构体为 KPCR。事实上，再加上另外的验证条件来验证结构体的准确性即可。

可以用来验证的条件非常多，包括 KdVersionBlock 内的“KDBG”字符串等等许多只有 KPCR 才有的特征。虽然这些 KPCR 自有的特征能够排除其他应用程序或病毒偶发性的干扰，但是并不能排除系统有可能对 KPCR 数据备份的情况。提出了一个更能排除干扰的检验方法，虚拟地址转换物理地址检验的方法。

3.1　虚拟地址转换验证算法

通过前人的研究知道，KPCR 中有 2 个指针，1 个指向 KPCR 自身，1 个指向 KPRCB 结构体。而通过结构体可以看出，2 个结构体是相连的，因此 KPCR 与 KPRCB 的物理地址相差为固定 0x120 个字节。而且两个结构体同在一个页面，因此，虚拟地址也是相差固定 0x120 个字节。同时研究者发现，其 KPCR 与 KPRCB 的虚拟地址永远是

0xffdff000 和 0xffdff120。

根据上一节的表述,直接搜索该字符串有可能会出现定位到其他同样串的位置的情况。但是,可以用一种办法来验证该位置是否为 KPCR 结构体,那就是利用 KPCR 的虚拟地址转换为物理地址之后,检查与当前的位置是否相符。

转换虚拟地址到物理地址,需要知道页目录表的地址。对于所有的系统的虚拟地址来说,其页目录表的物理地址都是存放在 CR3 寄存器当中。通过 KPCR,在其偏移 0x120 处可以找到 KPRCB 的位置,而在 KPRCB 结构的偏移 0x1C 处,有一个_KPROCESSOR_STATE 的结构体 ProcessorState,该结构体记载了所有的 CPU 的元件状态,包括特殊寄存器。在该结构的偏移 0x2CC 处是一个 KSPECIAL_REGISTERS 结构体,里面记载了 CR0-CR4 以及其他许多寄存器的当前值,只需要在此结构体的偏移量 0xC 处即可找到 CR3 寄存器,其值也就是系统页目录表的基址。以上结构体均可以参考网站所给出的结构信息。

根据 CR3 的系统页目录表基址,可以根据上一节的方法,将 0xffdff000 和 0xffdff120 2 个虚拟地址转换为物理地址,然后再确认当前位置的物理地址是否与转换后的物理地址相符。具体检验流程如图 1 所示。

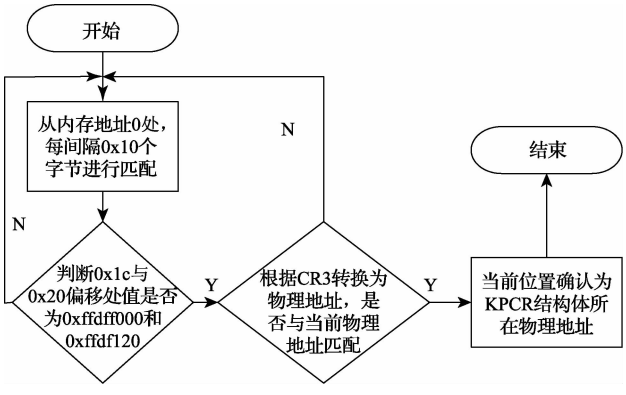

图 1 基于虚拟地址转换验证 KPCR 定位算法

3.2 直接定位方法

基于虚拟地址转换验证的算法在检测的数十台不同环境,不同版本的 Windows XP 操作系统(包括 SP1、SP2 以及 SP3)的机器上都能完美的定位到内核结构 KPCR 的位置。但是验证方法需要对内存数据进行暴力搜索,而且验证算法也比较复杂,如何以更快的速度,更有效的算法来定位 KPCR 结构呢?

经过了大量的实验之后,发现一个意外的规律,所有搜索到的 KPCR 结构都是在物理地址 0x40000 的位置,也就是 0x40 帧的位置。通过另外大量的寻找不同的 Windows XP 操作系统验证之后发现,KPCR 结构确实都加载在 0x40000 的位置。因此,提出了直接定位物理地址 0x40000 为 KPCR 的算法,流程如图 2 所示。

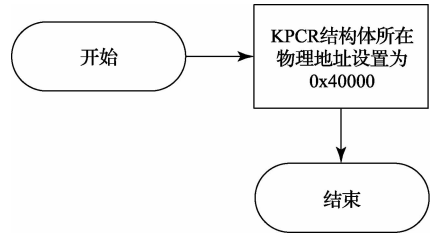

图 2 直接定位的 KPCR 定位算法

该算法可以不需要进行任何搜索即可直接定位 KPCR 结构,且实验准确率也能达到 100%。

以上就是 2 个不同的 KPCR 内核变量定位算法。值得注意的是,2 个算法均是针对 Windows XP 系统的内核变量定位算法。

4 内核变量及应用

4.1 内核变量相关结构体

KPCR 是非常重要的数据结构,定位其主要目的之一就是查看诸多其他内核变量的位置,这是因为 KPCR 中偏移位置 0x34 处有一个属性 KdVersionBlock,这是一个虚拟地址,指向的是_DBGKD_GET_VERSION64 结构体,该结构体包含了 MajorVersion,MinorVersion 等的版本信息,甚至还包括了 PsLoadedModuleList—加载模块列表信息。从某个研究内存数据块内容中进行实验,通过 KPCR 获得 KdVersionBlock 的值后进行虚拟地址到物理地址的转换,得到_DBGKD_GET_VERSION64 结构体物理地址为 0x5532B8,其数据块信息如图 3 中第一个黑框指定区域所示。

```
005532B0  48 C1 56 80 00 00 00 00  0F 00 28 0A 06 00 03 00
005532C0  4C 01 0C 03 2D 00 00 00  00 80 4D 80 FF FF FF FF
005532D0  A0 34 56 80 FF FF FF FF  F4 EF 69 80 FF FF FF FF
005532E0  F4 EF 69 80 F4 EF 69 80  00 00 00 00 00 00 00 00
005532F0  4B 44 42 47 90 02 00 00  00 80 4D 80 00 00 00 00
00553300  72 3A 4E 80 00 00 00 00  00 00 00 00 00 00 00 00
00553310  2C 01 08 00 18 00 00 00  40 5D 4E 80 00 00 00 00
00553320  60 E4 92 7C 00 00 00 00  A0 34 56 80 00 00 00 00
00553330  58 95 56 80 00 00 00 00  60 96 56 80 00 00 00 00
00553340  10 BA 56 80 00 00 00 00  20 A9 56 80 00 00 00 00
00553350  14 82 55 80 00 00 00 00  7C 24 56 80 00 00 00 00
00553360  58 25 56 80 00 00 00 00  A0 2E 56 80 00 00 00 00
00553370  40 0E 56 80 00 00 00 00  D8 8B 56 80 00 00 00 00
00553380  D0 8B 56 80 00 00 00 00  98 79 55 80 00 00 00 00
00553390  48 7C 56 80 00 00 00 00  60 7C 56 80 00 00 00 00
005533A0  48 7E 56 80 00 00 00 00  C8 30 56 80 00 00 00 00
005533B0  C0 30 56 80 00 00 00 00  58 34 56 80 00 00 00 00
005533C0  E0 79 56 80 00 00 00 00  C4 78 55 80 00 00 00 00
005533D0  A4 7E 56 80 00 00 00 00  A8 7E 56 80 00 00 00 00
005533E0  C4 7C 56 80 00 00 00 00  00 7E 56 80 00 00 00 00
005533F0  18 31 56 80 00 00 00 00  E0 7C 55 80 00 00 00 00
00553400  E4 7C 55 80 00 00 00 00  14 31 56 80 00 00 00 00
... ...                            ... ...
00553550  48 34 56 80 00 00 00 00  EC 78 56 80 00 00 00 00
00553560  E0 78 56 80 00 00 00 00  00 00 00 00 00 00 00 00
00553570  01 00 00 00 00 00 00 00  58 B8 55 80 5C B8 55 80
```

图 3 _DBGKD_GET_VERSION64 与_KDDEBUGGER_DATA 6 数据

_DBGKD_GET_VERSION64 包含有 13 个元素,共计

0x28 字节长度。以上数据可以解析如下：

/ * 0x000 * / MajorVersion(000F)；

/ * 0x002 * / MinorVersion(0A28)；

/ * 0x004 * / ProtocolVersion(0006)；

/ * 0x006 * / Flag(0003)；

/ * 0x008 * / MachineType(014C)；

/ * 0x00a * / MaxPacketType(030C)；

/ * 0x00c * / MaxStateChange(2D)；

/ * 0x00d * / Simulation(00)；

/ * 0x00e * / Unused(0000)；

/ * 0x010 * / Kernbase(FFFFFFFF804D8000)；

/ * 0x018 * / PsLoadedModuleList(FFFFFFFF805634A0)；

/ * 0x020 * / DebuggerDataList(FFFFFFFF8069EFF4)；

紧随其后的是一个 8 个字节的结构体，是打印等待队列 KdPrintCircularBufferPtr，包含两个 4 字节的虚拟地址，分别为 Flink 与 Blink。而在之后，就是一个非常重要的结构体_KDDEBUGGER_DATA64，数据块信息从图 3 中偏移 0x5532E8 开始到 0x553578 结束。包含了非常多的重要的内核变量数据，除了 0x10 个字节的头结点以外，其他的数据均为长度为 8 字节的内核变量，如下所示：

```
typedef struct _KDDEBUGGER_DATA64 {
DBGKD_DEBUG_DATA_HEADER64 Header;
ULONG64 KernBase;
ULONG64 BreakpointWithStatus;
ULONG64 SavedContext;
USHORT ThCallbackStack;
USHORT NextCallback;
USHORT FramePointer;
USHORT PaeEnabled:1;
ULONG64 KiCallUserMode;
ULONG64 KeUserCallbackDispatcher;
ULONG64 PsLoadedModuleList;
ULONG64 PsActiveProcessHead;
ULONG64 PspCidTable;
……
ULONG64   MmPhysicalMemoryBlock;
……
}
```

4.2　内核变量作用举例

以下例举几个比较重要的内核变量，简要说明一下其意义。

_KDDEBUGGER_DATA64 中偏移量 0x40 处，也就是物理地址 0x553328 处开始的方框内为变量 PsLoadedModuleList，该变量与 _DBGKD_GET_VERSION64 偏移 0x18 处的低位 4 个字节数据内容一致，指向的是当前系统加载模块列表。通过该虚拟地址进行转换之后，可以得到一个双向循环的队列的头结点，根据该头结点的向前以及向后的指针，就可以依次将整个系统加载的所有的模块信息列举出来。

偏移量为 0x48 处，也就是物理地址 0x553330 处开始的方框内为变量 PsActiveProcessHead，该变量指向的是当前系统加载进程的列表信息。通过该虚拟地址进行转换之后，可以得到一个进程的双向循环链表，以同样的方法对该链表的前后结点进行搜索之后，就可以得到整个系统加载进程链表信息，包括所有隐藏的进程等，这对于内存取证，查杀病毒等，都有非常重要的作用[10]。

在偏移量为 0x50 即物理地址 0x553338 处的 PspCidTable 变量是指向了对象表信息，其位置存放了所有系统所有的进程与线程对象信息的表格。通过该变量，可以分析出，所有系统的进程与线程的对象句柄，从而分析出所有线程与进程对象，哪怕这些进程与线程并不存在于进程与线程链表当中。

而在偏移地址为 0x268 处，也就是物理地址 0x553550 处的 MmPhysicalMemoryBlock，指向的是物理内存块运行表，从这个结构体中，可以得知，物理内存所使用的运行数据是哪些，而基于这个对象，提出了基于 MmPhysicalMemoryBlock 的内存数据快速导出算法，接下来将以该算法为例，说明内核变量数据的应用。

4.3　基于 MmPhysicalMemoryBlock 的内存数据快速导出算法

研究计算机内存的数据，将内存数据导出是一个必不可少的部分，在读取地址空间的数据的时候，通常是使用系统提供的接口获得内存大小 MemorySize，从而导出 0-MemorySize 地址空间内所有的内存数据。但是在分析内存数据的时候发现，经常会有物理地址超过 MemorySize 的情况出现，真正的尝试查看 MemorySize 以上的地址空间时，也会发现，确实有数据存在。另外，在导出过程中还发现，并不是所有的内存空间数据都能够顺利的获得。在某些情况下，会出现导出某些数据之后，会导致屏幕黑屏甚至于死机的情况出现。如何完整的获得所需要的数据，而且又能够不影响计算机的正常使用，这还需要进一步的研究。

通过仔细研究计算机的组成原理之后得知，计算机的内存地址空间并不仅仅的包含物理内存，同时还将显卡等等许多其他设备的存储空间进行了统一的编址。因此，仅仅认为内存地址空间为 0-MemorySize 的做法是明显错误的，实际的内存地址空间大小，还需要加上本身独立显卡以及各种连接的存储设备的内存。

虽然内存地址空间中包含了许多非物理内存的数据，但是大部分的非物理内存的数据均为设备专用。例如显卡的内存是用来存放显示缓冲区，以及显卡自己运算所需要

使用的内存空间等。通过仔细的对比研究之后发现,这些空间的数据读取:一方面会使得显卡运行不正常,导致黑屏甚至死机的情况;另一方面,对于内存研究用处不大。因此,完全可以忽略这些数据,仅导出所需要的物理内存数据。

如何获得所需要的物理内存数据是所有的内存分析与取证研究中不可或缺的一部分,而 MmPhysicalMemoryBlock 为获得这些数据提供了十分便利的条件。

MmPhysicalMemoryBlock 是一个内核变量,是操作系统提供的关于物理内存在内存地址空间中的分配情况记录。如图 4 所示,整个地址空间被分成了许多的小块。其中,所有操作系统使用到物理内存空间为图中的 Run[0],Run[1]……Run[n]所指的数据块,都以运行段的方式记录在 MmPhysicalMemoryBlock 中。而在运行段与运行段之间,有一部分的内存地址空间被分配给了各种不同的设备,例如显卡的显存。而存在一些没有任何含义的空隙页面,处在不同的块之间。在这些空隙页面中,有个比较特殊页—第 1 页,物理地址从 0~0x1000,这一页只有操作系统在启动过程中使用到,启动完成之后,则直接将该页也作为空隙页处理了。

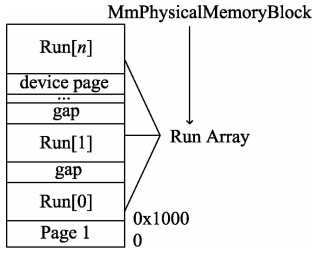

图 4 内存地址空间分布

因此,通过检验 MmPhysicalMemoryBlock 变量指向的运行段数据,就可以知道所有的物理内存数据在内存地址空间中的区域了,从而仅仅导出该区域内的数据,其他区域用指定的数据填充,这样就可以快速完整的导出所需要的物理内存数据内容。

获取运行段数据的流程如图 5 所示。

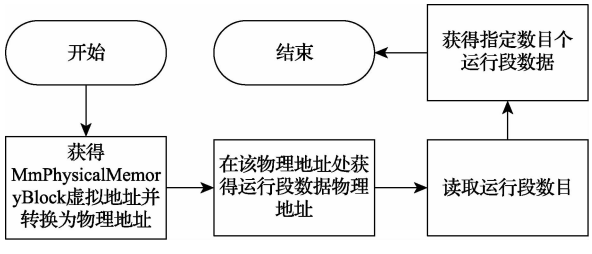

图 5 获得运行段数据流程

接下来通过实例数据来解析运行段数据分析过程。

从图 3 可知,MmPhysicalMemoryBlock 的值为 0x80563448,转换为物理地址后为 0x563448。在物理地址 0x563448 处存放的就是运行段数据的虚拟地址,查询 0x563440 处开始 0xF 个字节数据如图 6 所示。

```
40 34 56 80 40 34 56 80 80 CD 7F 86 2F 1A 03 00
```

图 6 MmPhysicalMemoryBlock 指向地址处数据

获得该虚拟地址为 0x867FCD80,通过转换之后得到物理地址为 0x67FCD80,转到该地址之后数据内容如图 7 所示。

```
03 00 00 00 2C FE 03 00   01 00 00 00 9E 00 00 00
00 01 00 00 FF 0E 00 00   00 10 00 00 8F EE 03 00
```

图 7 运行段数据

运行段数据结构如下所示:

```
typedef struct _PHYSICAL_MEMORY_DESCRIPTOR {
    ULONG NumberOfRuns;
    // NumberOfPages * PAGE_SIZE 为物理内存大小
    PFN_NUMBER NumberOfPages;
    //运行段数组
    PHYSICAL_MEMORY_RUN Run[NumberOfRuns];
} PHYSICAL_MEMORY_DESCRIPTOR, * PPHYSICAL_MEMORY_DESCRIPTOR;
```

其中_PHYSICAL_MEMORY_RUN 结构体定义如下所示:

```
typedef struct _PHYSICAL_MEMORY_RUN {
    PFN_NUMBER BasePage;
    PFN_NUMBER PageCount;
} PHYSICAL_MEMORY_RUN, * PPHYSICAL_MEMORY_RUN;
```

在 32 位的 Windows XP 操作系统中,PFN_NUMBER 的长度为 4 个字节。因此可以解析得出,运行段数目 NumberOfRuns 为 3,运行段的页面总数 NumberOfPages 为 0x3FE2C,而每页的大小为 0x1000 个字节,因此总共为 0x3FE2C000 个字节(1022.17 MB),而这个值与物理内存的总量相等。

接着就是 NumberOfRuns 个运行段数据,运行段由两个属性组成,起始页 BasePage 和页面数 PageCount。从而可以分析出表 1 的运行段数据。

表 1　运行段数据解析

	运行段 1	运行段 2	运行段 3
起始页	0x01	0x0100	0x1000
页面数	0x9E	0x0EFF	0x3EE8F
物理	0x1000	－0x9F000	0x100000
地址	－0xFFF000	0x1000000	－0x3FE8F000

因此，通过解析可以得知，此机器运行段数据在内存地址 0x1000-0x9F000、0x100000-0xFFF000、0x1000000-0x3FE8F000 中。在导出物理内存数据过程中，仅仅导出这些区域数据，其他区域数据直接用标志字段替代。

5　结　　论

以 Windows XP 为例，提出了 2 种内核变量定位方法，一种是基于前人算法进行改进的基于虚拟地址转换的 KPCR 定位算法；另一种是通过大量实验得出的直接快速的内核变量定位算法。2 个算法都经过大量的实验验证，证明其有效。且比前人算法容错性更强，能够应付更多不同情况下的操作系统机器。其次，简要分析了部分内核变量的应用方法，并且以 MmPhysicalMemoryBlock 变量为例，提出了基于 MmPhysicalMemoryBlock 的物理内存数据快速导出算法，能够快速的定位物理内存在内存地址空间中的位置，从而准确的将需要的物理内存数据导出，避免触碰不需要的显示缓冲区等数据，引发黑屏或死机的情况，从理论上说明了算法的有效性，并在多个不同的机器上进行实验，验证了物理内存导出算法的可行性。

参考文献

[1] 贾宝安. 内存取证技术的研究及应用[D]. 四川：电子科技大学，2013，1-2.

[2] OKOLICA J，PETERSON G. Windows operating systems agnostic memory analysis [J]. Digital Investigation，2010，7：48-56.

[3] 郭牧，王连海. 基于 KPCR 结构的 Windows 物理内存分析方法[J]. 计算机工程与应用，2009，45(18)：74-77.

[4] 韩敬伟，李树彪. 浅析 WINDOWS 环境下的内存分页机制[J]. 国外电子测量技术，2008，27(6)：35-38.

[5] THOMAS S，SHERLY K K，DIJA S. Extraction of memory forensic artifacts from windows 7 RAM image [C]. Information & Communication Technologies (ICT)，2013 IEEE Conference. 2013：937-942.

[6] XIANG T，GOU M L. Memory forensics based on Windows 8 physical memory dumps [J]. Computer Engineering and Applications，2013，49(19)：63-67.

[7] STEVENS R，CASEY E. Extracting Windows command line details from physical memory [J]. Digital Investigation，2010，7：57-63.

[8] ZHANG SH H，WANG L H，ZHANG L. Extracting windows registry information fromphysical memory[C]. IEEE，2011，85-89.

[9] ZHANG R CH，WANG L H，ZHANG SH H. Windows Memory Analysis Based on KPCR [C]. 5th International Conference on Information Assurance and Security，2009.

[10] 陈龙，敬凯，董振兴，等. 基于 EPROCESS 特征的物理内存查找方法[J]. 重庆邮电大学学报：自然科学版，2013，25(1)：122-125.

作者简介

车生兵，教授，硕士学位。主要研究方向为网络与信息安全、图像处理、模式识别、智能系统和农业信息化。E-mail：cheshengbing727@163.com

易文，硕士研究生。主要研究方向为内存取证。