Software Engineering and Applications 软件工程与应用, 2012, 1, 47-53
http://dx.doi.org/10.12677/sea.2012.12009 Published Online December 2012 (http://www.hanspub.org/journal/sea.html)

Software Protection Method Based on Shell Technology

Yuanpeng Sun[1], Wenyu Chen[2], Wen Li[2], Lingli Guo[2], Weishun Li[2]

[1]Shenzhen Zhenhua Microelectronics Co., Ltd., Shenzhen
[2]School of Computer Science and Engineering, University of Electronic Science & Technology of China, Chengdu
Email: sunyp@zhm.com.cn, cwy@uestc.edu.cn

Received: Oct. 30th, 2012; revised: Nov. 9th, 2012; accepted: Nov. 23rd, 2012

Abstract: Shell encryption is the most common and safest software encryption technology. The thesis propose new software protection model based on shell technology which is based on the integration of software shell protection model, the so-called "integration" is making the shell program and the original program combined together. When the shell is taken off by cracker, then the part of the original program was also given away, thus reach the purpose of protection software. This thesis also puts forward the solution implementing the model which is based on the code-of-unordered mechanism which means that the original program with shell code are out of order for reaching the goal of integration, at the same time for defensing static recompilation we will add junk code and use SHE technology for anti-dynamic tracking. Ultimately we will implement packing based on the integration of software shell protection model by this solution. The software shell protection model based on the integration has strong software protection ability.

Keywords: Software Protection; PE Files; Shell; Integration

基于壳技术的软件保护方法

孙元鹏[1]，陈文宇[2]，李　文[2]，郭凌立[2]，李维顺[2]

[1]深圳市振华微电子有限公司，深圳
[2]电子科技大学计算机学院，成都
Email: sunyp@zhm.com.cn, wy@uestc.edu.cn

收稿日期：2012 年 10 月 30 日；修回日期：2012 年 11 月 9 日；录用日期：2012 年 11 月 23 日

摘　要：壳技术是软件加密中最常见、最安全的一种技术。本文提出了新的基于壳技术的软件保护模型：基于融合的软件壳保护模型，即将外壳程序与原程序相互结合在一起，当破解者将外壳脱掉的同时，也除去了部分的原程序，从而到达保护软件的目的。本文同时提出了该模型的实现方案：基于代码乱序的机制，该机制将原程序与外壳程序代码进行乱序来到达融合的目的，并且在代码乱序的同时加入了花指令来防御静态反编译；在外壳程序中使用了 SHE 技术来反动态跟踪。通过实例验证了基于融合的壳软件保护模型具有很强的软件保护能力。

关键词：软件保护；PE 文件；壳；融合

1. 引言

随着计算机软件技术的发展，恶意软件感染破坏能力的增强、破解者水平的大幅提升，传统的软加密和硬加密方案对于安全性要求高的软件来说，保护效果不明显，现在已经发展到其他很多成熟的方式：注册验证、软件水印、反跟踪技术、加壳技术[1]等。其中壳加密技术是软件加密中最常见、最安全的一种技术。

加壳技术保护方式是指：利用某种算法，对 PE (Portable Executable)文件进行加密、压缩，给 PE 文件加上一个外壳。软件壳其实是一段为了防止软件被非法使用、篡改、拷贝等的程序。通常这部分代码比原

程序优先执行并且取得控制权，这样就起到保护软件的作用。加了壳的 PE 文件仍然能直接运行，这个外壳程序负责把 PE 文件原程序解密解压到内存里，并且把控制权还给解压解密后的原程序。整个过程都是运行在内存里，并且程序运行速度基本没有影响，解密、解压过程完全透明[2]。

2. PE 文件格式

PE 二进制文件格式是微软操作系统系列中的子集。图 1 是 PE 文件格式总体结构的层次布局。从图 1 中可以看出，PE 文件格式二进制数据流是线性的。它从一个 MS-DOS 头部结构开始，一个模式的程序残余紧随其后，接着就是 PE 文件头，其中 PE 文件头包含映像文件头(File Header)和可选文件头部(Optional Header)，这部分后面的结构就是区块表头，紧跟在区块表头之后就是全部的区块(Selection)，PE 文件的尾部是其他数据，其中包含 Code View 调试信息、COFF 符号表信息、COFF 行号信息[3]等。从大体上分为五大部分：

1) DOS 首部(DOS Header)；

2) PE 文件头(PE Header)；

3) 区块块表(Section Table)；

4) 块(Section)；

5) 调试信息。

其中，PE 文件格式结构包含三个非常重要的表头：

1) DOS Header：为了兼容以前的平台；

2) PE Header：PE 文件相关配置信息；

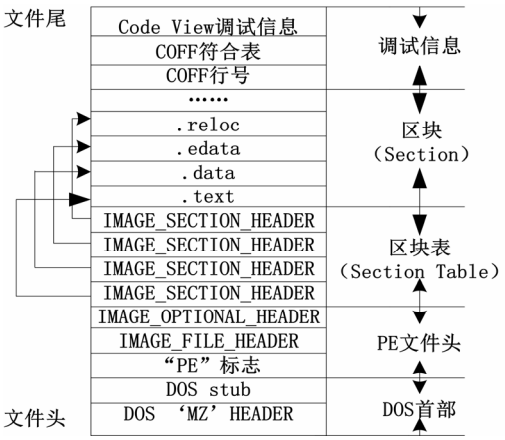

Figure 1. PE file format overall structure
图 1. PE 文件格式总体结构

3) Section Header：PE 文件各个区块的相关信息。

PE 装载器加载 PE 文件的时候，就是根据这个三个表头所提供的文件信息把文件各部分加载映射到内存中各自相应的位置，保证与磁盘数据结构布局的一致性。通过遍历在磁盘里 PE 文件内部信息来决定哪一部分被映射，一旦文件被加载到内存里之后，内存中的数据结构布局与外存(磁盘)上的数据结构布局的一致性[4]。因此，如果清楚在外存(磁盘)中数据结构中存在的内容，那么基本上都可以在内存映射文件里找到一样的数据；但是在内存映像文件中，通常已经改变了这些数据之间的相对位置，数据偏移地址已经和在外存上不一样了，无论如何，这些全部的数据都可以将外存文件偏移与内存映像文件偏移进行转换[5](如图 2 所示)。

3. 壳技术分析与研究

软件保护中的重要手段之一就是软件加密技术。这些加密技术措施给非法拷贝或者破解造成了非常大的困难，在相当程度上增强了破解难度[6]。现在加密程序的重要手段之一就是对程序加壳。

3.1. 加壳原理分析与研究

加壳结构如图 3 所示。对 PE 文件加壳过程为：

1) 首先对需要加壳的 PE 文件的 Section 区块数据进行压缩或者解密。

2) 然后把解压、解密的数据附加到区块里，同时在原文件的区块头中增添相应的区块表。其中这些数据也就是所谓的外壳程序。

3) 最后修改原 PE 文件程序的入口地址，让其指向外壳程序。

加了壳的 PE 文件程序执行过程示意图如图 4 所示。加了壳的 PE 文件执行过程为：

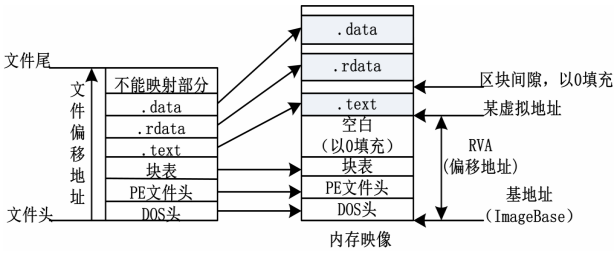

Figure 2. External and internal memory image structure
图 2. 外存与内存映像结构图

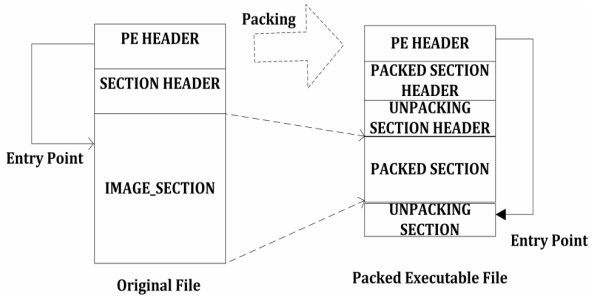

Figure 3. The packing structure
图 3. 加壳结构

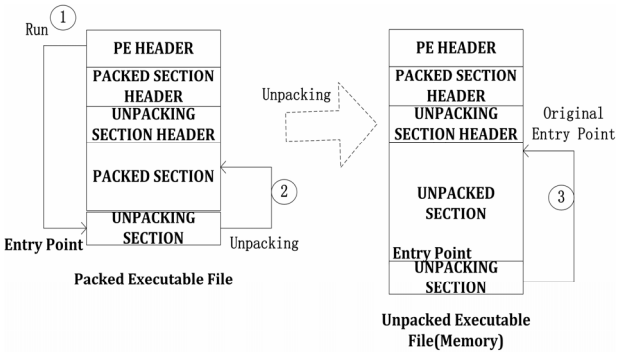

Figure 4. Execution process of packed PE file
图 4. 加了壳的 PE 文件执行过程

当加了壳的 PE 文件执行时，Windows 装载器首先加载 PE 文件到虚拟内存中，然后 PE 文件找到程序入口地址开始运行，而这时的程序入口地址就是外壳程序的入口地址。

外壳程序得到控制权，开始解压和解密原程序数据。

找到原文件的入口地址，把控制权交给原程序，让原程序继续执行。

3.2. 外壳加载

在上述 PE 文件程序执行过程中的第二步就要外壳的加载的过程。

1) 保存入口参数：外壳程序在刚开始时要把寄存器的值保存起来，当该程序结束的时候，要用来恢复到现场，最后再将控制权交给原程序。

2) 获取壳要用的 API 地址：事实上，外壳有时还需要其他的 API 函数，来完成它相应的任务[6]。为了把这些所需要的 API 隐藏起来，外壳程序通过相应的函数来动态加载这些输入函数。

3) 解密原程序中所有区块(Section)的数据：一般加壳程序是按照一个区块一个区块进行压缩加密的，那么当外壳程序也必须按这种方式对区块进行解压解密，并且把这些解压解密之后得到数据按照区块表的信息还原到内存相应位置[7]。

4) 初始化 IAT：对 PE 文件加壳后，通常都会对原 PE 文件的 IAT 表进行加密等处理，自己创建一个输入表，并且设置 PE 文件头里的输入表指针，让它指向这张自己建立的输入表。所以原 PE 文件的输入表的必须让壳程序来填写。

5) 重定位：但是 Windows 操作系统不能确保每次运行 DLL 类型的动态链接库文件的基址是一样的，那么需要“重定位”了，通常这个时候外壳中同样需要有能够“重定位”的程序代码，否则原程序是不能正常运行的[8]。

6) HOOK-API：外壳程序通常都把原 PE 文件程序的 Import Table 给破坏修改了，然后模仿着 Windows 操作系统处理来把相关数据写到 Import Table 里。在填写过程中，外壳程序可以在输入表写入 HOOK-API 代码的地址，通过这种间接的方式便能够地取得程序的控制权[9]。

7) 跳转到原 PE 文件程序的真正入口地址：最后所有的外壳程序运行完之后，都必须跳转到原 PE 文件的原始入口，让原程序得到控制权，继续执行。

3.3. 脱壳分析与研究

脱壳的目的就是把加了外壳的软件程序恢复到原来的状态，并且能使脱壳后的软件能正常执行。其中脱壳方式分为自动脱壳与手动脱壳[10]。自动脱壳，就是使用脱壳机实现软件的脱壳，但是现在很多外壳都没有相应的脱壳机，必须要分析 PE 文件程序和外壳等，然后进行手动脱壳。当该种脱壳过程分为三个步骤：

1) 找到原程序的真正入口地址(即所谓的“OEP”)：加了外壳的 PE 文件运行的时候，首先运行壳程序代码，外壳模块在内存中解压解密 PE 文件的原程序，同时把控制权交给被还原的程序，再跳转到 PE 文件的原来程序入口地址，在这里，壳一般都有一个明显的“分界线”，这个还原后的真正程序入口地址称为 OEP(Original Entry Point)[11]。寻找入口地址可以有以下几种方式。

根据跨段指令寻找程序入口地址：大部分的加了壳 PE 文件都会新加入区块，用来存放外壳程序代码

数据。当外壳程序运行完之后，就要跳转到真正原程序本身上，所以能根据跨段指令节来寻找到 OEP。

使用内存访问断点来寻找程序入口地址：该方法其实第一种方法的变形，只是不用一步步跟踪，当对代码段设置内存访问断点时，就肯定会在外壳程序对代码写入的那条指令上出现中断。

2) 抓取文件内存映像内容：该步骤就是常常说到的 Dump(“转存”)。是指读取出内存中指定位置的 PE 文件映像，接着用文件等方式存下来。通常情况下，进行 Dump 文件的开始位置在寻找到程序 OEP 的时候。

3) 重建 PE 文件输入表：通常在对 PE 文件进行加壳的时候，都会破坏原程序的 Import Table。使用在脱壳过程中，必须重新建立原程序的输入表。

程序运行时，首先执行外壳程序，把 PE 文件原程序数据进行解压解密还原到内存中。就在这个时候，内存里面的数据就是加壳前的原文件程序了，适机把这些内容取出来，再进行一定的修正就可以恢复到没有加壳时的情况了。

4. 壳保护模型

4.1. 基于融合的软件保护模型

如果要实现脱壳，第一步也是最重要的一步就是找到 PE 文件原程序入口地址，然后抓取内存 PE 文件的映像，还原程序。而从加了壳的 PE 文件运行来看，壳加载本身就提供了这样的一个明显断点给脱壳者，即无论什么壳最终都要转的原程序的入口地址，然后成功脱掉外壳。因为最终都要把控制权还给原程序，因此不能够将入口地址这个分界点进行隐藏。只能办法针对脱壳的第二步，采取一定保护措施让脱壳者即使得到的内存 PE 文件映像也没有很好的办法还原成原程序。本文就是基于这个问题作为出发点的。提出壳融合的软件保护模型。所谓融合，简单的说就是让外壳和 PE 文件原程序融合在一起。该种壳保护模型如图 5 所示。

从该模型可以看出：即使脱壳者，找到了原程序入口地址，然后抓取了 PE 文件内存映像，再把壳给脱了，也很难还原 PE 文件原程序。因为一旦把壳给脱了，那么原程序的一部分也给脱掉了，那么这样就不能将原程序还原了，从而到达保护软件的目的。

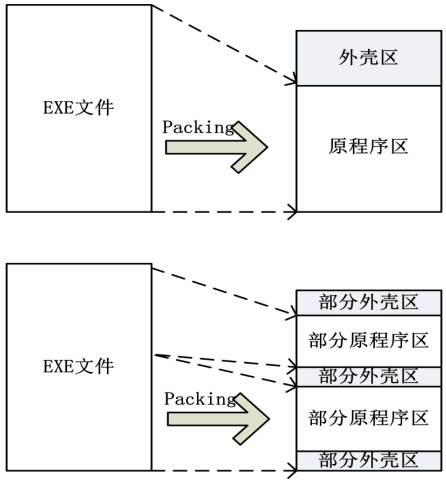

Figure 5. Software protection model based on integration
图 5. 基于融合的软件保护模型

如何实现这样的软件保护模型，本文将采取代码乱序的技术。代码乱序的原理很简单，就是把原来的流程打乱。很多脱壳者、破解者在逆向程序的时候，会把一个程序或者内存 PE 文件的映像直接通过反汇编等方式到得原程序或者流程图。而代码乱序的作用就是要把原程序打乱，让脱壳者很难分析。该模型正是基于这个原理，将原程序和外壳代码进行乱序，到达原程序与外壳的融合，大大增强保护软件的能力。并且在实现的加壳过程中还可以结合多种软件保护技术来增强抵御分析、破解的能力，到达更好的保护软件的目的。例如使用当前最流行的花指令来抵御静态分析和反汇编，使用 SHE 来反跟踪[12]等，当然代码乱序这种方式本身也具有这些功能。

4.2. 保护模型框架

本文所要实现的加壳分为两大部分：原 PE 文件处理和外壳程序。

原 PE 文件处理包括加壳程序和 PE 文件重组过程：将要加壳的 PE 文件从磁盘加载到内存里面，接着对 PE 文件内存映像每个相关头、区块进行处理，主要是对映像文件区块数据进行加密，计算代码段的 CRC32 值并写入到 PE 文件的头里面，特殊数据处理，输入表的构建，最终把外壳程序附加到该 PE 文件的内存映像里，这样就组成了新的加了壳 PE 文件了。

外壳程序：加了壳的 PE 文件开始运行壳程序的部分，它将对原 PE 文件程序就行 CRC32 校验，解密区块数据，输入表的重建，程序乱序，最后跳转到原

程序入口，把控制权移交给原程序。

本系统框架结构图如图 6 所示。

从文件格式分布角度来看，加了壳的 PE 文件由四部分组成。分别是：PE 文件头、被加密的文件区块数据、外壳程序、附加数据。原 PE 文件的区块数据以加密的方式存放。原 PE 文件将新增一个区块，用来存放外壳程序数据。其中在该新增的区块数据中，以 ShellEnd0 为界线，在这之前的数据(主要是资源中的一些数据如图标等)以加密的方式存放，而在之后的数据没有做任何处理，也就是以明文的方式存放的。加了壳的PE文件程序入口地址指向外壳的入口地址：ShellStart0。引导部分进行一些初始化工作，有时如果加密了后面一部分外壳程序，那在这里将进行解压。之后利用来进行校验 PE 文件的完整性，防止脱壳者动态跟踪、脱壳、反汇编。如果校验成功，那么外壳程序继续执行解压还原 PE 文件程序、代码乱序、输入表处理等等。

4.3. 保护模型解决方案

通过以上分析可以将该系统可以分为四个大模块：加密、外壳、PE 文件头处理、壳添加模块，关系如图 7 所示。

1) 加密部分：对内存映像文件的区块数据、资源数据进行加密。这些被加密的数据将被外壳程序解密。

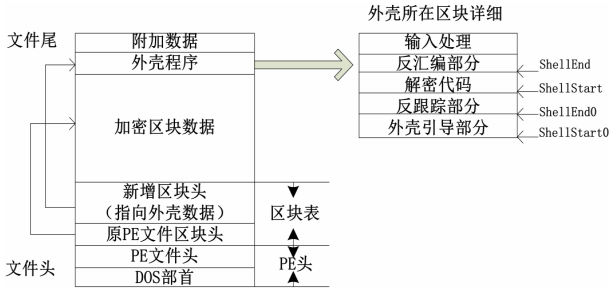

Figure 6. The packing system framework structure
图 6. 加壳系统框架结构图

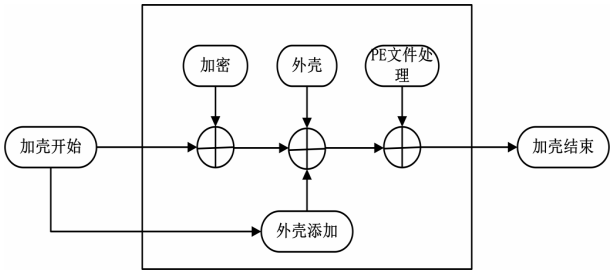

Figure 7. Total system design
图 7. 系统总设计

2) 外壳部分：就是本系统中的外壳程序，当程序运行的时候，这部分外壳程序首先运行，解密被加密的数据，修改相关信息，还原 PE 文件的原程序。

3) 外壳添加部分：在 PE 文件中新建区块，用于存放外壳部分数据，同时将 PE 文件的区块表部分添加新区块表头。

4) PE 文件处理部分：将 PE 文件读入的内存，然后对 PE 文件内存映像的附加数据、Import Tabel、资源数据等进行处理。

4.4. 实现方案

由于本系统是通过基于乱序来实现融合的保护模型，所以外壳程序也必将实现程序乱序的功能。从“乱序”这个概念就可以看出，就是将原程序的代码进行乱序。在本文的乱序的过程还会加入花指令增强反汇编防御静态分析的能力。程序代码乱序分为以下几步：

1) 找出程序代码中的跳转指令(jmp)、函数调用指令(call)和条件专用指令(jcc)。(这里可以反汇编引擎)。

2) 筛选上面的地址，如果跳转 16 位以下的直接不考虑，如果等于 16 位，看偏移是否足够跳转到花指令的 VAR。如果高于 16 位，直接跳转。

3) 当遇到偏移的时候，保存原指令要跳转的地址。然后跳转到花指令空间，然后在花指令执行完的地方增添一个跳转到原先要跳转的那个地址。

其中重要的记录乱序指令的结构体如下：

```
typedef struct _CODE_FLOW_NODE
{
    struct  _CODE_FLOW_NODE  *pNext;//下一个节点
    BOOL bGoDown;//是否向下跳
    DWORD dwBits;//跳转范围
    DWORD dwType;//指令类型
    BOOL bFar;//是否是远跳
    DWORD dwMemoryAddress;//当前内存地址
    LPBYTE pFileAddress;//当前文件地址
    DWORD  dwGotoMemoryAddress;//跳转后的内存地址
```

```
        LPBYTE  pGotoFileAddress;//跳转后的文件地址
        DWORD dwInsLen;//指令长度
        union
        {
          BYTE bOffset;
          WORD wOffset;
          DWORD dwOffset;
        };//偏移
    }CODE_FLOW_NODE,
*PCODE_FLOW_NODE;
```

5. 模型测试

5.1. 功能测试

本文采取实验对比方式，即用其它加壳软件对 PE 文件加壳和本系统对 PE 文件加壳，然后用脱壳、调试软件来分别对这些被加了壳的 PE 文件进行检测或者脱壳。通过实验得出的对比结果如表 1 所示，表中“fail”表示不能被逆向分析，“success”表示能被逆向分析。

其他加壳软件：ACProtect、ASPack、ASProtect、PECompact。

脱壳工具：File Scanner、FileInfo_V4.02、PEid。

从表 1 中实验结果数据来看，本系统在很大程度上提升了软件的防御能力，从侧面反应了基于融合的壳软件保护模型具有很强的软件保护能力。

5.2. 性能测试

为了查看可执行文件加壳前后内部程序数据的变化，本文使用 LordPE 工具来打开加壳其后的可执行文件，如图 8 所示，红色的字段就表示有变化。

该可执行文件加壳前后部分数据变化可以从图

Table 1. Resulting data
表 1. 结果数据

	ACProtect	ASPack	ASProtect	PECompact	本系统
File Scanner	1	3	4	7	8
PEid	2	4	3	5	9
FileInfo_V4.02	3	7	6	7	8

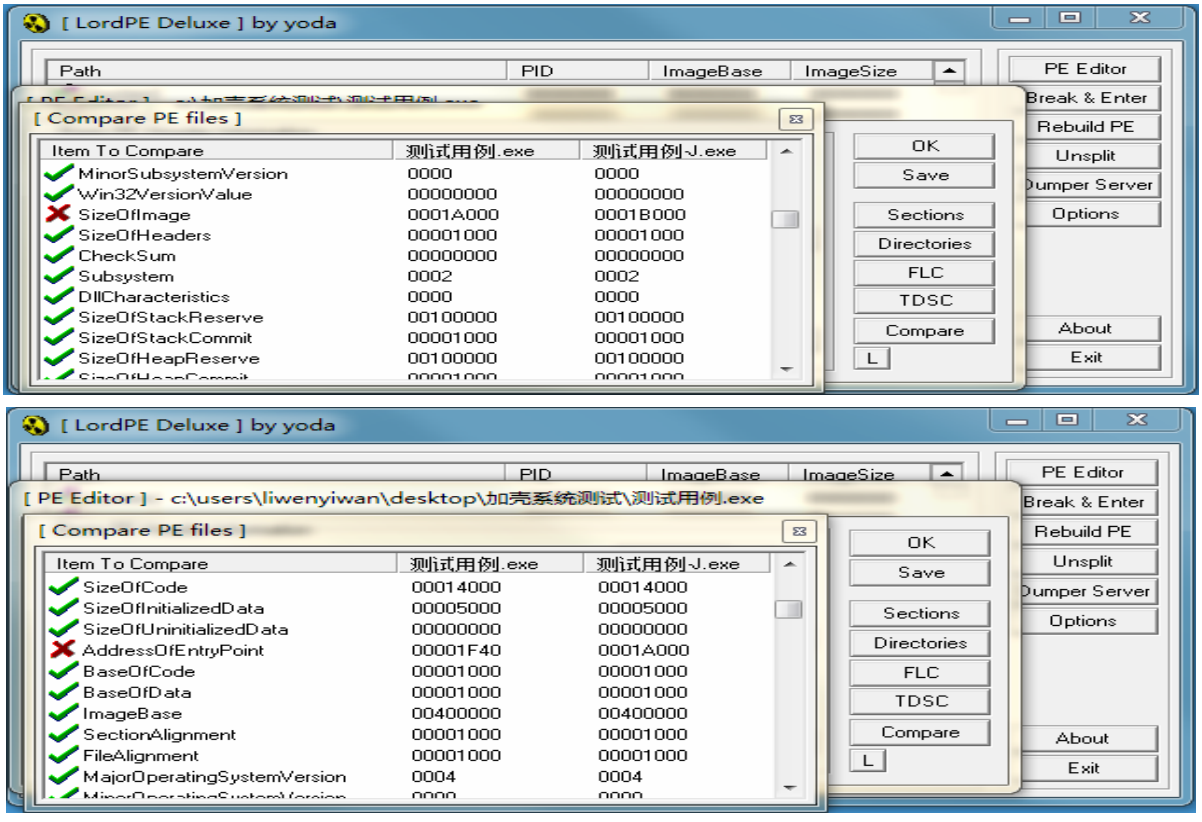

Figure 8. Test cases data comparison of .exe files before and after packing
图 8. 测试用例.exe 文件加壳前后数据对比

中看到。其中图 8 中给出了加壳过程中数据字段最明显的变化：程序入口地址(AddressOfEntryPoint)、内存中映像文件大小(SizeOfImage)的改变，其中以前的偏移地址是 00001F40，加壳之后的偏移地址为 0001A000。

6. 结束语

本文研究课题是基于壳技术的软件保护，从软件保护背景开始，深入分析研究了加壳技术和脱壳技术，提出了一种新的基于壳技术的软件保护模型。该软件保护模型具有很强前瞻性，本文认为如果外壳程序与原程序融合得越紧密，那么起到保护软件的效果就越好，让外壳与原程序成为一体是最终的趋势。所以找到实现该模型的解决方案是关键，本文提出的基于代码乱序的解决方案只是基本上初步实现了该模型，从一定程度上验证该模型的可行性，但是还没有找到更好的解决方案来实现该模型，并且其应用还需要更多的时间研究与检验。

参考文献 (References)

[1] 赵东方. 基于壳技术的软件保护技术研究[D]. 同济大学, 2009.
[2] 锻钢. 加密与解密(第三版)[M]. 北京: 电子工业出版社, 2008.
[3] Portable executable. http://en.wikipedia.org/wiki/Portable_Executalbe
[4] 秦杰. 基于 IAT 加密的加壳程序研究[D]. 电子科技大学, 2009.
[5] 张跃洋. 基于软件壳的研究与实现[D]. 电子科技大学, 2008.
[6] 姚为光. 软件加壳技术的研究[D]. 电子科技大学, 2011.
[7] 徐武华. 软件保护与分析技术的研究与实现[D]. 北京邮电大学, 2011.
[8] J. Wang. Hardware-assisted protection and isolation. Virginia: George Mason University, 2011.
[9] C. S. Collberg, C. Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. IEEE Transactions on Software Engineering, 2002, 28(8): 735-746.
[10] Microsoft. Microsoft portable executable and common object file format specification revision8.2_September21, 2010@2005-2010 Microsoft Corporation.
[11] 微软. Windows IFS kit and DDK 3700. Microsoft Corporation, 2003
[12] H. Yin, C. Lin, F. Qiu, et al. A survey of digital watermarking. Journal of Computer Research and Development, 2005, 42(7): 1093-1099.

