• 计算机工程与应用 •

静态修改 PE 输入表注入 DLL 的检测方法研究

于永斌[1*]，余文健[1]，莫洁虹[1]，康峥非[2]

(1. 电子科技大学信息与软件工程学院 成都 610054；2. 电子科技大学成都学院 成都 611731)

【摘要】该文研究静态修改 PE 输入表注入 DLL 的检测，提出了基于合法范围的普通检测方法和基于异常回溯的深度检测方法。第一种方法从静态的角度，对所有 DLL 的数据结构排列范围进行计算，无需解析 DLL 的功能来推断其是否恶意。第二种方法将调试的思想用于恶意 DLL 检测，控制目标程序的运行，跟踪目标程序初始化阶段中的 DLL 加载过程，并将调试 API 用于异常捕获，以实现检测。使用 C++设计 DLL 检测实验，将编写的具有下载功能的 DLL 注入到目标程序，设计开发检测工具 DLL Detector 进行检测；实验成功地从静态阶段和程序初始化阶段检测出可疑模块。两种方法均支持 32 位和 64 位可执行文件，可防御恶意代码。

关 键 词 DLL 检测； DLL 注入； 输入表； PE 文件格式

中图分类号 TP309.5 文献标志码 A doi:10.12178/1001-0548.2019301

Research on Detection of Dynamic Link Library Injected by Static Modifying Import Table of Portable Executable File

YU Yong-bin[1*], YU Wen-jian[1], MO Jie-hong[1], and Kang Zheng-fei[2]

(1. School of Information and Software Engineering, University of Electronic Science and Technology Chengdu 610054;
2. Chengdu College of University of Electronic Science and Technology of China Chendu 611731)

Abstract To study the detection of dynamic link library (DLL) injected by static modifying import table of portable executable (PE) file, a common detection method on legal scope and a depth detection method on exception backtracking are proposed. The first method calculates the range of data structure arrangement of all DLLs from a static point of view, without parsing the DLL's function to infer whether it is malicious. The idea of debugging is used to detect malicious DLLs in second method, which control the running of the target program, and track the DLL loading process in the initialization phase of the target program. Also the debugging API is used for exception capture to realize detection. C++ was used to design DLL detection experiment: injected the DLL with download function into the target program. The detection tool DLL Detector was designed and developed for detection. The experiment successfully detects suspicious modules from the static phase and the program initialization phase. Both methods support 32-bit and 64-bit PE files and can be used to guard against malicious code.

Key words DLL detection; DLL injection; import table; PE file format

DLL 可用于扩展应用程序功能和减少应用程序体积，是 Windows 平台提供的一种模块共享和重用机制，它不能直接独立运行，但可以被加载到其他进程中执行[1]。基于这种特性，DLL 常被用于注入进程实现恶意行为。DLL 的注入方式有很多种，如远线程注入、Windows 消息钩子、修改注册表和静态修改 PE 输入表注入等。通过挂钩技术[2-5]，能检测到前 3 种注入行为，而静态修改 PE 输入表注入 DLL 不会使用相关 API，因此不能通过挂钩技术进行检测。除了挂钩技术，检测 DLL 是否恶意的方法还有动态检测和二进制数据检测。

从动态角度进行检测，方法有：模块枚举[6]和 DLL 抢占式注入[7]。这些方法均需使被检测程序完全进入运行状态，程序一旦进入运行状态，就会产生恶意行为或反检测行为。“环境感知”型恶意软件可以在运行期间检测运行环境，确定自己是否

收稿日期：2019－12－27；修回日期：2020－04－12
基金项目：国家自然科学基金国际青年科学家研究基金 (61550110248)；四川省科技厅重大科技专项 (2019YFG0190)
作者简介：于永斌 (1975－)，男，博士，副教授，主要从事人工智能方面的研究. Email：ybyu@uestc.edu.cn

被分析，以便修改自身行为逃避检测[8]。文献 [6] 从动态的角度检测静态修改 PE 输入表注入的 DLL，通过运行未被修改的程序，遍历其中的模块建立合法模块列表；然后再运行被修改的程序，遍历模块进行对比，不同者即为可疑模块。但是程序中的模块必定随着版本更新和需求变化而相应地增加或减少，这种检测方法就会失效。基于此，本文研究了 DLL 的加载原理和过程，提出了在进程初始化阶段进行检测的方法，无需使被检测程序完全进入运行状态。

从二进制数据角度进行检测，方法有：提取字符串或函数调用[9-10]、提取 VERSIONINFO 资源[11] 和利用卷积神经网络进行特征提取[12] 等。这些方法均需要解析 DLL 获取具体数据与良性 DLL 进行对比，以判断该 DLL 是否恶意。但是 DLL 容易被指令等价替换处理[13]，或者使用加壳压缩[14-15] 等技术改变原始结构，使得以上方法无法提取到正确信息。基于此，本文研究了 PE 文件输入表的数据排列规则，提出了从数据范围的角度进行检测的方法，无需获取 DLL 内部的具体数据。

本文研究静态修改 PE 输入表注入 DLL 的检测方法，其主要工作归纳如下：

1) 提出了基于合法范围的普通检测方法和基于异常回溯的深度检测方法。基于合法范围的普通检测方法对所有 DLL 的数据结构排列范围进行计算，无需解析 DLL 中的函数调用和字符串等具体信息来推断 DLL 的功能是否恶意。基于异常回溯的深度检测方法可控制目标程序的运行，跟踪目标程序的初始化阶段中的 DLL 加载过程。2) 将调试的思想用于恶意 DLL 检测。调试 API 用于编写调试工具，捕获程序运行中的错误，从而帮助开发者修改错误提高程序的健壮性。本文将调试 API 用于 DLL 加载过程中的异常捕获，从而实现检测功能。3) 开发检测工具 DLL Detector，支持检测 32 位和 64 位可执行文件，并具有清除、恢复和备份功能。

1　PE 文件格式

PE 文件格式是 Windows 系统下可执行文件的格式，exe、dll、sys 等文件均属于该格式。PE 文件包含了可执行文件需要使用的所有信息，这些信息以二进制的形式按一定规则存放在一起。

图 1 为 PE 文件格式图。PE 文件主要包括 DOS 部分、PE 文件头、节表、节数据等。PE 文件结构在磁盘和内存中是相同的，这是因为被载入内存中的 PE 文件的数据结构不会改变，改变的只是数据之间的偏移[16]。PE 文件格式，不仅常用于二进制攻防领域，在生成胎记来防止盗版软件方面也发挥了巨大作用[17-19]。

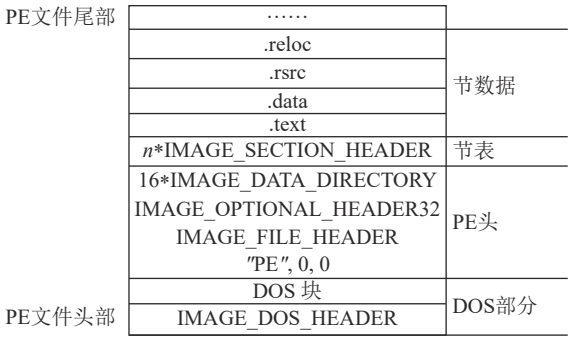

图 1　PE 文件格式

PE 头的末尾部分，是数据目录表，它是一个 IMAGE_DATA_DIRECTORY 结构数组，一共有 16 项，指向输出表、输入表、资源块等数据。该结构定义如下：

```
typedef struct _IMAGE_DATA_DIRECTORY
{
DWORD VirtualAddress;
DWORD Size;
}IMAGE_DATA_DIRECTORY,    *PIMAGE_DATA_DIRECTORY;
```

其中 VirtualAddress 字段是指向对应数据的 RVA(relative virtual address)，Size 字段是其指向的数据的大小。

数据目录表的第 2 项对应于输入表，此时其 VirtualAddress 字段指向一个 IID (IMAGE_IMPORT_DESCRIPTOR) 结构数组，该结构定义为：

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR
{
union
{
DOWRD Characteristics;
DWORD OriginalFirstThunk;
}
DWORD TimeDateStamp;
DWORD ForwarderChain;
DWORD Name;
DWORD FirstThunk;
```

}IMAGE_IMPORT_DESCRIPTOR, *PIMAGE_IMPORT_DESCRIPTOR;

该结构重要的字段是第 1 个、第 4 个和最后 1 个，分别是指向导入名称表 (INT)、DLL 名称和导入地址表 (IAT) 的 RVA。

每个 IID 结构对应于一个隐式链接的 DLL，当 PE 文件被载入内存时，操作系统会处理并加载输入表中指定的 DLL 模块，并填充程序的 IAT。所以如果能在输入表被处理之前，向输入表中增加一个项目，使其指向要加载的目标 DLL，那么新进程的主线程在输入表初始化阶段就会主动加载目标 DLL。

2 DLL Detector 的原理与设计

使用 C++语言和 MFC 开发检测工具 DLL Detector。图 2 是 DLL Detector 的界面示意图，scan 按钮实现文件选择功能；中间的列表视图控件用于显示文件的输入表数据；底部的 4 个按钮分别为普通检测、深度检测、清除+保存、恢复，两个复选框为置顶和备份。下文将详细介绍 DLL Detector 主要功能的设计原理。

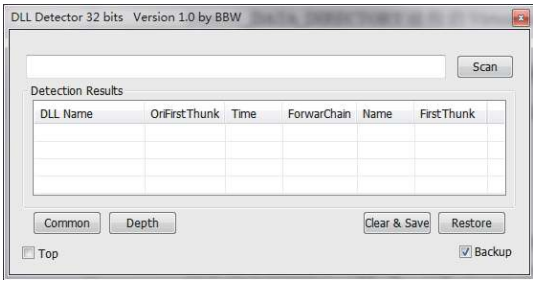

图 2 DLL Detector 检测工具界面

2.1 读取并解析 PE 文件输入表

调用 CreateFile 打开目标 PE 文件，调用 CreateFileMapping 和 MapViewOfFile 创建内存映射文件以获得目标 PE 文件在内存中的首地址。

根据首地址并按照 PE 文件格式得到输入表 IID 结构数组的地址，循环遍历每个 IID 结构即可获取目标 PE 文件的具体输入表数据，呈现在工具中部的列表视图控件上。

2.2 基于合法范围的普通检测方法的设计原理

本文深入研究了以 notepad.exe 为代表的 PE 文件的数据排列规则，发现通过编译链接形成的 PE 文件，其输入表的具体数据是排列在一起的。图 3 为 notepad.exe 所有合法 DLL 的 IAT 数据图，图 4 为 notepad.exe 所有合法 DLL 的 INT 数据图。由图可知，通过正常流程生成的可执行程序，其输入表数据按顺序排列，符合一定规则。而手动添加 DLL 的相关数据，位置在节空隙、节扩展部分或新节，导致与合法数据具有不一致性，这为检测非法模块提供了新的思路，本文根据这种思路提出了基于合法范围的普通检测方法。

```
00000400h: C8 6F DA 77 F0 6B DA 77 7D 8F DC 77 FD D5 DC 77
00000410h: 83 78 DA 77 1B 76 DA 77 CC D7 DA 77 00 00 00 00
00000420h: ED D2 18 77 00 00 00 00 23 59 F0 77 12 34 F2 77
00000430h: B1 5B F0 77 A6 6C EF 77 A6 6A F0 77 74 81 EF 77
00000440h: CF F8 F0 77 32 35 F2 77 79 B0 EF 77 E8 7C EF 77
00000450h: A2 40 F2 77 7C EA F1 77 F1 5F EF 77 23 83 EF 77
00000460h: A2 58 EF 77 BA 8B EF 77 3B 6A EF 77 F5 7B EF 77
00000470h: 0B 5D EF 77 95 81 EF 77 B6 E3 F0 77 5F E4 F0 77
00000480h: F7 A8 EF 77 A0 59 EF 77 00 00 00 00 37 97 80 7C
00000490h: AC 92 80 7C 17 A4 80 7C C1 C9 80 7C C0 9F 80 7C
000004a0h: 79 7C 82 7C D5 84 82 7C 19 01 81 7C 82 00 81 7C
000004b0h: 85 0E 81 7C 9E 93 80 7C E5 17 80 7C 16 1E 80 7C
000004c0h: 0D E0 80 7C 86 03 81 7C 77 1D 80 7C 29 B5 80 7C
000004d0h: EE 1E 80 7C 2F FE 80 7C 72 17 81 7C 5D 99 80 7C
.....
```

图 3 所有合法 DLL 的 IAT 数据

```
00006ac0h: 00 00 00 00 00 00 00 00 00 00 00 00 A2 7C 00 00
00006ad0h: B6 7C 00 00 C4 7C 00 00 D4 7C 00 00 E4 7C 00 00
00006ae0h: F8 7C 00 00 90 7C 00 00 00 00 00 00 48 7B 00 00
00006af0h: 00 00 00 00 D0 81 00 00 C4 81 00 00 BA 81 00 00
00006b00h: AE 81 00 00 DA 81 00 00 8A 81 00 00 7E 81 00 00
00006b10h: F2 81 00 00 5E 81 00 00 A2 81 00 00 E6 81 00 00
00006b20h: 50 81 00 00 3E 81 00 00 30 81 00 00 20 81 00 00
00006b30h: 0A 81 00 00 FA 80 00 00 02 82 00 00 14 82 00 00
00006b40h: 20 82 00 00 2A 82 00 00 3C 82 00 00 50 82 00 00
00006b50h: 6E 81 00 00 00 00 00 00 24 80 00 00 14 80 00 00
00006b60h: FA 7F 00 00 EA 7F 00 00 D4 7F 00 00 C2 7F 00 00
00006b70h: B0 7F 00 00 A2 7F 00 00 92 7F 00 00 74 7F 00 00
00006b80h: 5E 7F 00 00 3A 80 00 00 54 80 00 00 68 80 00 00
00006b90h: 98 80 00 00 B6 80 00 00 C6 80 00 00 DA 80 00 00
.....
```

图 4 所有合法 DLL 的 INT 数据

数据目录表的第 13 项是 Import Address Table，其 VirtualAddress 字段指向所有合法 DLL 的 IAT，而 INT 并没有由一个表项来指定，因此需要使用算法进行解析对比。

检测模块的 IAT 是否在合法 IAT 之内的核心代码如下：

dwIAT 为需要检测的模块的 IAT。

BOOL bResult1 = dwIAT < pImportTable->VirtualAddress || dwIAT > pImportTable->VirtualAddress + pImportTable->Size;

检测模块的 INT 是否在合法 INT 之内的核心代码如下：

m_pDosHeader 是 PE 文件的 DOS 头部；

dwINTStart 是合法 INT 的头部，来自于所有 DLL 模块中的最小 INT 的模块；

dwINTLast 来自于所有 DLL 模块中的最大 INT 的模块，意为该模块的头部；

dwINTEnd 是合法 INT 的尾部；

RVA2FileOffset 函数是将 RVA 转换为文件偏移；

FileOffset2RVA 函数是将文件偏移转换为 RVA。

```
BOOL IsInLegalINT(DWORD dwINTtoTest)
{
DWORD dwFileOffsetOfLastINT = RVA2FileOffset(dwLastINT);
d = (DWORD*)((DWORD)m_pDosHeader + dwFileOffsetOfLastINT);
//一个模块的 INT 以 0x00000000 结尾
while (*d){ d++; }
//计算出合法 INT 的结尾
dwINTEnd = FileOffset2RVA((DWORD)d - (DWORD)m_pDosHeader);
//测试某个模块的 INT 是否在合法 INT 范围内
//不在则返回假，反之返回真
if (dwINTtoTest < dwINTStart || dwINTtoTest > dwINTEnd)
return FALSE;
return TRUE;
}
```

2.3　基于异常回溯的深度检测方法的设计原理

为了处理软件中的异常，程序员可以使用异常处理[20-22]。发生异常时，可以将其恢复到正常状态或自动终止，而不影响操作系统。

基于合法范围的普通检测方法不涉及异常，而基于异常回溯的深度检测方法则与异常有关。前一种方法的有效性在于恶意人员修改 PE 文件的范围不大，其欲注入 DLL 的对应数据只是简单的添加在节空隙、节扩展部分或新节。而对于更恶意的情况——恶意人员将数据伪造于合法数据范围内，该方法将失效。基于此，本文提出了基于异常回溯的深度检测方法，目标程序被主动触发产生异常来达到检测目的。

基于异常回溯的深度检测方法的原理如下：

1) 程序编译并链接后，输入表中对应的 DLL 均是程序自身所需的 DLL。程序在运行时需要调用其库函数，因此程序与这些 DLL 具有主动关联性；2) 注入 DLL 的目的只是在加载时执行 DllMain 中的代码。程序自身没有任何代码调用其库函数，因此程序与注入的 DLL 具有被动关联性。

图 5 为 exe 与 DLL 的关联图，可见 exe 与合法 DLL、注入的 DLL 有不同的依赖关系。

图 6 为清除程序所需 DLL 后引发 0xC0000005 异常的结果图。由图 5 可知，若清除了程序所需要的 DLL，进程创建时该 DLL 不会被映射到进程的虚拟地址空间，程序原有的调用该 DLL 函数的代码就会因访问无效的地址而导致进程出现 0xC0000005 访问异常；若清除了注入的 DLL，由于程序没有任何主动调用其库函数的代码，所以不会导致进程出现 0xC0000005 访问异常。

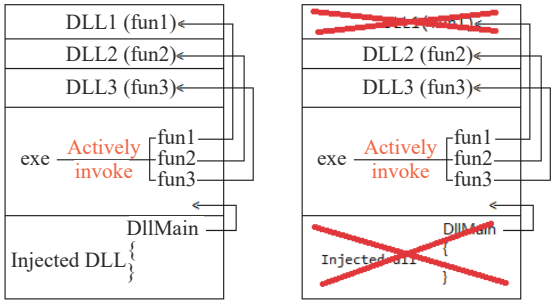

图 5　exe 与 DLL 的关联图

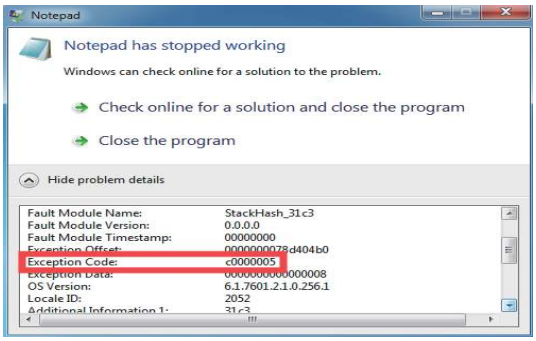

图 6　清除程序所需 DLL 后引发 0xC0000005 异常

根据以上描述，检测时可以以调试模式创建子进程，接管该子进程发生的 EXCEPTION_DEBUG_EVENT 类型异常。图 7 为深度检测方法流程图，如果发生了 EXCEPTION_DEBUG_EVENT 异常且异常类型码为 0xC0000005，说明清除了进程所需的 DLL。此时就需要恢复其输入表中的 DLL 项，然后清除下一个 DLL 项，再次以调试模式创建子进程。整个过程以输入表的表项作为循环次数。如果没有发生该异常，说明清除了无关的 DLL，即可证明其是注入的 DLL。

由于进程的创建模式是调试模式，父进程可以控制子进程运行与否，所以在整个深度检测期间，父进程可以完全控制子进程初始化阶段的 DLL 加载工作。一旦发现可疑 DLL，父进程可以终止子进程，这样就能避免可疑 DLL 做出后续的感染行为。

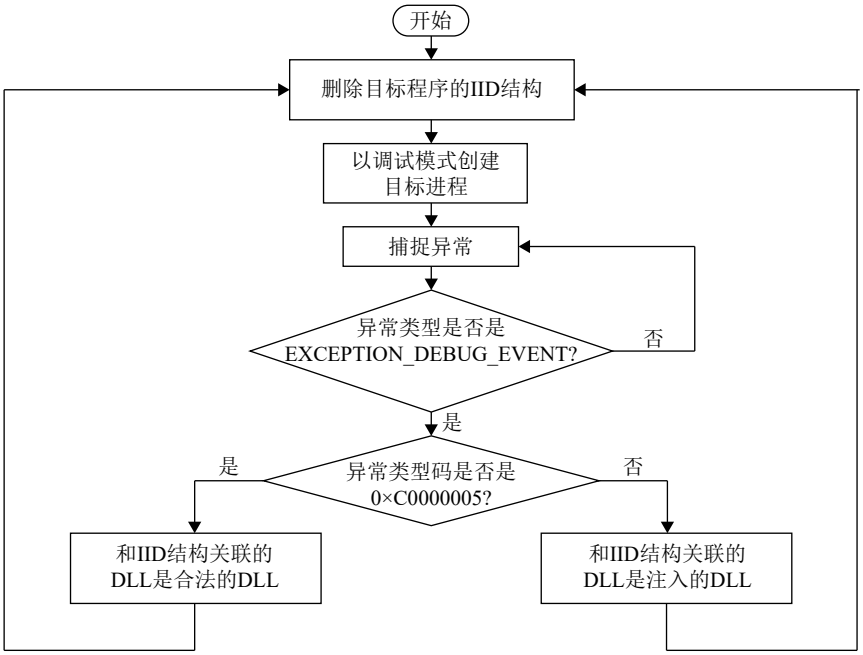

图 7 深度检测方法的流程

3 DLL 检测实验

本文对 Windows XP 下的 32 位 notepad.exe 和 Windows 7 下的 64 位 notepad.exe 同时进行了修改。修改版本 notepad_1.exe 为手动添加的 DLL 相关数据不在合法数据范围内，修改版本 notepad_2.exe 为手动添加的 DLL 相关数据伪造于合法数据范围内。本文以 Windows XP 下的 32 位修改版本 notepad_1.exe 和 notepad_2.exe 为例。欲注入的 DLL 为自备的 DownLoad.dll，其主要功能是在 DllMain 中执行相关代码，从网上下载一个文件到系统目录。

3.1 使用基于合法范围的普通检测方法进行检测

使用 DLL Detector 的普通检测功能分别对 notepad_1.exe 和 notepad_2.exe 进行检测。

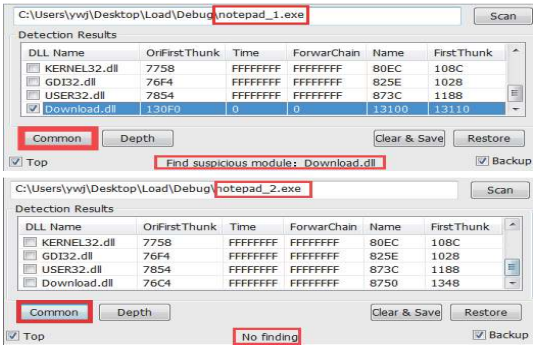

图 8 使用基于合法范围的普通检测方法的检测结果

图 8 为基于合法范围的普通检测方法针对 notepad_1.exe 和 notepad_2.exe 的检测结果，实验结果表明，手动添加的 DLL、相关数据不在程序本身的合法数据范围内，可以根据此方法检测出可疑模块，而相关数据被伪造于合法数据范围内，此检测方法将失效，可疑模块将绕过此方法实现其恶意行为。

3.2 使用基于异常回溯的深度检测方法进行检测

使用 DLL Detector 的深度检测功能分别对 notepad_1.exe 和 notepad_2.exe 进行检测。

图 9 为基于异常回溯的深度检测方法针对 notepad_1.exe 和 notepad_2.exe 的检测结果，可见，不管数据有没有被伪造于合法数据范围内，基于异常回溯的深度检测方法均成功检测出可疑模块。

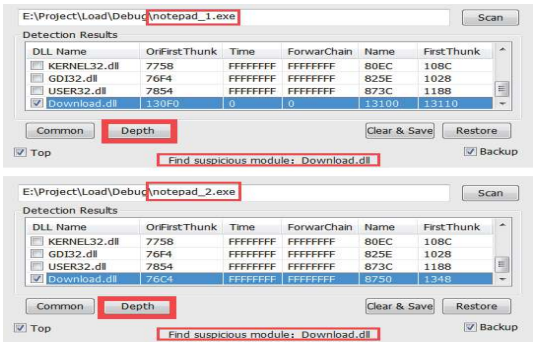

图 9 使用基于异常回溯的深度检测方法的检测结果

4 结 束 语

本文针对静态手动修改 PE 输入表注入 DLL 技术，提出了基于合法范围的普通检测方法和基于异常回溯的深度检测方法，能够有效检测出可疑模

块，为防御该注入技术提供了新的思路。两种方法同时支持检测 32 位和 64 位程序。

两种检测方法依然具有一些缺陷。普通检测方法的缺陷在于恶意数据被伪造于合法数据范围内，导致检测失效；深度检测方法的缺陷在于会耗费大量的时间进行异常捕获和进程重启。所以本文研究的两种方法并不完美，一般不作为首选，但可作为检测技术的补充。

将来还会致力于其他 DLL 注入技术的防御研究，同时会深入研究 PE 文件结构，以做出更好的保护措施。

参 考 文 献

[1] BERDAJS J, BOSNIC Z. Extending applications using an advanced approach to DLL injection and API hooking[J]. Software: Practice and Experience, 2010, 40(7): 567-584.

[2] SHEN Jian-fang, CHENG Liang-lun, FU Xiu-fen. Implementation of program behavior anomaly detection and protection using Hook technology[C]//2009 WRI International Conference on Communications and Mobile Computing. Yunnan, China: IEEE, 2009, 3: 338-342.

[3] LIU Xin, LIU Ren-ren, WU Xiang-bo. A secret inline Hook technology[C]//2013 8th International Conference on Computer Science & Education. Colombo, Sri Lanka: IEEE, 2013: 913-916.

[4] SONG Yu-chen, SHEN Yong-jun, ZHANG Gui-dong. The new INLINE Hook technology combination of hard-code technology and independent code injection[C]//2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS). Beijing, China: IEEE, 2016: 521-525.

[5] YOSHIZAKI K, YAMAUCHI T. Malware detection method focusing on anti-debugging functions[C]//2014 Second International Symposium on Computing and Networking. Shizuoka, Japan: IEEE, 2014: 563-566.

[6] 陈庄, 王津梁, 张騛. 手工 DLL 注入的检测方法研究实现[J]. 信息安全研究, 2017, 3(3): 246-253.
CHEN Zhuang, WANG Jin-liang, ZHANG Ti. Research and implementation of detection method of manual DLL injection[J]. Journal of Information Security Research, 2017, 3(3): 246-253.

[7] GUO Yu-cheng, WU Peng, LIN Ju-wei, et al. A way to detect computer trojan based on DLL preemptive injection[C]//2011 10th International Symposium on Distributed Computing and Applications to Business, Engineering and Science. Wuxi, China: IEEE, 2011: 255-258.

[8] SHAID S, MAAROF M. In memory detection of Windows API call hooking technique[C]//2015 International Conference on Computer, Communications, and Control Technology (i4CT). Kuching, Malaysia: IEEE, 2015: 294-298.

[9] JOPHIN S, VIJAYAN M, DIJA S. Detecting forensically relevant information from PE executables[C]//2013 International Conference on Recent Trends in Information Technology (ICRTIT). Chennai, India: IEEE, 2013: 277-282.

[10] KI Y, KIM E, KIM H K. A novel approach to detect malware based on API call sequence analysis[J]. International Journal of Distributed Sensor Networks, 2015, 2015: 1-9.

[11] JANG M, KIM H, YUN Y. Detection of DLL inserted by Windows malicious code[C]//2007 International Conference on Convergence Information Technology (ICCIT 2007). Gyeongju, South Korea: IEEE, 2007: 1059-1064.

[12] POONGUZHALI N P, RAJAKAMALAM T, UMA S, et al. Identification of malware using CNN and bio-inspired technique[C]//2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN). Pondicherry, India: IEEE, 2019: 1-5.

[13] 吴伟民, 范炜锋, 王志月, 等. 基于特征码的 PE 文件自动免杀策略[J]. 计算机工程, 2012, 38(12): 118-121.
WU Wei-min, FAN Wei-feng, WANG Zhi-yue, et al. PE file auto free-antivirus strategy based on characteristic code[J]. Computer Engineering, 2012, 38(12): 118-121.

[14] LI Ang, ZHANG Yue, ZHANG Jun-xing, et al. A token strengthened encryption packer to prevent reverse engineering PE files[C]//2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF). Harbin, China: IEEE, 2015: 307-312.

[15] LI Lu, LIU Qiu-ju, XU Ting-rong. Research and implementation of compression shell unpacking technology for PE file[C]//2009 International Forum on Information Technology and Applications. Chengdu, China: IEEE, 2009, 1: 438-442.

[16] YOUSAF M S, DURAD M H, ISMAIL M. Implementation of portable executable file analysis framework (PEFAF)[C]//2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST). Islamabad, Pakistan: IEEE, 2019: 671-675.

[17] KIM Y, MOON J, CHO S J, et al. Efficient identification of windows executable programs to prevent software piracy[C]//2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. Birmingham, UK: IEEE, 2014: 236-240.

[18] CHOI J C, HAN Y M, CHO S, et al. A static birthmark for MS windows applications using import address table[C]//2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. Taichung, Taiwan, China: IEEE, 2013: 129-134.

[19] WANG Xin-ran, JHI Y C, ZHU S, et al. Detecting software theft via system call based birthmarks[C]//2009 Annual Computer Security Applications Conference. Honolulu, HI, USA: IEEE, 2009: 149-158.

[20] CABRALl B, MARQUES P. A transactional model for automatic exception handling[J]. Computer Languages, Systems & Structures, 2011, 37(1): 43-61.

[21] CHANG H, MARIANI L, MAURO P. Exception handlers for healing component-based systems[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2013, 22(4): 30.

[22] GEORGE V, NEVILLE N. Forced exception handling[J]. Communications of the ACM, 2017, 60(6): 31-32.

编　辑　漆　蓉