动静结合的攻击代码检测方法

赵 帅[1,2] 丁保贞[1,2] 沈备军[1] 林九川[2]
(上海交通大学软件学院 上海 200240)[1] (公安部第三研究所 上海 201204)[2]

摘 要 缓冲区溢出攻击是近年来最主要的安全问题之一,攻击者利用缓冲区溢出漏洞执行远程代码,从而达到攻击的目的。Shellcode作为攻击的载体,是缓冲区攻击检测的主要对象。随着检测技术的发展,攻击者更倾向于使用多态技术对Shellcode进行加密来绕过IDS的检测。针对MS Windows操作系统下的Shellcode,提出了一种将静态检测和动态执行相结合的新的攻击代码检测方法。在判断依据上做了新的定义,既使用动态模拟技术提高了对使用多态技术的Shellcode的检测率,也兼顾了检测的效率。基于该方法,设计和实现了一套原型系统,并进行了检测率、误报率和吞吐率等方面的测试。测试结果表明,该系统在检测Shellcode的准确率和性能方面都达到了令人满意的效果。

关键词 Shellcode,静态分析,动态执行

Method of Shellcode Detection Based on Static and Dynamic Mechanism

ZHAO Shuai[1,2] DING Bao-zhen[1,2] SHEN Bei-jun[1] LIN Jiu-chuan[2]
(School of Software, Shanghai Jiaotong University, Shanghai 200240, China)[1]
(The Third Research Institute of Ministry of Public Security, Shanghai 201204, China)[2]

Abstract Buffer overflow attack has been a major security problem in recent years, where attackers utilize buffer overflow vulnerabilities to control other computers. As the vehicle of attack, Shellcode is the main target of buffer overflow attack detections. Now attackers tend to employ polymorphic techniques to encode Shellcode, which makes it harder for signature-based NIDS to detect it. This paper proposed a new method to detect the Shellcode executed under MS Windows, which integrates static analysis and dynamic execution techniques. It made some new principles of Shellcode detection, which enhance both the accuracy and performance of polymorphic Shellcode detection. Then a prototype system was implemented and tested. The test results on both the accuracy and performance are quite encouraging.

Keywords Shellcode, Static analysis, Dynamic execution

1 引言

缓冲区溢出攻击是近年来最主要的安全问题之一,攻击者利用缓冲区溢出漏洞执行远程代码,从而达到攻击的目的。Shellcode是一段代码,它常常是缓冲区溢出攻击的载体,所以检测Shellcode是检测缓冲区溢出攻击的有效手段。传统的Shellcode检测方式可以分为静态和动态两种。静态的检测方式是基于签名的检测,比较知名的工具有Snort[4]和Bro[5]等;动态的检测方法一般采用模拟执行。静态的检测是从数据流中提取出特征,然后根据特征去判断是否有Shellcode存在,如sled指令。Akritids等人通过分析7种sled指令,提出了相应的检测sled指令的方法,并开发了STRIDE来检测sled部分[1]。

随着多态技术在攻击代码上的应用,静态的检测技术在检测成功率上也逐渐显示出了不足,动态的虚拟执行的方式可以很好地解决这个问题。Zhang Qinghua提出了一种使用模拟执行的动态监测方法[2],即首先使用静态分析的方法查找getPC格式指令,然后使用从getPC格式指令开始查找解密程序的循环体,再根据两个属性:a)循环体中是否存在直接或间接的写地址操作,b)存储地址或偏移量的寄存器是否在循环体的执行过程中被更新来判断其是否包含恶意代码。Michalis Polychronakis[3]等人也提出了一种基于模拟执行的方法来检测使用了多态技术的Shellcode,并取得了良好的检测效果。

本文针对MS Windows操作系统下的Shellcode,提出了一种新的攻击代码检测方法。不同于以上的方法,本方法采用了静态和动态相结合的检测方法,在判决条件上做了新的定义,既综合了静态检测速度上的优势,又具有动态模拟技术在检测准确率上的优势。

2 Shellcode分析

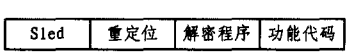

图1 Shellcode结构

通常,Shellcode可以分为4个部分:sled、重定位、解密程序、功能代码,其结构如图1所示。

1)sled

由于代码在执行过程中加载位置的不固定,通常在编写

到稿日期:2011-01-24 返修日期:2011-05-16 本文受信息网络安全公安部重点实验室开放课题资助。

赵 帅 硕士生,主要研究方向为信息安全,E-mail:zhaosspirit@sjtu.edu.cn;丁保贞 硕士生,主要研究方向为信息安全;沈备军 博士,副教授,主要研究方向为软件工程;林九川 助理研究员。

Shellcode 的时候，需要在 Shellcode 的开始添加一段 nop 指令或者类 nop 指令，来提高 Shellcode 的成功执行率，如使用 HeapSpray 技术的攻击代码；有时为了方便编写，会在之前添加一些调试代码，最后编写完成时将这些代码替换成 nop 指令或者类 nop 指令。但是 sled 部分并不是 Shellcode 的必要组成部分，比如在 MS Windows 下面的某些漏洞利用代码中，可以通过细致的编码来避免使用 sled，所以 sled 部分不是检测方法所关注的。

2)重定位

为了保证 Shellcode 能够正确地访问数据，需要获取当前指令的地址(PC)。一般来说有这样几种方式：一种是使用 call 指令，call 指令的执行过程中会将函数返回地址压入栈中，可以通过读取返回地址获得当前的指令地址；另一种是使用 fstenv 指令，保存当前的 FPU 操作环境；第三种是利用 MS Windows 的 seh 机制，这在 MS Windows 早期版本中适用。

3)解密程序

为了防止 Shellcode 被 IDS 检测到，在 Shellcode 的编写过程中会使用多态技术，使得 Shellcode 需要一段解密程序解密后才能正确执行。解密程序通常采用循环的方式对 Shellcode 进行异或等操作。

4)功能代码

功能代码是 Shellcode 成功运行后解密出来的最终代码。攻击者通过这些代码获取目标机器控制权，实现具体的攻击操作。

Shellcode 一般由这 4 个部分组成。静态检测的方法一般注重于 sled 部分和重定位部分。由于多态性导致解密程序的形式多样，因此无法准确地对这两个方面进行检测。在动态虚拟执行的检测方法中，重定位和解密程序是可以被虚拟执行的部分，所以通常这种方法注重于这两个部分。

3 一种新的检测方法

根据 MS Windows 下 Shellcode 的执行过程的特征，针对 MS Windows 下 Shellcode 的检测，本文提出了一种新的方法。方法的原理如下所述。

首先将从网络上截取数据流，利用静态扫描的方式从数据流中找到 Shellcode 的重定位相关指令。16 进制分别为 0xe8(call 指令)、0x9b(fstenv 指令)、0xd9(fnstenv 指令)，将这些指令作为虚拟执行的开始。

然后从这条指令开始，将指令放入虚拟机中虚拟执行，并检测其执行过程。如果网络流中存在 Shellcode，那么这些指令就可以连续地在虚拟机中得到执行，否则虚拟执行就会可能因为一些原因如非法指令、地址不可访问等而中断。因此可以通过观察在虚拟机中连续执行的指令的条数来作为一个判决条件。不过，仅仅利用这个条件往往会产生误报。为了提高准确率，增加了另外两个判决条件。

如果 Shellcode 中存在解密程序，虽然其形式各异，但是执行过程中会对 Shellcode 本身造成改变。通过比较原始的网络流和虚拟执行后的网络流，查看模拟执行 n 条指令后网络数据流本身的改变量 m。如果满足我们的判断条件，则说明可能存在 Shellcode。

Shellcode 中一般会存在获得 kernel32 基地址的操作，通过 kernel32 的加载地址和 PE 文件结构，可以获得 LoadLibrary()和 GetProcAddress()函数来继续加载其他的动态链接库。攻击者一般通过这种方法调用其他的 MS Windows API，达到攻击目的。获取 kernel32 基地址，常用的方法有通过 PEB 获取、通过 SHE 获取等。由于获取的方式有很多种，因此把网络字节流在虚拟执行过程中是否出现访问 kernel32 基地址行为作为判断依据，来降低误报率。

在执行检测程序之前使用 GetModuleHandle()获得 kernel32 的基地址，然后在虚拟执行的每一步中判断寄存器中是否出现了该地址。如果出现，则该序列中存在 Shellcode 的可能性就会增加。

总的来说，我们的检测方法是在动态虚拟执行过程中检测网络数据流是否满足如下 3 个特征。

1)虚拟执行步长；

2)虚拟执行后网络字节流改变量；

3)被改变的网络数据流中是否存在访问 kernel32 地址指令。

如果满足了这些特征，那么网络数据流中就有可能存在 Shellcode。如果满足如上 3 个条件中的一个或者两个，那么也可能存在 Shellcode，但是存在的几率也相对较低。

基于以上原理，设计出详细的检测方法，如图 2 所示。

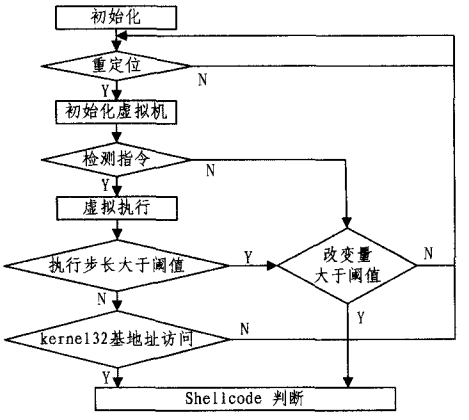

图 2 检测方法

在完成初始化之后，如果在网络流中检测到重定位指令，则初始化虚拟机，然后检测指令合法性。如果指令合法，则放到虚拟机中虚拟执行，否则计算网络流改变量。如果虚拟执行步长超过阈值，也会进行网络流改变量的计算。当网络流改变量大于阈值时，则认为满足了 Shellcode 判断的其中一个特征，同时在虚拟执行过程中观察是否出现访问 kernel32 基地址的指令。如果出现，则满足了 Shellcode 判断的另一个特征。满足的特征越多，说明网络流中出现 Shellcode 的可能性越大。

4 原型系统的设计和实现

基于所提出的检测方法，设计并实现了一个原型系统。

4.1 原型系统的设计

为了简化系统的实现过程，将网络字节流以文件方式存储，然后从文件中读取数据，从而进行反汇编并进行重定位指令的扫描。这一步由 ScanRelloc 完成。当检测到重定位指令时，系统会调用 InitVM 初始化虚拟机，然后调用 Valid 判断指令是否合法。如果合法，则调用 EXEC 放到虚拟机中去执行，并动态地监测虚拟执行状态。执行完成后再处理下一条指令。当虚拟执行步长超过配置文件中设置的虚拟执行步长

阈值或者指令不合法时，虚拟执行结束。系统会调用 ScanMem 比较原始数据流和虚拟执行后的数据流，并计算数据流的改变量。如果数据流改变量超过阈值，则该数据流中可能包含 Shellcode。如果在 EXEC 过程中监测到有访问 kernel32 基地址的行为，那么出现 Shellcode 的可能性增加。系统结构如图 3 所示。

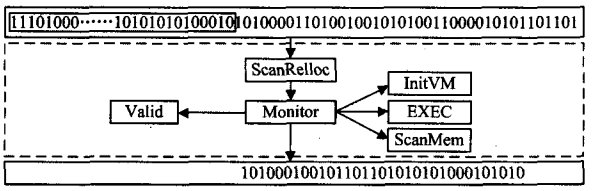

图 3　原型系统结构图

4.2　虚拟机的设计与实现

将一个填充了 nop 指令的进程作为虚拟执行的虚拟机，通过设置加载地址来模拟 Shellcode 加载位置不固定这一事实，使其重定位部分生效，这个地址是可以配置的。为了防止解密过程中出现非法操作，设置内存读写模式/SECTION:.text RWE。

MS Windows 提供了一些调试的接口，如 getDebugInfo，setThreadContext 等，通过这些接口来监测网络序列在虚拟机中的运行状态，获取与判决条件相关的数据。

当虚拟机遇到如下指令时，应该停止虚拟执行：1）Int 指令。2）跳转指令的目的地址位于不允许的范围内。3）写指令目的操作数地址在不允许的范围内。

5　测试

5.1　测试技术和方法

为了方便测试，把网络字节流以文件的方式作为检测程序的输入，同时在检测程序的配置文件中设置不同的参数，观察测试结果并进行综合比较和分析。

借助工具 Metasploit Framework v3.4[7] 产生测试用的 Shellcode。Metasploit Framework v3.4 提供了许多 Windows 下的 Shellcode 以及不同的加密方式。针对使用不同加密方式所产生的 Shellcode 均进行了检测率和漏报率的测试，并且选择了一些不包含 Shellcode 的数据流进行了误报率方面的测试。

5.2　测试结果分析

用控制变量法依次改变输入字节流大小、虚拟执行步长、网络流改变量阈值，观察实验结果。

1）检测率和漏报率的测试

首先将产生的 Shellcode 直接当作输入进行检测，测试其检测率和漏报率。图 4 所示是检测率和漏报率随指令改变量阈值变化的实验数据。从图中可以看出，当网络流改变量阈值为 50 时，检测率达到了 80%以上。

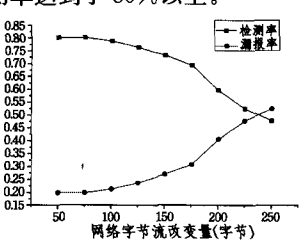

图 4　检测率和漏报率随网络流改变量阈值变化图

2）误报率的测试

对约 5GB 的普通数据流共 6740 个文件进行了测试，测试结果如图 5 所示。由测试结果可以看出，随着网络流改变量阈值增大，系统误报数逐渐减少。在网络流改变量阈值为 50 字节时，误报数为 24，误报率约为 0.0047 个/M。这个误报率是非常低的。然后用访问 kernel32 基地址的特征去进一步检测这些数据，均未出现该访问特征。

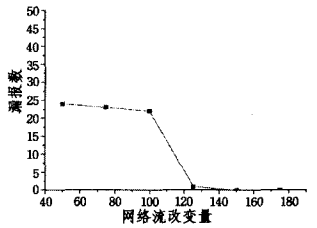

图 5　误报数随网络流改变量阈值变化图

3）吞吐率的测试

用同样的方法进行吞吐率的测试，测试结果如图 6 所示。由图 6 可以看出，随着虚拟执行步长和网络字节流改变量阈值的增大，吞吐率呈下降趋势。当设置网络字节流改变量阈值为 50 字节时，系统在一般情况下吞吐率为 10M/s。

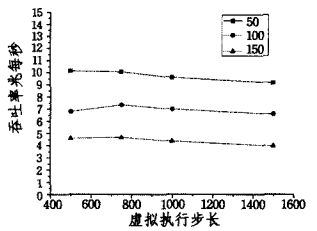

图 6　吞吐率变化图

结束语　本文针对 MS Windows 操作系统下的 Shellcode，提出了一种新的攻击代码检测方法，并实现了原型系统。本方法结合静态检测和动态执行技术，用静态的方式扫描重定位特征和访问 kernel32 基地址的特征，用动态虚拟执行的方式来监测其执行状态，既综合了静态检测速度上的优势，又具有动态监测准确率上的优势。

本文对所提出的方法和系统进行了检测率和性能等方面的测试。通过分析测试数据可以看出，这是一种实时检测 Shellcode 的有效方法，在检测使用了多态技术的 Shellcode 方面取得了比较好的结果。但是这种方法也有一些不足。首先，这种方法具有一定的针对性，主要针对 MS Windows 系统下使用了多态技术的 Shellcode。其次，这种方法在性能方面还需要进一步提升，一方面可以通过对各种参数的调整来达到一个比较好的检测效果，另一方面可以通过优化检测程序的性能来提高网络字节流的虚拟执行速度。

目前本文使用的是调试 API 的方式来获取指令虚拟执行的信息，这种方式在时间性能上存在一定的不足。在接下来的工作里，将使用其他方式，如使用 PIN[6] 来获取指令虚拟执行状态信息等。同时将注重误报率的降低，从而使系统性能得到提升。

参 考 文 献

[1] Akritidis P, Markatos E P, Polychronakis M, et al. Stride: Polymorphic Sled Detection Through Instruction Sequence Analysis [C]// IFIP Advances in Information and Communication Technology. 2005, 181: 375-391

（下转第 155 页）

时间为5s～7s。确定模式内语义近似属性间的关系所花费的时间较少，大约10ms。由于Books为出版物记录，在Web上可检索到book的相关信息，用GND来衡量实例相似性。当基于聚类的相似度θ小于0.3时，则需要再计算两聚类中心点间的距离作为语义相似性。表格中"—"表示无需计算或者不可计算。

表1 Books数据模式匹配

	书号	书名	作者	出版社	…
sh	0.2042,0.632	0.0001,0.031	—,0.014	0.0091,0	…
sm	0.0003,0.0	0.5904,—	0.8550,—	0.0162,0	…
zz	—,0.0301	0.2147,0.004	0.3000,0.7466	0.2809,0	…
cbs	—,0.0143	0.0904,0.031	0.3035,0	0.3072,0.8	…
…	…	…	…	…	…

另外，在商品信息集合和机械产品集合上也进行了实验。将本文所提出的方法与基于实例的DUMAS方法[6]在重复记录数量不同时的性能进行了比较，在重复记录比较少时，所提出的匹配方法MIC性能较好(见图5)。

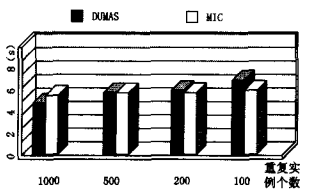

图5 重复实例个数与匹配时间的关系

将方法MIC与Cupid方法相结合，与基于实例提取部分函数依赖的PFD方法[8]进行比较，本文方法在查准率、查全率、全面性方面(见图6、图7)有所提高。

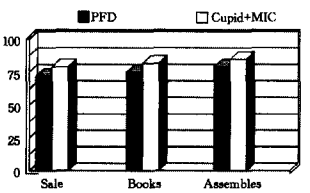

图6 PFD与Cupid+MIC查准率对比

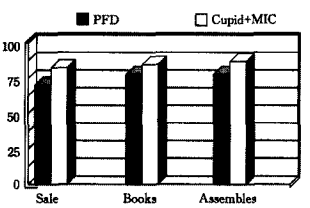

图7 PFD与Cupid+MIC查全率对比

通过设定合理的阈值来提高属性取值匹配度计算方法的效率，实验证明该方法能得到较全面的、正确的属性匹配关系。经验证，在缺少已知模式信息时，基于实例聚类分析的匹配方法是有效的。

结束语 模式信息包括各属性名称及其类型等属性的信息、模式的结构信息、数据实例。在进行匹配时，可以使用语义说明相互效果的多种模式信息，来提高匹配的精确度。本文所提出的模式匹配方法可以作为基于模式信息的一个辅助手段、有力补充，检验并纠正错误匹配，在缺少模式信息的场合，如果数据实例较全面体现了模式语义信息，借助领域知识，可利用本文所提方法来解决多数据源的模式匹配问题。通过把确定模式中属性对匹配的问题看作一个整体，从而避免将具有大量相同数据的属性实例集合对应的两个语义有差异的不匹配属性错误判定为匹配的。这一方法的优点在于即使数据实例中无重复描述的情形，但根据属性类型、实例的语义特征及部分属性之间的关系能分析出属性的语义相似性，可得到正确的模式匹配结果。

未来研究工作中将考虑模式匹配中存在的复杂匹配关系的发现，在模式的基础上解决实体识别的问题；并将就动态数据集成环境中数据源模式动态变化时的模式匹配方法进行研究。

参考文献

[1] Madhavan J, Bernstein P, Rahm E. Generic Schema Matching with Cupid[C]// VLDB 2001. 2001:49-58

[2] Hong Hai-do, Rahm E. COMA-A system for flexible combination of schema matching approaches[C]// VLDB 2002. 2002: 610-621

[3] Hernández M A, Popa L, Ho H, et al. Clio: A Schema Mapping Tool for Information Integration[C]// The8th International Symposium on Parallel Architectures. Algorithms, and Networks, ISPAN. 2005:11

[4] Zerdazi A, Lamolle M. Computing Path Similarity Relevant to XML Schema Mapping[C]// OTM Workshops. 2008:66-75

[5] Wang Ji-ying, Wen Ji-rong, Lochovsky F, et al. Instance-based Schema Mapping for Web Databases by Domain-specific Query Probing[C]// Proc. 30th VLDB. Conf. 2004:408-419

[6] Bilke A, Naumann F. Schema Mapping using Duplicates [C]// ICDE 2005. 2005:69-80

[7] Arndt C. Information Measures: Information and its description in Science and Engineering[M]. Berlin: Springer, 2001:370-373

[8] Li Guo-hui, Du Xian-kun, Hu Fang-xiao. A schema matching method based on Partial functional dependencies[C]// Frontier of Computer Science and Technology (FCST2008). 2008:131-138

(上接第127页)

[2] Zhang Qing-hua, Reeves D S, et al. Analyzing Network Traffic to Detect Self-decrypting Exploit Code[C]// Proceedings of the ACM Symposium on Information, Computer and Communications Security(ASIACCS). 2007

[3] Polychronakis M, Anagnostakis K G, Markatos E P. Emulation-based Detection of Non-self-contained Polymorphic Shellcode [C]// Proceedings of the Third Conference on Detection of Intrusions and Malware and Vulnerability Assesment(DIMVA). 2006

[4] Snort[EB/OL]. http://www.snort.org/

[5] Bro[EB/OL]. http://www.bro-ids.org/

[6] Pin[EB/OL]. http://www.pintool.org/

[7] Metasploit[EB/OL]. http://www.metasploit.com