文章编号：2096-1472(2017)-02-12-04

基于系统调用短序列的软件漏洞检测方法研究

葛立欣
（包头职业技术学院，内蒙古 包头 014030）

摘 要：软件受到攻击后将在所执行的系统调用状况中有所体现，因此可将基于系统调用的入侵检测技术应用于软件漏洞的检测。本文针对无源码的可执行程序，引入系统调用节点和系统调用上下文信息的概念来刻画软件行为的动态特性和漏洞的位置信息，利用改进的STIDE算法构造软件正常行为特征库来检测并定位漏洞。实验结果表明该方法能够准确获取软件行为信息，且具有较强的漏洞检测能力。

关键词：漏洞检测；行为建模；系统调用短序列；STIDE算法；函数调用链
中图分类号：TP393.08 **文献标识码**：A

A Study of the Software Vulnerability Detection Method Based on the Short Sequence of System Calls

GE Lixin
(*Baotou Vocational & Technical College*,*Baotou* 014030,*China*)

Abstract: After the software is attacked,the influence will be reflected in the status of the executed system call. Therefore,the intrusion detection technology based on system calls can be applied into the detection of software vulnerabilities.In order to analyze the executable program without source code,the concept of the system call node and the context information are introduced to depict the dynamic behavior characteristics of the software and the localization information of the vulnerabilities in this paper.Furthermore,the vulnerabilities can be detected and located by building the normal behavior characteristics library based on the improved STIDE algorithm.The experiment results show that the behavior information of the software can be obtained and the vulnerabilities can be detected accurately by applying the above method.

Keywords:vulnerability detection;behavior modeling;the short sequence of system calls;STIDE algorithm;function call chain

1 引言(Introduction)

软件漏洞是存在于软件系统内的一组弱点或缺陷，这组弱点或缺陷被恶意主体(攻击者或攻击程序)加以利用，可达到访问未授权信息或损坏系统的目的。软件由于其功能及行为的复杂性不可避免地存在一些漏洞，这给整个软件系统带来了极大的隐患。传统的漏洞检测技术分为针对源代码进行检查的静态检测技术和在缺少源代码的情况下对软件漏洞进行检测和防护的动态检测技术，但这些技术仅能检测出已知漏洞类型，能检测的对象有限，且存在检测规模大、误报率高等缺点[1]。

研究表明，软件受到攻击后将在所执行的系统调用状况中有所体现，并且系统调用之间的关系也在一定程度上反映了软件的分支结构。考虑到软件系统在未遭到攻击时，安全漏洞(而非引起系统崩溃的错误)并未给系统造成损失，可在软件系统交付使用后，当其遭到攻击的时候拦截攻击并检测出漏洞以便日后进行修补[2-5]。因此将基于系统调用的入侵检测技术应用于软件漏洞的检测是很有必要的。

本文针对无源码的可执行程序，提出一种基于系统调用短序列的软件漏洞检测方法。将目标程序在安全环境下运行，监测其系统调用序列和堆栈信息，利用改进的STIDE算法将系统调用序列“分割”为短序列，并建立正常行为特征库和位置信息库。再将目标程序暴露于攻击下，采用同样的算法得到短序列进行模式匹配，通过计算横向异常值和纵向异常值来判断是否发生行为偏离。当行为偏离超过阈值时告警，并根据可疑系统调用的位置特征与位置信息库中的位置特征进行比对，定位漏洞。

2 相关研究(Correlational research)

目前对软件行为的动态检测大多采用异常检测方式，在可信环境下通过静态分析软件代码或者动态监控软件实例，从中提取软件行为特征建立软件的正常行为模型，然后监控

软件的实际运行，提取其行为特征并与正常行为模型进行比较。如果软件行为发生的偏差超过指定阈值，则判定软件行为不可信。

目前软件行为模型已经取得了一些研究成果，按照不同的组织模式，相关研究主要包括基于短序列的N-gram模型[5]、Var-gram模型[6]，以及引入系统调用频率特性[7]、数据挖掘理论或隐式马尔可夫链的模型；基于自动机的FSA模型、Abstract Stack模型、Call Graph模型、Execution Graph模型；基于系统调用参数的Dyck模型、DTEMSB-BTCS模型；基于虚拟路径的Vt-Path模型、VPstatic模型。

现有研究将基于系统调用的入侵检测模型应用于漏洞检测技术，具体方法为：采用动态训练方式，结合系统调用的PC值构造出程序执行路径的有限状态自动机，自动机的节点为系统调用对应的PC值，而状态转换为系统调用。当每个系统调用发生时，在检测程序中存储当前的函数堆栈，根据异常发生的状态机所属函数来判断可能的函数漏洞在哪个函数中。基于状态机的漏洞检测方法需要建立自动机，存在时间、空间消耗较大及不可能路径等问题。

针对上述已有技术存在的不足，本文将基于系统调用短序列模型和STIDE算法加以改进并应用于软件漏洞检测，其目的在于发挥短序列模型汇聚时间短、运行时间负载较低的优势，同时克服由于缺乏定位信息和系统调用上下文信息导致模型对上下文不敏感、粒度粗等缺点，增强了模型的检测能力。

3 基于系统调用短序列的软件漏洞检测模型 (Software vulnerability detection model based on short sequence of system calls)

3.1 基本概念

定义1：系统调用短序列*SCS*(System Call Sequence)是指包含一定数量的系统调用节点的有序集合：

$$SCS_n = < SC_1, SC_2, \ldots, SC_n >$$

其中，n表示短序列长度，*SC*表示系统调用节点。

定义2：系统调用节点*SC*(System Call)是指含有系统调用相关信息的一个系统调用向量：

$$SC = < SCI, SCP, SCC >$$

其中，*SCI*表示系统调用基本信息，*SCP*表示系统调用属性，*SCC*表示系统调用上下文。这三个向量值唯一标识一个系统调用节点。

定义3：系统调用基本信息*SCI*(System Call Information)：

$$SCI = < SysCall\_No, SysCall\_Name >$$

其中，*SysCall_No*表示系统调用号，*SysCall_Name*表示系统调用名。

定义4：系统调用属性*SCP*(*System Call Properties*)：

$$SCP_n = < n, SysCallArg_1, SysCallArg_2, \ldots, SysCallArg_n >$$

其中，$n(n \leqslant 6)$表示系统调用参数个数，*SysCallArg*表示系统调用参数。

定义5：系统调用上下文信息*SCC*(*System Call Context*)：

$$SCC = < PC, CSV >$$

其中，*PC*表示当前系统调用发生时的*PC*值，*CSV*表示函数调用链值，通过如下算式得到：

$$CSV = Hash(CS)$$

CS(Call Stack)为函数调用链，当系统调用发生时，记录函数堆栈里的返回地址所对应的函数名称，并将其存于位置信息库CS_Table中，作为该表索引：

$$CS = < FunctionName_1, FunctionName_2, \ldots, FunctionName_n >$$

即 $FunctionName_1$ 调用了 $FunctionName_2$，$FunctionName_2$ 调用了 $FunctionName_3$，…，$FunctionName_{n-1}$ 调用了 $FunctionName_n$。其中，n表示函数调用链的长度。

上述系统调用信息的获取采用修改中断向量表的方法，准确高效。函数返回地址与名称的对应方法参考ELF文件格式分析方法，从可执行ELF文件中提取出相对地址对应的函数名称。

3.2 模型概述

基于系统调用短序列的软件安全漏洞检测模型，如图1所示，分为两个阶段：离线训练阶段与在线检测阶段。离线训练阶段要求目标软件在安全环境下运行，在线检测阶段则将目标软件曝露于恶意攻击中。

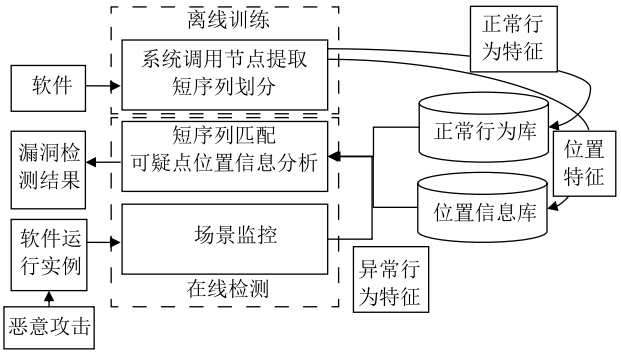

图1 基于系统调用短序列的软件安全漏洞检测模型

Fig.1 Software security vulnerability detection model based on system call short sequence

(1)离线训练

在离线训练阶段，由于系统调用序列是动态生成的，因此采用STIDE算法以固定长度值为n的滑动窗口对正常行为序列进行扫描，将生成的一系列短序列存储到正常行为特征库

中。正常行为特征库是由一系列上述系统调用短序列SCS_n组成的，n为滑动窗口大小也即短序列长度。此外，还需维护一个位置信息库CS_Table，记录函数调用链CS_n，其索引是CSV。

(2)在线检测

在线检测阶段中，同样采用STIDE滑动窗口机制，STIDE算法使用计算海明距离的方法来判定待测序列中的异常发生情况。此外，针对系统调用节点SC的变动，相应改进STIDE匹配算法，给出定位算法。

3.3 改进的STIDE算法及定位算法

第一步，采用长度为n、步长为1的滑动窗口机制对正常行为产生的系统调用序列CS_1、CS_2…进行扫描，并记录每个窗口内各个系统调用节点从自身i开始到其后$n-1$个位置所生成的短序列SCS_n。$SCS_n = < SC_i, SC_{i+1}, \cdots SC_{i+n-1} >$，以这种包含系统调用节点之间的位置关系信息的短序列SCS为结果建立特征库，并将其存储至描述正常行为的特征库中。

第二步，用同样的方式将未知行为的系统调用序列进行划分，与已建立的包含正常行为的数据库进行短序列元素关系匹配，匹配算法如下：

(1)系统调用节点匹配算法：对待测的系统调用节点SC，计算系统调用偏差值D。

$$D=\{Hash(SC)-Hash(SC'), SC' \in Normal\_Behavior\_DB\}$$

其中，SC为当前待匹配的系统调用节点元素的字符串拼接，SC'为特征库中系统调用节点元素的字符串拼接。

$D=0$当时，系统调用无偏差，SC与SC'匹配；

$D<m$当时，系统调用无偏差，SC与SC'匹配；

$D>m$当时，系统调用有偏差，SC与SC'不匹配。

m的取值与$Hash$函数的选择有关，在实现时采用安全套接层工具包OpenSSL包默认的散列函数来计算系统调用上下文值。

(2)系统调用短序列偏差值算法：对待测系统调用短序列SCS，计算其与特征库中短序列SCS'的系统调用短序列偏差值V。

$$V=\sum_{1}^{n-1} D_i$$

其中，n为滑动窗口大小，D_i为短序列中SCS第i个系统调用节点与SCS'中第i个系统调用节点的系统调用偏差值。

(3)对待测序列采用滑动窗口机制划分系统调用短序列SCS，计算该序列对比特征库中每条短序列SCS'的海明距离，记录最小值，记为待测短序列SCS的最小海明距离$d_{min}(SCS)$。

$$d_{min}(SCS)=min\{d(SCS,SCS'), SCS' \in Normal\_Behavior\_DB\}$$

其中，$d(SCS,SCS')$为短序列SCS与SCS'中系统调用节点SC的不匹配个数，匹配算法采用系统调用节点匹配算法。

(4)找到特征库中与待测短序列SCS匹配后海明距离最小的SCS'，记为SCS'_{min}。采用系统调用短序列偏差值算法，计算SCS与SCS'_{min}的系统调用短序列偏差值V。可能存在多个SCS'_{min}，选取最小的系统调用短序列偏差值V_{min}，记为待测短序列SCS的最小系统调用短序列偏差值$V_{min}(SCS)$。

(5)计算横向异常值V_A和纵向异常值V_B。

$$V_A = \max\{d_{min}(j), \forall j \in New\ \ Syscall\ \ Sequence\}$$

$$V_B = \max\{V_{min}(j), \forall j \in New\_Syscall\_Sequence\}$$

其中，j为待测系统调用短序列，横向异常值V_A记录当前所有待测系统调用短序列的最小海明距离d_{min}的最大值。纵向异常值V_A记录当前所有待测系统调用短序列的最小系统调用短序列偏差值V_{min}的最大值。

(6)当横向异常值V_A和纵向异常值V_B超过一定阈值，则挂起用户进程。两个阈值的选取分别基于滑动窗口大小和(1)中m值的大小。接着从最近一次未超过阈值的短序列中，找到系统调用偏差值D最大的系统调用节点的CSV值，并在位置信息库CS_Table中找出对应的函数调用链CS到当前系统调用节点的函数调用链CS，这之间判为可疑漏洞的位置。

4 实验及分析(Experiment and analysis)

4.1 实验环境及实验方法

实验主机的硬件配置为：CPU为主频2.53GHz的Intel(R) Core(TM)2 Duo T9400，内存为4.00 GB；运行内核版本为2.6.32的redhat 6.5操作系统，编译器版本为gcc4.4.7。

在上述实验环境下进行了模型实现，实现方法如下：在离线阶段，首先为系统新增一个LKM模块，用来修改中断向量表和恢复中断服务程序。其次获取当前系统调用并保存相关信息。随后将截获的系统调用划分成短序列，并输出到短序列集合文件中。最后读入该文件，构造系统调用短序列森林，形成正常行为特征库。

在线检测阶段采用CVE漏洞库中sendmail进程的exploit来攻击sendmail进程，获得异常数据。获取异常行为与离线阶段的方法一致。此外，获取当前系统调用时的ebp进行栈回溯，将函数堆栈的所有返回地址存于返回地址集合文件，并根据改进的STIDE算法在特征库中进行匹配，主要工作如下：

①计算横向异常值V_A和纵向异常值V_B；

②根据异常值偏差程度报告漏洞检测结果；

③如果异常值未超出阈值，则调用原系统调用；如果超出阈值，则报警并结束进程。

图2和图3分别展示了离线训练阶段和在线检测阶段的实现流程。

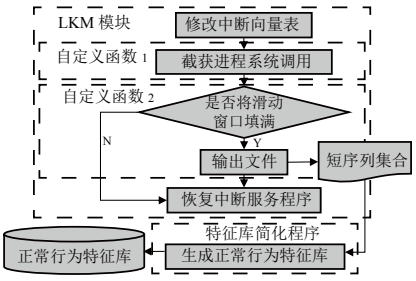

图2 模型离线训练阶段实现流程图

Fig.2 Flow chart of the off-line training phase

4.2 阈值确定

本文针对sendmail漏洞进行了实验，实验常值如表1所示。

表1 阈值及其方差参数

Tab.1 Threshold and its variance parameters

	横向异常值V_A						纵向异常值V_B					
阈值	1	2	3	4	5	6	21	36	43	54	65	75
方差	2.9	8.9	1.6	6.3	8.7	5.7	6.9	1.3	6.7	8.7	2.9	2.9

CVE 2009-1490Sendmail X-header头堆溢出漏洞。

CVE 2006-0058Sendmail异步信号处理漏洞。

CVE 2003-0681SendmailRuleset Parsing缓冲区溢出漏洞。

CVE 2002-1827Sendmail文件锁机制拒绝服务攻击漏洞。

CVE 2001-0653Sendmail修改内存提升特权漏洞。

CVE 1999-1109Sendmail ETRN拒绝服务漏洞。

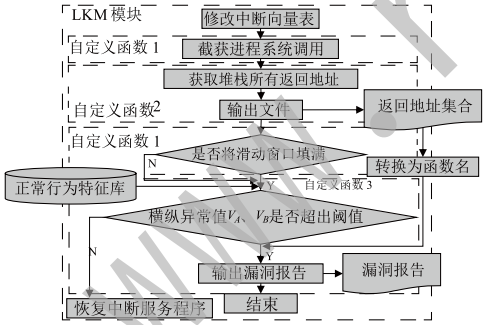

图3 模型在线检测阶段实现流程图

Fig.3 Flow chart of the model online detection

针对不同的异常值取值，选取部分漏洞和正常数据进行异常度测试，统计同一阈值下各漏洞的异常度，计算其标准偏差的方差。实验采用大小为30的局部帧，异常度为不匹配数目。

实验结果表明纵向异常值阈值取3、横向异常值阈值取36时，方差最小，异常度聚合程度最高。

4.3 漏洞检测能力

根据上述阈值，针对不同漏洞检测其漏洞所在函数及函数所在文件，实验结果如表2所示。

表2 漏洞所在函数检测结果

Tab.2 Vulnerability function detection results

CVE号	所在函数	存在文件
2009-1490	fwrite	mail.local.c
2006-0058	mi_signal_thread	signal.c
2003-0681	prescan	parseaddr.c readcf.c
2002-1827	flock	conf.c
2001-0653	tTflag	main.c, srvrstmp.c
1999-1109	sleep	Smtpd.c, smtpd_check.c

参照漏洞标准描述，其中四个漏洞所在函数和存在文件与披露的漏洞所在文件一致，其余两个漏洞未披露漏洞所在文件，但检测报告中函数调用链与披露的漏洞成因一致。

5 结论(Conclusion)

目前基于系统调用的软件行为研究方兴未艾，漏洞的在线检测方法更是研究热点之一。基于系统调用短序列的软件漏洞检测模型引入系统调用节点和系统调用上下文信息的概念来刻画软件行为的动态特性和漏洞的位置信息，利用改进的STIDE算法构造软件正常行为特征库来检测并定位漏洞。实验结果表明模型能够准确获取软件行为信息，也具有较强的漏洞检测能力。

参考文献(References)

[1] Simple Nomad.Sendmail 8.12.x-'X-header' Remote Heap Buffer Overflow Vulnerability[EB/OL].https://www.exploit-db.com/exploits/32995.

[2] redsand.Sendmail<=8.13.5-Remote Signal Handling Exploit PoC[EB/OL].[2014-6-12].https://www.exploit-db.com/exploits/2051.

[3] Giffin J T,et al.Environment-Sensitive Intrusion Detection[J]. Lecture Notes in Computer Science,2006:185-206.

[4] 傅建明,等.基于对象的软件行为模型[J].软件学报,2011,22(11):2716-2728.

[5] 忽朝俭,等.基于可执行代码的漏洞检测技术[J].清华大学学报:自然科学版,2009,S2:225-233.

[6] 张林,曾庆凯.软件安全漏洞的静态检测技术[J].计算机工程,2008,(12):157-159.

[7] 单国栋,戴英侠,王航.计算机漏洞分类研究[J].计算机工程,2002,28(10):3-6.

作者简介：

葛立欣(1970-)，男，硕士，副教授.研究领域：计算机网络，软件工程.