

# 进程完整性及其在软件漏洞检测中的应用

曾凡平<sup>1,2</sup>, 陈明辉<sup>1</sup>, 尹凯涛<sup>1</sup>, 王煦法<sup>1,2</sup>

(<sup>1</sup>中国科学技术大学计算机系, 安徽 合肥 230026;

<sup>2</sup>安徽省计算与通讯软件重点实验室, 安徽 合肥 230026)

【摘要】文中提出了一种进程完整性的度量方法,并应用于软件漏洞的检测。该方法利用源码转换和函数调用时的状态信息来实现进程完整性的检测与分析。通过检查进程在运行过程中的完整性度量是否保持为真,来判断进程是否存在漏洞。实验结果表明,这种方法能够在不带来大的性能损失的情况下准确地检测出缓冲区溢出漏洞。

【关键词】安全关系;进程完整性;源码转换;函数调用;缓冲区溢出

【中图分类号】TP391.76 【文献标识码】A 【文章编号】1009-8054(2009) 10-0091-04

## Process Integrity and Its Application in Software Vulnerability Testing

ZENG Fan-ping<sup>1,2</sup>, CHEN Ming-hui<sup>1</sup>, YIN Kai-tao<sup>1</sup>, WANG Xu-fa<sup>1,2</sup>

(<sup>1</sup>Department of Computer, University of Science and Technology of China, Hefei Anhui 230026, China;

<sup>2</sup>Anhui Provincial Key Lab of Software in Computer and Communication, Hefei Anhui 230026, China)

【Abstract】This paper presents a measurement method for process integrity. This measurement method is applied in the test of software vulnerability. The method uses source-code conversion and the state information of function-call to check and analyze the process integrity. By checking whether the measurement for process integrity is true, the fact that whether there is a vulnerability in the software is determined. The experiment shows that this method could accurately check the attack of buffer overflow, in the circumstances of with no large performance loss.

【Keywords】safety relation; process integrity; source-code conversion; function-call; buffer overflow

## 0 引言

软件漏洞是使软件遭受安全威胁和破坏其可信性的根本原因。如果能检测并排除软件中的漏洞<sup>[1-2]</sup>,则可以从根本上提高软件系统的安全性和可信性。从某种程度上说,软件漏洞是通过破坏软件所对应的进程完整性进而威胁系统的安全性的。比如对于众所周知的缓冲区溢出漏洞,就是通过破坏进程所属堆栈的完整性而导致进程的崩溃或被控制以达到攻击目标的。如果能定义进程的完整性,并用某种方法在进程的动态运行中检测出对这种完整性的破坏,就能检测出软件漏洞。因此,定义并检测进程的完整性,对

于检测软件漏洞具有重要的意义。

在借鉴 Kyung-suk Lhee 漏洞分类和正确性关系的基础上<sup>[3]</sup>,提出了一种进程完整性的度量和检测方法,并将该方法应用于缓冲区溢出漏洞的检测。实验结果表明,该方法可以在基本不影响被测程序性能的情况下检测出缓冲区溢出漏洞。

## 1 相关工作

国外首次提出进程完整性检测方法的是美国普度大学信息安全研究和教育中心的 Kyung-suk Lhee(2005)。他在其博士论文《Integrity Checking For Process Hardening》中提出并利用进程完整性检测方法来增强进程的安全性。Kyung-suk Lhee 的论文把程序漏洞分为限数量的错误类别,并给出了不出现该错误类别的安全关系,用于增强进程的安全性。

国内的联想集团于2005年提出了“实时检查进程完整性的方法与系统”,并申请了发明专利<sup>[4]</sup>。该发明提出了一种实时检查进程完整性的方法,所述方法首先收集程序在内存中

收稿日期:2009-04-10

作者简介:曾凡平,1967年生,男,副教授,研究方向:信息安全和软件测试;陈明辉,1983年生,男,硕士研究生,研究方向:信息安全;尹凯涛,1985年生,男,硕士研究生,研究方向:信息安全;王煦法,1948年生,男,教授,博导,研究方向:信息安全和计算智能。

第一次执行的进程的散列值和与所述进程相对应的整个程序的散列值,并存储在进程参考表中,然后对同一程序的后续进程,则利用进程参考表中的散列值,来验证所述进程和与所述进程相对应的整个程序的完整性。

在利用进程完整性检测软件漏洞方面,国内外的研究处于起步阶段,尚未有研究成果发表。

## 2 基于进程完整性检测的漏洞测试方法

所谓的进程完整性就是程序在运行过程中保持为正确的属性,否则程序违背了进程的完整性。可以用进程完整性来监测程序中某些点,若这些点在某时刻产生脆弱性异常,则说明了程序存在脆弱性。测试方法主要由两部分组成,分别是源码转换和检测机制,如图1所示。其中源码转换部分主要是识别一些与安全有关的函数、变量等必要的元素,并且在程序的适当位置插入基于函数调用的审查语句;脆弱性检测部分则是该方法的核心,主要是在具体的脆弱性种类中检测和分析进程完整性度量,而检测算法和解决方案都是在进程完整性度量的基础上得到的。

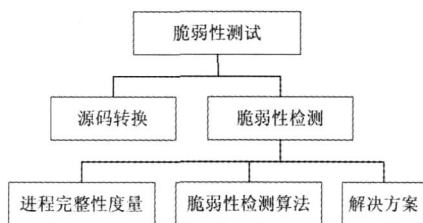


图1 脆弱性测试框架

### 2.1 源码转换

源码转换利用了程序插装的思想,它在被测试程序保持原有逻辑完整性基础上,通过在程序中插入的一些探针,来抛出程序的运行特征数据。程序插装分为源代码插装(SCI)和目标代码插装(OCI)两种。鉴于SCI比较简便易行,效率也比较高,文中选择SCI方式,在源代码中插入探针以监测状态信息的变化,从而以动态监测方法根据程序执行时的属性监控程序的运行,能够监控到某些与运行相关的异常,对程序进行安全测试,举例如下:

```

void Fun ()
{.....
Save 模块; // 保存状态信息
.....
Check 模块; // 检查状态信息是否符合安全关系
.....}
  
```

理想的结果是完成一个自动或半自动插装工具,其主要原理就是对源码进行扫描匹配,识别出所需要的函数、变量等,并且最终完成检测代码的插入。源码转换的过程

如下:首先,对C源代码进行词法、语法分析,这里借鉴YACC进行分析。其次,对源码进行插装,主要是在现有函数体内加入脆弱性检测代码。必要的时候根据函数调用关系有选择性地加入检测代码,特别是对于涉及危险函数调用(比如strcpy())的函数一定要加入检测代码,这样可以增加程序的执行效率。最后,形成程序插装器,并在条件允许的情况下,可以根据安全关系来指导测试用例的自动或半自动生成。

关于插装器目前还没有最终完成,文中实验部分的源码插装是由手工完成的。

### 2.2 进程完整性度量

如果进程在其执行过程中能保持某种粒度的完整性,则进程在该粒度上是安全的。在此,进程完整性度量的定义和检测是关键。安全关系<sup>[3]</sup>本质上是对脆弱性类型本质特征的形式化描述,这个关系在程序执行的过程中必须保证为True,否则,关系值为False时说明发生错误,这意味着程序很可能受到脆弱性的威胁。受此启发,把安全关系作为进程完整性的一种度量,这种度量在进程的运行过程中必须保持为True,这里借助函数调用状态值来描述该进程的完整性度量。以下给出缓冲区溢出所对应的进程完整性度量:

进程完整性度量1:输入的字符串大小,缓冲区的大小。对缓冲区进行边界检查,保证输入的字符串不跨过缓冲区边界;

进程完整性度量2:RA0, RA1, 其中RA0为函数调用开始时的返回地址,RA1为函数调用结束前的返回地址。这个主要用于堆栈缓冲区溢出的检测,目的是检测是否因为缓冲区溢出而造成函数返回地址被修改;

进程完整性度量3:AR, BR, 其中BR(Before Return)表示函数返回前的计数器,当有函数返回时,在RET前由被调用函数执行加1操作,而AR(After Return)类似,只不过是在函数成功返回时,由外部的调用体执行加1操作。这个关系本质上是保证函数能够正常返回(返回异常的时候,AR<BR),从而保证函数调用轨迹的正确性。

目前,只根据脆弱性类别,通过分析该类别的安全关系,找到与脆弱性相关的进程完整性度量。也可能还有其他的一些进程完整性度量,比如通过分析进程的运行环境(包括配置)从而得出与环境相关的进程完整性度量,这些可能与具体的程序密切相关,需要对程序作细致的分析,将在未来的工作中进行深入研究。

### 2.3 进程完整性度量的检测与判断

在定义了进程完整性度量后,可以根据该度量的定义推导出检测算法,这是对进程完整性度量所对应的逻辑断言的

直接描述。逻辑断言与计算机系统是相对独立的，可以转化为相对的数据结构和程序。比如对于缓冲区溢出，根据进程完整性度量 1，可以保存缓冲区的大小，然后在用户往缓冲区写入数据时，判断输入串的大小，如果  $\leq$ ，则结果为 True，否则发生溢出异常。

### 3 缓冲区溢出检测的实现

采用源码转换方法，在函数的入口和出口插入审查语句，这里根据 2.2 节的完整性度量 2 来完成。插入的方法举例如下：

```
void Fun (){
    Save 模块; // 保存返回地址
    ..... // 安全相关的代码
    Check 模块; // 检查返回地址
}
```

可以通过 EBP 来获取返回地址。通过表 1 中的代码，可以拿到 Fun 的返回地址(Save 模块)。

表 1 Save 模块的两种实现

Linux(AT&T)	Windows(Intel)
<code>_asm(“</code>	<code>_asm{</code>
<code>movl %ebp, %eax</code>	<code>mov eax, ebx</code>
<code>movl 4(%eax), %edx</code>	<code>mov edx, [eax+4]</code>
<code>movl %edx, ii</code>	<code>mov ii, edx</code>
<code>“);</code>	<code>};</code>

为了实现缓冲区溢出的检测，整个过程如图 2 所示。

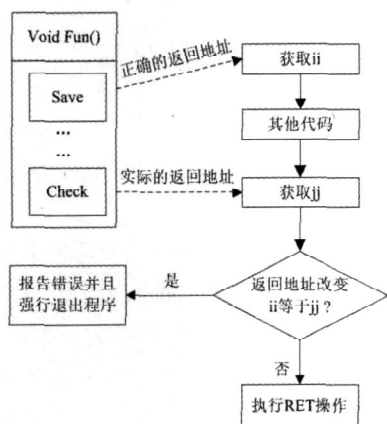


图 2 检测框架

由 Save 模块把返回地址保存到 ii，然后由 Check 模块来检查当前时刻的返回地址 jj 与 ii 是否相等，若相等，则执行 RET 操作，使函数调用正常返回，若不相等，则报告错误并且强行退出程序。

### 4 实验结果及其分析

这里给出一个缓冲区溢出程序，并且利用源码转换技术

来检测出缓冲区溢出：

```
#include <string.h>
#include <stdio.h>

void echo(){printf("Welcome to function echo!\n"); }

void test(){
    char buf[6];
    char a[]={"AAAAAAAAAAAAAAAAA"};
    strcpy(buf, a)
    .....省略打印部分.....
}

int main() {
    test();
    return 0;
}
```

#### 4.1 正确性分析

程序在 Windows2003 环境下的 Visual C++6.0 中调试执行，此时函数 test() 的返回地址为 0x00414141，系统报错，指出 0x00414141 引用的内存不能为 written。接下来修改字符串 a 为 char a[]= {"AAAAAAAAAAAAAAAA\x0f\x10\x40"}，此时 test() 的返回地址为 0x0040100f，这刚好与 echo() 的地址一样，也就是把 test() 的返回地址引导到 echo() (执行了 echo 中的 printf 语句)，从而改变程序的执行轨迹。

然后进行源码转换，在程序中(主要是 test() 中)增加了 Save 模块和 Check 模块，此时返回地址还是 0x0040100f，但是 Check 模块检测到了返回地址被修改的异常，于是报告错误给程序员，并且强行退出程序。

#### 4.2 时间分析

这里只测试 test 函数在源码转换前后的执行时间，而且只测试了在程序不出现缓冲区溢出情况下的性能。为了提高分析的精确度，这里用三重循环来测试函数的运行时间，并且取平均值。其中，N 为内层循环，表示 test 函数中增加的循环空操作次数；T 为次内层循环，表示 test 函数运行的趟数，一次 T 循环完成时，测试一次总运行时间；M 为外层循环，表示测试时间的次数。测试表明，转换前后的程序运行时间几乎相等。经过多次测试，记录了一组最坏情况下的额外开销，如表 2 所示。

表 2 最坏的额外开销

(M, T, N) (10 <sup>3</sup> )	源码转换前 (ms)	源码转换后 (ms)	Overhead (%)
(2, 2, 0)	0.000037157	0.000039688	6.8118267
(2, 4, 1)	0.000066195	0.000068385	3.3087399
(2, 4, 2)	0.000316963	0.000321689	1.4908359
(1, 4, 3)	0.002695952	0.002706236	0.3814486
(1, 4, 4)	0.026560413	0.026636650	0.2870315
(1, 4, 5)	0.241572685	0.242484533	0.3774631
(1, 4, 6)	2.739642354	2.755408909	0.5754969


从表2可以看出,随着函数体的增大,其运行时间也相应增大,从而使得插入检测代码的时间性能影响也相应降低,由7%左右下将到0.5%左右,并且最终稳定在0.5%左右。由此可见该方法对性能的影响比较小。

## 5 结语

文中提出基于安全关系的进程完整性度量,并利用源码转换和函数调用状态信息来实现进程完整性度量的检测与分析。将该方法成功地应用于对缓冲区溢出攻击的检测,结果表明该方法简单而且高效。目前的进程完整性度量是从漏洞分类的安全关系中得到的,能否从进程的操作系统环境中找

到进程完整性度量,是下一阶段的重点研究内容。

### 参考文献


- [1] 高静峰,林柏钢,倪一涛.基于粗糙集理论的漏洞检测技术研究[J].信息安全与通信保密,2007(01):63-66.
- [2] 屈晔,张昊. bugscam 自动化静态漏洞检测的分析[J].信息安全与通信保密,2007(03):105-107.
- [3] Kyung-suk Lhee. Integrity Checking For Process Hardening[D]. USA: Purdue University, 2005.
- [4] 联想(北京)有限公司. 实时检查进程完整性的方法与系统:中国,200510134081[P]. 2007-06-27. 

(上接第87页)

变得日益重要<sup>[5]</sup>。文中就是在这种情况下,设计了一种可行的一次性口令失步后重同步的方案。在这种方案中,我们兼顾了实现的高效性和安全性,并考虑到实现的便利性,为重同步问题的解决提供了一种解决方案。

### 参考文献

- [1] 刘立阳,张景川,吴至真. 一次性口令在身份认证中的应用[J]. 科技咨询报,2007(22):22.

- [2] 康丽军. 基于时间的一次性口令认证的原理与实现[D]. 太原:太原理工大学,2002.
- [3] 高雪,张焕国. 一种改进的一次性口令认证方案[J]. 计算机应用研究,2006,32(04):149-150.
- [4] 罗婵. 基于事件的一次性口令系统的研究与实现[D]. 西安:西安建筑科技大学,2007.
- [5] 张宏,陈志刚. 一种新型一次性口令身份认证方案的设计与分析[J]. 计算机工程,2004,30(17):112-113. 

(上接第90页)

别,API相关调用多将采用隐式调用。另外,病毒的壳的种类会更加多,同时,未来壳具有自我保护性和反调试性,使得杀毒软件引擎脱壳困难。病毒在运行方面,会更加频繁地利用系统以及软件的漏洞,同时采用捆绑等方式,使用户一旦被一种病毒感染,则会被许多病毒感染<sup>[8]</sup>。在病毒利用互联网不断更新换代的时代,杀毒软件也应该提高病毒库发布的频率,使得新变种的病毒能够在第一时间被杀毒软件查杀,提高杀毒软件的查杀率。

### 参考文献

- [1] 孙旭东. Windows XP 注册表与 BIOS 设置实用指南[M]. 北京:航空工业出版社,2002(12):89-90.
- [2] 霍格兰德. Rootkits-Windows 内核的安全防护[M]. 韩智文,译. 北京:清华大学出版社,2007:182-184.
- [3] 赵佐,蔡皖东,田广利,等. 基于异常行为监控的僵尸

网络发现技术研究[J]. 信息安全与通信保密,2007(09):60-69.

- [4] 孙翔,陈念伟. 传统局域网的 QoS 保障机制的分析与应用[J]. 通信技术,2007,40(08):71-73.
- [5] Xrayn. Hide Your SSDT Hooks[EB/OL]. (2008-11-13)[2009-03-9]. <http://www.rootkit.com/newsread.php?newsid=922>.
- [6] 汪小帆,戴跃伟. 信息隐藏技术——方法和应用[M]. 北京:机械工业出版社,2001:135-138.
- [7] 肖红光,谭作文,王键. 一种前向安全的代理盲签名方案[J]. 通信技术,2009,42(05):75-76.
- [8] 北京瑞星国际软件有限公司. 2008年中国大陆地区电脑病毒疫情 & 互联网安全报告[EB/OL]. (2008-11-18)[2002-04-15]. <http://it.rising.com.cn/new2008/News/NewsInfo/2008-11-18/1226970618d50435.shtml>. 