基于运行时类型分析的整形漏洞二进制检测和定位系统

肖 海 陈 平 茅 兵 谢 立
(南京大学计算机科学与技术系软件新技术国家重点实验室 南京 210093)

摘 要 整形漏洞(Integer-based vulnerability)是一种存在于C或C++代码中的漏洞,具有极其严重的破坏性。2006年CVE指出缓冲区溢出漏洞呈下降趋势,而其他一些漏洞,如整形溢出、符号转换错误等呈上升趋势。设计并实现了一种针对整形漏洞的二进制实时检测和定位的方法。针对整形漏洞攻击,首先将二进制文件转化为一种中间语言VEX;然后在运行时将与外部输入相关的数据着色,截获相关语句并记录信息;最后依据制定的检测策略对着色的数据进行检测并定位。选用常见的含有内存漏洞的程序来测试系统的有效性及其性能损耗。实验结果表明,该工具可以检测并且定位软件中绝大多数的整形漏洞,而且误报和漏报率都很低。

关键词 计算安全,软件安全,整形漏洞,整形溢出

New Binary System for Detecting and Locating Integer-based Vulnerability on Run-time Type Analysis

XIAO Hai CHEN Ping MAO Bing XIE Li
(State Key Laboratory for Novel Software Technique, Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract Integer-based vulnerability is an extremely serious bug for programs written in languages such as C/C++. Common Vulnerability and Exploit(CVE) shows that as the percentage of buffer overflow has declined, there has been an increase in related vulnerability types, including integer overflows and signedness errors. Here we presented the design, implementation, and evaluation of a tool for run-time detecting and locating integer-based vulnerability. We first translated the binary code into intermediate language VEX on Valgrind, then intercepted integer related statements at run-time, recorded the necessary information, and finally detected and located vulnerability based on the checking scheme. We chose several utility applications, which contain real integer-based vulnerability, to evaluate the effectiveness and run-time performance of our system. Preliminary experimental results are quit promising, it can detect and locate most of integer-based vulnerability in real software, and has very low false positives and negatives.

Keywords Computer security, Software security, Integer-based vulnerability, Integer overflow

1 引言

整形漏洞是臭名昭著的软件安全漏洞之一,在C或C++程序中尤为突出。整形漏洞分为4类[3]:(1)符号转换错误(Signedness Error);(2)整形下溢出(Integer Underflow);(3)整形溢出(Integer Overflow);(4)赋值截断(Assignment Truncation)。2006年,CVE(Common Vulnerability and Exploit)[4]指出缓冲区溢出漏洞(Buffer Overflow)呈下降趋势,而其他一些漏洞如整形溢出、符号转换错误等呈上升趋势。值得注意的是,近年来整形溢出在系统漏洞中的排名一直较高,其威胁程度仅次于缓冲区溢出。更重要的是,整形漏洞往往会触发很严重的攻击,如任意代码执行(arbitrary code execution)、拒绝服务攻击(denial of service)等。图1显示的是近几年CVE[17]报告的整形漏洞,可以看出其呈明显上升趋势。

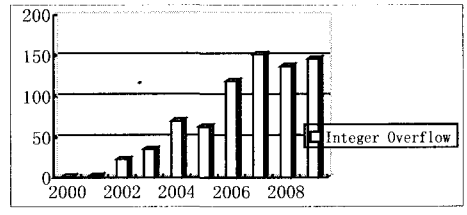

图1 近10年CVE报告的整形漏洞数量

整形漏洞一般与程序语义相关,因此很难被检测和定位。程序员经常忽视,甚至对整形漏洞知之甚少。标准编译器(如GCC)对整形漏洞也不能提供警告或错误信息。更糟糕的是,整形漏洞常常与某些攻击方法结合,共同产生内存错误。现有的安全工具,例如地址随机化工具[14]对内存进行随机化处理,控制流分析工具[16]保护关键的控制流对象,WIT[15]通过静态分析和运行时检测防止内存写错误,它们几乎都不能检测并定位整形漏洞。尽管这些工具能有效地检测内存错误,

到稿日期:2010-02-09 返修日期:2010-04-26 本文受863国家高技术研究项目(No. 2007AA01Z448),国家自然科学基金(60773171)和江苏省自然科学基金(BK2007136)资助。

肖 海 硕士,主要研究方向为软件安全,E-mail:dabaosod011@gmail.com;陈 平 博士,主要研究方向为软件安全;茅 兵 教授,博士生导师,主要研究方向为系统安全与分布式系统;谢 立 教授,博士生导师,主要研究方向为分布式系统。

但是它们对整形漏洞却“视而不见”。这些工具报告的漏洞信息一般仅仅是整形漏洞的“表象”，而不是漏洞的根源。因此设计能有效检测和定位整形漏洞的工具很重要。

2 整形漏洞

我们研究了 CVE 发布的 748 例整形漏洞，采用了 David Brumley 的分类[3]：(1)符号转换错误(Signedness Error)；(2) 整形下溢出(Integer Underflow)；(3) 整形溢出(Integer Overflow)；(4) 赋值截断(Assignment Truncation)。如图 2 所示，整形溢出在整形漏洞中所占比重最大，赋值截断出现的频率最低。

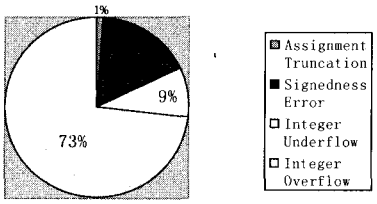

图 2 整形漏洞分类

2.1 整形漏洞的根源

整形漏洞的根源是整形操作数的类型所能表示的范围有限。通常在赋值操作(不同类型的操作数之间的赋值)或者算术运算(乘法、加法、减法)时，操作数的值被溢出或者替换，导致结果发生错误。

2.1.1 符号转换错误

符号转换错误(Signedness Error)的根本原因是误将有符号数当成无符号数使用，或者相反。虽然它们的机器码相同，但其由于可表示的数值范围不同，会被解释成不同的值。如，32 位数值 0xFFFFFFFF 可以被当成有符号数的 −1 使用，也可以被当成无符号数 4294967295 使用。

2.1.2 整形下溢出

整形下溢出(Integer Underflow)的根本原因是某种类型的变量存在最小值。例如 32 位无符号型整数最小值是 0。整形数下溢出通常由减法或除法操作引起。

2.1.3 整形溢出

整形溢出(Integer Overflow)的根本原因是某种类型的变量存在最大值。在算术运算(通常是加法或者乘法)中，所得的值可能超过了变量类型所能表示的最大值，使得结果小于预期值。下面是一个整形溢出的例子(CVE-2008-3732)：

```
1  int open(object_t * p_this)
2  { …
3      i_size=sizeof(uint32_t) * N;
4      p =(uint8_t * )malloc(i_size);
5      stream_Read(p,i_size);
6      p_sys =(uint32_t * )malloc(i_seektable_size);
7      for(i=0;i < p_sys->i_totalframes;i++)
8          p_sys[i]=GetDWLE(&p[i * 4]);
9      …
10 }
```

假设 N 的值为 2^{30}，由于 sizeof(uint32_t)和 N 的乘积发生了溢出，在第 3 行 i_size 被置为 0。这将导致第 6 行 p_sys 被分配的空间为 0，第 8 行进行拷贝操作时，会发生堆溢出。

2.1.4 赋值截断

赋值截断(Assignment Truncation)的根本原因是不同宽度类型(width type)变量之间的赋值。在实际攻击中，目前赋值截断很少出现。

2.2 检测整形漏洞的关键

检测符号转换错误的关键是识别变量的符号类型(Signedness Type)。在此基础上找到类型使用冲突的变量，即那些既当作有符号数又当作无符号数使用的变量，并判断它的值是否是负数。检测整形下溢出的关键是判断减法操作的结果是否发生下溢出(忽略除法操作是因为由除法操作引起的整形数下溢出很少)。检测整形溢出的关键是判断加法或乘法操作的结果是否超过变量类型所能表示的最大值。检测赋值截断的关键是识别变量的宽度类型(width type)，判断所赋的值是否超过变量所能表示的最大值。

结合上述漏洞检测关键点，需要对以下两类整形操作进行检查：(1)赋值操作(通常是不同宽度或符号类型的变量之间的赋值)；(2)算术运算(加、减、乘)。

3 系统简介

针对上面提到的检测整形漏洞的关键，我们在 Linux 平台实现了一个检测系统，在二进制级别检测整形漏洞。本节介绍该系统的结构以及工作流程。我们需要 4 类信息来检测和定位整形漏洞：(1)变量的类型信息；(2)变量的最新值；(3)与变量相关的最近的算术运算；(4)赋给变量最新值的指令地址。利用前 3 类信息，可以检测整形漏洞。对于符号转换错误和赋值截断，检测变量的值是否在其类型范围内；而对于整形溢出和整形数下溢出，检测算术运算的结果是否在变量的类型范围内。最后一类信息可以用来定位漏洞代码。

系统建立在动态二进制分析工具 Valgrind[1] 及其类型引用(type inference)插件 Catchconv[2] 的基础上。其结构如图 3 所示。标签为①的数据流代表符号类型信息(Signedness type information)，而标签为②的数据流表示如下信息：宽度类型(width type)、操作数的值、指令地址以及算术运算。基于 Valgrind Core 和 Catchconv 提供的功能，系统提供了两个主要的功能组件，即信息记录器：当程序在 Valgrind 上运行时记录必要的信息；漏洞检测器：利用检测机制对整形漏洞进行检测和定位。

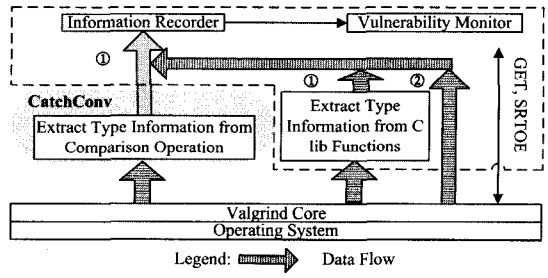

图 3 系统结构图

系统的工作流程可以分为 4 步：首先，将二进制码转换成 Valgrind 的中间语言 VEX。其次，程序运行时，将与外部输入相关的数据着色；截获相关的语句并记录信息(图 3 中标签为①，②的数据流信息)。最后，通过检测机制，结合记录的信息，检测着色的数据，并定位整形漏洞。

4 设计与实现

4.1 Valgrind 中间语言表示

系统建立在 Valgrind 3.2.2 平台上，使用的是 VEX 中间

语言。VEX 采用 RISC 指令集。其中有专门针对寄存器变量和内存变量的读写操作。对寄存器变量而言，“GET”操作将寄存器变量中的值读取到临时变量中，而“PUT”操作将临时变量的值写到寄存器变量中。对于内存变量而言，读操作是“LOAD”，写操作是“STORE”。语句“IMark”标记了某一条机器指令的地址。

4.2 数据着色

将与外部输入相关的数据着色要考虑以下 3 点：

(1)整形漏洞的特点。整形漏洞通常由外部输入引起，攻击者利用整形漏洞，精心构造输入数据，从而产生攻击。我们研究的 748 例漏洞均与外部输入相关。

(2)降低误报率。整形漏洞是逻辑漏洞，容易引起误报。编译器或程序员通常会在代码中有意或者无意使用一些操作，而这些操作通常与整形漏洞的特点相似。例如，编译器 gcc-3.4.0 在编译 if($x>=-3\&\&x<=0x7ffffffc$)时，如果使用了优化选项，会将这段代码翻译成如下汇编指令：

```
mov eax,x;   // eax=x
add eax,3;   // eax=eax+3
js target
```

如果 x 是 int 型，并且其值是 0x7fffffff 时执行 add eax，值是 3 时产生溢出。这种情况下，容易被认为是整形溢出漏洞。然而，如果 x 与外部数据无关，则是一个正常的操作，而非漏洞。因此，将与外部输入相关的数据区分开，能够减少误报率。

(3)提高性能。将与外部输入相关的数据着色，可以缩小检测的范围。特别是算术运算，由于获得类型信息较晚，系统需要重新计算算术运算，这将大大降低系统的性能。虽然数据着色会增加系统的开销，但实验表明，系统的性能有显著提高，由原来性能损耗 50 倍减低到现在的 30 倍。

系统的着色机制建立在 Valgrind 的工具 Flayer[19] 基础上。截获外部输入函数，将外部数据标记，并根据程序动态运行时的操作传播标记。着色方式依赖于操作数的最新值，即当操作数被赋予最新值时，根据该值，标记其是否与外部数据相关；当数据发生变化，标记也作出相应的调整。此外，具有值依赖的操作数，其对应的标记做同样的调整。

4.3 信息记录器

信息记录器的功能是程序在 Valgrind 上运行时，提取并记录本文第 3 节提到的 4 类信息。

类型信息：类型信息可以分为两类：宽度类型信息和符号类型信息。Valgrind Core 可以提供宽度类型信息，但不能提供符号类型信息。我们通过两种途径获得操作数的符号类型信息：(1)利用 Catchconv 从比较操作中提取符号类型信息。注意到比较操作通常与条件跳转指令(conditional jump instruction)相关联，条件跳转指令可以分为两类：无符号条件跳转(Unsigned Conditional Jumps)和有符号条件跳转(Signed Conditional Jumps)。根据不同的条件跳转指令，可以区分有符号操作数和无符号操作数[18]。(2)利用 C 库函数的参数定义来获取符号类型信息。实验表明，使用比较操作和 C 库函数的定义确定类型信息是充分的。当某个操作数既被用作有符号数，又被用作无符号数时，我们将其符号类型记录为“conflict”。

操作数的最新值：确定操作数类型后，还必须知道操作数的值。然而，程序执行过程中操作数的值在 Valgrind 中没有记录。此外，确定操作数类型时，原始的错误值可能已经发生改变，因此需要记录运行时每个操作数的最新值。

与操作数相关的最近的算术运算：当操作中包含算术运算表达式的时候，不能通过比较操作数的值及其类型来检测，因此需要记录与操作数相关的最近的算术运算。

指令地址：为了定位整形漏洞代码，需要记录赋给操作数最新值的指令地址，包括算术运算地址。

4.4 检测机制

本节针对每种整形漏洞的特点，分别设计检测机制。我们的检测机制是建立在本文 4.2 节的着色技术和 4.3 节记录的信息基础之上的。对着色的操作数根据漏洞的特点进行检测并且定位。

4.4.1 整形溢出

分两类来检测整形溢出：由乘法操作和加法操作引起的整形溢出。对于每一类，分两种情况来检测：(1)n 位有符号乘法和加法操作；(2)n 位无符号乘法和加法操作。对于 n 位的加法和乘法操作，我们首先截获类型信息为“Unsigned n”或“Signed n”的操作数，然后判断该操作数是否被着色。当确定该操作数与外部输入相关，回溯到相应的加法或乘法操作，检测是否有溢出发生。当检测到整形溢出时，找到包含算术运算的那条指令，这条指令就是整形溢出漏洞代码。

4.4.2 整形下溢出

与整形溢出相似，当设置操作数的类型为 Unsigned n 或者 Signed n 时，判断该操作数是否被着色。如果被着色，回溯到相应的减法操作，并检测是否发生溢出。当检测到整形下溢出时，漏洞代码地址是包含算术运算的那条指令地址。

4.4.3 符号数转换错误

将类型变量标记为“Conflict n”时，意味着可能发生了符号数转换错误。检测与该类型相关且与外部输入相关的操作数的值，判断其值是否为负数。当检测到符号数转换错误时，定位与操作数相关的那条指令地址即是漏洞代码的地址。

4.4.4 赋值截断

在 VEX 中，当把一个小于 32 位的数值写到内存变量时，这个值连同宽度类型信息由“GET”表达式传给寄存器变量，然后由“STORE”操作存入内存变量。因此对寄存器变量的读操作“GET”进行检查：首先判断该数值是否与外部输入相关，接着判断该值是否在其类型范围之内。如果不在，再检测下一个操作是否为“STORE”。如果是，意味着写入内存变量的值超过了其类型所能表达的范围。检测到赋值截断后，定位到包含“STORE”操作的指令地址，即是漏洞代码地址。

4.5 实验分析

我们选取了一些具有代表性的软件进行试验。实验的主要目的是评价我们的系统检测整形漏洞的有效性以及它的性能损耗。实验平台是 Intel Pentium(R) Dual E2180 2.00GHz，256MB memory 和 Linux 2.6.15 kernel。测试程序均由 gcc-3.4.0 编译，以及 glibc2.3.2 连接。

由于整形漏洞很难被模拟，在实验中，从每个软件中提取漏洞代码，加上导致错误的输入，转换成相对较小的、独立的漏洞程序进行检测。这些程序在系统的监控下运行，表 1 是有效性测试结果，漏洞均被检测并定位。

表 1　系统的有效性测试

CVE#	Program	Types	Detected?
2008-1384	PHP 5.2.5	Integer Overflow	√
2008-3794	VLC Media Player 0.8.6i	Signedness Error	√
2005-0199	ngIRCd-0.8.1	Integer Underflow	√
2003-0326	slocate-2.7	Integer Overflow	√
2001-0144	openssh-2.2.1	Assignment Truncation	√

我们还测试了系统检测的误报率。实验表明，在检测整形溢出和整形下溢出时，误报率微乎其微。由于这两种漏洞通常发生在算术运算时，而我们的系统只有在一些关键点（如比较操作和 C 库函数）才对算术运算进行重演，因此降低了一些可能的误报，例如编译器或程序员有时会根据需要故意产生运算溢出。然而，我们的系统仍不可避免地会有一些误报。此外，在检测符号类型错误和赋值截断错误时，也仅有很低的误报率，通常发生在程序员故意使用不安全的 C 操作的时候，如显式强制转换（explicit casts）等。为了进一步降低误报率，可以通过专门的漏洞判断测试工具[20]或人工审查的方式来实现。

我们同样使用一些实用软件来测试系统的性能。对每一个软件运行 3 次，分别在没有工具的监控下运行、Memcheck[9]监控下运行以及在我们系统的监控下运行。Memcheck 是一个在 Valgrind 上运行的应用很广泛的工具，它通过标记影子内存（shadow memory）来检测内存错误，然而 Memcheck 不能检测整形漏洞。表 2 显示了 Memcheck 和我们系统的性能测试结果。从表 2 可以看出，我们系统的性能损耗稍高于 Memcheck。前 4 个软件测试的结果是，性能损耗大约是 Memcheck 的 1.5 倍，因为这些软件存在大量的算术运算操作，系统在检测时需要花额外的时间来计算和检测这些操作。而后 3 个软件测试的结果显示系统的性能损耗接近 Memcheck，甚至低于 Memcheck（apache-2.2.0 和 gcc-3.4.0）。因为这些软件算术运算相对较少，节省了检测时间。平均性能下降约 32 倍。

表 2　系统的性能损耗

Program and Benchmark	Time(s)	memcheck	Our System
tar-1.15.1 archive 36MB data	4.999	3.2X	30.7X
pine-4.6.4 open a given url	3.087	3.9X	42.3X
apache-2.2.0 ab	2.249	25.9X	35.1X
Proftp-1.2.10 dkftpbench	2.607	11.7X	19.7X
average	3.236	9.2X	32.0X

5　相关工作

目前，针对整形漏洞防御的工具分为 4 类：①静态分析工具；②编译器补丁；③安全语言；④二进制检测。

静态分析工具通过分析源代码来检测整形漏洞。LCLint[11]使用类型系统和局部数据流分析检测由算术运算产生的整形漏洞。Dipanwita Sarkar et al.[12]提出了基于 PREfast AST infrastructure[13]的整数漏洞检测算法。PREfast 是一种源代码静态分析工具。然而，静态检测方法只能检测潜在的整形溢出和整形数下溢出漏洞，因为程序操作数的值只有在程序运行时才能得到，所以静态检测工具有很高的误报率。另外，静态检测方法不能检测符号数转换错误和赋值截断漏洞。

有的编译器补丁为 C/C++编译器如 GCC 提供安全函数，其主要思想是在程序运行中截获一些算术运算操作并且对结果进行检测。例如，GNU GCC 3.4.0 以后的版本都提供“-ftrapv”命令行选项，使得 GCC 在编译代码的时候插入安全函数[3]。但是，这些安全函数仅仅是针对算术运算的，因此不能充分地检测整形漏洞。也有一些是 C/C++编译器的补丁，但它们大多需要源代码从中提取类型信息，并且具有较高的误报和漏报率。

整形漏洞的根源在于不安全类型（type unsafe），因此防御这种漏洞的根本方法是设计类型安全的语言。有的语言是类型安全的，例如 Standard ML[21]和 Ada[22]。它们使用基于正则类型理论定义（formal type-theoretic definition）的类型系统（type system）为每个整形操作提供了类型安全规则。安全语言可以在编译时静态截获整形漏洞，同时可以在运行时比较类型与值，并给予警告。但是，很难将 C 语言转换成这些安全语言。也有的工作致力于将 C 语言转换为安全语言，如 CCured[7]和 Cyclone[8]。虽然这些方法可以抵挡多种漏洞，包括整形漏洞和缓冲区溢出漏洞，但是通常需要人工将 C 语言转换成这些安全语言。

前面提到的技术都需要源代码才能对整形漏洞进行保护。尽管像 RICH[3]这样的工作能较有效地保护整形漏洞，但是在实际防御中源代码很难得到。另外，有些信息在源代码中也很难分析。例如，RICH 不能处理与指针别名相关的整形漏洞，因为它很难从源代码分析出两个不同类型的指针指向同一个内存空间。而在二进制代码检测时，两个不同类型的指针访问同一个内存采用的是共享地址的方式，因此很容易识别。目前已有一些二进制级检测技术，例如 UQBTng[5]，一个在 Win32 二进制代码中检测整形溢出的工具，它使用 UQBT[6]来反汇编 Win32 PE 文件，并且在每次内存分配时，检测分配空间的值有没有被溢出。但是 UQBTng 不能检测与指针相关的整形溢出，也不能检测其他类型的整形漏洞，如符号数转换错误和赋值截断。Catchconv[2]是 Valgrind 上检测整形数转换错误（integer conversion error）的工具，它的类型引用（type inference）方法对我们的工作有很大的帮助。

结束语　本文设计并实现了针对整形漏洞的二进制级实时检测和定位系统。主要用 4 类信息来检测和定位整形漏洞：变量的类型信息、变量的最新值、与变量相关的最近的算术运算以及赋给变量最新值的指令地址。利用前 3 类信息可以检测整形漏洞。对于符号转换错误和赋值截断，检测变量的值是否在其类型范围内；而对于整形溢出和整形数下溢出，检测算术运算的结果是否在变量的类型范围内。最后一类信息可以用来定位漏洞代码。实验表明，我们的系统可以有效地检测和定位整形漏洞，而且误报率和漏报率都很低。

参考文献

[1] Nethercote N, Seward J. Valgrind: A framework for heavy weight dynamic binary instrumentation[C]// Proceedings of PLDI 2007. San Diego, California, USA, June 2007

[2] Molnar D A, Wagner D. Catchconv: Symbolic execution and runtime type inference for integer conversion errors[C]// Proceedings of EECS. 2007

[3] Brumley D, Chiueh Tzi-cker, et al. RICH: Automatically Protecting Against Integer-based Vulnerabilities[C]// Proceedings of

the 14th Annual Network and Distributed System Security. Symposium(NDSS07). 2007
[4] Vulnerability Type Distributions in CEV[EB/OL]. http://cve.mitre.org/docs/vuln-trends/vuln-trends.pdf,2007
[5] Wojtczuk R. UQBTng:a tool capable of automatically finding integer overflows in Win32 binaries. November 2005
[6] Cifuentes C, et al. UQBT[EB/OL]. http://www.itee.uq.edu.au/.cristina/uqbt.html
[7] Necula G C, McPeak S, Weimer W. CCured: type safe retrofitting of legacy code[C]//Proceedings of the Symposium on Principles of Programming Languages. 2002
[8] Jim T, Morrisett G, Grossman D, et al. Cyclone: A safe dialect of c[C]//USENIX Annual Technical Conference. 2002
[9] Seward J, Nethercote N. Using valgrind to detect undefined value errors with bit-precision[C]//Proceedings of the USENIX05 Annual Technical Conference. Anaheim, California, USA, April 2005
[10] Howard M. Integer overflow and operator::new[EB/OL]. http://blogs.msdn.com/michael_howard/archive/2005/12/06/500629.aspx, Dec. 2006
[11] Evans D, Guttag J, Horning J, et al. LCLint: A tool for using specification to check code[C]//Proceedings of the ACM SIGSOFT 94 Symposium on the Foundations of Software Engineering. 1994:87-96
[12] Larus J, et al. Righting software[C]//IEEE SOFTWARE. IEEE Computer Society, 2004
[13] Sarkar D, et al. Flow-insensitive Static Analysis for Detecting Integer Anomalies in Programs[C]//Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering. 2007
[14] PaX Project. The PaX project[EB/OL]. http://pax.grsecurity.net/,2004
[15] Akritidis P, et al. Preventing memory error exploits with WIT [C]//IEEE Symposium on Security and Privacy. May 2008
[16] Budiu A M, Erlingsson M, Ligatti J. Control-flow integrity[C]//Proceedings of the 12th. ACM Conference on Computer and Communications Security(CCS05). 2005
[17] http://www.cve.mitre.org/cgi-bin/cvekey.cgi? keyword=integer
[18] Intel Corporation. IA-32 Intel Architecture Software Developers Manual-Volume1: Basic Architecture[S]. November 2006
[19] Drewry W, Ormandy T. Flayer: Exposing Application Internals [C]//Proceedings of the First USENIX Workshop on Offensive Technologies(WOOT '07). Boston, Massachussetts, USA, August 2007
[20] Lin Z, Zhang X, Xu D. Convicting exploitable software vulnerabilities: An efficient input provenance based approach[C]//Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks(DSN'08). Anchorage, Alaska, USA, June 2008
[21] Gilmore S. Programming in Standard ML[Z]. 1997
[22] Ada95 Language Reference Manual[M]. ISO/IEC, 1995

(上接第 121 页)

AES 的 128 位密钥的 CBC 模式加密来保护传输的数据，使用 MD5 算法做 ESP 阶段的消息认证码计算，移动终端客户程序采用 handyGet。测试得到的数据如表 1 所列。从测试的数据中可以看到，加入 SSL 处理后，传输时间增加大约 20%，其并没有由于传输文件的增大而增加。

表 1 系统接入速度的传输效率

文件大小(MB)/应用协议	隧道建立时间(s)	ftp 协商时间(s)		数据传输时间(s)		传输速率(kB/s)	
		不使用 VPN	使用 SSL 隧道保护	不使用 VPN	使用 SSL 隧道保护	不使用 VPN	使用 SSL 隧道保护
1/ftp	5	12	17	192	215	5.2	4.7
2.66/ftp	5	14	15	524	731	5.0	3.6
7.5/ftp	5	15	13	1595	1916	4.7	3.9

结束语 本文提出的移动 SSL VPN 方案采用在应用层级别上利用 Socks5 协议进行数据转发搭建安全接入 VPN 的方式，将客户端应用程序到局端应用服务的 TCP 连接利用 Socks5 代理中继分为 3 段，Socks5 代理功能由异地的终端和局端共同实现。系统改善了 SSL VPN 在无线网络中的连接性能，减小了无线网络速率极不稳定、易断线的缺点带来的连接性能影响，很好地支持了移动漫游时的安全稳定接入。在跨越上海、江苏、浙江的外场移动漫游的联动实测验证了系统的安全性能和使用效率。目前该系统已初步在物流运输部门投入实际运用，并表现了较高的现实意义和应用价值。

参 考 文 献

[1] Kent S. Security Architecture for the Internet Protocol [S]. IETF RFC 2401. Nov. 1998
[2] Vaarala S, Klovning E. Mobile IPv4 Traversal across IPSec-Based VPN Gateways[S]. IETF RFC 5265. 2008
[3] Devarapalli V, Eronen P. Secure Connectivity and Mobility Using Mobile IPv4 and IKEv2 Mobility and Multihoming(MOBIKE)[S]. IETF RFC 5266. 2008
[4] Benenati D. A seamless mobile VPN data solution for CDMA 2000, UMTS and WLAN users[J]. Bell Labs Technical Journal, 2002,7(2):143-165
[5] 薛海波. 移动 VPN 的研究和实现[D]. 北京:北京交通大学, 2006
[6] Nokia Inc. White Paper: The Evolution of Mobile VPN and its Implications for Security [EB/OL]. http://www.nokia.com, 2009
[7] BirdStep Corp. Introducing Birdstep Intelligent Mobile IP, v2.0 Universal Edition [EB/OL]. http://www.birdstep.com, 2009
[8] Cisco Systems. Enterprise Mobile Wireless Data Solutions 1.0 [EB/OL]. White paper. http:/www.cisco.com/, Aug. 2003
[9] Dierks T, Allen C. The TLS Protocol Version 1.0[S]. IETF RFC 2246. January 1999
[10] Wireless Transport Layer Security Specification[EB/OL]. WAP Forum, February 2000
[11] Kim K, Hong J, Lim J. A Secure and Efficient Communication Resume Protocol for Secure Wireless Networks[Z]. International Federation for Information Processing, 2005
[12] Columbitech Wireless VPN technical Description [EB/OL]. Columbitech AB. ttp://www.columbitech.com/Products/WVPN.asp, 2004
[13] Goutham Rao. NET6 Hybrid-VPN Gateway [EB/OL]. http://www.citrix.it/REPOSITORY/docRepository/id_900_111297 9921897309.pdf, 2008
[14] Marcus L. SOCKS Protocol Version 5[S]. IETF RFC 1928. 1996