基于攻击特征签名的自动生成

王国栋 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 签名可以基于攻击特征的相关信息生成。在栈上针对控制流攻击中对函数调用返回值和函数调用指针的攻击以及非控制流中对与判断相关联的数据的攻击,结合动态分析技术生成二进制签名。首先,识别出漏洞相关指令;然后,用虚拟机监控运行上述指令;最后,修改虚拟机以在监控到恶意写行为时报警并生成签名。同时生成的补丁文件记录恶意写指令以便后继执行时跳过。签名可迅速分发给其他主机,在轻量级虚拟机上监测程序运行。实验表明,二进制签名具有准确、精简的优点,可以防御多态攻击,同时具有较低漏报率,结合使用轻量级虚拟机可使签名生成和后继检测都快速高效。

关键词 计算机安全,软件安全,软件漏洞,二进制程序签名,二进制补丁
中图法分类号 TP309.5 **文献标识码** A

Automatic Generation of Attack-based Signature

WANG Guo-dong CHEN Ping MAO Bing XIE Li
(State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract Signatures can be generated based on characteristics of attacks. Using dynamic program analyzing skills we generated binary signatures for control flow attack to return value of function call and function call pointer, and non-control flow attack to decision-related variable. First, we identified instructions related to the vulnerability. Second, we monitored these instructions using a modified virtual machine. At last, we alerted and generated signature after finding any malicious write behaviors. Patch recorded malicious write instructions could be generated meanwhile to ignore these instructions in future execution. Generated signature could be sent to other computers to monitor the same software's execution using lightweight virtual machine. Experiment results show that binary level signature has simplified form and precise functionality and low false negative and is effective to defense polymorphic attack. Besides, lightweight virtual machine makes use of the signature fast.

Keywords Computer security, Software security, Software vulnerability, Binary signature, Binary patch

1 引言

软件漏洞一直是困扰业界的一大难题,CVE的统计显示,各种软件漏洞总数一直呈逐年增加之势,且多种软件漏洞连年频发。

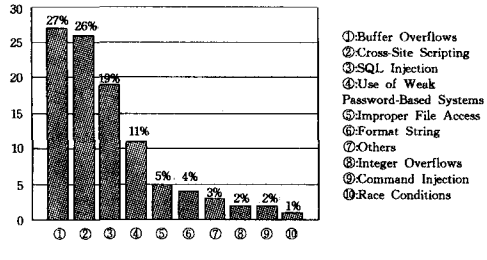

图1 2006年CVE公布的漏洞类型分布

由图1可见,缓冲区溢出攻击、字符串格式化攻击等溢出攻击在总攻击类型中占有很大比例(约31%),这些攻击也因其相关漏洞普遍存在和易于发起而影响许多常用软件(如Apache、WU-Ftp等)的运行,因此对这几种攻击进行研究和防御具有重要意义。

针对上述常见漏洞,本文从控制流和非控制流两个方面进行研究,其中控制流主要关注针对函数返回值和函数指针的攻击;在非控制流方面,其关注对判断相关变量(decision-making related data)进行篡改的攻击。方法首先使用虚拟机对漏洞点进行定位,然后动态模拟运行程序并修改虚拟机来定位执行恶意操作的指令和被恶意操纵的目标地址。最终,将恶意指令的地址和被恶意操纵的目标地址及其他辅助信息作为签名,并且生成二进制补丁,使程序在后继执行中跳过恶意写指令,让程序不用终止或崩溃即可继续运行。在此过程中还可以得到漏洞点在源代码中的位置,以提供给软件厂商

到稿日期:2011-03-30 返修日期:2011-07-21 本文受国家863高技术研究发展计划(2007AA01Z448),国家973重点研究发展规划(2009CB320705),国家自然科学基金(60773171),江苏省自然科学基金(BK2007136)资助。

王国栋 硕士,主要研究方向为软件安全、网络安全,E-mail:shepherderwang@gmail.com;陈 平 博士,主要研究方向为软件安全、系统安全;茅 兵 教授,博士生导师,主要研究方向为系统安全和分布式系统;谢 立 教授,博士生导师,主要研究方向为分布式系统。

作为参考。

方法可以防御多态攻击(Polymorphic attack)。多态攻击的特点是攻击(病毒、蠕虫等)改变自身形态,在各个攻击中呈现不同形式以使得传统特征值检测方法失效。我们抽取出精确的漏洞点信息而不是仅考虑攻击本身形式,因此可以有效防御多态攻击。

2 相关工作及背景

程序签名作为一种重要的防御手段被广泛研究,最初针对网络攻击的系统 Bro[1] 和 SNORT[2] 在服务器端针对特定协议监控网络流,借用手动分析的病毒特征字符串进行模式匹配,其低效而且反应迟缓;随后出现的 Honeycomb[3]、Autograph[4]、EarlyBird[5] 等系统将网络流分类为良性网络流和可疑网络流,在可疑网络流上训练提取各网络包中某一连续字母串组成的不变部分来作为签名,并使用良性网络流对签名进行调整和优化,但这几种方法对随后出现的多态病毒无效;Polygraph[6]、Hamsa[7]、LISABETH[8] 等分析多态病毒的组成,针对可疑网络流使用不同算法提取各网络包中多态病毒体的不变部分作为签名;文献[3-8]中的系统使用模式匹配的方法在对网络流进行分类时都受到精确性上的限制,各自都具有不同程度的漏报或者误报。文献[9-11]更是针对上述方法提取不变模式作为签名的特点,在良性网络流中插入精心构造的伪病毒体来破坏签名,令得到的签名对真正的病毒无效。除上述方法之外,文献[12,13]提取运行程序的控制流图来监控程序可能出现的异常行为,这一类方法可以防御多态攻击,但抽取程序运行时控制流图使用的静态程序分析方法会带来非常大的运行开销,并且具有自身的局限性。

在基于漏洞类型的签名生成方面,J. Newsome 和 D. Song 等在文献[14,15]中通过对内存单元染色,来追踪数据在代码和内存中的传播路径,并借助动态程序分析方法对非法写内存等行为提供准确的程序签名,但他们对外部数据传播路径上的所有内存进行染色,致使生成签名的开销很大。类似于软件方法上的染色,Loki[16] 在硬件级别对内存打标签,但这种硬件增强的方法明显不具有通用性。ARBOR[17] 比较正常外部输入和程序被攻击前后的输入,发现异常时将后者作为程序签名,但没有分析和使用漏洞本身的信息就生成了签名。D. Brumley 等在文献[18]中寻找外界输入和程序漏洞点之间的关系,使用静态程序分析方法分析得到两者之间的一个以函数调用图为形式的程序切片,并最终以切片中被执行到的指令作为程序签名。Trag[19] 通过假设攻击可被再构造,根据运行状态日志再构造攻击,模拟得到崩溃前程序运行内存数据之间的依赖关系和条件分支路径,最后通过后向切片技术得到攻击时的指令序列。在以上两种方法中,切片技术本身的效率和准确性尚不确定。此外,前者签名的大小可能随着漏洞数量的增加呈指数级增长;后者签名由单次模拟执行路径上的指令组成,不一定对所有同类攻击都有效。

在本文提出的方法中,签名只由动态定位到的触发漏洞的恶意写指令和被恶意写向的地址组成,签名本身简洁、生成过程高效,虽然借助执行路径提供定位信息,但是签名本身不包括执行路径;另一方面,签名不涉及外部输入本身,因此对多态攻击有效,同时使用轻量级检测工具执行后继运行时的检查,使得生成的签名形式简洁且能高效使用。

3 方法描述

在控制流相关的攻击中,考虑的是针对函数返回值(ret 指令)和函数指针(call 指令)的攻击;在文献[20]中,P. Chen 等将常见非控制流攻击分为 4 类,其中对判断相关变量的攻击通过改变普通变量的值间接影响控制流的转变,也引起我们的重点关注。

3.1 针对控制流攻击的防御

控制流攻击是指通过篡改控制流相关的指令(如 ret 指令、jmp 指令、函数调用指令)来改变程序执行控制流的攻击方法,其中针对函数返回值(ret 指令)和函数指针(jmp 指令)这两种情况占有大部分比例,本文主要关注这两种控制流攻击。

3.1.1 针对函数返回值攻击的防御

针对函数返回值的攻击(文中称之为攻击类型一),其特点是:不管最终使用什么样的手段,篡改函数调用时保存的返回地址,使被调用函数执行 ret 指令时偏离正常的控制流,而跳转至事先插入的恶意指令继续执行。常见的篡改方式包括缓冲区溢出攻击和格式化字符串攻击,作用原理如图 2 所示(RA 即返回地址,图中通过向 temp 拷贝字母串导致越界而将其篡改)。

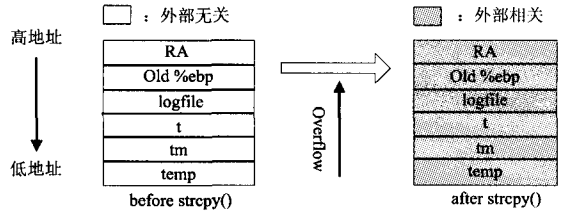

图 2 常见的篡改返回地址方式

针对此类攻击的特点,我们借助于 Valgrind 虚拟机,修改 Valgrind 的染色增强工具 Flayer,来识别并保存函数调用时存在栈中的返回地址。

定义 1 每次函数调用时保存的返回地址称为备份返回地址,记作 backup_addr。

定义 2 每次被调函数返回时,将此时栈中的返回地址值称为当前返回地址,记作 cur_addr。

然后识别出 ret 指令,当被调用函数执行到 ret 指令时,执行规则 1。

规则 1

if (curr_addr == backup_addr)
then continue_execute
else
generate sig_ret and alert

如发现二者不同,则将此次函数调用的 call 指令地址和出错的 ret 指令地址作为程序签名,签名生成过程示意如图 3 所示。

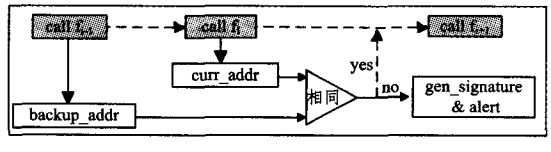

图 3 签名 sig_ret 生成过程

3.1.2 针对函数指针攻击的防御

针对函数指针的攻击(攻击类型二),通过篡改操作数位于内存之中的 call 指令的操作数值,也即被调用函数的入口地址,来改变程序执行的控制流。常见的篡改方式包括缓冲区溢出攻击和格式化字符串攻击等,作用原理参考图 2。

我们的方法基于以下设定 1:

设定 1 在针对函数指针的攻击中,程序崩溃前最后一条向 call 指令的操作数执行写操作的指令是导致崩溃的恶意写指令。

定义 3 程序崩溃前所有的对内存执行写操作的指令为 write_1,write_2,…,write_n。

定义 4 导致程序控制流转移并导致程序崩溃的 call 指令称为恶意 call 指令,记作 call_evil;修改 call 指令操作数的写指令称为恶意写指令,记作 write_evil。

我们的方法记录程序运行时所有的对内存执行写操作的指令 write_i,同时识别出每一条 call 指令。程序执行因函数调用导致控制流转移并最终崩溃时,最后一条 call 指令即为 call_evil 指令。识别出 call_evil 后,使用规则 2 来定位 write_evil。

规则 2 初始化 i=n (注:operator 为取操作数操作)

```
repeat:
    if (operator(call_evil) ! = operator(write_i))
        then i=i-1
            goto repeat
    else
        write_evil=write_i
    end
```

定位到 write_evil 之后,write_evil 指令的地址可用于下面提到的二进制程序补丁,而 call_evil 指令的地址则用于程序签名,整个过称如图 4 所示。

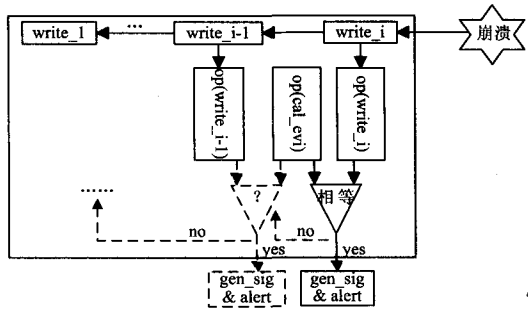

图 4 签名 sig_call 的生成过程

3.2 针对非控制流攻击的防御

关注文献[20]中提到的对判断相关变量的非控制流的攻击(攻击类型三),此类攻击篡改与判断相关的变量来影响程序在条件判断语句处的分支,间接更改程序运行控制流。常见的篡改变量方式包括缓冲区溢出攻击、格式化字符串攻击、整形溢出攻击等。

针对此类攻击的特点,在局部变量之间插入保护变量来对其进行识别和防御,程序局部变量间的保护变量需事先手动插入。

定义 5 与判断相关的变量为关键变量(Critical Value),关键变量在内存中的地址记作 addr_cri。

定义 6 局部变量之间插入的保护变量为 Garget,每个 Garget 的地址记作 addr_gar_i,初值为 garget_value_i,后继读取时的当前值称为 cur_value_i。

在小端(Small Endian)体系结构计算机上,插入保护变量后的程序,其每个变量在低地址方向的相邻处都有一个 Garget,如图 5 所示。每个变量插入 Garget 的一个初始值,即 garget_value_i。

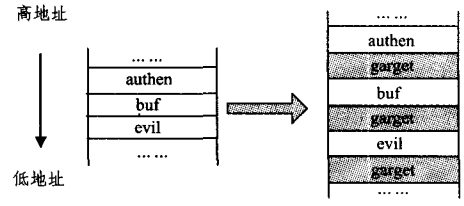

图 5 在局部变量间插入 Garget

程序运行时,对程序的监控和签名的生成按如下步骤执行:

1)在二进制级别识别条件判断语对应的 cmp 指令;

2)通过 cmp 指令识别出关键变量;

3)分析出关键变量在内存中的地址 addr_cri;

4)然后根据规则 3 计算与该关键变量相关联的 Garget 的地址 addr_gar_i;

5)读取 addr_gar_i 中的值,即 cur_value_i,使用规则 4 将之与 garget_vaule_i 对比,判断其是否有对关键变量的攻击。

规则 3(Little Endian)

addr_gar_i=addr_cri-4

规则 4

```
if (cur_value_i == garget_value_i)
    then continue_executeelse
    generate sig_noncon and alert
```

一旦检测到 Garget 数值被更改,便可认为外部输入有篡改程序内部变量、尤其是关键变量的企图,此时,将该 Garget 的地址 addr_gar_i 和使用关键变量的 cmp 指令的地址作为程序签名,整个流程示意如图 6 所示

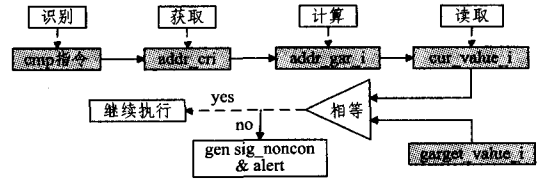

图 6 签名 sig_noncon 生成流程

3.3 生成程序补丁

我们拓展签名生成过程,同时生成程序的二进制级别补丁,二进制补丁由导致程序崩溃的恶意写指令构成,用于在程序的后继执行中跳过恶意写指令。判断指令为恶意指令的依据为:

定义 7 导致函数调用返回值、函数调用指针操作数、判断相关变量被更改的写指令为恶意写指令。

生成二进制程序补丁的思路和过程为:

1)定位到出错指令(比如,出错 call 指令);

2)识别出错指令的操作数地址(比如,出错 call 指令的被

调用函数入口地址的存储地址)；

3)在出错指令之前的所有写指令中，查找目标地址为出错指令操作数的最后一条写指令，该写指令即为恶意写指令；

4)将恶意写指令、被恶意写向的地址、出错指令的下一条指令作为二进制的程序补丁；

生成二进制程序补丁之后，对补丁的使用形式如图 7 所示。即二进制程序补丁的使用方式为：若检测到某些触发了程序漏洞的二进制恶意写指令，则在程序的后继执行中跳过这些恶意指令。

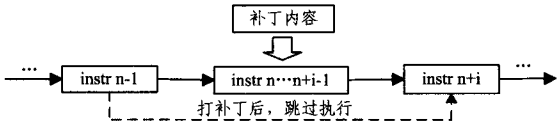

图 7　二进制补丁的使用形式

4　设计与实现

4.1　系统设计框架

利用程序染色技术跟踪外部输入在被监控程序中的传播情况。随着程序的执行，记录传播外部输入数据的相关指令及指令操作数地址，最终在漏洞触发时识别并回溯处理外部数据传播路径上的指令。系统通过修改开源虚拟机 Valgrind 及其上的工具 Flayer 来实现。

Valgrind 是一个动态二进制代码分析平台，它将二进制代码转换成 VEX 指令(一种类 RISC 指令集)的中间代码再执行。Valgrind 提供的染色功能记录外部输入数据在程序中的传播路径，对同一外部数据可对传播路径上涉及的所有内存地址打上相同的特征标签，即执行同一颜色的染色，工具 Flayer 对这一染色功能进行增强，可以执行比特级别的细粒度染色。

如图 8 所示，系统主要包括如下几个部分：

①，④：实现程序二进制代码和 VEX 中间代码的相互转换，由 Valgrind core 自动完成；

②：识别二进制指令对应的 VEX 中间代码指令并记录外部输入的传播路径，同时对传播路径上的内存地址进行跟踪染色，通过修改 Flayer 工具来实现；

③：针对 3 种不同类型的攻击，在识别出攻击时生成程序签名以及二进制补丁，通过修改工具 Flayer 来实现；

⑤：生成签名和补丁后，用轻量级检测工具 Pin 监控同一程序的后继执行，监控可在本机执行，也可将签名和补丁文件与程序一起分发到其他机器执行。

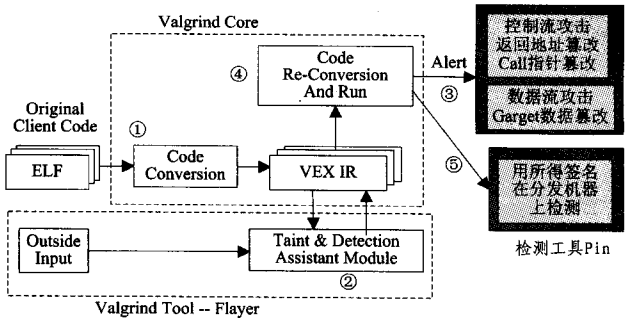

图 8　系统实现结构图

4.2　系统实现

实现时，假设程序本身在运行时不会崩溃，只有外部输入造成的异常行为会导致程序崩溃。针对上述控制流攻击和非控制流攻击，提出相应方法来生成程序签名。

4.2.1　针对函数返回值攻击的防御实现

如定义 1、定义 2、规则 1 所述，首先识别 call 指令来分析备份返回地址 backup_addr，以实验程序为例，call 指令及其对应 VEX 中间代码指令序列为：

```
8049de7:        E8 94 04 00 00      call 804a280
------------------------------------------------
->t26 = Sub32(t21,0x4:I32)
->PUT(16) = t25
->STle(t26) = 0x8049DEC:I32
```

(横线上方 8049de7 为 call 指令地址，E894040000 为 call 指令的二进制编码；横线之下为对应的 VEX 指令序列。后文格式含义同此处。)

每次识别出顺序出现的 Sub32、PUT、STle 指令即是识别出了一条 call 指令，而 ret 指令及对应 VEX 指令序列为：

```
804a35c:        c3                  ret
------------------------------------------------
->t12 = LDle:I32(t16)
->t17 = Add32(t16,0x4:I32)
->PUT(16) = t17
```

在实现中，我们模拟实现一个函数调用栈 call_stack，按照函数调用顺序记录 call 指令的地址，根据栈的后入先出原理，遇到一个 ret 指令时，便知其与当前 call_stack 栈顶 call 指令对应。根据程序执行行为，正常情况下函数调用的返回地址为当前 call 指令下一条指令的地址，而 call 指令的长度为 5 个字节，于是在实现中用以下代码来进行判断，其中 tp_inst_addr 为当前指令的地址：

```
if(tp_inst_addr-5==call_stack[call_number-1])
```

若判断成真，从 call_stack 中弹出栈顶 ret 指令即可；若判断为假，便知返回地址已经被修改，系统将给出警告并且生成针对该攻击的程序签名，签名的形式以试验中软件 Ghttpd-1.4 为例，即为：

```
0x08049de7
0x0804a35c
```

其中 0x08049de7 为出错 ret 指令对应的 call 指令 0x0804a35c 为导致出错的 ret 指令。

4.2.2　针对函数指针攻击的防御实现

实现针对操作数在内存的 call 指令，这样的 call 指令之前有一条 mov 指令，其 VEX 指令序列为：

```
8048594:    A1 94 98 04 08      mov 0x8049894,%eax
------------------------------------------------
->t4 = LDle:I32(0x8049894:I32)
->PUT(0) = t4
```

call 指令的 VEX 指令序列仍如 4.2.1 节中所示。

实现中，首先捕捉程序的出错信息以定位出错时的 call 指令，然后取出 call 指令的操作数，也即存储被调用函数入口地址的内存地址。利用 Flayer 的染色功能，判断该内存地址内数据是否与外部输入数据具有相同颜色。如没有被染色，那么此次调用安全；否则被调函数入口地址已被外部输入篡改，并导致程序运行出错，此时需要生成程序签名。

生成程序签名时，根据函数入口地址的存储地址扫描出错前调用该存储地址的 call 指令，将每次调用的良性函数入口地址作为白名单，与出错的 call 指令一起加入签名文件，签名文件形式为：

- 良性函数入口地址
- ……
- 良性函数入口地址
- 出错call指令地址

4.2.3 针对非控制流攻击的防御实现

实现时，预先向被监控程序的局部变量之间插入保护变量 Garget，如定义 6，Garget 具有设定的初始值 garget_value_i。

根据 3.2 节，首先识别 if 语句对应的二进制 cmp 系列指令，试验中为 cmpl 指令，其 VEX 指令序列为：

```
8048447:  83 7d f4 00   cmpl $0x0,-0xc(%ebp)
------------------------------------------------------------
->t5 = GET:I32(20)
->t4 = Add32(t5, 0xFFFFFFFF:I32)
->t2 = LDle:I32(t4)
->IR-NoOp
->PUT(32) = 0x6:I32
->PUT(36) = t2
->PUT(40) = 0x0:I32
->PUT(44) = 0x0:I32
```

识别 cmpl 指令后，取其操作数地址 addr_cri；根据规则 3 计算对应保护变量地址 addr_gar_i，取出 addr_gar_i 的当前数值 cur_value_i，再根据规则 4，比较其和保护变量初始值 garget_value_i。若无变动，此次判断可信；否则，有攻击通过篡改判断相关变量来改变程序执行路径，此时应生成程序签名。程序签名呈以下形式：

- 此次if判断语句对应cmpl指令的地址
- 该cmpl指令的操作数对应的保护变量的地址*addr_gar_i*
- 保护变量的初始值*gatget_value_i*

4.2.4 生成二进制程序补丁的实现

生成签名的攻击相关信息，也可用于生成由恶意写指令构成的二进制程序补丁。

以针对函数指针攻击的程序签名为例，根据 3.3 节中的补丁生成过程生成签名时可定位到程序运行的出错指令（call 指令）；再分析出错指令的操作数地址（被调函数入口地址的存储地址）；最后，使用该存储地址逆序查找程序崩溃前向该地址执行写操作的最后一条指令，即是导致程序崩溃的恶意写指令。然后便可生成二进制补丁。

4.2.5 轻量级检测工具 Pin 的使用

在实现中生成程序签名后，将程序和二进制签名文件、补丁文件一起分发给其他机器，使用轻量级检测工具 Pin 监测程序的后继运行。Pin 是一个轻量级虚拟机。针对上述不同类型攻击，Pin 使用对应签名文件或补丁文件，在执行到用作判断位置的指令（如出错 call 指令）时取出对应的辅助信息（如 4.2.2 节中的良性函数入口地址）执行比对操作以及对应的补救操作。检测工具 Pin 的整合使用，加快了二进制签名和补丁的检测过程。

5 实验结果

为评估方法的有效性，进行了一系列实验，本节将详细描述实验结果。实验测试的平台参数为：Pentium Dual E2180、2G 内存以及 Linux Kernel 2.6.15，gcc-3.4.0，glibc-2.3.3。

对签名文件和补丁文件的使用可参考第 3 节。

5.1 有效性测试

选用软件 Ghttpd-1.4（一款轻量级的开源服务器程序，具有格式化字符串漏洞，测试攻击类型一）、NCompress-4.2.4（一款压缩程序，具有缓冲区溢出漏洞，测试攻击类型一）、我们自己编写的程序 vulnerability.c（具有缓冲区溢出漏洞，测试攻击类型二）以及修改自 CVE 2001-0144 的程序 authentication.c（具有缓冲区溢出漏洞，测试攻击类型三），使用检测工具 Pin 对生成的签名文件和补丁文件进行测试。

测试结果如表 1 所列，我们的程序防御住了所有 3 种类型的攻击，并针对函数指列针攻击生成了有效的二进制程序补丁，其中 Test type 表明测试的是签名文件还是二进制补丁文件。

表 1 有效性测试结果

CVE	Program	Test type	Effection
CVE-2002-1904	Ghttpd-1.4	signature	√
CVE-2001-1413	NCompress-4.2.4	signature	√
#	vulnerability.c	signature	√
CVE 2001-0144	authentication.c	signature	√
#	vulnerability.c	patch	√

5.2 对多态攻击的测试

针对 Ghttpd-1.4 和 NCompress-4.2.4，使用不同形式的输入进行攻击，如表 2 所列，实验结果表明，我们的程序能够防御住针对这两款软件的多态攻击。

表 2 对多态攻击的测试

CVE	Program	Test type	Effection
CVE-2002-1904	Ghttpd-1.4	多态攻击	√
CVE-2001-1413	NCompress-4.2.4	多态攻击	√

5.3 效率测试

使用 Ghttpd-1.4 和 NCompress-4.2.4 这两款公开发布的开源软件进行测试，并将之和 Newsome 等人在文献[15]中的工作（简称 VSEF）进行比较。如表 3 所列，使用我们的方法对程序进行监控具有较小的性能损耗，比文献[15]中的工作提升 7～10 倍。

表 3 性能测试

	Ghttpd-1.4	NCompress-4.2.4
有监控	0.770s	3.224s
无监控	0.640s	2.848s
性能损耗	20.3%	13.9%
VSEF 的性能损耗	约 140%	

6 讨论与后续工作

出于功能和效率上的考虑，我们的方法存在的一定的局限性，第一，模拟攻击和进行防御实现的时候都关闭了 Linux 系统的地址随机化功能；第二，在使用二进制程序补丁时，采取跳过恶意写指令的策略，不知这样对程序的正确执行是否有影响；第三，在针对函数指针攻击的防御中，只考虑 call 指令的操作数在内存中的情况；第四，在针对非控制流攻击的防御中，对被监控函数局部变量间的保护变量的插入，需要提前手动完成，而且，当保护变量的初始值被攻击者猜中的时候，对攻击的判断会失误，方法也会失去防御效果；最后，对于被攻击程序的后继监控，需要将可该程序文件和签名文件一起分发，且检测只对同一类型的攻击有效。

到目前为止，我们的工作主要在栈上进行，接下来可能针对程序的染色效率及能够防御的攻击类型范围进一步研究，并且考虑对堆上攻击的防御。

结束语 本文针对控制流攻击中对返回地址的攻击、对

函数指针的攻击以及非控制流攻击中对判断相关变量的攻击提出相应的防御方法，借助具有内存染色功能的虚拟机 Valgrind 及其上的工具 Flayer 进行实现，生成针对各种攻击的二进制签名文件，并同时生成二进制补丁文件。使用轻量级检测工具 Pin 进行检测表明，生成的二进制签名文件和补丁文件能高效使用。

参 考 文 献

[1] Vern P. Bro: A System for Detecting Network Intruders in Real-Time[C]//Proceeding USENIX Security Symposium. 1998

[2] Martin R. Snort-Lightweight Intrusion Detection for Networks [C]//Proceeding of LISA '99

[3] Christian K, Jon C. Honeycomb Creating Intrusion Detection Signatures Using Honeypots [C] // Proceeding of the Second Workshop on Hot Topics in Networks. 2003

[4] Hyang-Ah K, Brad K. Autograph: Toward Automated, Distributed Worm Signature Detection[C]//Proceeding of the USENIX Security Conference. 2004

[5] Sumeet S, Cristian E, George V, et al. Automated Worm Fingerprinting[C]//Proceeding of OSDI'04

[6] James N, Brad K, Dawn S. Polygraph: Automatically generating signatures for polymorphic worms [C] // Proceeding of IEEE S&P. 2005

[7] Li Zhi-chun, Manan S, Yan Chen, et al. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience[C]//Proceeding of IEEE S&P. 2006

[8] Lorenzo C, Andrea L, Luca M, et al. LISABETH: automated content-based signature generator for zero-day polymorphic worms [C] // International Conference on Software Engineering, Proceedings of the fourth international workshop on Software engineering for secure systems. 2008

[9] Aleg K, Wenke L. Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic[R]. Georgia Tech College of Computing, 2004

[10] Roberto P, David D, Wenke L, et al. Misleading Worm Signature Generators Using Deliberate Noise Injection[C]//Proceeding of IEEE S&P. 2006

[11] James N, Brad K, Dawn S. Paragraph: Thwarting Signature Learning by Training Maliciously [C] // Proceeding of RAID. 2006

[12] Ramkumar C, van den Berg E. A fast static analysis approach to detect exploit code inside network flows[C]//RAID. 2005

[13] Christopher K, Engin K, Darren M, et al. Polymorphic worm detection using structural information of executables[C]//RAID. 2005

[14] James N, Down S. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[C]//NDSS. 2005

[15] James N, David B, Dawn S. Vulnerability-specific execution fltering for exploit prevention on commodity software[C]//Proceedings of NDSS. 2005

[16] Nickolai Z, Hari K, Michael D, et al. Hardware enforcement of application security policies using tagged memory[C]//OSDI. 2008

[17] Liang Zhen-kai, Sekar R. Automatic generation of buffer overflow attack signatures: An approach based on program behavior models[C]//Proceedings of ACSAC. 2005

[18] David B, James N, Dawn S, et al. Towards automatic generation of vulnerability-based signatures[C]//IEEE S&P. 2006

[19] Susanta N, Tzi-cker C. Execution Trace-Driven Automated Attack Signature Generation[C]//ACSAC. 2008

[20] 陈平，韩浩，沈晓斌，等. 基于动静态程序分析的整形漏洞检测工具[J]. 电子学报，2010，38 (8)：1741-1748

(上接第 112 页)

的地址，测试容错路由算法的运行情况。如图 5 所示，出错节点的数量从 1 增至 300，建立平均路径长度与出错节点数量之间的关系。(12,4) CDCN 网络架构中拥有 5,000 台服务器，若其中 300 台出错，其出错概率已达到 6%，网络传输中依然没有发生任何包丢弃的现象。

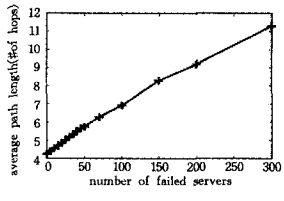

图 5 CDCN 容错路由策略中平均路径长度

参 考 文 献

[1] Amazon elastic compute cloud[EB/OL]. http://aws.amaz-on.com/ec2/

[2] Rabbe L. Powering the Yahoo! network[J]. Nov. 2006

[3] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture[C]//Proceedings of SIGCOMM' 08. 2008

[4] Mysore R N, et al. Portland: A scalable fault-tolerant layer 2 data center network fabric[C]//Proceedings of SIGCOMM' 09. 2009

[5] Cisco data center infrastructure 2.5 design guide[C]//Dec. 2007

[6] Guo C, Wu H, Tan K, et al. Dcell: a scalable and fault-tolerant network structure for data centers[C] // Proceedings of SIGCOMM' 08. 2008

[7] Li D, Guo C, Wu H, et al. FiConn: Using backup port for server interconnection in data centers[C]//Proceedings of INFOCOM' 09. 2009

[8] Guo C, Lu G, Li D, et al. BCube: A high performance, server-centric network architecture for modular data centers[C]//Proceedings of SIGCOMM' 09. 2009

[9] Leighton F T. Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes[M]//M. Kaufmann, 1992

[10] Jeng M, Siegel H. A fault-tolerant multistage interconnection network for multiprocessor systems using dynamic redundancy [C]//Proceedings of ICDCS' 86. 1986