动态信息流分析的漏洞利用检测系统

唐和平 黄曙光 张 亮
(解放军电子工程学院 合肥 230037)

摘 要 安全相关的函数使用了来自网络用户输入或配置文件的非可信数据,由于未经过严格验证,引发了软件安全问题。大量软件漏洞都与非可信数据传播相关。非可信数据传播分析的漏洞利用检测系统将从网络用户输入或配置文件中获得的非可信数据标记为污染数据,使用信息流方法分析污染数据的传播范围,对可能使用污染数据的函数使用多种策略进行污染检查。借助开源的虚拟机代码实现动态信息流跟踪的漏洞检测原型系统,并优化了漏洞利用检测过程。

关键词 动态污染分析,漏洞利用检测,信息流分析,污染场景分析
中图法分类号 TP309 **文献标识码** A

Dynamic Information Flow Analysis for Vulnerability Exploits Detection

TANG He-ping HUANG Shu-guang ZHANG Liang
(School of Electrical Engineering of PLA, Hefei 230037, China)

Abstract Untrusted data originating from network user input and configuration files, causes many software security problems, when operated by security-critical function without strict data validation. Keeping track of the propagation of untrusted data is the main idea of dynamic taint analysis for vulnerability exploits detection. Data derived from network user input and configuration files were labeled as taint. Executed taint propagating algorithm by virtue of data flow analysis, and carried out several taint detection policies for each security-critical function. A vulnerability exploits detection prototype system was implemented on open source emulator and many optimization mechanisms were deployed.

Keywords Dynamic taint analysis, Vulnerability exploits detection, Information flow analysis, Tainted scenes analysis

软件漏洞利用检测经历了人工代码检查、编译器检查、自动/半自动代码检查3个阶段,目前已经提出了多种对抗漏洞利用的方法。这些方法按照采取措施的性质,可以分为被动防御和主动检测两大类。被动防御试图通过修改程序堆结构、栈结构、库函数和内存地址随机化等机制最大限度地阻止漏洞利用,相应地提出了安全堆、栈和库函数调用,例如StackGuard, StackShield, Visual C++. NET /GS, ProPolice, Libsafe, FormatGuard等方法。而主动检测就是在程序运行过程中采取主动措施监视系统的状态,在不安全状况发生前,切断数据来源或者终止程序运行,动态污染信息跟踪就是主动对抗漏洞利用的一种方法。

1 污染信息传播分析研究现状

动态污染信息跟踪应用越来越广泛。在安全领域,污染分析能够阻止多种攻击类型,包括缓冲区溢出[1,2]、格式化字符串攻击[3]、SQL注入攻击[4,5]、跨站脚本攻击[6]等。动态污染信息跟踪主要从以下4个方面开展研究:动态污染传播的通用框架、漏洞利用检测和阻断、数据流分析策略、软件测试和调试等。

由于对污染数据传播分析的一般框架研究较少,还未能提出完善的污染源标记、污染传播、漏洞检测等通用分析方法。Lam和Chiueh[2]提出使用虚拟机技术来实现污染标记和传播的一般方法,该方法不足之处在于需要对代码进行重新编译,这对于第三方软件和函数库是不现实的,其次它不支持多种污染类型标记。

动态污染分析主要用于检测软件漏洞利用类型,覆盖攻击是常见的漏洞利用类型。Newsom和Song[3]提出基于动态污染传播的方法以阻止覆盖攻击,首先标记从网络读取的任何数据为污染数据,在程序执行过程中跟踪污染数据,最后加强数据使用时的安全性检查,防止污染数据被用作跳转地址、函数返回地址、格式化字符串、系统调用参数等。动态污染分析亦用于阻止SQL注入攻击。Nguyen Tuong[6]提出了在PHP构建的Web应用程序中加入污染传播分析的机制。

制定信息流安全策略是保证信息系统机密性的重要手段。RIFLE[9]提供了基于结构支持的信息流安全检测方法,能够同时分析显示和隐式的信息流。

除了软件安全分析,研究者开始将动态污染分析应用到软件测试和软件调试中。COMET[10]系统使用动态污染技术和智能搜索技术来增加C语言程序的测试覆盖率。

到稿日期:2009-08-25 返修日期:2009-11-09
唐和平(1981—),男,博士生,CCF会员,主要研究方向为信息安全、程序静态分析,E-mail:tangheping2007@163.com;黄曙光(1960—),男,教授,博士生导师,主要研究方向为信息安全技术、网络仿真;张 亮(1982—),男,博士生,主要研究方向为信息安全、程序自动验证。

2 信息流跟踪分析基本框架

动态信息流跟踪的漏洞利用检测主要围绕 3 个问题展开：

(1)标记污染数据，确定污染数据源；

(2)污染数据传播过程，即污染的数据是如何通过程序的执行过程传播的；

(3)污染告警机制，制定漏洞利用检查策略。

一般从网络中接收到的各种数据和本地读取的各种文件数据在未加验证情况下均有可能是被污染的，网络、用户输入、配置文件等都是潜在的污染源。在系统函数调用时需要判断函数的操作性质，判断函数是否从污染源读取数据，来自污染源的数据应该打上污染标记。漏洞利用警告是动态污染分析的目标，在污染数据被敏感函数使用前，漏洞利用警告模块将阻断污染数据。

基于信息流跟踪的漏洞检测系统框架如图 1 所示，整个检测系统作为 X86 程序监视器的监控例程。X86 程序监视器是虚拟执行环境，提供了虚拟 CPU 运行程序和一系列分析工具，能够用于程序的内存调试和代码剖析，并允许用户对其功能进行扩展。

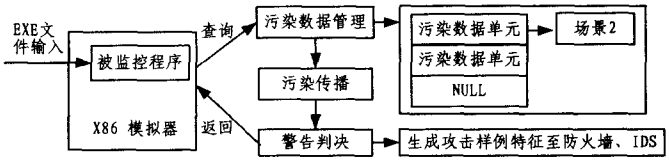

图 1 基于信息流跟踪漏洞检测系统一般框架

被监控程序运行到新的代码块，X86 监视器首先将二进制指令转换为 RISC 指令操作码，然后把操作码输出到污染数据管理模块，并挂起被监视程序。如果某个操作数是污染过的，系统则申请一个新的污染数据单元，存储每一个污染过的全局变量、局部变量、函数指针、跳转标志等，并记录栈的部分内容，这些数据就成为污染数据使用的场景。接着对操作码块进行静态分析，模拟程序在计算机上的执行。污染传播模块根据代码静态分析结果执行污染数据和非污染数据的传递运算，并将结果更新到污染数据单元。对于使用了污染数据的不安全函数，告警模块将发出警告，指明污染数据的来源，显示从污染数据引入到不安全函数操作的中间执行路径。对于其他函数操作，监视器直接返回，等待下一次对代码块的污染分析。

在污染跟踪过程中，污染数据传递过程形成污染数据使用的场景链表，系统将记录每次传递的操作代码、地址、变量内容等信息。污染传播用于分析当前操作与历史污染数据的联系和已知的污染数据对操作结果的影响。在发出漏洞利用警告后，污染传播分析器接着分析污染传播的各个场景，寻找污染源与危险操作之间的执行路径，揭示漏洞利用的全过程。

3 污染传播算法实现

3.1 污染传播分析一般流程

漏洞利用检测系统输入汇编代码块，对代码块进行相应的操作数标记，提取变量符号，剔除程序中的常量符号，识别代码中的库函数调用，生成代码对应的控制流图。

抽象控制流图是在程序控制流图的基础上，省略了常数运算、堆栈操作等与数据流分析无关的操作。结合抽象控制流图和别名分析结果可以得到最终的污染流图。污染流图生成主要是完成函数参数传递、结果返回、中间操作规约、别名约简、无关数据消除、污染数据推测等过程。污染数据流图比控制流图更简洁、直观。图 2 是经过 IDA 反汇编的结果，图 3 是生成的污染流图，图 4 是化简后的污染流图。经过污染传播分析，得到化简后的污染流图与人工分析代码的数据流图是一致的。

```
.text:00411A7E  mov   eax, ds:dword_4250D8
.text:00411A83  mov   [ebp+var_C], eax
.text:00411A86  mov   ecx, ds:dword_4250DC
.text:00411A8C  mov   [ebp+var_8], ecx
.text:00411A8F  lea   eax, [ebp+var_78]
.text:00411A92  push  eax
.text:00411A93  call  sub_411505
.text:00411A98  add   esp, 4
.text:00411A9B  lea   eax, [ebp+var_C]
.text:00411A9E  push  eax
.text:00411A9F  lea   ecx, [ebp+var_78]
.text:00411AA2  push  ecx
.text:00411AA3  call  sub_41120D
.text:00411AA8  add   esp, 8
.text:00411AAB  test  eax, eax
.text:00411AAD  jnz   short loc_411ABE
.text:00411AAF  push  offset aNameMatched ; "name matched!"
.text:00411AB4  call  sub_4114AB
.text:00411AB9  add   esp, 4
```

图 2 反汇编代码

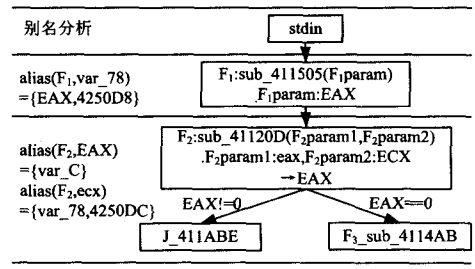

图 3 别名分析与污染传播

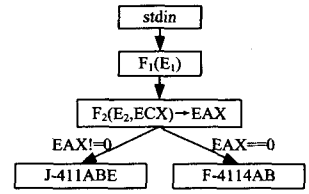

图 4 简化的污染流图

3.2 漏洞利用类型检测策略

计算机软件漏洞类型多种多样，形成的原因和漏洞利用的方式也不一样。漏洞利用检测需要针对不同的漏洞类型分别制定漏洞利用告警规则。表 1 列出了动态污染传播漏洞利用检测的几种常见策略，涵盖了控制流劫持、缓冲区溢出漏洞、目录遍历漏洞、字符串格式化漏洞、跨站脚本漏洞、SQL 注入漏洞等漏洞类型。在多种漏洞检测策略的支持下，使用污染传播分析能够检测大部分漏洞利用类型。

表 1 漏洞利用检测策略

漏洞类型	污染检查策略
控制流劫持	Jmp(addr) | excute(addr)；addr→tainted
缓冲区溢出	Strcpy(,str)；str→tainted
目录遍历	File_function:=execute_file(path,) | Open(path,) | CanonicalizePathName(path,) Path:"\..\path"→tainted
字符串格式化漏洞	printf(fmt,)；fmt::"…(%n)t…"

跨站脚本漏洞	html_print_function(str); str →tainted
SQL 注入	SQL_query_function(query); query:"…(sqlmetachar)…"

缓冲区溢出漏洞是由于字符串拷贝函数使用了污染字符串,用户提供足够大的数据块覆盖本地缓冲区,进而覆盖函数返回地址、异常指针、栈底等。目录遍历漏洞是使用污染数据作为文件操作函数的路径参数,使得".\..\"字符串将会跳出原始的路径范围,导致信息泄漏或者引起缓冲区溢出。字符串格式化漏洞是污染数据作为格式化输出函数的字符格式参数,特定的字符(如"%n")可以改写任意内存,不带格式控制参数将导致信息泄漏。跨站脚本漏洞是输出的动态网页包含污染的数据,导致其他合法用户遭受跨站攻击。SQL 注入漏洞是在提交的 SQL 查询语句中包含了一些特殊用途的 SQL 关键词和 SQL 语句,如"'; execute"等,导致在数据库中执行任意的 SQL 语句。

3.3 动态污染跟踪算法

污染数据链表是存储污染数据的单向链表,每一个污染数据传递到达目的寄存器或者内存地址就建立一条从目的寄存器或内存地址指向污染源的链表。对于单个函数而言,函数的每个输入参数均可能被污染,这样需要对函数每个参数进行污染跟踪,为每个参数建立一条污染链表。链表的每个节点记录了污染数据的存储器名称、内容、污染类型、赋值操作代码地址等。数据存储器包括通用寄存器、标志寄存器、栈和堆内地址。污染类型是指被污染数据的类型。在污染传播时,可能存在一个节点污染多个节点的情况,即一个污染数据传播到多个存储器。

当前污染集合是在程序运行当前时刻被污染的数据集合。如果污染数据被正常数据覆盖,污染数据就被清除,即从当前污染集合中删除该污染数据。

污染数据链表结构:

Taintedlink:={Node | Node:=〈Tainedaddress, value, taintedtype, pparentTaint, operation〉};

当前污染数据集合:

TaintedObject:=

{object|object:=〈Tainedaddress, value, taintedtype〉}

可执行程序的运行过程中,污染的数据都被标记。在进入函数体内时,被污染的数据称为函数的污染环境,包括函数参数 argv、全局变量 global 以及外部输入等。污染跟踪算法首先获得被分析的汇编代码块,由 X86 监视器把汇编代码块输出到污染跟踪分析模块。按照指令的运算方法,区分每一条指令的类型(赋值型、栈操作型或者算术运算型),提取指令相应的操作数,供污染传播算法使用。在指令分析过程中,先判断源操作数是否被污染。如果被污染,则目的操作数即刻被污染,否则就清除目的操作数的污染标记。

污染链表和污染数据集合的构建与代码执行同步进行。污染链表记录了对污染数据操作的场景,包含污染数据操作地址、值、类型、源数据以及污染操作等信息。

在应用程序中,函数需要使用用户提供的数据作为参数,它们若未经过安全检查就被安全敏感函数使用,则可能引发安全问题。污染数据跟踪分析就是检查敏感函数是否使用了受污染的数据,其主要方法就是判定来自污染源的数据是否能够通过一定的传播途径到达安全敏感的函数。

如果污染数据经过若干次的运算或者转换最终被不安全函数使用,那么告警模块将发出警告,指出软件可能存在漏洞。对于危及系统安全的漏洞利用,应及时终止函数执行,向系统发出运行异常。在污染链表中记录着不同污染源数据的传递过程、操作和污染数据内容。污染源追踪就是从污染链表中寻找从污染源到不安全函数参数之间的路径。

在污染告警模块中,并不是所有使用了污染数据的不安全函数必然存在漏洞。如果在污染数据传递的过程中,污染数据经过某些操作使得漏洞利用不会发生,这样漏洞利用就被消除了,污染数据被清洁了。

4 实验结果

4.1 模拟分析

图 2 是用于测试的代码片断,实现将用户输入字符串与常量字符串进行比较的功能。图 3 是污染流图。污染数据流图指出污染数据的来源、流经的函数、控制条件分支,但没有记录污染数据经过的详细操作。污染链表详细记录了污染数据经过的寄存器、内存单元和操作代码。污染数据链表内容如图 5 所示,"→"表示污染数据传递。

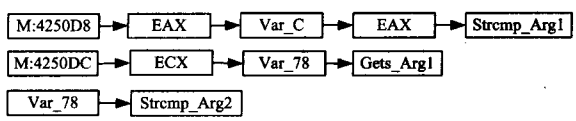

图 5 示例程序的污染链表

在图 5 中,Gets()读取用户从键盘输入的数据,存入到局部变量 Var_78 中,外部用户就可以控制变量 Var_78 的内容,因此 Var_78 变量是污染数据,是非可信的数据源。随后,变量 Var_78 的内容传递到 Strcmp()函数,作为字符串比较函数的一个参数。

4.2 漏洞污染数据跟踪实例分析

MS08-067 是一个远程执行代码漏洞。由于 Windows 的一项服务不能正确地处理特制的 RPC 请求而导致漏洞利用发生,漏洞利用类型属于目录遍历而引发的缓冲区溢出,成功利用此漏洞的攻击者可以远程控制受影响的系统。图 6 是 MS08-067 漏洞的污染传播图。污染数据首先从 RPC 输入开始,RPC 输入是 MS08-067 漏洞发生的污染源,输入数据被拷贝至目录变量 Directory,随后污染数据传递到长度变量 length,变量 length 限制了请求调用命令字符串的大小。因此攻击者可以控制污染数据 E4,构造格式如"\x.\..\yyyyyyyyyyyyy"的污染数据。在函数 CanonicalizePathName()中,对输入字符串进行标准的 Windows 目录处理,由于标准化处理中存在一个逻辑错误,程序不断寻找".\"字符。与此同时,攻击者在栈顶部构造一个残留映象字符".\",最终导致程序用请求的文件名在未检查拷贝字符串大小的情况下覆盖栈顶,同时覆盖了本级函数的返回地址。漏洞利用检测系统检查函数调用时,能够发现 CanonicalizePathName()的参数是污染数据,并且带有危险字符串".\..\",因此发出漏洞利用的警告。基于污染传播分析方法可以记录漏洞利用的全

过程，准确发出漏洞利用的警告。

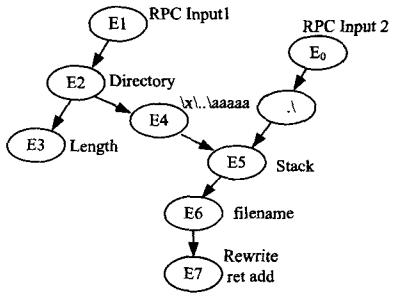

图 6 MS08-067 漏洞污染传播分析

结束语 到目前为止，人们已经提出了许多检测漏洞利用的方法，按照数据截获的位置，这些方法可分成两类：一类是外部阻断型，着重检测从网络接收的各种数据，对数据进行字符、字符串特征过滤，以阻止攻击字符串进入系统；另一类是内部截获型，深入到模块内部甚至内核，监视和检测污染数据，提前发现可能的漏洞利用。基于动态信息跟踪的漏洞利用检测使用污染源标记技术，识别非可信的数据，跟踪并记录污染数据传递的过程，提供污染传播场景信息，对 Open、exec、vfprintf、数据库操作函数、strcmp 等敏感函数参数进行检查，能够检测包括命令注入、格式字符串、缓冲区溢出、SQL 注入、跨站脚本、目录遍历等更加广泛的漏洞利用过程。

参 考 文 献

[1] Kong J, Zou C, Zhou H. Improving Software Security via Runtime Instruction-level Taint Checkingp[C] // Proc. of the 1st Workshop on Architectural and System Support for Improving Software Dependability. Califorria: ACM Press, 2006: 18-24

[2] Lam L, Chiueh T. A General Dynamic Information Flow Tracking Framework for Security Applications[C]// the 22nd Annual Computer Security Applications Conference. Miami Bench, Florida: IEEE Computer Scociety, 2006: 463-472

[3] Newsome J, Song D. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software[C] // Proc. of the Network and Distributed System Security Symposium. Sandiego California, 2005

[4] Alfond W, Orso A, Manolios P. Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks[C]// Proc. of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2006: 175-185

[5] Pietraszek T, Berghe C. Defending Against Injection Attacks Through Context-Sensitive String Evaluation[C]// Proc. of Recent Advances in Intrusion Detection. Seattle, Washington, 2005

[6] Nguyen T A, Guarnieri S, Greene D, et al. Automatically Hardening Web Applications Using Precise Tainting[C]// Proc. of the 20th IFIP International Information Security Conference. Chiba, Japan, 2005

[7] Suh G, Lee J, Zhang D, et al. Secure Program Execution via Dynamic Information Flow Tracking[C]// Proc. of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2004: 85-96

[8] Qin F, Wang C, Li Z, et al. LIFT: A Low-overhead Practical Information Flow Tracking System for Detecting Security Attacks [C]// Proc. Of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. Florida: IEEE Computer Society, 2006: 135-148

[9] Vachharajani N, Bridges M, Chang J, et al. RIFLE: An Architectural Framework for User-Centric Information-Flow Security [C]// Proc. of the 37th Annual IEEE/ACM International Symposium on Microarchitecture. Washington: ACM Press, 2004: 243-254

[10] Leek T, Baker G, Brown R, et al. Coverage Maximization Using Dynamic Taint Tracing[R]. TR-1112. MIT Lincoln Laboratory, 2007

(上接第 136 页)

结束语 为了保证日益复杂的 Web 应用软件的质量和可靠性，需要不断加强对 Web 应用软件的测试研究。实际上，Web 应用软件的有效测试依赖于对其进行充分的分析和理解，提出良好的测试模型，并基于测试模型提出测试策略和测试方法。本文通过对现有测试模型的研究和分析，提出了一种面向对象的 Web 应用软件测试模型 WATM，该模型包括对象模型和导航模型，实现了对 Web 应用软件静态结构和动态行为的有效表示。此外，本文还提出了基于 WATM 来选择和设计导航测试用例和状态测试用例的方法，从而对 Web 应用软件进行导航测试和状态行为测试，更好地保证了 Web 应用软件的质量和可靠性。

参 考 文 献

[1] Leung K, Hui L, Yiu S, et al. Modeling Web Navigation by Statechart[C]// Proc. of Computer Software and Application Conference. 2000

[2] Yang J, Huang J, Wang F, et al. An Objected-Oriented Architecture. Supporting Web Appliction Testing[C]// Proc. of Computer Software and Applications Conference. 1999

[3] Yang Ji-Tzay, Huang Jium-Long, Wang Feng-Jian. A Tool Set to Support Web Application Testing[Z]

[4] Ricca F, Tonella P. Analysis and testing of Web application[J]. IEEE Computer, July 2001

[5] Ricca F, Tonella P. Web site analysis: Structure andevolution[C]// Proceedings of the International Conferenceon Software Maintenance. San Jose, California, USA, 2000: 76-86

[6] Kung D c, Liu C H. An Object-oriented Web test model for testing Web application[J]. IEEE Computer, January 2002

[7] Liu C-H, Kung D C, Hsia P, et al. An object based data flow testing approach for Web applications[J]. International Journal of Software Engineering and Knowledge Engineering, 2001, 11(2): 157-179

[8] Liu Chien-Hung, Kung D c, Hsia P. Structural testing of Web application[J]. IEEE Computer, March 2000