山东农业大学学报(自然科学版),2019,50(5):873-876 VOL.50 NO.5 2019
Journal of Shandong Agricultural University (Natural Science Edition) doi:10.3969/j.issn.1000-2324.2019.05.030

基于动态检测技术的软件设计安全漏洞查找方法

蔡荣文

杭州万向职业技术学院, 浙江 杭州 310023

摘 要: 计算机软件设计需要考虑安全性，这是保证软件系统长周期稳定运行的关键。动态检测技术是当前应用较广的软件安全漏洞检测方式，包括安全共享库、沙箱、内存映射、非执行栈、程序解释、非执行堆等技术，这些技术方式都有不同的特点。通过对比分析每一项技术的优缺点，为软件设计安全漏洞查找提供技术选择的依据。结果表明：动态检测技术可以在不更改程序源代码的情况下，对软件的运行环境实现全面的分析，提升了程序的保密性。

关键词: 动态检测; 安全漏洞; 软件设计; 源代码

中图法分类号: TP3 **文献标识码**: A **文章编号**: 1000-2324(2019)05-0873-04

A Method Searching for the Security Flaw in a Software Design Based on The Dynamic Detection Technology

CAI Rong-wen

Hangzhou Wanxiang Ploytechnic, Hangzhou 310023, *China*

Abstract: Computer software design needs to consider security, which is the key to ensure the long-term stable operation of the software system. Dynamic detection technology is a widely used method of software security vulnerability detection, including security shared library, sandbox, memory mapping, non-execution stack, program interpretation, non-execution stack and so on. These technologies have different characteristics. By comparing and analyzing the advantages and disadvantages of each technology, it provides a basis for technology selection for software design security vulnerability search. The results show that the dynamic detection technology can achieve a comprehensive analysis of the running environment of the software without changing the source code, and improve the confidentiality of the program.

Keywords: Dynamic detection; security vulnerabilities; software design; source code

软件是计算机与网络的重要组成部分，存在于软件中的安全漏洞，极有可能导致计算机与网络被非法入侵。软件的安全漏洞通常是在设计的时候就留下的，包括软件的具体运行及系统安全策略等方面的设计缺陷，因此对软件设计的安全漏洞及时找出其安全隐患并加以弥补修正，以降低软件正式投入运行后给计算机和网络带来的风险是当前计算机安全领域一个值得研究的课题。在软件设计与编写的过程中，植入安全代码是最直接的解决软件安全问题的方式，但这种方式与人为因素的关系较大，不能保证每一位程序编写者都能认真对待，所以检测软件安全漏洞是非常关键的一步[1]。涉及到检测技术的方式方法，当前使用较多的主要有三种，一是静态检测技术，二是动态检测技术，三是混合检测技术。其中动态检测技术是使用最广泛的软件安全漏洞检测方式，原因是动态检测技术可以最大程度地提升软件程序的保密性，而且检测的范围更全面，也非常地便于管理。

1 软件漏洞及动态检测技术

1.1 软件漏洞的含义

软件漏洞主要指隐含在软件中的弱点与缺点，这些问题极有可能造成软件运行时对危险因素的敏感度提高，严重时还有可能被非法分子所利用，成为攻击系统的一个突破口。出现漏洞的主要原因，是因为在软件的设计与开发时，程序编写人员出现操作上的失误而留下的，该漏洞通常分为安全性漏洞和功能性漏洞两种，这两者都关系到软件的安全运行。功能性漏洞将会影响到运行平台的性能，例如系统运行该软件时出现错误结果，或者系统的运行流程出现差错等；安全性漏洞不仅会影响到软件本身的正常运行，同时也会影响到计算机系统的运行，而且会被黑客等不法分子所利用，

收稿日期: 2018-10-05 **修回日期**: 2018-12-04

基金项目: 浙江省教育厅一般科研项目:基于“摩课书院”的课堂教学管理系统设计与开发(Y201738481)

作者简介: 蔡荣文(1974-),男,本科,副教授,主要研究方向为计算机软件开发与应用. E-mail:wxxycrw@126.com

数字优先出版:2019-10-25 http://www.cnki.net

对软件系统进行攻击，有可能会被植入恶性的执行代码，对软件系统发送错误指令[2]。相对而言，安全性漏洞具备更高的危险性。

1.2　软件漏洞产生的途径

根据漏洞的自身特点，主要产生途径有以下三条：一是程序编写者在设计与编写软件的过程中疏忽大意所造成的，软件的编写极富逻辑性，稍微有点偏差就会产生错误。二是软件本身在运算数据的过程中，也有可能会产生预料之外的错误，影响到程序模块的正常运行。三是系统的运行环境也可能会导致软件出现错误，从而产生安全漏洞，包括不同的系统版本或者不同的系统设置，都有可能会留下不同的安全漏洞。四是软件的各种安全漏洞之间也会相互影响，可能以前的漏洞被纠正与弥补的同时，新的漏洞却会不断出现。总之，解决软件漏洞问题是一项长期的、系统的工作，需要持之以恒，并用正确的方法去查找解决。

1.3　动态检测技术

计算机软件程序在设计与运行过程中，无论是系统控制还是数据传输，都具有一定的共性[3]。软件运行中的某一进程，其内存通常可以映射为部分，包括数据、代码、资源、栈、堆等，当操作系统不同的时候，这些部分所占的比例也会不同。动态检测技术就是根据这些比例的变化，在检测软件漏洞的时候，不用改变程序的源代码，主要是依靠改变软件进程的运行环境来达到检测的目的。通常使用的动态检测技术有：安全共享库、沙箱、内存映射、非执行栈、程序解释、非执行堆等。

2　不同动态检测技术的分析对比

2.1　安全共享库

在计算机网络不断发展的过程中，共享库成为提升软件系统性能的重要手段，但是当前有不少软件所使用的共享库并不安全，导致软件安全漏洞频发。尤为严重的是，在软件程序编写的语言中，C 语言和 C++语言里的一些函数还达不到安全的层次，例如 gets、strcpy 等。如果不能合适地使用这些函数，将难以达到预想的效果。利用安全共享库这一技术，可以有效地避免软件受到攻击。安全共享库的工作原理就是凭借动态链接，在计算机运行期间对有威胁因素的函数进行拦截，还能对运行中的函数进行风险检测。这项技术不仅在 Windows 系统上得到了很好的应用，还可以在 UNIX 系统上流畅使用。安全共享库技术还有一个特殊之处，就是能评估系统的内存情况，这样就能将数据写在评估边界的里面。该技术的应用并不难，开发也较容易，而且在软件中应用该技术不需要对系统设置作任何修改。从理论层面看，安全共享库的拦截率很高，能够拦截所有标准库函数的威胁，但是它也有自身的不足，就是不能对系统的一些变化起到保护作用，如果本地的数据段溢出攻击，安全共享库不能起到阻止作用。值得注意的是，安全共享库只能阻止标准函数的攻击，如果遭受到非标准函数的攻击，将起不到阻止作用。安全共享库不存在兼容问题，在标准库下运行的程序同样在安全共享库技术下也能运行，不会出现任何差错。安全共享库可以拦截标准函数中有威胁的函数，不会影响其它标准函数的正常工作，该技术可以将标准函数中有不安全因素的函数进行处理，为整个程序扫除了一些障碍，程序的运行更顺畅。但是要把所有程序中的函数都进行阻止，将会让程序的性能消耗增大 15%。目前，安全共享库的一些功能已经移到其它的标准库，最常见的就是 glibc。如果计算机配置 glibc 库，隐式安全性将得到提高。

2.2　沙箱

阻止某些程序的攻击还可以通过限制其访问的资源来实现。例如 C 语言中存在一些系统函数，像 execv、system 等，一般的小软件是不会涉及到这些系统函数的，一旦发现这些系统函数存在于某个大型软件中，就需要引起注意，很有可能导致软件受到威胁[4]。如果在计算机被威胁之前采用沙箱技术，不让一些软件访问到这些系统函数调用，针对这些软件的攻击将被阻止。沙箱技术的应用比较简单，不需要调整系统已有的程序，但是需要加一个步骤，就是在需要安全防护的程序上定义

访问策略，一般程序的访问策略定义都比较方便，如果程序较为复杂，则定义策略比较繁琐。沙箱技术阻止攻击的性能主要通过定义策略的功能来实现，策略定义的严格程度和程序的保护程度呈正比，严格程度越高，保护程度就越高。沙箱技术存在一个弱项，就是一些软件受到修改用户身份的威胁之时，就不能起到阻止作用。沙箱技术的依托就是安全策略，一般来说，其不会引发兼容性问题，如果应用的策略是一个严格度非常高的策略，也可能会造成程度无法正常使用，还可能造成系统调用函数时无法进行正常的审查工作。沙箱技术会消耗系统的性能，如果系统函数只是偶尔调用，这个消耗是几乎看不见的，但是系统函数调用次数较多，就能明显看出性能消耗。沙箱技术的应用范围比较窄，仅仅限于系统调用。如果沙箱技术不能清楚地知道某个程序的逻辑情况，将起不到阻止威胁的作用。

2.3　内存映射

内存是一些攻击者攻击的主要目标，他们利用特殊的字符将内存覆盖，这些字符通常以 NULL 结尾。对于该攻击，可以采用内存映射来阻止。因为内存映射代码本身也会出现带有 NULL 结尾的字符，这样就能扰乱攻击者。将这种内存代码随机映射在不同的内存地址之上，可以有效阻止部分凭借内存地址进行的攻击。有些软件上存在的一些溢出漏洞，例如缓存区溢出漏洞，攻击者能利用自身的技术和数据来覆盖内存地址，查找这些地址并不难，只要进行计算就可以得出，这样就能轻易达到攻击的目的[5]。一旦采用了内存映射，随机映射几个地址，会给攻击者增加攻击难度，因为安装了代码页之后，攻击者计算地址将会变得困难，需要花费大量的时间才能计算出。内存映射技术虽然起到了一定的拦截作用，但是应用起来比较麻烦，要对操作系统进行重新设置，使得代码页的映射能够在内存较低的区域进行。利用内存映射阻止攻击有一定的局限性，当攻击者采用的手段不是覆盖内存，而是用新的代码来进行攻击，内存映射就形同于虚设了。此外，内存映射还存在一个缺点，由于受到内存大小的约束，不可能将所有的内存都映射到代码页，所以内存映射只能应用一部分。内存映射的最好功能就是能够拦截依靠猜测地址的攻击，然而在实际中，这种情况较少[6]。内存映射对性能没有什么损耗，可以不用考虑。

2.4　非执行栈

有一些攻击者经常利用栈进行攻击，这是因为操作系统的一些数据是依靠栈来储存的，如果攻击者想办法将自己的代码运行到栈中，并执行这些代码，就能破坏软件和操作系统[7]。有关于栈攻击的方法和策略较为成熟，因此攻击者采用这种方法能够很容易达到自己的目的。想要阻止这种攻击，需要用一种程序来阻止栈攻击代码。如果有了这种阻止程序，即便栈中存在一些威胁代码，但是不能运行，就难以继续攻击。这种非执行栈技术运行起来较为麻烦，因为需要操作系统的配合，只有将操作系统进行一定的修改之后才能应用。此外，该技术使用中还存在一个问题，当系统存在栈溢出漏洞的同时，还有另外的漏洞，非执行栈技术就不能很好地解决问题了，只能将操作系统的内核做一些改变，才能让这种技术起到阻止攻击的作用。非执行栈技术存在着较多的缺点，这跟它的工作原理有关，因为该技术是通过对栈的检测来阻止攻击，如果攻击者巧妙避开检测，就可以摆脱它的干扰。该技术在使用的过程中还会引发不兼容现象，但在性能消耗方面则可以忽略不计，其原理只是多加了几个指令，这些指令对于程序的运行起不到任何的影响。

2.5　程序解释

阻止攻击的一个有效方法是在程序应用的过程中进行全程跟踪，同时检测程序的运行状态，以此来保证程序的安全，这种避免攻击的方法就是程序解释技术。但是该技术在使用中会造成较多的性能消耗，对于这个缺憾，有人试图采用动态优化来进行解决，其中程序检测器是一个不错的选择。程序检测器既可以起到检测程序安全的作用，还可以避免消耗系统的性能。程序解释技术应用比较方便，因为不需要对操作系统进行重新设置，对一些程序代码也没有影响，只有在启动的时候重新做一下链接即可完成。利用程序解释技术对运行中的程序展开实时监测，就能对试图控制程序运行

或者修改和程序相关数据的一些攻击进行有效的拦截，该技术虽然可以拦截大部分的攻击，却存在着技术缺陷，如果攻击者通过改变地址的方式来进行攻击，程序解释技术将达不到拦截的目的[8]。采用程序解释技术不仅能对运行中的程序进行安全监测，还能对动态生成的代码起到保护作用，防止它们受到不良攻击。

2.6　非执行堆

除了以上几种动态检测技术，还有一种就是非执行堆技术，但是该技术还没有发展成熟，应用程度也不高，主要是因为一旦处理不当就可能引发软件的瘫痪，导致系统不能正常运行。堆是一个区域的名称，一个内存区域。简单来说，堆是动态的，如果有一种技术，能够让堆正常运行，使攻击者的代码不能运行，就能达到防护的目的。考虑到该技术存在的缺陷，可以跟其它动态检测技术一起使用，例如结合非执行栈技术，可以在最大程度上让攻击者的代码不能运行，起不到任何的破坏。非执行堆技术应用起来较为方便，只是需要对操作系统进行更改，尽管目前的发展还不成熟，但应用性较好，有一定的推广价值。整体来说，该技术的防御范围非常广，只要是攻击者采用代码进行攻击，都可以起到有效的拦截作用，当攻击者采用函数攻击，就难以发挥作用。该技术对系统的性能损耗极少，只是在使用的过程中有可能产生兼容问题，导致系统不能正常运行。

3　结 论

动态检测技术的发展，让软件设计安全漏洞的查找方法更加的全面，以前的静态检测目标只是对安全漏洞进行跟踪检查，如今还可以实现软件系统运行环境的扫描和防护。软件设计的核心内容是程序编写，因此通过动态检测技术达到程序保护的目的，是软件设计人员需要认真思考的问题。在程序执行的时候，对程序的运行环境进行检测分析，包括栈、堆、内存等，从而查找出软件中存在的安全漏洞，这是动态检测技术的优势，可以在不更改程序源代码的情况下全面分析运行环境，提升了程序的保密性。在动态检测过程中，要考虑到是否影响到软件系统的性能、其覆盖面是否广泛、检测效果是否良好等因素，并通过分析对比，选择适合本软件的动态检测技术。

参考文献

[1] Fouda RM. Security vulnerabilities of cyberphysical unmanned aircraft systems[J]. IEEE Aerospace and Electronic Systems Magazine, 2018,33(9):4-17

[2] Ravishankar M, Rao DV, Kumar CRS. A Game Theoretic Software Test-bed for Cyber Security Analysis of Critical Infrastructure [J]. Defence science journal, 2018,68(1):54-63

[3] Sousa BL, Bigonha MAS, Ferreira KAM. An exploratory study on cooccurrence of design patterns and bad smells using software metrics[J]. Software: Practice and Experience, 2019,49(7):1079-1113

[4] Hayden CM, Smith EK, Hardisty EA, *et al*. Evaluating Dynamic Software Update Safety Using Systematic Testing[J]. IEEE Transactions on Software Engineering, 2012,38(6):1340-1354

[5] Dharmalingam JM, Eswaran M. An Agent Based Intelligent Dynamic Vulnerability Analysis Framework for Critical SQLIA Attacks: Intelligent SQLIA Vulnerability Analyzer Agent[J]. Journal of Intelligent Information Technologies, 2018,14(3):56-82

[6] Cruz DM, Eduardo HM, Navaux POA. Modeling memory access behavior for data mapping[J]. Journal of high performance computing applications, 2017,31(3):212-228

[7] Nashimoto S, Homma N, Hayashi Y, *et al*. Buffer overflow attack with multiple fault injection and a proven countermeasure[J]. Journal of cryptographic engineering, 2017,7(1):35-46

[8] Hosni M, Idri A, Abran A. Evaluating filter fuzzy analogy homogenous ensembles for software development effort estimation[J]. Journal of Software: Evolution and Process, 2018,31(2):1-10