云计算环境下基于随机化的安全防御研究

傅建明[1),2)] 林 艳[3)] 刘秀文[1),2)] 张 旭[1),2)]

[1)](空天信息安全与可信计算教育部重点实验室(武汉大学) 武汉 430072)

[2)](武汉大学国家网络安全学院 武汉 430072)

[3)](新加坡管理大学信息系统学院 新加坡 178902)

摘 要 云计算利用云端的资源向用户提供计算和存储服务,改变了个人和企业对信息资源的处理模式.但是,现有的云计算环境的安全加固方法无法应对源于云计算环境服务的单一性和自身漏洞引发的外部攻击.该文从云服务本身、云服务接口和网络接口三个方面研究了云计算平台存在的安全威胁.为了应对这些安全威胁,剖析和比较了应用于云服务本身、云服务接口和网络接口的随机化方法.接着,提出了多层次、层间随机化感知和分布式协同的随机化部署模型,讨论了模型部署中存在的随机化参数感知问题、不同虚拟机上同层服务的协同问题,并给出了相应的解决思路.最后,讨论随机化方法的安全性度量和局限性,进而指出未来研究工作方向.

关键词 云计算;随机化;云服务;云服务接口;网络接口

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2018.01207

Survey of Randomization Defenses on Cloud Computing

FU Jian-Ming[1),2)] LIN Yan[3)] LIU Xiu-Wen[1),2)] ZHANG Xu[1),2)]

[1)](*Key Laboratory of Aerospace Information Security and Trusted Computing of the Ministry of Education, Wuhan University, Wuhan* 430072)

[2)](*School of Cyber Science and Engineering, Wuhan University, Wuhan* 430072)

[3)](*School of Information Systems, Singapore Management University, Singapore* 178902)

Abstract Cloud computing has changed the processing mode on resources of individuals and industries by providing computing and storage services to users. However, existing defenses on cloud, such as virtual machine monitoring and integrity detection, cannot counter against attacks result from the homogeneity and vulnerability of services effectively. In this paper, we have investigated the threats on cloud computing platform from the perspective of cloud service, service interface and network interface, such as code reuse attack, side channel attack and SQL injection. Code reuse attack chains code snippets (gadgets) located in binaries to bypass Data Execution Prevention (DEP). Side channel attack can infer the internal information of an application, such as the encryption key, by analyzing the interaction between the application and the execution environment. SQL injection means the attacker uses malicious SQL statements to control a web application's database server. In order to counter these threats, various randomization approaches that can be applied to cloud service, service interface and network interface have been studied and compared, including address space layout randomization, instruction-set randomization, data randomization and system service interface randomization. We classify them into two categories according to

收稿日期:2016-06-28;在线出版日期:2017-06-29.本课题得到国家自然科学基金(61373168,U1636107)资助.**傅建明**,男,1969 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为软件安全、网络安全.E-mail:jmfu@whu.edu.cn.**林 艳**,女,1990 年生,博士研究生,主要研究方向为软件安全、系统安全.**刘秀文**,女,1991 年生,博士研究生,主要研究方向为软件安全、系统安全.**张 旭**,男,1990 年生,硕士研究生,主要研究方向为软件安全、系统安全.

whether they need de-randomization. Those that need de-randomization are called synergetic randomization, including instruction-set randomization, data randomization and system service interface randomization, the others are called self-contained randomization. The core idea behind them is to make the attacker cannot easily guess the accurate address of the code or data in memory. Then, a multi-layered randomization model on cloud has been proposed, which can achieve the perception of randomization approaches between different service layers and the synergy between different virtual machines. We also discussed the potential problems in the actual deployment of this model, and proposed feasible ways to solve these problems. In general, services running in the upper layer need to use resources in the lower layers, so there is a need to make the upper layer know the randomization approaches used in the lower layers. In order to make different service layers can perceive what kinds of randomization approaches are used, each service layer should have its own management unit to deliver related randomization arguments, such as the name and type of the service, the randomization approach and so on. On the other hand, the same application can be randomized with different options and deployed to different virtual machines. However, it makes software patch difficult as applications in the cloud are keeping running, we cannot simply re-randomize the patched application and deploy it. Therefore, we propose an online patching approach to solve this problem. Moreover, cloud computing also has the vulnerability of buffer overflow, format string and integer overflow and so on. We propose a distributed fault diagnosis approach to capture the context of faults, such as the value of PC, registers and the call stack frame, which can be used to extract the Shellcode and the conditions that trigger the vulnerability. Finally, the security measurement and limitations of this randomization model have been analyzed, and the future research directions have been pointed out.

Keywords cloud computing; randomization; cloud service; service interface; network interface

1 引 言

云计算利用云端的资源,如大型硬件平台、数据中心、应用服务中心等向互联网用户提供资源租用、应用托管、服务外包等服务.用户无需购买大量的硬件设备,也无需安装和配置软件环境,从而改变个人和企业对信息资源的处理、存储和交换的传统模式.例如,用户数据(博客、相册、网盘、邮件、文档等)的存储以及企业信息系统逐渐从终端向云端进行转移,给用户使用和体验互联网业务带来极大的便利,同时也将以往的终端安全问题迁移到云端.例如云计算的多租户模式会导致信息泄露;恶意用户可以租用云服务,对云服务软件、架构、安全机制进行分析研究以及漏洞挖掘.目前,云计算环境的安全加固和安全监控不足以抵御未知攻击,为此,保证云计算的安全性成为目前研究的热点.

云计算需要确保用户和云服务提供商本身的可信和安全.但从本质上讲,云计算的根本问题是确保云服务提供商的安全.如果云服务提供商无法抵挡恶意用户的DDoS(Distributed Denial of Service)攻击、基于漏洞的攻击以及各种恶意代码攻击[1],那么仅仅确保用户的安全是空谈.现有云计算平台的安全性保护主要包括虚拟机的安全监控和云计算平台的完整性检测.

虚拟机的安全监控是保障虚拟机安全运行的基础,有内部监控和外部监控两种结构[2].在内部监控中[3-4],信息收集组件部署在目标虚拟机中,负责收集和拦截某些事件.当VMM(Virtual Machine Monitor)捕获到特定的事件时,VMM通知安全域中安全组件实施事件的安全审查和入侵检测.在外部监控中[5],信息收集组件部署在VMM中,拦截目标虚拟机中的事件,重构该事件的高级语义并传递给安全域中的安全组件.外部监控的所有组件独立于目标虚拟机,对攻击者不可见.CloudVisor将虚拟机监视器资源管理功能与安全保护功能分离,给出了一种基于嵌套虚拟化的解决方案[6],该方案部署在VMM下层.即使虚拟机监视器和虚拟机都已被

恶意入侵，也能够保证用户数据的隐私和虚拟机的一致性. Laureano 等人利用 VMM 监控到的信息检测入侵，从外部监控系统调用序列，根据调用序列判断进程行为是否异常[7].

云计算平台的完整性检测主要包括文件完整性、内核代码完整性和虚拟机管理代码的完整性检测[2,8-15].

上述安全技术没有考虑对云服务本身进行细粒度地保护，难以防御和检测因云计算服务本身存在的安全漏洞引发的攻击. 例如，攻击者利用云服务本身的栈溢出、堆溢出、格式化字符串漏洞、整数溢出、释放后再引用漏洞、类型混淆、API 误用等[16]劫持软件的执行流程导致云计算环境被劫持或者信息泄露. 例如，2009 年 5 月，VMware 虚拟化软件的 Mac 版本存在一个严重的安全漏洞，攻击者利用该漏洞通过 Windows 虚拟机在 Mac 主机上执行恶意代码[17]. 2012 年，仅 Xen Hypervisor 4.1.4 版本就修复了18个关键漏洞，其中 Hypervisor 的缺陷会导致攻击者获得对虚拟平台的控制. 2012 年，发现了 GAE (Google App Engine) 漏洞[18]；2016 年，挖掘出 QEMU 关键组件的十大漏洞①②. 因此，由漏洞引发的攻击是无法避免的.

其次，云服务实例的单一性使得云计算平台更容易受到攻击. 例如，云计算平台可以同时运行多个 VM(Virtual Machine)，这些 VM 上可能安装相同的软件，当某台 VM 上的软件因为漏洞受到攻击时，其它安装相同版本软件的 VM 也会很容易受到攻击. 利用随机化技术可以增强云计算环境下服务实例的多样性，可以避免因服务版本的同构引发的攻击传播，减小攻击面.

同时，上述安全监控技术仅仅能够在攻击发生时或者发生后进行监控，并不能预防攻击的发生. 云计算平台的完整性检测存在 TOC-TOU(Time Of Check to Time Of Use)攻击③，如加载时代码完整性检测无法预防加载后的攻击. 在云计算环境下，随机化技术使得服务的状态与攻击者事先获取的状态不一致，攻击者事先构造的攻击代码在随机化服务实例中将无法成功执行. 本文从云服务面临的各种安全威胁出发，研究云环境中基础设施即服务、平台即服务、软件即服务可能存在的安全问题，提出了结合现有的随机化方法来防御云服务可能会遭受到的攻击情况，丰富了云计算环境的多样性和异构性.

本文首先从云服务自身、云服务接口和网络接口三个方面概述了云计算存在的安全问题；第 3 节从云服务、云服务接口和网络接口三个方面研究和比较了现有的随机化方法；第 4 节提出了云计算环境下的随机化部署模型；第 5 节分析了随机化方法的安全性和可能会产生的影响；最后总结全文，给出了将来的研究重点.

2　云服务安全概述

本节概述云计算服务栈结构和云计算服务面临的安全威胁.

2.1　云计算服务栈.

目前，云计算提供三种基础的服务：基础设施即服务(Infrastructure as a Service，IaaS)、平台即服务(Platform as a Service，PaaS)以及软件即服务(Software as a Service，SaaS)，如图 1 所示.

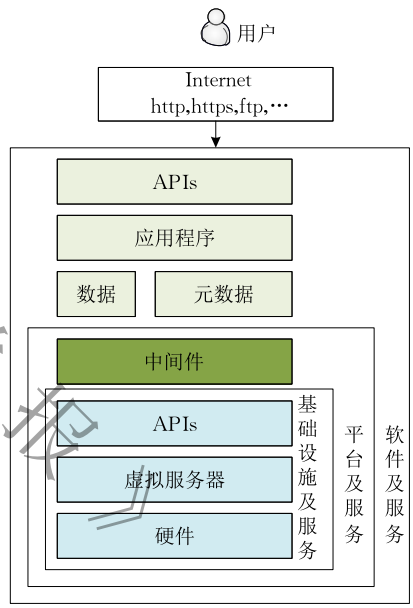


图 1　云计算服务栈

云服务的底层为基础设施即服务 IaaS，提供基础的虚拟硬件资源，即将计算资源和存储资源作为服务出租，如 Amazon 的 EC2(Elastic Compute Cloud). 实际上，IaaS 具有在特定服务质量约束的情况下出租计算机或数据中心的能力，使用户可以选择安装自己的操作系统和应用软件；PaaS 为用户提供服务平台和开发接口，如分布式数据库，Google

① CVE-2016-4439. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4439 2016

② CVE-2016-5106. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5106 2016

③ Time of check, time of use race condition. https://www.owasp.org/index.php/Time_of_check,_time_of_use_race_condition 2009

《计算机学报》

公司的 GAE(Google App Engine)是 PaaS 的代表;云计算服务栈的顶层为软件即服务 SaaS,它是云计算服务层中最为广泛的一层,将应用以基于 Web 的方式提供给用户,例如 Gmail.

通过云计算提供的三层服务栈,用户无需为应用的安装、升级以及服务器、操作系统等的维护操心.同时,云计算的这三层服务具有一定的依赖关系.一个 SaaS 层的服务不仅需要用到 SaaS 层本身的技术,还需要依赖于 PaaS 层提供的开发和部署平台,或者直接部署于 IaaS 层提供的虚拟硬件资源上.同时,PaaS 层的服务也可能构建于 IaaS 层服务之上.因此,服务栈中层与层之间的调用接口需要保持一致,其修改和更新也需要协调和同步.

2.2 安全威胁

云计算环境承载着多种操作系统、中间件和服务软件,一旦这些系统或软件存在漏洞缺陷并被利用,将导致大量服务被破坏或者控制.同时,云计算平台上的服务一般通过 Web(Browser)进行管理① 和对外提供服务,所以 Web 接口中存在的漏洞也会直接影响到云计算平台提供的服务.本节将分别从云服务本身、云服务接口和网络接口三个方面介绍云计算平台可能受到的安全威胁.

2.2.1 云服务自身

云计算平台提供的云服务本身与正常应用程序一样,直接安装在 VM 或者主机上.云服务(应用)本身存在漏洞缺陷将会对使用该云服务的所有用户产生影响.针对云服务本身的攻击威胁主要有代码注入攻击(Shellcode)、代码复用攻击和侧信道攻击三种.Shellcode 和代码复用攻击利用服务本身存在的漏洞进行攻击,侧信道攻击利用云计算平台能被多个用户所共享的特性获取服务中包含的敏感信息.

(1) Shellcode.利用应用程序本身存在的内存错误漏洞,劫持程序的控制流,使其指向攻击者注入的恶意代码②.其攻击目的区别于终端应用.云计算环境下的代码注入攻击主要用于非法获取证书信息和用户数据.同时,其虚拟化特征扩大了代码注入攻击带来的安全威胁,因为多租户模式会引发信息泄漏和窃取服务攻击.

(2)代码复用攻击.操作系统提供的数据执行保护(Data Execution Prevention,DEP)机制[19] 使得内存页无法同时具有既可写又可执行的属性,将外部 Shellcode 直接部署在堆栈上的攻击已经变得十分困难.为此,攻击者将应用程序本身存在的、有用的指令片段进行连接,来绕过 DEP 机制,这类攻击称为代码复用攻击[20-24].代码复用攻击常常通过对控制流完整性进行检测:检测应用程序的执行流程是否为事先得到的控制流图中的一条路径[25-31].但是控制流完整性检测需要对应用程序的每一条间接分支转移进行目的地址比对,在对性能要求较高的云环境下控制流完整性检测并不适用.虽然 Lin 等人[32] 提出利用动态代码优化工具来避免对每一条间接分支转移进行检测,但是对于一些间接分支转移较多的程序,该方法仍然存在较大的性能开销.

(3)侧信道攻击.分析应用与执行环境之间的交互作用,可以推测应用程序的内部状态,如加密密钥、内存地址信息等.常见的用于侧信道攻击的信息有时间消耗、能量消耗和电磁消耗等.在云计算环境中,PaaS 可以为多租户提供开发环境,即多个租户的实例将会运行在同一个操作系统中,更容易被攻击者利用侧信道攻击获取应用的内部信息.例如 Zhang 等人[33] 利用 Flush-Reload 攻击在 PaaS 上获取应用的秘密信息.同时租户可以通过 IaaS 创建自己的 VM,如果攻击者可以创建与受害者共享同一物理服务器的 VM,攻击者同样可以通过侧信道攻击获取受害者 VM 中的秘密信息.

2.2.2 云服务接口

云计算平台的三层服务栈都提供了与用户进行交互的接口,然而这些接口往往是不安全的,不安全接口会影响云服务.针对云服务接口③ 的攻击主要包括 SQL 注入攻击、跨站脚本攻击(Cross-Site Scripting,XSS)、跨站点请求伪造(Cross-Site Request Forgery,CSRF)、XML 签名绕过攻击(Signature Wrapping Attack)和拒绝服务攻击(Denial of Service,DoS).

(1) SQL 注入.由于没有对用户输入进行合法性检查或过滤,攻击者通过构建特殊的输入作为参数传入 Web 应用,执行 SQL 语句,并完成攻击者所要的操作④.

(2) XSS.利用网页开发时留下的漏洞,注入恶意指令到网页,并使用户加载并执行攻击者构造的包含恶意 JavaScript 的网页④.例如 2010 年

① Browser Security Handbook, http://code.google.com/p/browsersec/S 2012

② Common Vulnerabilities and Exposures. http://cve.mitre.org/

③ CERT Vulnerability Note VU#282403. http://www.kb.cert.org/vuls/id/282403 2002

④ Cross-site scripting. http://www.ouah.org/SPIcross-site-scripting.pdf

Amazon.com 上存在的 XSS 攻击使得攻击者可以获得该站点上的证书①. 攻击者利用 ebay 上的 XSS 攻击盗窃用户密码②.

(3) CSRF. 利用浏览器存储的用户认证信息，攻击者诱导受害者访问攻击者精心构造的请求网页时，触发该网页中嵌入的跨域请求③. 一旦这些伪造的请求涉及敏感数据更新操作，会引发用户的财产损失、信誉降低或者蠕虫的泛滥等危害④. 例如 2007 年，谷歌 Gmail 遭受了严重的 CSRF 攻击⑤，攻击者通过恶意网站伪造对受害者 Gmail 中邮件的转发请求，使得成千上万用户的邮件遭到泄露.

(4) XML 签名绕过攻击. 云计算平台为用户提供了多种服务接口，便于用户友好使用服务. 但是这些接口可能会被攻击者利用，使得攻击者可以创建任何云服务. 例如，IaaS 提供 SOAP 接口使用户可以方便地创建、运行或者终止某个 VM. SOAP 基于 XML，在接收到一个 SOAP 请求时，服务器仅仅根据时间戳和 SOAP 的 body 域验证 XML 签名是否正确，并没有对 SOAP 消息的整体结构进行验证，所以可以在一个 SOAP 消息中增加、删除、复制任意的 XML 语句块，从而使得 XML 签名绕过攻击可以成功[34].

(5) DoS. 拒绝服务攻击是一类资源耗尽型攻击，攻击者向目标主机发起大规模处理请求，试图耗尽系统资源、瘫痪正常的软硬件服务. 云计算资源的集中分配方式使得拒绝服务攻击的破坏程度进一步加剧. 云服务接口大都基于 XML、HTTP 和 REST 技术[35]，其拒绝服务攻击主要基于 XML、HTTP 和 REST 技术. 特别是并发量大的高负载请求容易产生 DoS 攻击.

2.2.3　网络接口

云服务接口的底层涉及网络接口的实现，如域名、IP 地址、端口. 攻击者利用端口扫描获得云服务所在的 IP 地址和端口，则可以向提供该服务的服务器发起拒绝服务攻击.

除侧信道攻击外，上述攻击都可能存在于云计算环境 IaaS、PaaS、SaaS. SaaS 因感知的共享信息较少，难以存在侧信道攻击，但该攻击在 IaaS 和 PaaS 较多.

除了以上提到的攻击外，云计算平台还会受到其它的攻击，例如云端数据的安全问题、用户的身份认证问题、云服务接口的对象反序列化问题等. 存储在半可信云端的大量数据一旦泄露，将会给用户带来巨大的损失. 因此需要采用密码学方法增强用户数据的安全性和隐私性，如基于属性加密方案、同态加密方案等[17]，本文不考虑该类安全问题.

3　随机化防御

攻击者在实施攻击时，常常依赖于对应用或者系统的实现具有较为详细的了解，其设计的攻击代码会利用程序在内存中的代码或者数据. 随机化技术可以使这些代码或者数据在内存中的位置难以预测，从而阻止攻击代码的执行. 其次，随机化技术可以增加云服务版本的多样性，不同的随机化参数可以抑制因漏洞引发的攻击传播. 最后，随机化是一种事先防御的方法，可以缓解 TOC-TOU 攻击，可以更好地确保云计算平台的安全性.

随机化方法的基本出发点是对软件或者程序的某些属性进行变换，使得软件的某些属性不可预测. 在云计算环境下，云服务提供商为租户提供各种服务，这些服务的本质仍然是软件，这些软件由服务提供商或服务运营商进行部署，随机化防御技术可以增加云环境下 0-Day 漏洞攻击的难度，同时利用随机化防御技术可以提升云环境下服务的多态和多样性. 根据 2.2 节提到的安全威胁，本节将分别介绍和比较针对云服务本身、云服务接口和网络接口的随机化方法.

3.1　云服务随机化

云服务随机化直接对云环境下的服务自身进行随机化操作，使得攻击者无法轻易获取云服务的敏感属性. 在云计算平台下，云服务本身和普通应用程序一样安装在物理机或者 VM 上，所以对云服务本身进行随机化操作类似于对普通应用程序的操作. 这类随机化方法主要有指令随机化、地址随机化、数据随机化和接口随机化. 根据是否需要进行去随机化操作，本文将其分为两类：自包含随机化和协同式随机化. 自包含随机化不需要在运行时进行去随机化操作，如地址随机化. 协同式随机化需要在运行时

① Godin D. The register: Amazon purges account hijacking threat from site. http://www.theregister.co.uk/2010/04/20/amazon_website_treat/2010

② Swati K. Simple ye effective eBay bug allows hackers to steal passwords. http://thehackernews.com/2016/01/ebay-hacking.html 2016

③ Cross site request forgery: An introduction to a common web application weakness. https://www.icir.org/vern/cs161-sp17/notes/CSRF_Paper.pdf

④ Kenneth Corbin. Facebook hit with new CSRF worm. http://www.internetnews.com/security/article.php/3849616/Facebook+Hit+With+New+CSRF+Worm.html. 2009

⑤ US-CERT. Google gmail cross-site request forgery vulnerability. http://www.kb.cert.org/vuls/id/571584 2007

动态地进行去随机化，如指令随机化、数据随机化和接口随机化.

3.1.1 地址随机化

根据地址随机化作用的对象，可以分为基地址随机化和相对地址随机化两种.

(1)基地址随机化主要对程序的代码段、数据段以及堆栈的基地址在加载时进行随机化，使得程序每次在运行时具有不同的加载基地址，这种随机化方法一般由操作系统实施.但是基地址随机化容易被信息泄露①、部分覆盖、蛮力破解[36]等绕过，需要与其它随机化方法相结合以提高基地址的不可预测性.

(2)相对地址随机化对程序内部的代码块、函数例程以及变量的相对偏移做随机化变换，使得代码块、函数例程以及变量的相对偏移无法预测.这种随机化方法常常部署在软件的生命周期中，例如在程序的编译阶段、加载阶段、运行阶段等，每一个阶段都可以实施相对地址随机化操作，如图 2 所示.

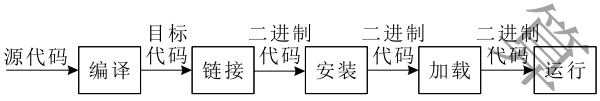


图 2 软件生命周期

根据随机化对象的不同，可以分为指令级、基本块级、函数级以及程序级四类相对地址随机化方法.

(1)指令级随机化对基本块中的指令进行转换操作，是最细粒度的随机化操作，包括以下几类：

①等价指令替换.应用软件在生成后，其内部的某些指令往往可以替换为其它等价指令，使得程序的语义不发生变化，例如将“a=a+b”指令替换为“a=a-(-b)”.指令等价替换在一定程度上会减少攻击者可以利用的信息.

②指令重排序.对于没有依赖关系的指令，其执行顺序对程序的运行并没有影响，并且指令重排序在很大程度上改变了程序的内存布局，可以有效地防御代码复用攻击.

③无关指令填充.在程序中填充一些对程序的执行不会产生影响的指令，可以增加攻击者获取程序内部信息的难度，例如在程序中添加 NOP 指令使得指令间的相对偏移难以预测.

(2)基本块级随机化对一个函数中的基本块进行转换，主要包括：

①基本块重排序.程序的执行流都会体现在基本块的执行上.一个基本块的最后一条指令会给出程序将要执行的下一个基本块的位置，如果能够对基本块的位置进行重排序，则攻击者无法精确地推断出下一个执行的基本块位置.

②虚假基本块插入.如果能够在一个基本块后利用条件跳转插入一个不会执行的基本块，将会增加攻击者获得精确内存布局的难度.

③基本块分解.将一个基本块中关键指令序列进行分解，可以在一定程度上减少攻击者寻找到有用指令序列的机会.

(3)函数级随机化是一种较粗粒度的随机化方法，主要包括：

①栈布局随机化.程序执行时，都会动态地为自己分配一个函数栈帧，攻击者可以利用栈溢出打破栈内对象边界、越界读取敏感信息.其随机化方法主要包括栈逆向增长、栈帧填充和栈变量重排序.

②函数参数随机化.攻击者想要实现攻击目的，往往需要调用共享库中的一些函数.如果事先对函数参数的顺序或者函数参数的个数进行随机化操作，那么攻击者调用的函数将不会正常执行.

(4)程序级随机化是最粗粒度的相对地址随机化操作，主要包括：

①函数重排序.应用程序都由一个一个的函数组成，对这些函数进行重排序操作使得函数间的相对偏移变得不可预测，在一定程度上增加攻击者获取程序内存布局的难度.

②库函数入口地址随机化.攻击代码往往需要调用共享库中的库函数.如果能够使得库函数的入口地址不可预测，则攻击者发起的攻击将难以成功.

③相关数据结构随机化.除了根据应用程序的代码部分泄漏程序的有关信息，攻击者如果能够获得与数据相关的信息，同样可以猜测出程序的内部信息，例如通过某个全局变量的地址信息获得程序的整个内存布局.为此，对于数据相关的信息进行随机化操作在一定程度上也会增加攻击者获得程序信息的难度，包括堆布局随机化、结构体布局随机化和全局变量重排序等.

现有的地址随机化方法如表 1 所示，其中 C 表示编译阶段，I 表示安装阶段，L 表示加载阶段，E 表示运行阶段.同时由于这些方法通过随机化操作使得攻击者原来寻找到的有用信息的位置发生了变化，主要防御针对云服务本身的代码复用攻击和代码注入攻击.

① Fermin J Serna. Cve-2012-0769. The case of the perfect info leak. http://zhodia.hispahak.com/mystuff/seurity/Flash_ASLR_bypass.pdf. 2012

表 1　地址随机化方法

方法	随机化阶段	基地址随机化	指令级			基本块级			函数级		程序级		
			等价指令替换	指令重排序	无效指令填充	基本块重排序	无效基本块插入	基本块分解	栈布局随机化	函数参数随机化	函数重排序	数据结构	库函数入口点
Forrest 等人[39]	C	√		√	√	√			√			√	
Chew 等人[40]	C,L								√				√
PaX①	L	√											
Bhatkar 等人[41]	I	√							√			√	
Kil 等人[42]	I	√									√	√	
Bhatkar 等人[43]	C	√							√		√	√	
Jacob 等人[44]	I		√										
Williams 等人[45]	L									√			
Jackson 等人[46-47]	C				√								
Wei 等人[48]	E		√										
Pappas 等人[49]	I		√	√									
Hiser 等人[50]	I			√									
Giuffrida 等人[51]	C	√			√		√				√	√	
Wartell 等人[52]	L					√							
陈平等人[53]	C											√	
Collberg 等人[54]	C									√		√	
Homescu 等人[55]	C					√							
Gupta 等人[56]	I										√		
Davi 等人[57]	L				√	√							
Homescu 等人[58]	C				√								
辛知等人[59]	C											√	
Chen 等人[60]	I								√				
Junod 等人[61]	C				√			√					
Liang 等人[62]	I								√				
Kim 等人[63]	I			√									
Chen 等人[64]	C											√	
Crane 等人[65]	C,L										√		
Braden 等人[66]	C										√		
Fu 等人[67]	I										√		
Xu 等人[68]	C						√						
Chen 等人[69]	C,E					√							
Koo 等人[70]	I			√									
Williams-King 等人[71]	E										√		

除了对主进程的地址进行随机化操作外，也可以对子进程进行随机化操作以抵抗克隆攻击[37]. 利用动态二进制插装工具 Pin[38] 对调用 fork()函数产生的子进程的地址空间进行随机化操作，使得攻击者无法利用子进程中的有用信息发起攻击.

3.1.2　协同式随机化

针对服务本身的协同式随机化方法需要在运行时对服务进行去随机化操作，主要有指令随机化、数据随机化和接口随机化三类.

(1) 指令随机化的基本思想是为每一个运行的进程创建特有的指令集，使得攻击者注入的代码与现在的指令集不兼容而无法正常运行. 指令随机化常常在应用程序加载前或加载时对指令进行加密操作，在执行时对指令进行解密操作. 当一个功能是由多个服务共同完成时，需要事先知道服务所使用的随机化方法，在执行该服务时按照正确的去随机化方式执行正常的指令.

(2) 数据随机化的基本原理是将内存中的指针或非指针数据作为随机化对象，通过随机化变换(加密操作)，使得数据在内存中以密文存储，使用时解密数据.

(3) 接口随机化将与底层进行交互的接口进行随机化，现有的接口随机化通常将系统调用接口的相关信息作为随机化对象，一般包括系统调用号、系统调用参数. 经过随机化变换，使得系统内部的系统调用动态、随机.

这些随机化方法在经过随机化之后需要将其所使用的随机化参数告知底层，使得底层可以正常地进行去随机化操作，否则服务将无法正常运行，所以它们属于协同式随机化方法. 同时由于这些随机化方法大都只在程序加载前或加载时进行，所以它们又称为静态协同式随机化. 现有的静态协同式随机化方法如表 2 所示，可知，静态协同式随机化方法主要抵抗代码注入攻击.

① Team P. PaX address space layout randomization (ASLR). https://pax.grsecurity.net/docs/aslr.txt 2003

《计算机学报》

表 2 静态协同式随机化方法

随机化方法	方法概述	随机化手段	可抵抗的攻击
Cowan 等人[72]	在编译阶段生成的中间代码中插入随机化功能代码对函数指针、数据指针等读写操作进行异或处理	数据随机化	非控制数据攻击
Kc 等人[73]	在程序装载前利用 32-bit 密钥对其代码段的每 32-bit 代码块进行异或操作	指令随机化	代码注入攻击
Oyama 等人[74]	对 Linux 下标准库函数中存在系统调用的指令序列添加进行随机化调用号和参数的 XOR 指令	接口随机化	通过 glibc 库调用系统调用的代码注入攻击
Rajagopalan 等人[75]	将原始的系统调用增加额外的参数并且替换掉原来的系统调用,利用这些额外的参数对该系统调用进行认证	接口随机化	代码注入攻击
Jiang 等人[76]	对系统调用号和系统调用位置进行 DES 加密处理	接口随机化	直接调用系统调用的代码注入攻击
Nguyen 等人[77]	对 Native API 的分配号进行重排序	接口随机化	直接调用 Native API 的代码注入攻击
Cadar 等人[78]	在编译阶段,对指针数据和非指针数据在写入内存操作时利用 32-bit 密钥进行异或操作	数据随机化	非控制数据攻击
Bhatkar 等人[79]	在源码级对所有类型的数据进行异或操作	数据随机化	非控制数据攻击
Liang 等人[80]	在编译阶段对 Linux 下系统调用的调用号进行随机化	接口随机化	直接调用系统调用的代码注入攻击
Portokalidis 等人[81]	在程序加载之前,对其代码段利用 16-bit 密钥进行异或操作	指令随机化	代码注入攻击
Papadogiannakis 等人[82]	对硬件进行修改,在程序加载前和加载时对代码段进行异或或者换位操作	指令随机化	代码注入攻击
Fu 等人[83]	在程序加载时,利用二进制插装工具进行等长指令替换	指令随机化	代码注入攻击

如 2.2.1 节所述,在云计算环境中多个租户的实例将会运行在同一个操作系统中.如果此时攻击者与受害者位于同一操作系统,则可以通过侧信道攻击获取受害者程序中的秘密信息.对应用软件进行一次静态随机化操作并不能有效地防御侧信道攻击[84-85].为此,提出了基于动态随机化方式的侧信道攻击防御方法.

Crane 等人[86]提出利用动态控制流随机化来防御基于 Cache 的侧信道攻击.对于需要保护的程序抽取那些可能含有敏感信息的函数或者基本块,并对这些函数或者基本块进行克隆操作,以得到多份拷贝.然后对每一份拷贝进行随机化操作,具体的随机化操作由侧信道攻击的类型决定.程序在运行时动态地选择利用哪一份拷贝执行.

Bigelow 等人[87]提出每隔一定时间间隔对程序的内存布局进行随机化,其采用的时间间隔为输出型的系统调用之后,输入型系统调用之前,这样可以有效地防止攻击者根据输出信息获得相关有用信息.

Hataba 等人[88]提出动态消除程序中的条件分支转移来抵抗侧信道攻击.利用 LLVM(Low Level Virtual Machine)[89]对每个函数中的 if 语句进行分析,然后标记对其进行的随机化操作.在程序动态运行时,利用 LLVM 提供的及时编译器对每个程序块进行编译,使得不同随机化参数的程序块被执行,同时如果发现该代码块是已经被编译后的代码块,将会对其分配一个不同的对条件分支进行处理的标记.由于条件分支的消除使得程序的执行时间发生了变化,该方法可以抵抗基于时间的侧信道攻击.

3.2 云服务接口随机化

云计算平台提供的服务需要通过 Web 接口与用户进行交互,所以确保云服务接口自身的安全至关重要.如 2.2 节所述,与用户端密切相关的云服务接口可能会受到的攻击包括 XSS 攻击、SQL 注入攻击、CSRF 攻击、拒绝服务攻击等.云服务接口随机化方法主要对脚本或网页进行随机化操作,包括 SQL 语句、(X)HTML 等,使得它们的关键字或者标签发生变化,从而使攻击者注入的脚本或代码无法正常运行,降低攻击者对云服务进行攻击的概率.例如通过对(X)HTML 进行随机化使得 XSS 攻击无法成功,从而抵抗 XML 签名绕过攻击.

Boyd 等人[90]对 SQL 语句的标准关键字进行随机化(替换)来抵抗 SQL 注入攻击,通过在数据库服务端添加一个代理来进行去随机化操作.

van Gundy 等人[91]提出利用随机化方法来抵抗 XSS 攻击,在服务器端将(X)HTML 内容划分为不同的信任级别,然后对不同信任级别内容中的标签和属性添加一个随机标识符.将该随机化后的(X)HTML 文件和服务器端指定的 policy 发送给客户端,客户端通过判断该(X)HTML 文件是否满足 policy 来判断该(X)HTML 文件是否可信.由于攻击者无法获得该随机标识符,所以如果是存在 XSS 攻击的(X)HTML 文件,则无法通过验证.

CSRF Guard① 在 HTTP 请求中以参数的形式加入一个随机产生的 token，并在服务器端建立一个拦截器来验证这个 token. 如果请求中没有 token 或者 token 内容不正确，则认为可能是 CSRF 攻击而拒绝该请求.

3.3　网络接口随机化

如 2.2.3 节所述，对网络接口的攻击会直接影响云计算平台能否正常地为用户提供服务，所以同样需要对网络接口进行随机化操作以防御相应的攻击. 这类随机方法主要包括 IP 地址随机化、端口号随机化、MAC 地址随机化和域名随机化. IP 地址随机化将主机的 IP 地址在每隔一定时间内随机置换为该局域网内没有被其它主机所使用的并且可用的 IP 地址. MAC 地址随机化将物理子网中的 MAC 地址进行随机转换. 域名随机化指在进行反向 DNS 查询时，返回一个虚假的域名.

网络接口随机化同样为协同式随机化方法，因为他们在进行随机化操作之后往往需要告知网关其使用的随机化参数，以便进行去随机化操作，使得其可以正常地和其它主机进行通信. 表 3 列出了现有的网络地址随机化方法，可以发现现有的网络地址随机化方法主要针对 IP 地址和端口号进行. 由于网络地址随机化使得目标具有不同的 IP 地址，所以一般的端口扫描和拒绝服务攻击将无法成功.

表 3　网络地址随机化方法

随机化方法	随机化对象
Kewley 等人[93]	TCP/IP 报文头中的目的端口号和目的主机地址
Atighetch 等人[94]	IP 地址和端口号
Antonatos 等人[95]	IP 地址
Jafarian 等人[96]	IP 地址
Jafarian 等人[97]	IP 地址、MAC 地址和域名
Wang 等人[98]	IP 地址和域名

除了以上提到的随机化方法，国内邬江兴院士提出了拟态安全的概念，利用不确定性来对抗未知威胁[92]. 其基本思想为：在功能等价条件下，以提供目标环境的动态性、非确定性、异构性和非持续性为目标，动态地构建网络、平台、环境、软件、数据等多样化的拟态环境，从而增大未知漏洞、后门的攻击难度和代价.

4　云环境下随机化部署

将常见的随机化技术如地址随机化、指令随机化、数据随机化等部署在云环境中可以突破云计算环境的单一性. 同时，不同的随机化方法和策略可以抵抗不同的攻击，组合随机化方法和策略可以更有效地抵抗多种攻击. 本节针对云计算服务栈，介绍云环境下的随机化部署方案，并分析了这些方案的难点，以及对应的解决思路.

4.1　云环境下随机化部署的可行性

如图 3 所示，现有的云架构主要包含管理单元、软件部署单元、Hypervisor、网络单元、服务器和存储单元.

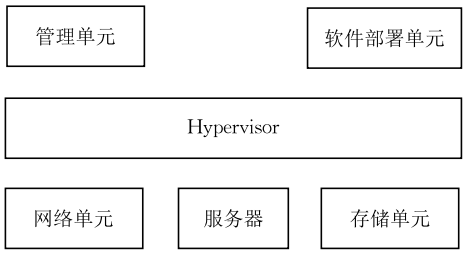


图 3　云架构

图 3 的 Hypervisor 即虚拟机管理器 VMM，它允许多个租户共享物理资源；软件管理单元负责维护和配置基础设施；软件部署单元负责将软件部署和集成在云计算平台上；网络单元允许租户通过网络与云服务相连接；服务器主要负责共享资源的计算和分配；存储单元保存了资源的多个副本，当某地的资源发生错误时，可以向其它地方获取资源. 由图 3 可知，各种随机化方法可以很容易的部署在云计算平台下，例如针对云服务本身的随机化方法可以在软件部署单元中通过随机化引擎进行部署，管理单元负责云服务接口的随机化，网络单元负责网络接口的随机化.

4.2　多层次随机化部署

如图 4(a)所示，用户与云服务之间以及云服务和云服务之间都通过 API 进行交互，攻击者可以位

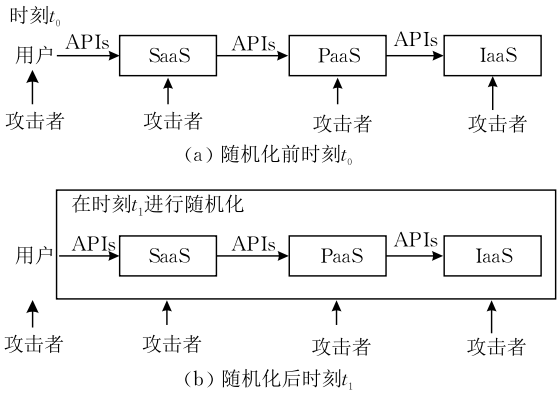


图 4　随机化在云环境下的作用

① OWASP CSRFGuard Project. https://www.owasp.org/index.php/CSRF_Guard

于交互过程中的任意位置.在时刻 t_0 由于没有进行随机化操作,攻击者可以获得与用户完全相同的云服务信息,使得攻击较容易就获得成功.在时刻 t_1,对云服务和调用接口进行了随机化操作,如图 4(b)所示,但此时攻击者只具有 t_0 时刻关于云服务的信息,所以攻击者在 t_1 时刻发起的攻击将无法成功.

同时,随机化方法部署在云服务本身,用户对此时的随机化方法是透明的,随机化操作不影响用户的任何访问.但是当对云服务接口进行随机化操作后,用户利用该接口访问某个云服务时,需要事先获得随机化 token,然后将该 token 插入访问接口中,由于攻击者无法获得关于该 token 的信息,则无法进行正常的去随机化操作.即使攻击者获得了该 token,随机化方法同样部署在 SaaS、PaaS 和 IaaS 中,攻击者如果无法得知这些云服务所使用的随机化密钥,攻击仍然无法获得成功.

第 3 节所述的随机化方法可以部署在不同的云服务上,不同的云服务可以部署的随机化方法如表 4 所示.其中平台即服务相对特殊,由于其主要提供与 Web 相关的服务,例如数据库、Apache 服务器等,所以与 Web 框架相关的随机化方法部署在该层.基地址随机化主要部署在基础设施即服务,使得软件的代码段、数据段等每次都被加载至不同的基地址.由于用户与各服务层进行交互都需要通过不同的 APIs,所以三层都可以部署接口随机化方法.不同的服务都可以有自己的网络地址,所以网络接口随机化可以部署在三层服务上.其它随机化方法既可以部署在软件即服务层,也可以部署在基础设施即服务层.

表 4 多层次随机化部署

随机化方法分类	随机化方法	云服务		
		软件即服务	平台即服务	基础设施即服务
自包含随机化	基地址随机化			√
	相对地址随机化	√		√
协同式随机化	指令随机化	√		√
	数据随机化	√		√
	接口随机化	√	√	√
	网络接口随机化	√	√	√
	云服务接口随机化		√	

值得注意的是,多个 VM 之间的内存共享在云计算环境中十分流行,因为内存共享会减少物理内存的消耗,允许创建更多的 VM[99].但是现有的地址随机化方法在应用每次运行时都会改变内存页布局,所以将地址随机化技术直接部署在云计算环境的 VM 中,将会带来内存共享问题.在部署时需要仔细考虑地址随机化会影响哪些内存共享区域.

4.3 层间随机化感知

在云计算环境下,底层服务可以承载不同的上层服务,上层服务可能跨层调用底层服务,如果上层应用不知道下层的随机化参数,则调用会失败.同时,云服务商可能会形成强强联合、协作共生的生态关系.如图 5 所示,不同云服务商提供的服务可以通过相应的管理网络进行相互访问.所以在进行随机化操作时,需要考虑层间随机化的感知问题,使得随机化后的服务可以正常运行.为此需要使用随机化

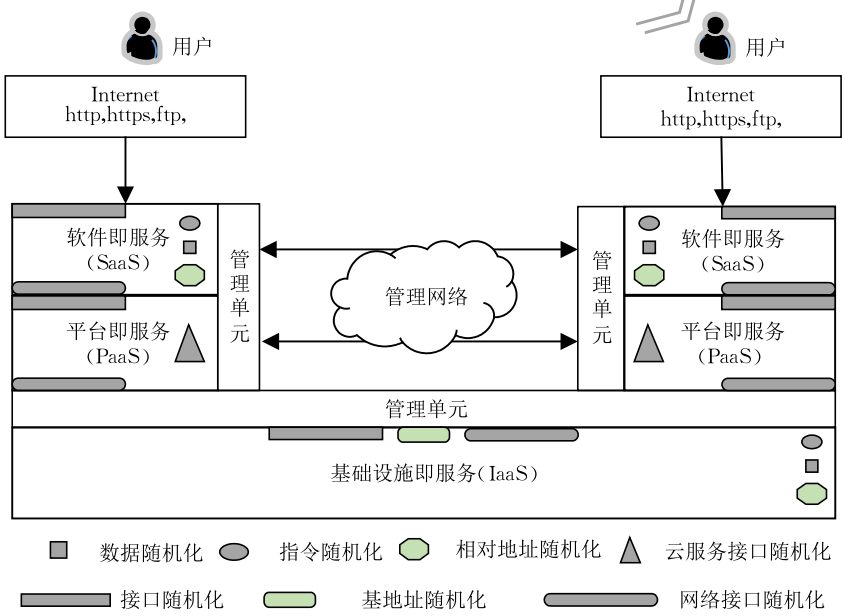


图 5 云环境下随机化部署模型

代理插件完成随机化参数的传递，如图 4(a)所示，各个服务层次之间通过 APIs 进行交互，攻击者可以位于该调用链的任意位置，在进行层间随机化参数传递时需要通过特殊的协议使得攻击者无法获得服务所使用的随机化密钥.

如图 5 所示，当在云计算服务栈中部署随机化方法的同时，云环境下的每一个服务提供层都需要有自己的管理单元通过特殊的协议负责随机化参数的传递. 管理单元负责传递的信息包括服务的名称、类型、参数、随机化方法名等.

4.4　分布式随机化协同

在云计算环境下，可以使不同虚拟机上运行不同随机化参数的同一软件，从而阻止攻击代码在该服务软件中传播. 同时，在随机化环境下运行着同一软件的不同变体，可以通过某一变体受到的攻击情况进行分布式故障诊断.

4.4.1　分布式协同免疫

如图 6 所示，应用软件在进行部署时，首先根据不同的随机化参数，通过随机化引擎进行不同的随机化处理得到不同的随机化版本，然后这些随机化版本将会部署在不同的虚拟机上. 但这种随机化部署方式给软件补丁的修补带来了新的挑战. 如果某一软件存在安全漏洞，需要对其进行及时修补，但是云计算环境的软件系统一般是在线运行的，即无法直接对修补后的应用程序进行再随机化操作，然后进行部署. 为此，需要在线补丁修补的方法.

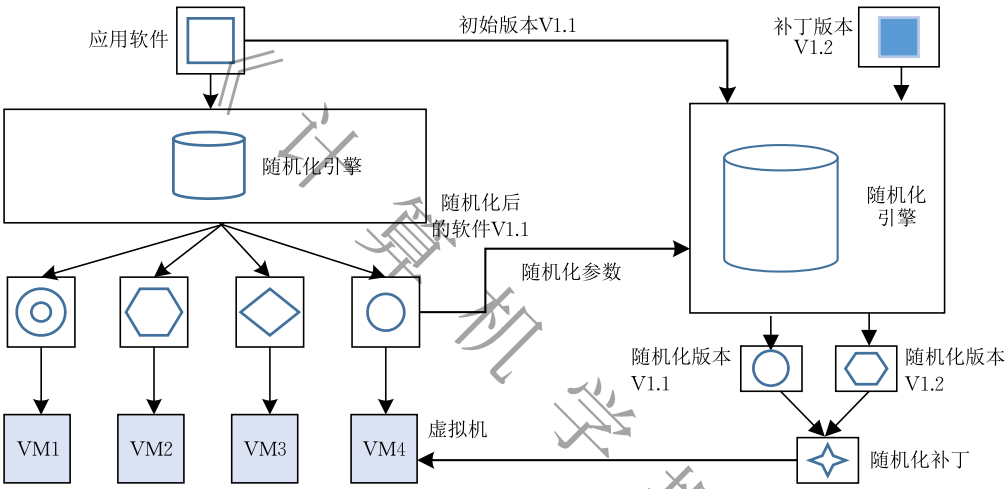


图 6　分布式随机化部署

如图 6 所示，运行在 VM4 上的随机化版本(V1.1)软件需要进行更新，为了不影响软件的在线运行，首先虚拟机 VM4 将自己的随机化参数(包括软件名、软件版本、随机化方法等)传递给随机化引擎，随机化引擎将原始版本和补丁版本利用相同的随机化参数进行随机化操作，然后对两个随机化后的版本进行比较获得现在的随机化补丁，将其发送给 VM4 以进行在线补丁修补操作.

4.4.2　分布式故障诊断

由于自身复杂性以及需求、设计、编码、测试的不完备，云计算环境的软件系统同样存在各种漏洞，如栈溢出、堆溢出、格式化字符串漏洞、NULL 指针漏洞、整数溢出、释放后引用漏洞、类型混淆等. 随机化技术和方法部署到云计算环境可以有效抑制这些漏洞攻击. 但其抑制的结果可能会导致程序崩溃. 另外，云计算环境的软件执行也会产生某些意外的、偶然的崩溃. 为此，需要研究面向故障的运行监控、分布式故障诊断，为攻击溯源和攻击免疫提供服务，达到一体化的防御效果.

图 7 为面向攻击的分布式故障诊断功能结构，从运行监控和异常捕获到故障诊断法，实现攻击的协同免疫. 其中监控模块主要负责部署监控点和捕获异常事件；诊断模块主要负责对故障进行诊断、定位故障源.

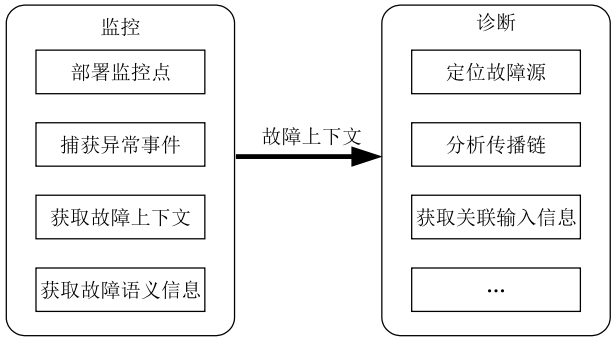


图 7　分布式故障诊断

计算机学报

可以利用陷入机制或者挂钩技术设计异常捕获插件.该插件可以位于某一个服务层(SaaS、PaaS 或者 IaaS).例如,在 Windows 操作系统中可以拦截用户态的 *KiUserExceptionDispatcher* 函数,或者内核态的 *KiDispatchException* 函数实现异常捕获.在 Linux 操作系统以及 VMM 中同样可以通过拦截异常处理函数来实现异常捕获.

当异常发生后,可以捕捉到故障的上下文信息,包括 PC 值、所有寄存器的值、当前的调用栈信息等,通过这些信息为提取 Shellcode 和漏洞触发条件提供帮助.

如图 8 所示为内存错误的漏洞攻击过程,主要包含四个阶段:关键数据的覆盖、关键数据的传播、EIP 跳转和 Shellcode 的执行.在该过程中,由于部署了随机化机制,每一阶段都可能会引发服务崩溃.第一类崩溃(Type1)发生在恶意数据的首次写入,利用故障诊断技术可以很容易就得到恶意数据片段;第二类崩溃(Type2)发生在恶意数据的传播中,该过程要通过内存访问读取恶意数据,所以会很容易得到恶意数据地址;第三类崩溃(Type3)发生在 EIP 跳转时,通常通过覆盖的恶意数据中预测的地址跳转到 Shellcode 所在的地址,所以可以获得 Shellcode 地址;第四类崩溃(Type4)发生在 Shellcode 的执行中,通过故障诊断可以获得恶意代码和 Shellcode 数据.

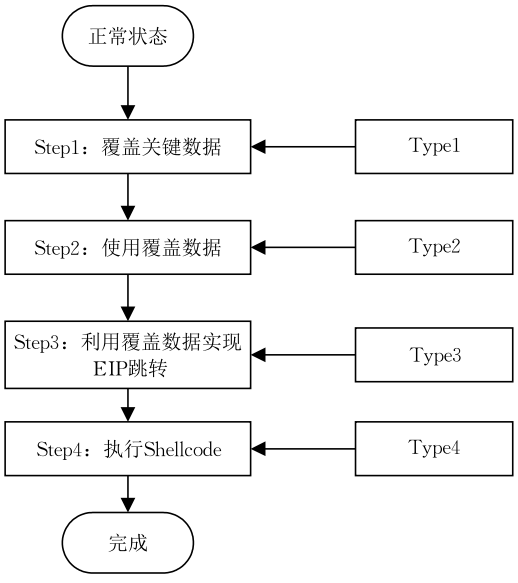


图 8 内存错误漏洞攻击过程

5 随机化方法分析

随机化方法可以有效地抵抗大部分攻击,本节分析随机化方法的安全性和随机化方法产生的影响.

5.1 安全性

随机化方法需要确保被保护对象的安全防御的全面性、及时性和不可预测性,其安全性与随机化时刻、随机化密钥、随机化对象、安全漏洞等因素密切相关.

首先,随机化方法的安全性与随机化和去随机化的时刻密切相关.假设服务的随机化时刻为 t_1,去随机化的时刻为 t_2,服务正常执行的时间为 t_3.对于地址随机化,有 $t_2=t_3$,即不需要对服务进行去随机化操作.对于协同式随机化方法,$t_2<t_3$,此时随机化方法的安全性与 t_2 和 t_3 之间的差值密切相关.如果 t_2 和 t_3 之间的差值较大,则攻击者可以利用该时间差值获得去随机化的信息.所以当服务执行时刻和去随机化时刻之间的差值越大说明随机化方法的安全性越低.

同时,随机化防御提供的是一种概率保护机制,将静态、已知的数据变为动态未知的数据,使得传统的攻击方法受阻.在随机化防御保护下,攻击者所需的关键信息均为动态的,不可预测的,因此攻击者一般需要通过暴力猜测攻击,猜测随机化密钥或者随机化后的关键信息.可以利用信息熵来衡量随机化方法的不确定性,熵值越大,则随机化方法越具有较高的不确定性,即安全性越高.

随机化技术的熵值还与随机化对象的粒度、漏洞出现概率、随机化技术的状态等有关.基地址随机化本质是一种移位变换,由于其仅仅对基地址进行随机化操作,所以其熵值往往较小,容易遭受穷举攻击、邻接信息攻击和基地址窃听攻击等.相比,相对地址随机化具有更好的随机化粒度,所以其熵值较大.要想绕过这类随机化方法,攻击者必须动态地获得有用信息,精心构造攻击代码.虽然地址随机化的安全性相对较低,但是其在运行时几乎不需要进行额外的操作,运行时性能开销较小,所以该随机化方法得到广泛应用.

数据随机化、指令随机化、接口随机化、网络接口随机化和云服务接口随机化本质是一种置换变换或者加密操作,其熵值相对较大.攻击者往往需要对所使用的密钥进行猜测,其安全性相对较高.但是这些随机化方法都需要进行额外的去随机化操作,会产生额外的运行时性能开销,在部署时需要平衡安全性和性能开销.

为了更好地对本文提出的随机化模型的安全性进行分析，本文假设接口随机化、数据随机化和指令随机化、云服务接口随机化和网络接口随机化使用不同的随机化密钥，且密钥分别为 n_0, n_1, n_2, n_3, n_4. 且基地址随机化进行随机化的地址位数为 n_5. M 为不同随机化版本的同一软件部署在 VM 的个数. 攻击者想要成功地同时对 M 个 VM 发起攻击则需要进行 M 次不同的猜测.

攻击者对一台 VM 发起攻击需要进行猜测的密钥数为 $n=n_0+n_1+n_2+n_3+n_4+n_5$. 攻击者对一台 VM 进行 t 次尝试后成功获得接口随机化密钥的概率可以表示为

$$\frac{2^{n_0}-1}{2^{n_0}}\times\frac{2^{n_0}-2}{2^{n_0}-1}\times\cdots\times\frac{2^{n_0}-t-1}{2^{n_0}-t}\times\frac{1}{2^{n_0}-t-1}=\frac{1}{2^{n_0}}$$

期望值为 $\sum_{1}^{2^{n_0}} t\times\frac{1}{2^{n_0}}=\frac{2^{n_0}+1}{2}\approx 2^{2^{n_0}-1}$.

则攻击者想要绕过所有的随机化方法需要进行猜测的总次数为

$$N=2^{n_0-1}\times 2^{n_1-1}\times 2^{n_2-1}\times 2^{n_3-1}\times 2^{n_4-1}\times 2^{n_5-1}\approx 2^n.$$

攻击者想要对 M 台 VM 同时发起攻击，则需要进行的猜测次数为 $N^M=2^{Mn}$.

例如对于安装 32 位操作系统的 VM，接口随机化，数据随机化和指令随机化所使用的随机化密钥为 32 位，同时对基地址的后 20 位进行随机化操作. 云服务接口和网络接口随机化所使用的随机化密钥也为 32 位，则总共的随机化密钥为 $n=180$，假设在该云环境下安装的 VM 数为 20. 则猜测次数为 $2^{20\times 180}$. 表明本文提出的随机化模型在很大程度上增加了攻击者发起攻击的难度.

5.2　随机化影响

随机化方法在增强安全性的同时，也会带来兼容性、维护性等问题. 协同式随机化方法需要云服务内部之间、云服务与用户之间的协同，必要时需要协同协议和协同插件，会影响云服务部署的兼容性. 另外，随机化云环境在服务升级、打补丁时需要额外的操作，增加维护成本.

随机化方法的弱点是容易被信息泄露攻击绕过. 例如，采用地址随机化的程序每次运行时映射的内存地址是随机的. 如果攻击者利用侧信道攻击或者其他手段获得某个全局变量的地址，然后通过固定的偏移计算得到漏洞程序运行时的内存布局，并精心构造攻击代码，则可以实施有效的攻击. 其应对的办法是针对云服务的所有组件都部署安全防御机制，减少攻击面，降低信息泄漏的风险.

6　总　结

随机化思想可以增强软件和系统的多样性、不确定性. 文献[100]介绍了自动化软件多样性方法，讨论了其对二进制代码重写、错误报告、补丁的影响. 文献[101]从攻击流程出发，介绍了移动目标技术对网络、平台、运行环境、软件、数据的作用以及存在的问题. 本文针对云计算服务栈，从云服务本身、云服务接口、网络接口三个方面剖析了云计算环境面临的安全威胁和对抗这些威胁的随机化方法；提出了多层次、层间随机化感知和分布式随机化协同的随机化部署模型，讨论了该模型的安全性取决于随机化方法、随机化时刻、随机化对象、随机化熵值等因素. 最后，指出云计算环境基于随机化的安全防御机制会带来的兼容性和维护性问题，以及应对办法.

除地址随机化得到广泛部署外，其他随机化方法还在研究和实践，将来的工作研究工作有：

(1) 针对具体的云计算平台和框架，部署和测试这些随机化方法，寻求安全性和性能开销兼顾的折中方案.

(2) 在部署静态协同随机化方法的同时，挖掘云服务中核心对象，并针对这些核心对象引入动态协同随机化方法，从而尽可能地减少去随机化时刻与代码(脚本)执行时刻的时间差.

(3) 针对云计算平台，设计轻量级的随机化协同协议，便于在服务与服务之间、服务与用户之间的随机化参数的协同感知.

(4) 采用随机化安全防御机制，设计高效的分布式 0-Day 安全漏洞检测、入侵响应系统，满足 0-Day 漏洞的自动检测、补丁的自动生成、漏洞攻击的自动免疫的目标.

(5) 探索云计算环境独有的随机化对象和随机化方法，如 JSON 模板的随机化、网络服务(协议)的随机化.

参　考　文　献

[1] Fu Jian-Ming, Liu Xiu-Wen, Tang Yi, et al. Survey of memory address leakage and its defense. Journal of Computer Research and Development, 2016, 53(8): 1829-1849(in Chinese)
(傅建明，刘秀文，汤毅等. 内存地址泄漏分析与防御. 计算机研究与发展，2016，53(8)：1829-1849)

《计算机学报》

[2] Xiang Guo-Fu, Jin Hai, Zou De-Qing, et al. Virtualization-based security monitoring. Journal of Software, 2012, 23(8): 2173-2187(in Chinese)
(项国富，金海，邹德清等. 基于虚拟化的安全监控. 软件学报，2012，23(8)：2173-2187)

[3] Payne B D, Carbone M, Sharif M, et al. Lares: An architecture for secure active monitoring using virtualization//Proceedings of the IEEE Symposium on Security and Privacy. Oakland, USA, 2008: 233-247

[4] Sharif M I, Lee W, Cui W, et al. Secure in-VM monitoring using hardware virtualization//Proceedings of the 16th ACM Conference on Computer and Communications Security. Chicago, USA, 2009: 477-487

[5] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2003: 191-206

[6] Zhang F, Chen J, Chen H, et al. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization//Proceedings of the 23rd ACM Symposium on Operating Systems Principles. Cascais, Portugal, 2011: 203-216

[7] Laureano M, Maziero C, Jamhour E. Protecting host-based intrusion detectors through virtual machines. The Journal of Computer and Telecommunications Networking, 2007, 51(5): 1275-1283

[8] Fu J, Lin Y, Zhang X, et al. Computation integrity measurement based on branch transfer//Proceedings of the 13th International Conference on Trust, Security and Privacy in Computing and Communications. Beijing, China, 2014: 590-597

[9] Sailer R, Zhang X, Jaeger T, et al. Design and implementation of a TCG-based integrity measurement architecture//Proceedings of the USENIX Security Symposium. San Diego, USA, 2004: 223-238

[10] Santos N, Gummadi K P, Rodrigues R. Towards trusted cloud computing//Proceedings of the Workshop on Hot Topics in Cloud Computing. San Diego, USA, 2009: 1-5

[11] Jin H, Xiang G, Zou D, et al. A guest-transparent file integrity monitoring method in virtualization environment. The Journal of Computers and Mathematics with Applications, 2010, 60(2): 256-266

[12] Azab A M, Ning P, Wang Z, et al. HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity//Proceedings of the 17th ACM Conference on Computer and Communications Security. Chicago, USA, 2010: 38-49

[13] Wang Z, Jiang X. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity//Proceedings of the 31st IEEE Symposium on Security and Privacy. Oakland, USA, 2010: 380-395

[14] Xiong X, Tian D, Liu P. Practical protection of kernel integrity for commodity OS from untrusted extensions//Proceedings of the 18th Annual Network and Distributed System Security. San Diego, USA, 2011: 114-130

[15] Wang J, Stavrou A, Ghosh A. HyperCheck: A hardware-assisted integrity monitor//Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection. Ottawa, Canada, 2010: 158-177

[16] van der Veen V, Cavallaro L, Bos H. Memory errors: The past, the present, and the future//Proceedings of the International Workshop on Recent Advances in Intrusion Detection. Amsterdam, Netherlands, 2012: 86-106

[17] Feng Deng-Guo, Zhang Min, Zhang Yan, et al. Study on cloud computing security. Journal of Software, 2011, 22(1): 71-83(in Chinese)
(冯登国，张敏，张妍等. 云计算安全研究. 软件学报，2011，22(1)：71-83)

[18] Google app engine Java security sandbox bypasses. Poland: Security Explorations, Technical Report: SE-2014-02, 2012

[19] Andersen S, Abella V. Changes to functionality in Microsoft Windows XP service pack 2, part 3: Memory protection technologies, Data Execution Prevention, Microsoft TechNet Library, 2004

[20] Schuster F, Tendyck T, Liebchen C, et al. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications//Proceedings of the IEEE Symposium on Security and Privacy. San Jose, USA, 2015: 745-762

[21] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)//Proceedings of the 14th ACM Conference on Computer and Communications Security. Alexandria, USA, 2007: 552-561

[22] Bletsch T, Jiang X, Freeh V W, et al. Jump-oriented programming: A new class of code-reuse attack//Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. Chicago, USA, 2011: 30-40

[23] Snow K Z, Monrose F, Davi L, et al. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization//Proceedings of the IEEE Symposium on Security and Privacy. San Francisco, USA, 2013: 574-588

[24] van der Veen V, Göktaş E, Contag M, et al. A tough call: Mitigating advanced code-reuse attacks at the binary level//Proceedings of the IEEE Symposium on Security and Privacy. San Jose, USA, 2016: 934-953

[25] Abadi M, Budiu M, Erlingsson U, et al. Control-flow integrity//Proceedings of the 12th ACM Conference on Computer and Communications Security. Alexandria, USA, 2005: 340-353

[26] Zhang M, Sekar R. Control flow integrity for COTS binaries//Proceedings of the 22nd USENIX Security Symposium. Washington, USA, 2013: 337-352

[27] Zhang C, Wei T, Chen Z, et al. Practical control flow integrity and randomization for binary executables//Proceedings of the IEEE Symposium on Security and Privacy. San Francisco, USA, 2013: 559-573

《计算机学报》

[28] Bletsch T, Jiang X, Freeh V. Mitigating code-reuse attacks with control-flow locking//Proceedings of the 27th Annual Computer Security Applications Conference. Orlando, USA, 2011: 353-362

[29] Mashtizadeh A J, Bittau A, Mazieres D, et al. Cryptographically enforced control flow integrity//Proceedings of the 22nd ACM Conference on Computer and Communications Security. Denver, USA, 2015: 941-951

[30] Yuan P, Zeng Q, Ding X. Hardware-assisted fine-grained code-reuse attack detection//Proceedings of the Research in Attacks, Intrusions, and Defenses. Kyoto, Japan, 2015: 66-85

[31] van der Veen V, Andriesse D, Göktaş E, et al. Practical context-sensitive CFI//Proceedings of the 22nd ACM Conference on Computer and Communications Security. Denver, USA, 2015: 927-940

[32] Lin Y, Tang X, Gao D, et al. Control flow integrity enforcement with dynamic code optimization//Proceedings of the 19th Information Security Conference. Hawaii, USA, 2016: 366-385

[33] Zhang Y, Juels A, Reiter M K, et al. Cross-tenant side-channel attacks in PaaS clouds//Proceedings of the ACM Conference on Computer and Communications Security. Scottsdale, USA, 2014: 990-1003

[34] Somorovsky J, Heiderich M, Jensen M, et al. All your clouds are belong to us: Security analysis of cloud management interfaces//Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop. Chicago, USA, 2011: 3-14

[35] Zhang Yu-Qing, Wang Xiao-Fei, Liu Xue-Feng, et al. Survey on cloud computing security. Journal of Software, 2016, 27(6): 1328-1348(in Chinese)
(张玉清，王晓菲，刘雪峰等. 云计算环境安全综述. 软件学报，2016，27(6)：1328-1348)

[36] Shacham H, Page M, Pfaff B, et al. On the effectiveness of address-space randomization//Proceedings of the 11th ACM Conference on Computer and Communications Security. Washington, USA, 2004: 298-307

[37] Lu K, Nürnberger S, Backes M, et al. How to make ASLR win the clone wars: Runtime re-randomization//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2016: 433-447

[38] Luk C K, Cohn R, Muth R, et al. Pin: building customized program analysis tools with dynamic instrumentation//Proceedings of the Programming Language Design and Implementation. Chicago, USA, 2005: 190-200

[39] Forrest S, Somayaji A, Ackley D H. Building diverse computer systems//Proceedings of the 6th Workshop on Hot Topics in Operating Systems. Los Alamitos, USA, 1997: 67-72

[40] Chew M, Song D. Mitigating buffer overflows by operating system randomization. Carnegie Mellon University, USA: Technique Report: CMU-CS-02-197, 2002

[41] Bhatkar S, DuVarney D C, Sekar R. Address obfuscation: An efficient approach to combat a broad range of memory error exploits//Proceedings of the 12th USENIX Security Symposium. Washington, USA, 2003: 105-120

[42] Kil C, Jun J, Bookholt C, et al. Address space layout permutation (ASLP): Towards fine-grained randomization of commodity software//Proceedings of the 22nd Annual Computer Security Applications Conference. Miami Beach, Florida, 2006: 339-348

[43] Bhatkar S, DuVarney D C, Sekar R. Efficient techniques for comprehensive protection from memory error exploits//Proceedings of the 14th Usenix Security Symposium. Baltimore, USA, 2005: 271-286

[44] Jacob M, Jakubowski M H, Naldurg P, et al. The superdiversifier: Peephole individualization for software protection//Proceedings of the Advances in Information and Computer Security. Kagawa, Japan, 2008: 100-120

[45] Williams D, Hu W, Davidson J W, et al. Security through diversity: Leveraging virtual machine technology//Proceedings of the IEEE Symposium on Security and Privacy. Oakland, USA, 2009, 7(1): 26-33

[46] Jackson T, Salamat B, Homescu A, et al. Compiler-Generated Software Diversity. Moving Target Defense. New York: Springer, 2011: 77-98

[47] Jackson T, Homescu A, Crane S, et al. Diversifying the Software Stack Using Randomized NOP Insertion. Moving Target Defense II. New York: Springer, 2013: 151-173

[48] Wei T, Wang T, Duan L, et al. INSeRT: Protect dynamic code generation against spraying//Proceedings of the International Conference on Information Science and Technology. Nanjing, China, 2011: 323-328

[49] Pappas V, Polychronakis M, Keromytis A D. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization//Proceedings of the IEEE Symposium on Security and Privacy. San Francisco, USA, 2012: 601-615

[50] Hiser J, Nguyen-Tuong A, Co M, et al. ILR: Where'd my gadgets go?//Proceedings of the IEEE Symposium on Security and Privacy. San Francisco, 2012: 571-585

[51] Giuffrida C, Kuijsten A, Tanenbaum A S. Enhanced operating system security through efficient and fine-grained address space randomization//Proceedings of the 21st USENIX Security Symposium. Bellevue, USA, 2012: 475-490

[52] Wartell R, Mohan V, Hamlen K W, et al. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code//Proceedings of the 2012 ACM Conference on Computer and Communications Security. Raleigh, USA, 2012: 157-168

[53] Chen Ping, Xing Xiao, Xin Zhi, et al. Protecting programs based on randomizing the encapsulated structure. Journal of Computer Research and Development, 2012, 48(12): 2227-2234(in Chinese)
(陈平，刑晓，辛知等. 基于封装结构随机化的程序保护方法. 计算机研究与发展，2012，48(12)：2227-2234)

《计算机学报》

[54] Collberg C, Martin S, Myers J, et al. Distributed application tamper detection via continuous software updates//Proceedings of the 28th Annual Computer Security Applications Conference. Orlando, USA, 2012: 319-328

[55] Homescu A, Neisius S, Larsen P, et al. Profile-guided automated software diversity//Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization. Shenzhen, China, 2012: 1-11

[56] Gupta A, Kerr S, Kirkpatrick M S, et al. Marlin: A fine grained randomization approach to defend against ROP attacks//Proceedings of the 7th International Conference on Network and System Security. Madrid, Spain, 2013: 293-306

[57] Davi L V, Dmitrienko A, Nürnberger S, et al. Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and ARM//Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security. Hangzhou, China, 2013: 299-310

[58] Homescu A, Brunthaler S, Larsen P, et al. Librando: Transparent code randomization for just-in-time compilers//Proceedings of the 2013 ACM Conference on Computer and Communications Security. Berlin, Germany, 2013: 993-1004

[59] Xin Zhi, Chen Hui-Yu, Han Hao, et al. Kernel Rootkit defense based on automatic data structure randomization. Chinese Journal of Computers, 2014, 37(5): 1101-1110(in Chinese)
(辛知, 陈惠宇, 韩浩等. 基于结构体随机化的内核 Rootkit 防御技术. 计算机学报, 2014, 37(5): 1101-1110)

[60] Chen X, Slowinska A, Andriesse, D, et al. StackArmor: Comprehensive protection from stack-based memory error vulnerabilities for binaries//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2015: 1-15

[61] Junod P, Rinaldini J, Wehrli J, et al. Obfuscator-LLVM: Software protection for the masses//Proceedings of the 1st International Workshop on Software Protection. Florence, Italy, 2015: 3-9

[62] Liang Y, Ma X, Wu D, et al. Stack layout randomization with minimal rewriting of android binaries//Proceedings of the International Conference on Information Security and Cryptology. Seoul, Korea, 2015: 229-245

[63] Kim S H, Xu L, Liu Z, et al. Enhancing software dependability and security with hardware supported instruction address space randomization//Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Rio, Brazil, 2015: 251-262

[64] Chen P, Xu J, Lin Z, et al. A practical approach for adaptive data structure layout randomization//Proceedings of the European Symposium on Research in Computer Security. Vienna, Austria, 2015: 69-89

[65] Crane S J, Volckaert S, Schuster F, et al. It's a TRaP: Table randomization and protection against function-reuse attacks//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. Denver, USA, 2015: 243-255

[66] Braden K, Crane S, Davi L, et al. Leakage-resilient layout randomization for mobile devices//Proceedings of the 2016 Network and Distributed System Security Symposium. San Diego, USA, 2016: 1-15

[67] Fu J, Lin Y, Zhang X. Code reuse attack mitigation based on function randomization without symbol table//Proceedings of the 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. Tianjin, China, 2016: 394-401

[68] Xu D, Ming J, Wu D. Generalized dynamic opaque predicates: A new control flow obfuscation method//Proceedings of the 19th Information Security Conference. Hawaii, USA, 2016: 323-342

[69] Chen Y, Wang Z, Whalley D, et al. Remix: On-demand live randomization//Proceedings of the 6th ACM Conference on Data and Application Security and Privacy. New Orleans, USA, 2016: 50-61

[70] Koo H, Polychronakis M. Juggling the gadgets: Binary-level code randomization using instruction displacement//Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. Xi'an, China, 2016: 23-34

[71] Williams-King D, Gobieski G, Williams-King K, et al. Shuffler: Fast and deployable continuous code re-randomization//Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. Austin, USA, 2016: 367-382

[72] Cowan C, Beattie S, Johansen J, et al. PointGuard™: Protecting pointers from buffer overflow vulnerabilities//Proceedings of the 12th Conference on USENIX Security Symposium. Bellevue, USA, 2003: 91-104

[73] Kc G S, Keromytis A D, Prevelakis V. Countering code-injection attacks with instruction-set randomization//Proceedings of the 10th ACM Conference on Computer and Communications Security. Washington, USA, 2003: 272-280

[74] Oyama Y, Yonezawa A. Prevention of code-injection attacks by encrypting system call arguments. University of Tokyo, Japan: Technical Report: TR06-01, 2006

[75] Rajagopalan M, Hiltunen M A, Jim T, et al. System call monitoring using authenticated system calls. IEEE Transactions on Dependable and Secure Computing, 2006, 3(3): 216-229

[76] Jiang X, Wang H J, Xu D, et al. RandSys: Thwarting code injection attacks with system service interface randomization//Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems. Budapest, Hungary, 2007: 209-218

[77] Nguyen L Q, Demir T, Rowe J, et al. A framework for diversifying windows native APIs to tolerate code injection attacks//Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security. Singapore, 2007: 392-394

[78] Cadar C, Akritidis P, Costa M, et al. Data randomization. Microsoft Research, USA: Technical Report: MSR-TR-2008-120, 2008

[79] Bhatkar S, Sekar R. Data space randomization//Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment. Paris, France, 2008: 1-22

[80] Liang Z, Liang B, Li L. A system call randomization based method for countering code-injection attacks. International Journal of Information Technology and Computer Science, 2009, 1(1): 1-7

[81] Portokalidis G, Keromytis A D. Fast and practical instruction-set randomization for commodity systems//Proceedings of the 26th Annual Computer Security Applications Conference. Orlando, USA, 2010: 41-48

[82] Papadogiannakis A, Loutsis L, Papaefstathiou V, et al. ASIST: Architectural support for instruction set randomization //Proceedings of the 2013 ACM Conference on Computer and Communications Security. Berlin, Germany, 2013: 981-992

[83] Fu J, Zhang X, Lin Y. An instruction-set randomization using length-preserving permutation//Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communication. Helsinki, Finland, 2015: 376-383

[84] Hund R, Willems C, Holz T. Practical timing side channel attacks against kernel space ASLR//Proceedings of the IEEE Symposium on Security and Privacy. San Francisco, USA, 2013: 191-205

[85] Seibert J, Okhravi H, Söderström E. Information leaks without memory disclosures: Remote side channel attacks on diversified code//Proceedings of the 2014 ACM Conference on Computer and Communications Security. Scottsdale, USA, 2014: 54-65

[86] Crane S, Homescu A, Brunthaler S, et al. Thwarting cache side-channel attacks through dynamic software diversity// Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2015: 1-14

[87] Bigelow D, Hobson T, Rudd R, et al. Timely rerandomization for mitigating memory disclosures//Proceedings of the 22nd ACM Conference on Computer and Communications Security. Denver, USA, 2015: 268-279

[88] Hataba M, Elkhouly R, El-Mahdy A. Diversified remote code execution using dynamic obfuscation of conditional branches//Proceedings of the 35th International Conference on Distributed Computing Systems Workshops. Columbus, USA, 2013: 120-127

[89] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation//Proceedings of the International Symposium on Code Generation and Optimization. Palo Alto, USA, 2004: 75-86

[90] Boyd S W, Keromytis A D. SQLrand: Preventing SQL injection attacks//Proceedings of the Applied Cryptography and Network Security. Yellow Mountain, China, 2004: 292-302

[91] van Gundy M, Chen H. Noncespaces: Using randomization to defeat cross-site scripting attacks. Computers & Security, 2012, 31(4): 612-628

[92] Wu Jiang-Xing, Zhang Fan, Luo Xing-Guo. Mimic computing and mimic security. Communications of China Computer Federation, 2015, 11(1): 8-14(in Chinese)
(邬江兴，张帆，罗兴国. 拟态计算与拟态安全防御. 计算机学会通讯，2015，11(1)：8-14)

[93] Kewley D, Fink R, Lowry J, et al. Dynamic approaches to thwart adversary intelligence gathering//Proceedings of the DARPA Information Survivability Conference & Exposition. Hilton Head, USA, 2001: 176-185

[94] Atighetchi M, Pal P, Webber F, et al. Adaptive use of network-centric mechanisms in cyber-defense//Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. Hakodate, Japan, 2003: 183-192

[95] Antonatos S, Akritidis P, Markatos E P, et al. Defending against hitlist worms using network address space randomization. Computer Networks, 2007, 51(12): 3471-3490

[96] Jafarian J H, Al-Shaer E, Duan Q. Openflow random host mutation: Transparent moving target defense using software defined networking//Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks. Helsinki, Finland, 2012: 127-132

[97] Jafarian J H, Al-Shaer E, Duan Q. An effective address mutation approach for disrupting reconnaissance attacks. IEEE Transactions on Information Forensics and Security, 2015, 10(12): 2562-2577

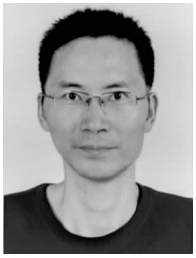
[98] Wang K, Chen X, Zhu Y. Random domain name and address mutation (RDAM) for thwarting reconnaissance attacks. PLoS One, 2017, 12(5): e0177111

[99] Suzaki K, Iijima K, Yagi T. Effects of memory randomization, sanitization and page cache on memory deduplication// Proceedings of the European Workshop on System Security. Bern, Switzerland, 2012: 1-6

[100] Larsen P, Homescu A, Brunthaler S, et al. SoK: Automated software diversity//Proceedings of the IEEE Symposium on Security and Privacy. San Jose, USA, 2014: 276-291

[101] Okhravi H, Thomas H, David B, et al. Finding focus in the blur of moving target techniques//Proceedings of the IEEE Symposium on Security and Privacy. San Jose, USA, 2014, 12(2): 16-26

计算机学报



FU Jian-Ming, born in 1969, Ph.D., professor. His main research interests include software security and network security.

LIN Yan, born in 1990, Ph. D. Her main research interests include software security and system security.

LIU Xiu-Wen, born in 1991, Ph. D. Her main research interests include software security and system security.

ZHANG Xu, born in 1990, M. S. His main research interests include software security and system security.

Background

Cloud computing has changed the processing mode on resources of individuals and industries by providing computing and storage services to users. Despite many security defenses on cloud are proposed in recent years, these approaches cannot change the homogeneity of services and interfaces on cloud, attackers can still construct attacks by exploiting the vulnerabilities of services running on cloud. To make the sensitive information of programs cannot be predicted easily, randomization is proposed. Although different kinds of randomization approaches increase the difficulty of predicting sensitive data, they are not combined with cloud and the difficulty of deploying them on cloud is not discussed.

Our paper surveys the state-art of randomization approaches. The novelty and contributions of our research are as follows. (1) Our paper focuses on the analysis and comparison of attacks and randomization approaches from the perspective of cloud services, service interfaces and network interfaces, including address space layout randomization, instruction set randomization, data randomization, interface randomization, dynamic service randomization, and dynamic network randomization. (2) We have proposed a multi-layered randomization model, which discusses the critical problems when deploying it, including the perception of randomization approaches between different layers in cloud and the deployment of patching when different versions of software are installed on different VMs. Meanwhile, we proposed potential ways to solve these problems. (3) We have discussed the security measurement of this randomization model and the impact of that on cloud, and the security of randomization approaches depends on randomization time, randomization key and randomization object. Moreover, randomization approaches are vulnerable to information leakage. And we pointed out our future research direction.

This paper is partly supported by the National Natural Science Foundation of China (No. 61373168, U1636107).

《计算机学报》