一种基于托架的自蜕变主动防御网络框架

吴承荣, 严 明, 金蒿林, 刘 巍, 张世永, 曾剑平

复旦大学计算机科学技术学院 上海 中国 200433

摘要 传统的防御技术存在静态和被动的特性, 即系统以及安全机制在一段时间内相对固定, 攻击方可持续对一个固定的目标进行研究和尝试, 总能找到弱点进行攻击。为了提高安全防御的主动性, 国际上启动了若干力图“改变游戏规则”的研究计划, MTD(moving target defense)以及拟态防御是新兴的主动防御思路的代表。然而, 要达到主动防御的目的, 除了需要在关键环节采用相关的单点技术外, 还需要一种框架, 使各层主动防御机制能够有机地协同, 形成一个总体可变可控的主动防御体系, 并且使得不具备内生性主动防御机制的传统信息系统也能在这种框架上运行, 并得益于主动防御机制所带来的防御能力提升。本论文提出了一种可以有效整合不同层面的主流主动防御技术和机制, 并兼容传统应用的框架: 自蜕变主动防御网络框架, 该框架可实现内生与外加式主动防御技术的有机整合, 多层次、多粒度主动防御技术的整合, 兼容传统的不具备内生性主动防御特性的应用, 并可为将来形成新一代内生性主动防御网络体系架构提供借鉴。

关键词 主动防御; 拟态防御; 移动目标防御; 虚拟化; 虚拟网络
中图法分类号 TN915.08 DOI 号 10.19363/j.cnki.cn10-1380/tn.2016.04.002

A Self-transforming Proactive Defense Network Framework based on “carrier”

WU Chengrong, YAN Ming, Jin Haolin, LIU Wei, ZHANG Shiyong, ZENG Jianping

School of Computer Science, Fudan University, Shanghai 200433, China

Abstract Traditional information security defense techniques have the feature of static and passive. Systems and security mechanisms are relatively fixed in a period of time. So attackers can continuously study the static target, and try to find the vulnerability of it. In order to improve the proactivity of security defense, some research programs that tried to “change the rules of the game” had been initialized throughout the world. MTD (moving target defense) and the Mimic-Defense are the emerging ideas of proactive defense. However, to achieve the purpose of proactive defense, in addition to the separate uses of techniques on key points, we also need a framework. Deferent defense mechanisms can be effectively cooperate in the framework to form a general “moving” and controlled proactive defense system. Traditional information system which does not have the build-in proactive defense mechanism can also run in this framework, and be benefited from the proactive defense mechanism to enhance the ability of defense. This paper presents a kind of framework that can effectively integrate different levels of proactive defense techniques and mechanisms, and is compatible with the traditional applications. We call it self-transforming proactive defense network framework. This framework can archive the integration of build-in with bolt-on proactive defense techniques, the integration of multi-level and multi-granularity proactive defense techniques. It is compatible with traditional applications which do not have the build-in proactive defense mechanism, and provides ideas.to form a new generation of build-in proactive defense network architecture in the future.

Key words proactive defense; mimic defense; moving target defense; virtualization; virtual network

1 背景

大量的研究和实践表明, 传统信息安全理念和技术在面对未来高强度对抗的网络攻击威胁时显得十分被动, 主动权往往在发起攻击的一方。传统的防御技术存在静态和被动的特性, 即系统以及安全机制在一段时间内相对固定, 攻击方可持续对一个固定的目标进行研究和尝试, 总能找到弱点进行攻击。此外, 传统的安全技术普遍采用分层的、松散的控制机制, 各类安全机制分别作用于各个层次, 缺乏有效的协同机制, 给了恶意代码长期生存的空间和网络访问途径。

为了提高安全防御的主动性, 一些具备“动态”特性的技术被引入到安全防御机制的相关环节。例

通讯作者: 吴承荣, 博士, 副教授, cwu@fudan.edu.cn。
收稿日期: 2016-09-16; 修改日期: 2016-10-13; 定稿日期: 2016-10-20

如, NAT 和动态 IP 地址分配机制、动态域名、随机会话密钥、动态验证码等。2009 年左右, 这种主动防御思路得到了拓展, 国际上启动了若干力图“改变游戏规则”的研究计划。美国 NITRD(Networking and Information Technology Research and Development Program)在 2009 年的年度报告中提出的 Moving Target Defense(MTD)[1], 即是通过可变性来提高系统防御能力的新方法。在早期的探索中, BBN 提出的主动网络防御[2]通过动态改变网络地址和端口来增强系统的安全性; 随后的一些研究[3-5]提出了用网络地址空间的随机化方法使木马和蠕虫失效。Al-Shaer[6]等提出了一个 MUTE(Mutable Networks)模型, 支持 IP 地址的随机分配及变化。除了基于网络的 MTD 方法, 国际上一些学者同时提出了基于系统的 MTD 方法, 包括内存地址空间的随机化方法 ASLR[7], 指令集随机化方法 ISR[8], 数据随机化方法[9]等。这些方法的本质是使系统产生多样性来抵御攻击。近年来围绕这种主动防御理念的相关的理论和技术又得到了进一步的发展, 将形式化描述和博弈论等理论以及虚拟化技术引入 MTD, 对拟采用的 MTD 机制进行分析和决策, 并利用虚拟化技术实现架构层面的 MTD。Bradley Schmerl 等提出一种自适应系统框架 Rainbow[10], 可用于对架构层面的 MTD 战术分类和编目, 根据安全和应用的要求对可采取的 MTD 战术和时机进行形式化推理, 并提出使用随机多方博弈模型来分析验证各类 MTD 场景中的行为。Seungwon Shin 等提出了一个称为 CloudRand 的框架[11], 实现了一个 Hypervisor 层面对于网络服务端口进行实时变迁的 MTD 系统。Scott A. DeLoach, Xinming Ou 等提出了一个 MTD 原型系统, 采用 VGM、OMACS 和 CAG 等运行期模型驱动自适应引擎形成相关决策, 对 IP 地址和端口、防火墙设置、应用种类和版本、虚拟机类型等系统配置进行变迁。Zhuang R 等提出了一个分析模型[13], 以分析在企业网络环境中 MTD 机制的有效性。Wei Peng 等分析了在具备异构和动态攻击表面的云服务中MTD机制的有效性[14], 构想了一个结合了虚拟机迁移、快照以及迁移的多样性和兼容性等云平台特性的云服务安全模型, 考虑攻击者对目标服务攻击表面情报的累计效应对服务攻击表面的异构和动态性建模, 估算服务被攻破的概率。

国内在“拟态计算”和“拟态安全”方面的研究有所进展。拟态安全(mimic security)是以提高运行环境或执行机构的不确定性为目标, 以计算或处理结构功能等价条件的主动跳变或迁移实现拟态环境, 以防御者可控的方式随机的改变系统体系结构, 对攻击者则表现为难以观察和预测的目标变化, 因而能从体系结构技术层面降低由病毒和木马或漏洞及后门引发的安全风险。

自从 MTD 的思想得到广泛关注以来, 诞生了一系列相关技术, 利用“移动”特性在相关环节起到了良好的抵御攻击的作用。但目前诸如 MTD 等主动安全防御技术分别针对若干环节的特定安全威胁(例如代码注入式攻击)进行增强, 却并没有形成一个具备内生的可自主变迁的框架结构。纯粹依靠某些特定环节的“移动”机制抵御攻击的效果有限。David Evans 等经过研究指出 Circumvention Attack、Deputy Attack、Brute Force and Entropy Reduction Attack、Probing Attack 等攻击是可以绕过或削弱 MTD 中若干技术(如 ASLR、ISR 等)的防御效果[15]。其次, 当前许多信息系统或应用软件并未内生地具备或缺省地开启“移动”机制, 而当前我们也不可能对已有系统进行重新开发使其具备“内生”的“移动”特性, 所以大量现有信息系统难以全面应用主动防御技术。

要有效达到主动防御的目的, 除了需要在关键环节应用相关的单点技术外, 更需要一种框架, 使得各个层面的主动防御机制能够有机地协同, 形成一个总体可变可控的主动防御体系, 并且使得传统的不具备内生性主动防御机制的传统信息系统也能在这种框架上运行, 并得益于主动防御机制所带来的防御能力提升。本论文提出了一种可以有效整合不同层面的主流主动防御技术和机制, 并兼容传统应用的框架: 基于托架的自蜕变主动防御网络框架。该框架采用一种称为托架应用承载机制, 以包容和承载不同粒度的应用系统, 采用相对独立的控制平面以及托架间的协同实现主动防御机制, 形成一个整体上具备可变和可控特性的主动防御网络。该框架具备以下优势:

(1) 提供“外加”式主动防御机制: 当前一些主流的操作系统已经具备内生性主动防御机制(如 ALSR 等), 但大量的上层应用并不具备内生的主动防御机制, 这些应用需要依靠所运行的环境所提供的 “外加”式的主动防御机制, 才能使整个系统具备“移动”特性。自蜕变主动防御网络框架将通过托架实现动态化、随机化等主动防御机制, 在对上层应用基本透明的情况下, 使运行于托架上应用系统具备主动防御特性。

(2) 可实现多层次、多粒度主动防御技术的整合。MTD、拟态防御等主动防御机制可以在多个层

次(如网络层、系统层、应用层)、多个粒度(如子网、单机、软件/进程)实施。在当前分层、松耦合的主体架构中, 缺乏关联、各自为战的单点防御技术使黑客拥有更大的空间各个突破。自蜕变主动防御网络框架将支持不同层次、不同粒度的主流主动防御技术的整合, 形成一个防御体系。

(3) 实现对正常应用 “稳定”以及对异常访问“移动”: MTD 类的主动防御技术以主动变迁攻击所依赖的固定因素来提升防御特性, 但是现实应用中某些因素是攻击和正常应用同时依赖的, 正常应用需要这些因素保持稳定, 这将限制类 MTD 机制的应用范围。自蜕变主动防御网络框架试图实现相关因素对正常应用保持稳定, 而对攻击主体呈现不可预测的随机变迁特性。

(4) 实现对主动防御机制自身的协同保护: 主动防御机制的本身也是由软件/硬件实现的, 不可避免存在漏洞。主动防御机制本身也不可能在所有环节均具备“移动”特性, 总有固定不变的地方, 这将成为攻击者可利用的弱点。因此主动防御机制本身需要有其他机制进行保护, 形成各机制相互保护的整体防御系统。自蜕变主动防御网络框架可支持这种相互协防。

本论文提出的自蜕变主动防御网络框架不仅致力于整合当前的主流主动防御机制, 以及兼容传统应用。其框架本身也体现了一种新型的可变可控网络架构, 为将来形成颠覆性的网络架构提供借鉴。

本论文的第 1 章介绍了传统被动防御技术的局限性以及 MTD、拟态防御等主动防御技术, 提出了自蜕变主动防御网络框架试图实现的目标和优势。第 2 章提出一个自蜕变主动防御网络框架, 以及采用托架方法构建可兼容传统应用的运算网元的基本思路。第 3 章介绍了利用当前主流的虚拟化技术实现自蜕变主动防御网络框架的相关方法。第 4 章介绍了一种在基于虚拟化技术的自蜕变主动防御网络框架上实现 URL 随机变迁的方法。第 5 章介绍了针对第 4 章所提出方法的实验及结果。第 6 章提出了今后研究工作的展望。

2 自蜕变主动防御网络框架

自蜕变主动防御网络框架是一种具备可变和可控特性的主动安全防御网络的框架结构。

其中“自蜕变”是指整个主动防御网络在网络层、系统层、应用三个层面可以根据即定的策略实现自主变迁, 以使多个层面的攻击表面发生迁移, 规避攻击。

这种网络框架借鉴了当前 MTD、拟态防御等主动安全防御理念及防护策略, 采用支持多层次动态变迁的网络和系统结构, 可支持不同层面和粒度的主流主动防御技术的部署和实现, 并可包容装入现有传统应用和以后可能有的等价变体应用, 使整个系统体现出主动防御的特性。

2.1 自蜕变主动防御网络的整体架构

自蜕变主动防御网络的核心思想包括两个方面。其一是采用动态可变的设计思路, 使得网络中的重要因素可以根据需要主动调整, 可调整的因素主要包括网络标识、转发路径、封装协议、基础软件组成部件、应用系统执行环境等。这种主动变迁使系统脆弱点在一段时间内无法被发现和利用, 使对手的攻击手段无法在变更的网络中奏效。其二是加强对网络和系统空间各类实体的细粒度访问控制, 使各类实体无法不受限制地访问网络和系统资源, 仅能实施合法的受限行为, 这样即使网络和系统空间被侵入和植入恶意实体, 其行为也将受到严格限制。上述主动变迁和控制的策略均可根据网络和系统需求进行调整, 因此称为“自蜕变主动防御网络”。

自蜕变主动防御网络在逻辑上由网络通道、网元和控制中心组成:

(1) 网络通道主要为自蜕变主动防御网络提供基本的通信支持。网络通道可以是任何采用传统的网络设备、传输介质、通信协议建立。自蜕变主动防御网络在逻辑上将传统网络设备(交换机、路由器、防火墙、加密机、网关等)和介质构成的物理网络看作“通道”, 所有的可变可控机制在传统网络设备和介质之外实现, 所以可兼容当前的主流网络技术。

(2) 网元是组成自蜕变主动防御网络节点的基本单元, 用于承载网络中的各类应用和数据, 进行相应的运算和处理, 并在总体策略的指导下实现对所承载应用的动态变迁和内部细粒度访问控制, 对中转的信息进行访问控制, 并对一个区域内的自蜕变主动防御网络的各类防御机制进行总体控制。一个自蜕变主动防御网络中, 有些网元承载系统中与业务相关的应用或数据, 这种网元类似于当前网络中的主机设备; 有些网元则承载专门部署在网络中实现某些主动防御功能的软件, 这种网元起到中转和变迁通信数据的作用; 还有一些网元中承载的是伪装应用, 起到扰乱攻击者的作用。网元与当前组网技术中的主机和网络设备的区别在于: 网元采用控制平面和数据平面分离的结构, 并具备变迁、控制等主动防御机制。

(3) 总控中心是自蜕变主动防御网络中, 实现总

控功能的特殊网元(控制网元)形成的集合, 负责一个区域内的自蜕变主动防御网络的总体控制。总控中心可以由一个主控制网元、一个或多个从控制网元组成, 主要功能是对整个自蜕变主动防御网络进行统一管理和控制。单个控制网元故障或失控, 将由其他控制网元接管。总控中心的控制范围覆盖网络层、系统层和应用层, 具备与各层安全机制的接口, 实现重要网络因素的主动变迁以及细粒度访问控制。

根据当前的主流网络构建方法, 以及主动防御的需要, 自蜕变主动防御网络主要包含“运算网元”、“存储网元”、“传输网元”、“控制网元”这四类网元。在自蜕变主动防御网络中, 不同类型的网元在自蜕变主动防御网络中的角色各不相同, 但是其基本的自蜕变和访问控制机制是一致的, 可以采用一种通用的软硬件结构作为构建不同网元的基础, 本论文中提出采用托架来构建(详见 2.3)网元。因此网元可以看作由相对通用的托架及其承载的不同类型的“应用”共同组成。基于以上组成部分, 自蜕变主动防御网络的基本结构如图 1 所示:

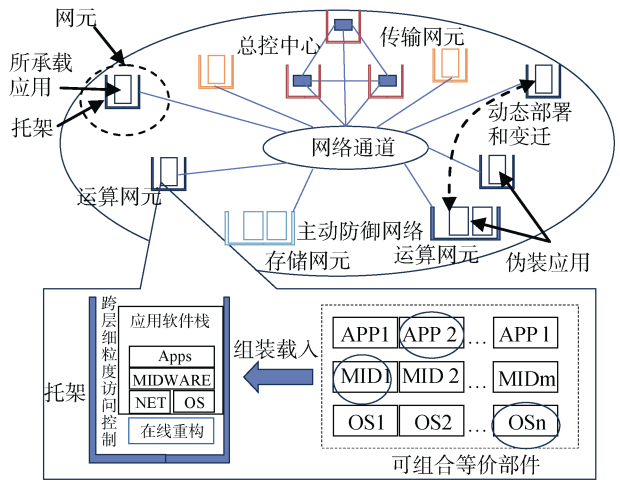


图 1 自蜕变主动防御网络的基本结构

2.2 自蜕变主动防御网络运作机制

2.2.1 基于网元间隧道的层叠逻辑网络

自蜕变主动防御网络是由网元和总控中心共同组成的一个具备自适应主动变迁、跨层细粒度访问控制、态势感知、动态重组等特性的安全网络。自蜕变主动防御网络建立在传统技术构建的网络通道基础上, 但是在逻辑上可形成多个动态可变的、对外不可见的层叠逻辑网络。

网络通道由传统网络设备以及连接这些设备的介质组成。传输网元可安插在关键通道和关键接入区域(如服务器, 终端接入区域)入口, 使得经过传输网元的数据流受到严格控制并对某些因素进行变迁。

在网络通道的基础上, 网元之间采用隧道技术互联, 在隧道技术基础上建立转发路径, 从而构成逻辑拓扑。根据所建立的网络所覆盖的范围, 可以选择适用的隧道协议建立隧道。如果整个网络覆盖范围是一个子网, 可以采用二层隧道协议, 如 L2TP 等; 如果覆盖若干个子网, 则可以采用三层隧道协议, 如 GRE、IPSEC 等。网元在总控中心统一控制下动态变迁逻辑转发路径, 从而逻辑拓扑也动态可变。利用转发控制和隧道机制, 可以在通用的网络通道上建立对外不可见的逻辑层隐性网络, 可以在原来的逻辑拓扑上透明地嵌入可实现特定主动防御功能的传输网元, 例如以拟态防御思路构建动态异构冗余体系的输入分配代理及输出仲裁部件。逻辑网络的转发路径、封装协议、访问控制机制动态可变。在此基础上形成的逻辑拓扑可在总控中心的控制下根据态势发展而变迁, 在网络和信息系统受到攻击、摧毁、侵扰的情况下自动愈合, 保证最重要的功能在系统受损情况下仍然具备一定的可用性; 可根据态势变化和策略的需求进行主动变迁, 使针对传统网络的安全威胁失效。如图 2 所示:

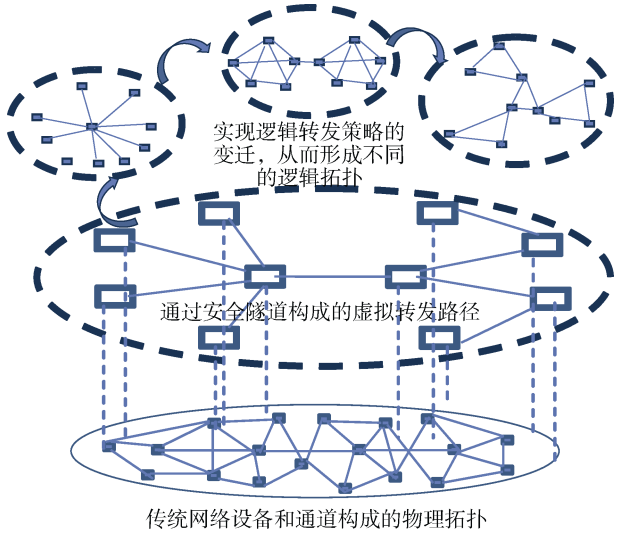


图 2 基于网元间隧道的逻辑转发路径变迁

2.2.2 网元内部的主动变迁机制

自蜕变主动防御网络的主动变迁机制不仅可以改变网络与系统的外部表现, 而且其系统内部构成也可改变。在自蜕变主动防御网络中, 运算网元是承载应用软件的核心单元, 运算网元中所承载的应用软件栈的代码动态可变; 同时, 各个应用中实体在应用层、系统层、网络层所具备的权限受到细粒度访问控制的制约。

承载服务端软件栈的运算网元, 采用自动和人

工参与结合的方式由多层次的等价模块组装形成, 不同模块之间采用标准接口, 可以形成大量等效应用服务软件栈。在 MTD 领域当前的最新研究成果(如 ISR 等)可用于生成等效应用服务软件栈, 在软件栈的操作系统层可以实现诸如 ASLR 的 MTD 技术, 只要能够组装成符合规范的自完整等效软件栈就可以装入自蜕变主动防御网络。

承载客户端应用软件栈的网元, 采用类似的方法预先组装形成等效候选软件栈集合, 通过总控中心和托架的协同通过可对软件栈进行升级和变更, 并采用配置和选通的方式挑选其中的某个等效软件栈激活, 并根据既定策略进行配置变更。

运算网元所承载的应用软件栈中的合规应用软件实体采用严格的适配过程在承载其的网元上进行注册, 这些应用软件实体(如进程)可以通过网元提供的可变机制实时定位到变更后的服务资源, 并利用网元之间的隧道连接相关的服务资源。而未经注册过程的实体(如使用过程中被注入的恶意代码)则无法实时跟踪变更后的服务资源。

2.2.3 跨层细粒度访问控制

自蜕变主动防御网络中的网元对其所承载的应用软件栈中各层实体的行为进行严格控制, 防止实体实施未授权行为, 并对所有行为进行严密监视。

(1) 在网络层: 系统间的传输数据严格控制, 并与系统层、应用层协同。经改造的网络协议中将携带用户和进程信息, 网元对所中转数据的访问控制将不仅局限于 IP 地址和端口, 特定时段对于特定主机仅允许合法用户控制下的合法进程的数据向限定的目标传输。

(2) 系统层: 各个时段允许登录的用户和允许启动的进程(应用软件)、以及允许启用的外设、进程允许访问的主机和网络资源(包括系统调用和启动其他进程)受到严格控制, 无关的进程将无法启动和对外通信。通过可信启动、完整性校验机制等防止系统关键部位被篡改, 通过增强的监控机制防止进程注入等假冒进程标识的攻击。

(3) 应用层: 提供与应用系统的接口, 应用层可获得网络层和系统层信息(如身份、物理上线端口等), 并可通过与总控中心交互对系统层和网络层的安全策略进行细粒度控制; 对于现有的(不具备内生性主动防御机制的)系统, 可以通过系统层和网络层的“包扎式”(wrapper)外部增强机制与总控中心交互, 在不修改应用系统本身的情况下可实现一定程度的安全。

所承载的系统中, 所有实体激活后, 只能连通最为有限的网络资源(如注册和身份验证主机), 在经过识别和验证后, 相应的可连通资源将会发生改变。随着应用进行到相应的阶段, 相关的通道和权限也将进一步发生变化。

2.2.4 与原有技术的兼容机制

传统计算机上的软件系统(包括操作系统、平台软件、应用软件和数据等)可以通过几种方式装载到自蜕变主动防御网络。

一是通过扫描、导出和组装机制从传统计算机上导出相关软件和数据, 构造可备系统装入的应用软件栈, 通过装入机制部署到相应网元上。

二是通过特殊设计的外置模式的托架与传统计算机接驳, 通过外置模式托架对原有计算机进行安全加固、结构改造和控制, 并采用软件 Agent 方式实现安全控制。这样, 采用外置托架+传统计算机的形式构成运算网元, 这种方式具备部分可变可控特性。

三是传统计算机通过网络连接承担接入中转角色的传输网元, 不对计算机本身作任何改造, 通过网络层的访问控制机制对访问行为进行控制, 通过安装安全重定向软件访问动态变迁的网络资源。这种方式具备有限的网络层安全防护能力。

2.3 基于托架构建网元的思路

自蜕变主动防御网络的各类网元可以根据即定的规范采用各种适合的技术实现, 而分层、通用、兼容是当前 IT 部件设计的主流趋势。充分借鉴这种主流趋势, 本文提出一种称为托架的通用软硬件结构作为实现自蜕变主动防御网络中网元的构建方法。

托架是一种根据自蜕变主动防御网络的总体框架, 专门设计的用于构建自蜕变主动防御网络中网元的通用设备, 它可承载传统应用软件和数据, 并实施主动变迁和跨层细粒度访问控制。托架及其承载的应用软件共同组成网元。

采用托架构建网元的主要思路包括两个方面:

(1) 采用通用的软硬件结构, 实现主动变迁、细粒度访问控制等共性主动防御机制。在这个通用的软硬件结构的基础上, 可以承载传统的不具备主动防御特性的应用和数据, 也可以承载具备内生性主动防御机制的应用和数据。对于后者可以提供更深入的主动防御支持。

(2) 将所有网元的托架部分组成一个实现网络整体主动变迁和细粒度访问控制的“控制平面”, 与这个网络所承载的应用节点和数据所构成的 “数据平面”适当的分离。使得不具备内生性主动防御功能的应用系统透明地运行于动态变更的自蜕变主动防御网络平台之上。

托架在自蜕变主动防御网络中的定位是:

(1) 托架是自蜕变主动防御网络中各类应用软件和数据的**装载器**;

(2) 托架是形成等效软件系统变体的**组装器**;

(3) 托架是网络、系统、应用各层变迁机制的**执行器**;

(4) 托架是传统网络应用的**兼容器**;

(5) 托架是实现态势感知以及跨层访问控制的伴生式**监控器**;

在托架上可以承载多套应用软件栈(操作系统、平台软件、应用软件等)和数据。托架根据总控中心的指令, 对托架间通信的网络协议、所承载的操作系统、应用平台以及相关的控制策略进行切换, 实现全局自主变迁。托架可以在总控中心的统一指挥下, 通过特定配置和重构机制, 改变所承载的软件栈, 转化成承担各类角色的网元。装载在托架上的应用软件系统中, 合规实体行为不受影响, 但非法注入的实体的行为将受到托架的控制和排斥, 无法持续生存。托架可以采用硬件实现, 也可以采用纯软件实现; 可以采用一体化的、可直接承载应用软件和数据的“主机”形态, 也可以采用包裹传统计算机的所有外联接口(网络、KVM、外设 I/O、电源)的“外置”设备形态。图 3 给出一个采用软硬件组合方式实现的, 可直接承载软件和数据的托架的逻辑结构:

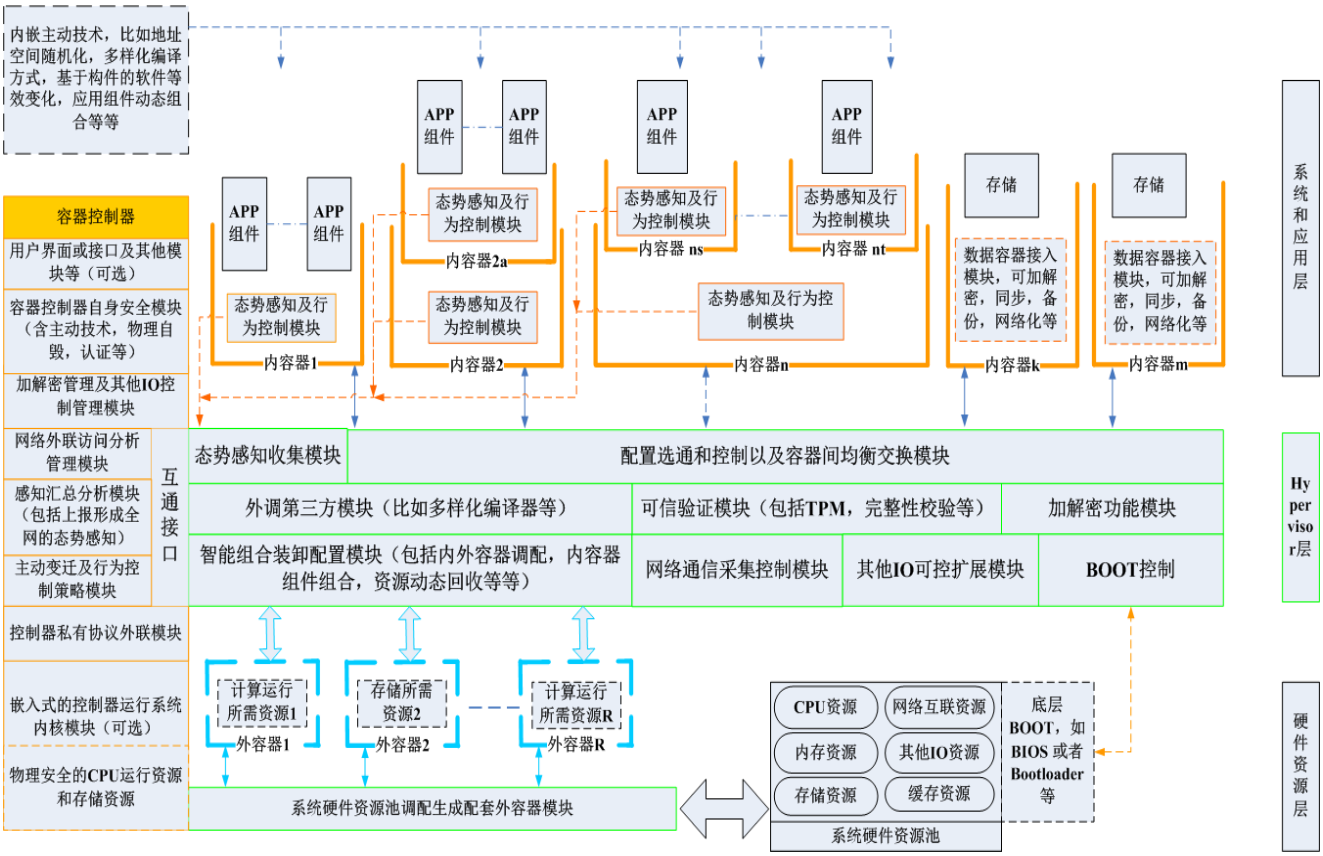


图 3　采用软硬件组合方式实现的托架的逻辑结构

当前主流计算机的 IT 架构大都是以冯-诺伊曼体系的通用计算机架构为基础, 提供通用的自完整的本机运算、存储、网络资源; 建立操作系统、平台软件、应用软件为主的多层松耦合软件栈; 以操作系统用户为主要主体建立管理层用户体系, 以应用系统用户管理为基础建立业务层用户体系; 以单个物理计算机为单元, 作为网络的基本运算节点。这个 IT 架构中没有对主动变迁和细粒度跨层严控提供直接的支持。图 3 所示的采用软硬件组合方式实现的托架, 将控制任务和计算任务分开, 所承担计算任务将在各层容器所控制的与传统主机兼容运行环境(传统计算机中的 CPU、内存和总线等核心部件)中执行; 控制任务在与计算任务相对独立的控制部件(包括容器控制器、外容器、内容器、Hypervisor 层)上执行, 实现可变和可控机制。

托架采用容器的方式装载应用, 容器分为外容器与内容器。外容器实现对硬件资源的分配组合包裹, 同时也实现对硬件资源的接管。其针对硬件资源, 根据应用需求动态调整, 支持资源根据应用实时分配并动态调整, 可适应目前的多核处理器(X86)技术,

支持资源在初始时以颗粒度较大方式静态分配。内容器对所承载的应用软件以及数据资源进行包裹, 可分为不同粒度的多层次的内容器, 并可嵌套。可以采用虚拟机方式包裹一个主机粒度的完整软件栈, 也可采用类似 Docker 的轻量级容器技术或者 wrapper 技术等方法包裹应用进程及其依赖于的执行环境。可对内容器承载的应用软件的内部组件采用多种主动安全机制, 包括等效软件变体, 基本配置变迁, 安全配置变迁等, 并植入跨层细粒度访问控制机制。对于仅承载数据的内容器只需实现存储功能, 可采用数据的可变密钥透明加解密、碎片化存储等主动防御机制。通过内容器对应用软件栈和数据进行包裹, 可以使处于“容器”之外的实体无法获得变迁的相关知识, 无法跟上变迁, 而容器之内的实体(正常应用)可以随动地适应变迁环境。内容器中配备态势感知和行为控制模块对其承载的软件的行为进行监控。

容器控制器是一个独立于业务系统外的智能核心控制部件。其主要实现对各层容器的管理和控制, 实现相关智能分析及变迁决策控制, 实现网络外联和管控(隧道建立、逻辑网络路由和转发控制等)。容器控制器还实现各个态势感知模块采集信息的汇总, 并对整个托架的安全机制进行管控, 包括加解密控制配置、可信验证, 完整性校验、网络行为的分析及控制(包括 IO 控制), 实现网络阻断等。

Hypervisor 层实现容器组件的组装和容器选择配置互通功能, 可扩展第三方主动安全工具(比如多样性编译工具), 便于容器动态组装。此外还实现对网络及其他 IO 资源的控制及感知, 完成所有容器的态势感知信息收集及行为控制指令下达, 执行具体加解密功能, 并在软硬件结合实现情况下对 Boot 完成接管, 实现可靠启动。

托架的自身安全防护, 可充分利用当前的软硬件安全技术的组合, 以完整性和可信性保障为重点。采用硬盘数据完整性检查, 防止重要数据被篡改; 采用数据存储透明加解密技术; 采用基于 TPM 的启动可信链; 对原有系统 Boot 部分进行接管, 防止后门激活; 在安全的系统启动前, 接管网络, 禁止系统外联; 在特殊情况下, 考虑配备系统自毁机制。

不同托架之间, 内容器包裹的应用软件之间数据传输的实际路径由托架之间的转发策略决定。托架之间采用隧道技术互联, 使用内部协议, 托架之间的转发路径和访问控制策略由总控中心统一控制。

原有应用软件栈由总控中心根据策略将相应的应用软件部署到相应的托架之上, 托架可根据部署和调动的需求, 由容器控制器在候选的应用软件栈中快速激活对应的镜像, 完成所需的网络、主机、软件系统的动态配置。在托架的支持下, 自蜕变主动防御网络上的相关机制可以顺利地实现, 并同时兼容传统 IT 技术。

以图 3 所示的结构为基础, 可以对相应的模块进行删减, 承载不同的应用软件栈来实现不同类型的网元。

3 基于虚拟化技术的自蜕变主动防御网络的实现方法

3.1 虚拟化技术为自蜕变主动防御提供的支持

传统 IT 技术在设计之初并未考虑对主动防御机制的支持, 因此不具备内生的“可变”、“可控”特性。采用软硬件组合方式实现的托架, 表现形态为一种新型的硬件设备, 需要较大的开发成本和产品化周期。虚拟化技术的发展可以为实现纯软件方式的自蜕变主动防御网络托架提供有力的支持, 也能实现一定程度的可变与可控。虚拟化技术一方面提供了动态创建相互隔离的虚拟运算和存储空间的支持, 另一方面可以在一定程度上实现“数据平面”与“控制平面”的分离。在虚拟化技术的支持下, 可以形成一个充分利用现有 IT 技术的自蜕变主动防御网络, 可承载传统的应用, 并符合当前主流的基于云计算平台的 IT 构建模式, 本文中称之为**“虚拟自蜕变主动防御网络”**。在当前的虚拟化技术开源项目中, OpenStack 作为一个开源云平台的管理架构和集合, 对实现自蜕变主动防御网络的四类主要网元的托架都能够提供有力的支持:

(1) 运算网元: 利用 OpenStack, 可在通用物理机上采用虚拟化技术构建虚拟自蜕变主动防御网络所需要的运算网元。物理机将通过 Hypervisor 的技术(例如使用最多的 KVM 技术)来提供虚拟化服务, 通过这一层来实现虚拟机的构建和操控, 实现了运算网元的托架。在此基础上采用虚拟机形式包裹“主机”粒度的应用软件, 在虚拟机构建完成后, OpenStack 提供的核心 Nova 模块将提供对运算资源的整合和集中的调度等。这就构成了虚拟化网络中的运算(compute)部分, 也就是虚拟自蜕变主动防御网络的运算网元。

(2) 存储网元: 采用集中化存储资源(如云存储系统), 配套相应的存储管理系统, 构成可承载应用数据的托架, 并内生地实现数据分片化、冗余化、透

明加密等主动防御机制。在此基础上根据应用需求分配, 采用虚拟存储目录建立存储网元和运算网元间的对应关系, 承载相关数据, 构成存储网元。在 OpenStack 中, 针对不同的使用需求, OpenStack 提供了若干个不同的模块来满足用户的需求, 可用于构建存储网元的托架。其中, Cinder 模块提供外部的磁盘卷, 可以为整个虚拟自蜕变主动防御网络提供存储服务, 通过调整模块的设置可以满足所需要的不同的策略, 包括对多个运算网元的支持等。而 Glance 和 Swift 模块则是提供保存镜像文件和保存磁盘文件等的需求, 提供对一些运算数据等等的镜像和备份服务。

(3) 传输网元: 采用具备 OpenFlow 支持的交换机/路由器, 可直接构成传输网元, 通过对流表的控制实现相关的变迁。此外, 还可以采用软件实现的虚拟交换机模块作为实现基本变迁机制的托架, 在此基础上承载特殊的变迁和控制软件(例如更改协议特定字段等), 构成具备特殊主动防御功能的传输网元。OpenStack 中通过 Neutron 模块负责对运算节点提供网络连接, 可使用 SDN 的技术来实现虚拟化网络中的路由机和虚拟交换机等等所实现的功能。

(4) 控制网元: 以云资源管理系统、SDN 控制器等为基础, 可形成托架, 承载特别开发的主动防御协同管理和决策系统软件, 以构成控制网元。在 OpenStack 中, 对于每一个重要的模块, 会有相应的节点来承担相应的处理, 例如在 OpenStack 中需要 Keystone 和 Network 节点等来承担对应的工作。此外, OpenStack 的 Horizon 模块所提供的 Dashboard 提供了一个完整的用户界面, 对系统几乎大部分的功能提供了一个比较完整的整合管理, 通过这一界面, 我们可以对整个系统的功能等按照我们的需要来进行对应的调整, 为构建控制网元提供了有力支持。

利用虚拟化技术构建托架, 形成虚拟自蜕变主动防御网络在虚拟网络、虚拟机层、虚拟执行环境三个层面都能实现良好的自蜕变机制, 并且可以实现多层自蜕变机制的协同。

3.2 基于虚拟网络技术的网络层自蜕变机制

在虚拟化环境中, 可以生成多个不同的虚拟网络, 形成逻辑隔离子网, 虚拟网络中存在可灵活设定转发表的虚拟交换器和虚拟路由器, 这给网络层自蜕变带来了很大的可行空间。SDN 技术的发展也使得 SDN 网络中控制器成为了有效的自蜕变主动防御网络控制中心托架的组成部分。

同时, SDN 技术使得建立在虚拟网络之上的虚拟拓扑和路由具备可变性, 可以充分利用 SDN 技术实现主动变迁。通过 SDN 控制控制器, 在传统技术构建的网络通道基础上, 建立逻辑上的多个动态可变的、对外不可见的层叠逻辑网络。

在 SDN 网络的基础上, 通过在各个虚拟机的虚拟网络接口建立安全容器(从虚拟网络的角度, 可以认为其控制了连接在这个虚拟网络接口上的虚拟节点, 实现外联网络访问控制模块的角色), 并拦截网络流量, 使各个虚拟机的网络通信必须通过安全容器进行。各个安全容器之间采用隧道技术互联, 在隧道技术基础上建立转发路径, 从而构成逻辑拓扑。在控制中心统一控制下动态变迁逻辑转发路径, 从而逻辑拓扑也动态可变。利用转发控制和隧道机制, 可以在通用的网络通道上建立对外不可见的隐性网络。逻辑网络的转发路径、封装协议、访问控制机制动态可变。

各虚拟机之间的信息传输均由安全容器实现。运行在各个虚拟机上各个应用软件之间数据传输的实际路径由安全容器之间的转发策略决定。安全容器之间可利用虚拟网络构建隧道互联, 可使用内部协议, 托架之间的转发路径和访问控制策略由总控中心统一控制。在此基础上形成的逻辑拓扑可在总控中心的控制下根据态势发展而变迁, 在网络和信息系统受到攻击、摧毁、侵扰的情况下自动愈合。

网络层自蜕变的实体是多样化的, 例如, 针对 IP 地址可以实现自蜕变, IP 地址所关联的实体是某一个终端。在一个多层次虚拟化环境中, 终端可以是一个物理机, 可以是一个虚拟机, 也可以是一个容器。针对端口自蜕变, 所对应的实体是一条具体的连接, 亦或只是一次交换的数据包(针对 UDP 等无连接协议)。其他一些协议中的字段, 例如 TCP 协议中的序号和窗口等字段, 都存在暴露某些额外信息的可能, 也是攻击者在实施信息收集时经常采用的手段, 这些也是网络层自蜕变所针对的实体对象。

这里以虚拟机 IP 地址的自蜕变为例, 介绍虚拟机 IP 地址的自蜕变具体流程。

当一个虚拟机 A 需要访问另一个虚拟机 B 时, 有两种情况, 一种是通过域名访问, 另一种是直接通过 IP 地址访问, 其中, 前者会有 DNS 服务器介入。由于需要确保 IP 地址变迁以后仍能被正确地传递, 变化范围限定在同一子网内。

以下过程描述了第一种情况下 IP 地址自蜕变的流程。

(1) 虚拟机 A 发出 DNS 请求虚拟机 B 的 IP 地址;

(2) 控制器发现这一请求, 将有关虚拟机 A 和虚拟机 B 的自蜕变规则与虚拟机 A 和虚拟机 B 的托架

进行规则同步;

(3) DNS 返回虚拟机 B 的真实 IP 地址 r2;

(4) 虚拟机 A 的托架收到 DNS 返回的结果, 将结果改写为虚拟机 B 的自蜕变 IP 地址 v2;

(5) 虚拟机 A 收到 DNS 响应, 以自身真实 IP 地址 r1 作为源地址, 目的地址 v2 发出数据包;

(6) 虚拟机 A 所在安全容器的变迁路由器收到数据包后, 根据自蜕变规则将源地址改写为虚拟机 A 的变迁 IP 地址 v1;

(7) 由于 v2 与 r2 处在同一个子网中, 网络中路由过程不变。或者控制器将具体路由规则下发到网络中的虚拟路由器中。具体取决于所选用的技术;

(8) 虚拟机 B 的托架收到数据包后, 根据自蜕变规则将目的地址改写为虚拟机 B 的真实 IP 地址 r2, 然后发往虚拟机 B;

(9) 虚拟机 B 收到数据包后, 发出响应数据包, 源地址为自身真实 IP 地址 r2, 目的地址为虚拟机 A 的自蜕变 IP 地址 v1;

(10) 虚拟机 B 的托架收到数据包后, 根据自蜕变规则将源地址改写为虚拟机 B 的自蜕变 IP 地址 v2;

(11) 由于 v1 与 r1 处在同一个子网中, 网络中路由过程不变;

(12) 虚拟机 A 的托架收到数据包后, 根据自蜕变规则将目的地址改写为虚拟机 A 的真实 IP 地址 r1, 然后发往虚拟机 A;

(13) 虚拟机 A 收到数据包, 一次通信过程结束。

以下过程描述了第二种情况(直接 IP 地址访问)下 IP 地址自蜕变的流程。

(1) 虚拟机 A 向虚拟机 B 发送数据包, 源地址为虚拟机 A 真实 IP 地址 r1, 目的地址为虚拟机 B 真实 IP 地址 r2;

(2) 虚拟机 A 的托架收到数据包后, 将与虚拟机 A 和虚拟机 B 关联的自蜕变规则与控制器进行同步, 同时验证请求是否合法;

(3) 虚拟机 A 的托架根据自蜕变规则将源地址改写为虚拟机 A 变迁 IP 地址 v1, 将目的地址改写为虚拟机 B 变迁 IP 地址 v2;

(4) 由于 v2 与 r2 处在同一个子网中, 网络中路由过程不变;

(5) 虚拟机 B 的托架收到数据包后, 根据自蜕变规则将目的地址改写为虚拟机 B 的真实 IP 地址 r2, 然后发往虚拟机 B;

(6) 虚拟机 B 收到数据包后, 发出响应数据包, 源地址为自身真实 IP 地址 r2, 目的地址为虚拟机 A 的变迁 IP 地址 v1;

(7) 虚拟机 B 的托架收到数据包后, 根据自蜕变规则将源地址改写为虚拟机 B 的变迁 IP 地址 v2;

(8) 由于 v1 与 r1 处在同一个子网中, 网络中路由过程不变;

(9) 虚拟机 A 的托架收到数据包后, 根据自蜕变规则将目的地址改写为虚拟机 A 的真实 IP 地址 r1, 源地址改写为虚拟机 B 的真实 IP 地址 r2, 然后发往虚拟机 A;

(10) 虚拟机 A 收到数据包, 一次通信过程结束。

得益于 SDN 技术的发展, 采用中央控制的同步方案将更加易于部署, 具有相当高的可靠性和可扩展性。在本例中, 可以使用 Open vSwitch 作为托架, 使用 OpenFlow 使中央控制器将流表规则下发于托架中, 以使托架可以根据流表规则对数据包进行改写和转发。

3.3 基于 Hypervisor 的虚拟机层自蜕变机制

虚拟化环境中, 可以迅速创建或销毁虚拟机, 虚拟机的重构由镜像重构模块实现, 通过对应用虚拟机的镜像进行操作, 可以实现虚拟机层的自蜕变机制:

将最新 MTD 机制注入原虚拟机, 并自动配置启用相关的 MTD 机制。

(1) 通过替换相关等价功能模块, 从一个原始虚拟机镜像构造出一系列等价虚拟机变体;

(2) 重配虚拟机相关参数, 进一步引入随机性;

(3) 结合虚拟机管理调度系统, 实现等价变体的动态启用;

(4) 结合异构冗余机制, 实现等价执行体的动态选用。

近年来在 MTD 领域的研究成果可以支持可定义主动防御网络上相关实体自身的主动变迁, 例如 ASLR、ISR、Just-in-Time compiler 等软件混淆技。可定义主动防御网络框架结构中可应用 MTD 领域的最新研究成果, 实现实体自身的主动变迁。

相关的应用软件栈将可以通过扫描、导出和人工参与的组装等方式形成可定义主动防御网络种的标准镜像, 通过虚拟机管理调度机制部署到云计算平台中。如下图 4 所示:

同时, 通过启用多台虚拟机, 可以实现动态异构冗余机制, 基本思路为:

(1) 通过虚拟拓扑变迁, 在原虚拟拓扑中插入异构冗余环节, 形成新拓扑;

(2) 利用虚拟机重构机制, 形成等价异构虚拟机备选集群;

(3) 构建主动防御功能虚拟机(如拟态防御组件),

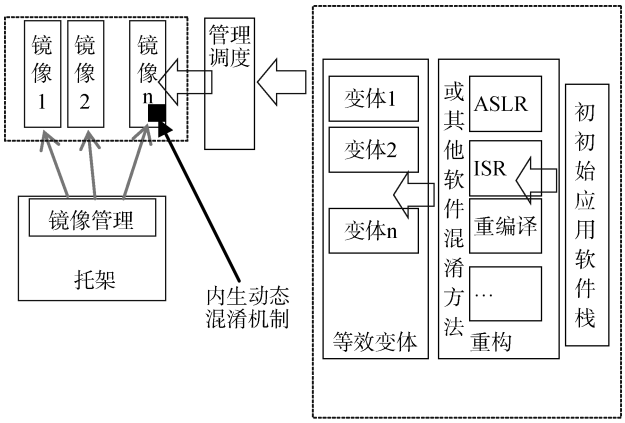


图 4　等效虚拟机镜像重构

包括输代理、裁决器、输出代理。通过虚拟网络管控中心导流;

(4) 通过虚拟机管理调度中心, 选取开启等价异构虚拟机和主动防御功能虚拟机;

(5) 根据策略, 动态变迁等价异构虚拟机和主动防御功能虚拟机。

3.4　基于容器技术的执行环境层面的自蜕变机制

容器技术是一种较新的轻量级虚拟化技术。使用容器技术实现应用执行环境自蜕变, 较之 Hypervisor 的虚拟机自蜕变技术, 有更小的细粒度, 消耗更少的资源。类似虚拟机层面的异构冗余、镜像构造等技术, 多数可以在容器技术上得以运用, 从而形成多层次的自蜕变系统。同时, 在容器技术的基础上可以形成自蜕变共生容器, 对相关应用软件进行包裹。该容器在原有的应用部件(如 Web 服务器和负载均衡器、业务逻辑处理器与数据库系统)信息系统的基础上, 通过对各个部件的相关对象自蜕变共生容器进行包裹来实现自蜕变防御。因此容器技术可以用于实现内嵌到运算网元所承载的应用中的蜕变和控制机制。

(1) 自蜕变共生容器截获原有的 IT 部件对外交互信息。在控制对外交互的过程中, 将交互涉及的拟变迁要素进行随机化处理。例如将原有的命名空间进行变迁。

(2) 自蜕变共生容器对相关的执行环境进行等价变迁(这种变迁没有改变接口, 不需要同步), 例如替换等价的依赖库等。

(3) 对于需要同步的变迁, 自蜕变共生容器之间通过安全的协同和同步机制实现关联, 协商生成同步的随机数因子, 通过反运算可以还原成原有正常信息。

(4) 经过自蜕变共生容器处理后, 在网络通道上传输的实际网络通信经过了随机化处理, 其中的相关字段已经不是原始的信息, 而是表现为具有随机特性的网络通信信息, 经过服务端部件的自蜕变共生容器进行反运算处理才能还原成原始的端对端应用交互信息, 实现对相关服务对象的正常访问。

(5) 网络通道上的攻击方所操控的 IT 部件(包括硬件设备、攻击软件等), 如果没有自蜕变共生安全容器的包裹, 无法在全局变迁的环境中定位应用资源, 在缺乏正常使用方的必要知识和容器环境的情况下, 随意生成的请求在经过相关应用单元的自蜕变共生安全容器的反运算所转化成的请求是“非法”的, 无法到达真实的服务对象, 将被进入伪装服务器, 得到经过伪装或混淆的信息。

(6) 即使被包裹的应用部件被攻破, 原 IT 部件完整性若遭到破坏, 针对原有 IT 部件适配的自蜕变共生容器不能与遭破坏的部件正常共生运行, 也就无法被攻击者所利用。此外, 自蜕变共生容器还将结合使用用户身份认证和行为模式监控等机制, 在没有获得有效身份认证因子(动态口令输入或身份认证辅助硬件), 行为模式严重偏离正常模式的情况下, 也无法生成“合法”的随机的应用请求。

3.5　基于分层控制中心的自蜕变机制的协同

多数自蜕变防御机制要发挥作用, 都需要假设攻击者没有侵入到正常使用者内部, 如果攻击者采用某种方法侵入到正常使用者内部, 那么就可以利用正常使用者获取相关知识或访问途径。在多数场景中, 攻击和正常使用所需的知识是重合的, 安全的协同机制就尤为重要, 它是使自蜕变机制发挥安全防护效果的前提。所以除了安全的通知和同步机制之外, 对正常使用者的防护也是至关重要的。

基于分层控制中心的自蜕变机制的协同是针对多层次虚拟化环境提出的多机制协同方案。与单层控制中心相比, 分层控制中心更容易部署, 管理更加方便, 逻辑更加清晰, 同时不容易出现故障, 从故障中恢复的速度也更快。

一种可行的分层控制方案是根据不同层次的虚拟化技术建立分层控制中心。针对宿主机, 有一个物理控制中心, 控制所有处于物理层面的实体的自蜕变机制的协同; 针对虚拟网络, 有一个虚拟网络控制中心, 控制所有处于虚拟网层面实体的自蜕变机制的协同; 针对虚拟机, 有一个虚拟机控制中心, 控制所有处于虚拟机层面的实体的自蜕变机制的协同; 针对容器, 有一个容器控制中心, 控制所有处于容器层面的实体的自蜕变机制的协同。此外, 还有一个

主动变迁及行为控制决策中心对各层控制中心进行协同调度

每一个层面的实体受到相应层面的控制中心的控制, 同时其本身处于比其低一个层面的实体中, 例如, 虚拟机实体存在于物理机实体中, 容器实体存在于一个虚拟机实体中。这样, 每一个上层实体实际也受到下层实体的自蜕变机制的保护, 其本身的自蜕变机制也对上层实体的保护提供支持。同时, 由于虚拟化技术在故障恢复重启方面的天然优势, 上层实体一旦遇到故障, 可以由下层提供的虚拟技术快速恢复。

为了实现架构上的简洁性, 自蜕变机制的协同由主动变迁及行为控制决策中心进行集中调度, 整个架构如图 5 所示, 其中态势分析模块和实体管理模块用于提供自蜕变决策所需的信息。整个决策中心和各个分层控制中心共同组成了自蜕变机制协同系统。各个分层控制中心收集各层面的实体信息以及威胁感知信息, 对本层面的威胁感知信息进行初步匹配和筛选, 并将经筛选的威胁感知信息和实体信息上报决策中心。决策中心根据收到的各层威胁感知信息和实体信息, 以系统管理员配置的基本策略为基础, 生成总体变迁规则, 推导出一系列具备一致性、协同性特性的分层主动变迁和访问控制规则并分别向各个分层控制中心发送; 各层控制中心形成具体变迁和访问控制指令下发到各层实体。如果某个层面的控制中心基于所收集到的本层态势感知信息, 根据预设的应急响应规则需要进行某项应急变迁时, 将本层应急变迁请求发送到决策中心, 由决策中心根据当前的各层实体信息, 生成各层次的协同变迁和访问控制规则发送到各层面控制中心, 继而实现其他层面实体的协同变迁和对应的访问控制。在协同变迁执行时, 下层实体作为上层实体的载体, 上层实体的通信会受到下层自蜕变机制和访问控制机制的保护, 同时也受下层变迁和访问控制的影响, 因此在协同变迁的时序方面, 一般采取先下后上的次序, 变迁的时序由决策中心掌控, 各层控制中心根据决策中心发出的执行时序要求, 在相应时机下发指令执行变迁。

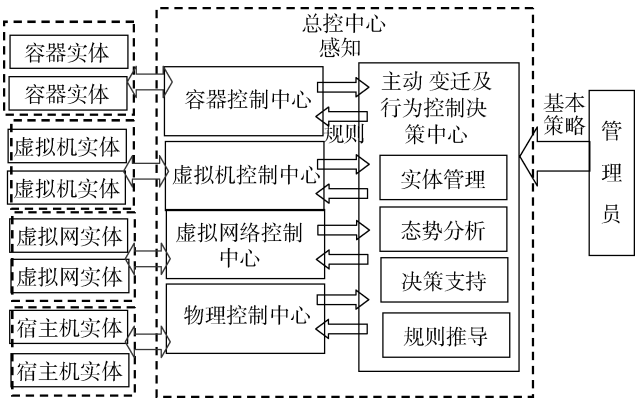


图 5 基于分层控制中心的自蜕变机制的协同系统

4 在虚拟自蜕变主动防御网络上实现随机 URL 机制的方法

采用虚拟化技术构建的“虚拟自蜕变主动防御网络”, 通过不同层面的主动防御机制的协同可以实现各类全局化的主动变迁机制。本章以 URL 随机化为例, 描述了采用虚拟自蜕变主动防御网络上多层协同机制实现相关因素主动变迁的方法。URL 随机化对于依赖固定 URL 的网络攻击具有防御效果, 可以使得攻击方无法锁定所要攻击的目标 URL, 也就无法发动有效的扫描和攻击。该主动防御机制可以通过虚拟自蜕变主动防御网络中多层协同的主动防御机制来实现。

4.1 随机 URL 的作用

针对 Web 的攻击首先需要确定攻击目标的 URL。传统 Web 应用的首页入口 URL 固定不变, 很容易让黑客确定目标。如果 URL 可以主动发生改变, 则黑客就不容易确定目标。当然其前提是正常用户能够即时定位变更中的 URL。利用虚拟自蜕变主动防御网络, 将 Web 客户端结合服务器装在到相应的网元托架上, 并且在虚拟自蜕变主动防御网络上部署实现 URL 变迁的 proxy 网元, 可以在虚拟网络、虚拟机、容器等多层自蜕变机制的协同作用下, 兼容传统应用习惯实现随机 URL 变迁。

在该方法中, 客户端和服务器端双方的代理是是针对其应用的自蜕变托架, 是 URL 变迁的执行者。客户端托架负责接收由客户发出的 HTTP 请求, 对 HTTP 请求进行相应“变迁”后, 发送至服务器托架。服务器托架承载 Web 服务器, 在收到由客户端托架“变迁”后的请求后, 验证该请求的合法性, 过滤并处理后转发至 Web 服务器, 如图 6 所示。通过托架, 在 HTTP 请求的 URL 中加入随机化生成的字段进行通信, 使每次 HTTP 访问所采用的 URL 均不相同, 以抵御针对固定 URL 的扫描和自动攻击行为, 即使原有的 Web Server 上存在漏洞和后门, 非法用户也因无法构造随机生成的合法 URL 而无法利用该漏洞和后门, 从而保护了 Web Server。由于 URL 自蜕变的过程中依赖于时戳、请求序号等时刻变化的信息, 从而保证了无法通过网络截取通信信息后重放访问相关网页, 避免了重放攻击。

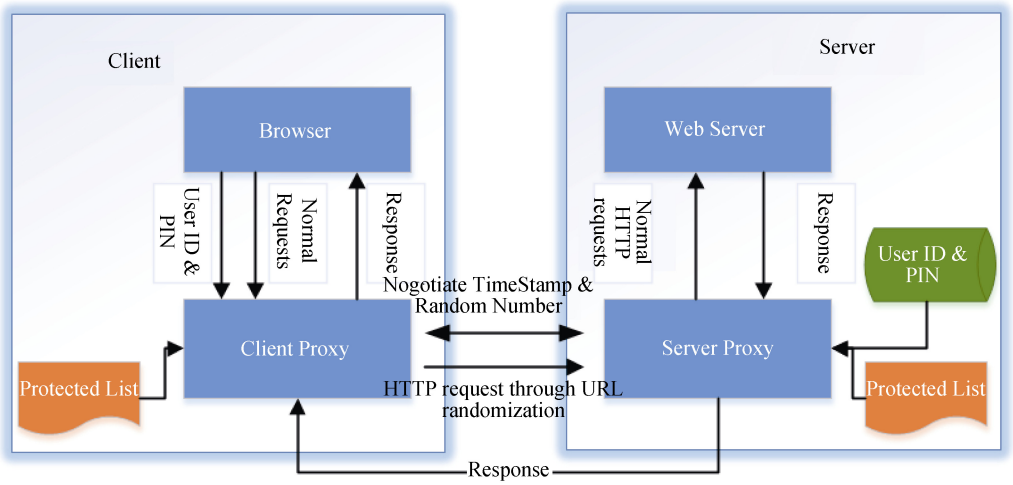


图 6 URL 自蜕变逻辑架构

4.2 URL 变迁流程

(1) 进行相关配置: 在客户端托架和服务器端托架配置所需要保护的网站列表; 在服务端托架配置合法用户的 ID 和 PIN 信息; 用户配置客户端托架的 IP 地址和端口; 在客户端托架和服务端托架配置 Diffie Hellman Group 号(Group 号确定 Diffie Hellman 密钥交换机制所采用的 p 和 g 参数, 其中 p 大素数, g 是这个大素数的原根, p 和 g 允许公开)。系统进行初始化, 读入配置, 运行进程, 监听服务端口, 建立空会话表项。

(2) 客户端通过客户端托架访问网站。

(3) 客户端托架收到 HTTP 请求, 比对客户向客户端托架发出的 HTTP 请求中含有的 HOST 字段与受保护的网站列表。所要访问的网站不属于受保护网站的, 采用标准的代理机制访问。所要访问的网站属于受保护站点的, 采用身份验证机制, 获取并记录用户 ID 和 PIN 信息。

(4) 客户端托架生成会话标识 s, 该会话标识 s 由客户端 IP、客户端 PORT、服务端 IP、服务端 PORT 这四项信息组成; 生成随机数 x, 生成 Diffie Hellman 公钥信息 c:

$$c = g^x \bmod p \tag{1}$$

(5) 向服务端托架发送初始随机数和时戳请求, 包含:(s、ID、c); 设定初始请求序号 n=0, 空闲标识 idle=0, 为这个会话生成本地会话表项, 在会话表项中记录(s, ID, PIN, x, n, idle), 因为该会话表项用 s 作为索引, 故下文中使用“会话表项 s”指代“为会话标识为 s 的会话所建立的会话表项”。

(6) 服务端托架获取本地时间信息 t, 根据需要采用相应的精度, 如 5 秒, 10 秒, 1 分钟等; 生成随机数 y, 生成 Diffie Hellman 公钥信息 d:

$$d = g^y \bmod p \tag{2}$$

(7) 向客户端托架发送应答信息, 包含: (s,t,d); 运算初始随机数 r:

$$r = c^y \bmod p \tag{3}$$

(8) 设定初始序号 n=0, 空闲标识 idle=0, 生成本地会话表项(s 为索引), 记录为相应会话分配的信息(s,ID,r, n, idle)。

(9) 客户端托架获取服务端托架的应答信息, 提取其中的时戳 t, 取本地时间 t’, 运算ΔT:

$$\Delta T = t - t' \tag{4}$$

(10) 提取存储在本地会话表中相应会话表项中的 x, 运算初始随机数 r:

$$r = d^x \bmod p \tag{5}$$

(11) (根据 Diffie Hellman 密钥交换机制, 该运算结果应该与步骤 5 中服务端托架运算的结果 r 相同); 在本地会话表项 s 中记录(ΔT, r)。

(12) 客户端托架提取会话表项 s 中的 ID, PIN, ΔT, r, n; 取本地时间 t’, 运算时戳信息 t 和摘要 h:

$$t = \Delta T + t' \tag{6}$$

$$h = HMAC(PIN, ID\_t\_r\_n) \tag{7}$$

(13) 生成随机目录字段 z=s_h。在原 URL 字段的首部增加将一个随机目录/z。将经过随机化处理的 HTTP 请求发送到服务端托架。更新会话表项 s 中的 n=n+1, 将会话表项中的空闲计数器 idle 清零。

(14) 服务端托架提取 HTTP 请求中的随机目录字段 s、h, 根据连接序列标识 s, 获取本地会话表中的 ID、r、n, 并根据用户 ID 获取用户库中相应用户的 PIN, 取本地时间信息 t。运算 h' :

$$h' = HMAC(\mathrm{PIN}, \mathrm{ID}\_t''\_r\_n) \tag{8}$$

(15) 将 h' 与所提取的随机目录字段的 h 进行异

或运算, 如结果为 0, 则去除 HTTP 请求中的随机目录字段, 将去除随机目录字段的请求提交 Web Server, 更新会话表项 s 中的 n=n+1, 将会话表项中的空闲计数器 idle 清零; 否则, 丢弃请求, 返回错误页面, 清除连接和会话表项, 结束本次会话。

(16) Web Server 返回页面信息到服务端托架, 该页面通过服务端托架、客户端托架返回到客户。

(17) 客户获取返回页面信息后, 进行后续访问时, 相关 HTTP 请求发送到客户端托架。客户端托架根据 IP 地址、PORT(或 IP 地址), 查找会话表判断是否已经为该用户分配会话表项, 如果未分配表项, 则为新会话, 跳转到(3)。否则, 查找到所分配的表项 s, 转步骤 7。

(18) 客户端托架和服务端托架如收到来自客户、Web Server 的连接中止信息, 都将关闭相应的连接, 清除相关会话表项。

(19) 客户端托架和服务端托架都将定时对所有生成的会话表项的空闲标识 idle 值增加 1, idle=idle+1。如果超过阈值, 标识该表项很长时间未用, 则关闭相应连接, 清除会话表项。

4.3 虚拟自蜕变主动防御网络上 URL 自蜕变的实现方法

随着虚拟化技术的发展与应用, 利用虚拟化技术实现诸如拟态防御的主动安全机制的一些想法也慢慢成为一种可能, 特别是近几年迅速发展的 SDN 技术, 使得网络的流向变得更好的控制。容器技术的发展, 也使得对资源的利用更加的高效, 迁移也变得更加的方便和有效。更加重要的事, 容器技术和虚拟化技术提供了更好的隔离, 使得各个程序隔离成一个个比较安全的环境。容器相比于虚拟机而言更加的轻量级, 从而使得其成为切换与迁移更好的选择。可以利用虚拟化技术和 Docker 实现更加有效的动态可变的、对外不可见的, 透明的 URL 自蜕变变迁。如图 7 所示。

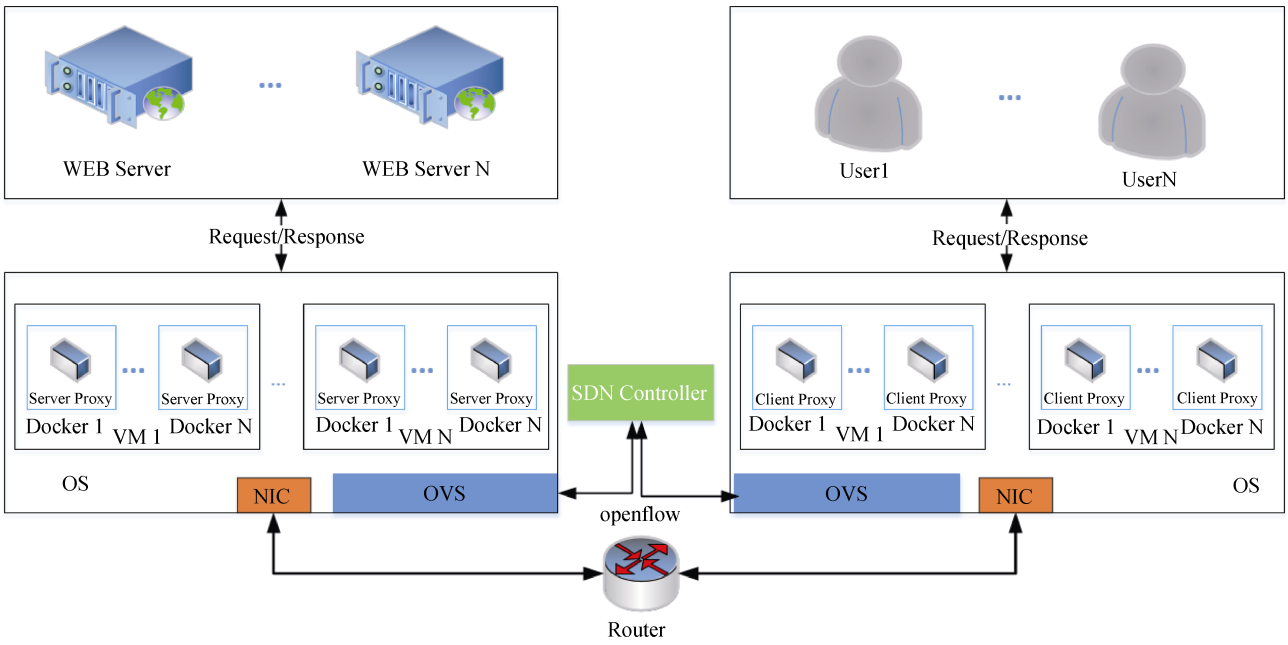


图 7 虚拟自蜕变主动防御网络上的 URL 自蜕变架构

(1) 基于虚拟网络/SDN, 实现传输路径强制途经自蜕变托架。

(2) SDN 利用 OpenFlow 将网络设备的控制面和数据面分离开来, 从而实现了网络流量的灵活控制, 使网络变的更加智能。在自蜕变的过程中, 利用 SDN 控制器, 通过 OpenFlow 协议, 给 Open vSwitch 下发一些流表规则来控制数据的流向, 使其经过客户端托架和服务器端托架, 对 Web Server 或者客户进行请求和响应。

(3) 基于虚拟机技术, 部署客户端托架、服务器端托架、Web Server 运算网元, 并且可存在多个功能等价, 实现异构的运算网元和自蜕变托架轮流切换或异构冗余。

(4) 在一台服务器上, 可以起多个的虚拟机, 在每一个虚拟机上部署一个客户端托架或者服务器端托架, 各个虚拟机之间可以通过 Bridge 进行信息交换。在变迁的过程中, 我们可以利用变迁技术, 进行等价的切换, 从而减少被攻击的可能性, 提高相对

应的安全性。

(5) 基于容器/Docker 实现配置/配置文件的动态变迁

(6) 利用 Docker 轻量级, 同时隔离的安全性, 可以在一台虚拟机启动多台 Docker, 并且在每台虚拟机上保存一些种子, 利用这个种子, 虚拟机上的所有 Docker 可以进行等价的配置, 并在需要的时候, 进行动态的切换。

4.4 防御效果的估算

在进行理论分析的之前, 我们需要进行一些假设, 首先, 我们认为攻击者是可以监听并获取一些基本的信息, 其次, 我们认为在托架内部保存的信息是安全的。攻击者有两种方法得到自蜕变后的 URL。一种是, 攻击者劫持了客户端托架和服务器端托架之间的连接, 然后利用连接生产自蜕变后的 URL。另外一种是, 攻击者直接通过暴力破解去猜出用户的 PIN 码, 然后去连接 Web Server。

(1) 从劫持的连接生产自蜕变后的 URL

因为在自蜕变的 URL 方法中, 我们实际上是在已有的 URL 基础上嵌入一个通过 HMAC 函数生产的目录。如果攻击者想要获取相同的目录, 他们需要得到 HASH 值。得到 HASH 值的方法有两种, 一种是利用暴力破解的方法, 另一种是通过获取随机数 r 或者 DH 私钥 x。我们假设服务器端托架每 5 秒钟获取一次当前的时间戳 t”。从第(8)步, 我们知道序列号 n 只有在从服务器返回正确的 URL 后才会改变。攻击者进行一次攻击, 至少需要一个 RTT 的时间去验证攻击结果的正确性。我们假设 RTT 的时间是 10ms,也就是 5 秒钟可以发 500 次。此外, 我们还假设一个 Web Server 可以同时保留 M(M>0)个连接, 攻击者同时可以发起 N(0<N≤M)个攻击。

利用暴力破解的方法去生产 HASH 值: 因为 SHA-1 的值为 160 个位, 如果攻击者想要构造相同的目录的可能性为:

$$P(Success)=\frac{500\times N}{2^{160}}\approx\frac{N}{10^{48}} \tag{9}$$

这个 $P(Success)$ 的值近似为 0, 即使 N 值非常大, 我们也认为这个暴力破解不起作用。

通过随机数 r 或者 DH 私钥 x 生产 URL: 根据我们之前的合理的假设, 攻击者要想猜出 URL, 必须需要知道 r 和 x。因为 x 是存储在客户端的 proxy 中, 也就是说:

$$P(Suceess)=0 \tag{10}$$

攻击者获取不到 x, 只能去试图得到 r。为了得到 r。只需要得到 y, y 又是由伪随机数生产的。随机数 r 在连接的过程中是保持不变的, 并且我们假设 keep-alive 的时间很长, 我们取 1 个小时。在 C++标准库中, 随机数算法是 srand(seed), 这个方法返回一个数介于 0 和 RAND_MAX 之间。类似的, 我们定义一个 mtdrand(seed), 并且这个函数返回一个 0 到 Q(232<Q)。我们假设攻击者可以同时发起 N 个连接进行攻击。

$$P(Success)=\frac{1\times 60\times 60\times\frac{1000}{10}\times N}{Q} \tag{11}$$

这个 P 的值由 N 和 Q 来决定。因为连接服务器的连接是一定的, 我们认为是一个常数, 因此如果 Q 足够大的话, 我们可以认为通过这个方法是行不通的。

(2) 直接通过暴力破解去生产 PIN 的值

如果攻击者设法跳过了客户端托架直接和服务器托架进行连接, 攻击者只需要知道用户的 PIN 然后利用 PIN 去和服务器托架协商生成 r。然后再利用 PIN 去生成 HASH 值。为了得到 PIN, 攻击者可以利用暴力破解的方法。我们假设 PIN 的长度为 L, 并且 PIN 由可打印字符组成。同时, 我们假设用户每一年改一次 PIN, 一个 RTT 为 10ms。因此, 如果攻击者一年的时间内同时利用 N 个连接去猜测 PIN 码, 则可能猜中的可能性为:

$$P(Success)=\frac{365\times 24\times 60\times 60\times\frac{1000}{10}\times N}{94^{L}} \tag{12}$$

这个可能性 P 由 N 和 L 决定, 如果 N 是 10 亿的话, 那么用户只需要设置 9 位以上的 PIN 码即可保证安全。

5 虚拟自蜕变主动防御网络上随机 URL 变迁机制的实验效果

5.1 环境描述

在前一个章节中, 我们论述了虚拟自蜕变主动防御网络上随机 URL 变迁机制的有效性。这一章节, 我们将通过两个实际的对比实验, 第一个对比试验, 用来验证我们所提出的虚拟自蜕变主动防御网络框架下随机 URL 变迁的可行性和有效性。第二个对比试验, 我们用来测试框架所带来的开销问题。实验总体架构如图 7 所示, 表 1 列出了实验环境所需要的软硬件信息。

在两台服务器上, 我们分别安装 CentOS 7 操作系统, 并分别安装 OpenStack 管理平台, 并分别起 10 个虚拟机(M≥1, 这里 M=10), 在虚拟机的基础上, 我们分别安装了 100 个(N≥1, 这里 N = 100)Docker

表 1 实验环境所需的软硬件信息

实验环境说明		
名称	数量	软硬件具体信息
物理机	2	Dell PowerEdge R730
物理机-上的 OS	2	CentOS 7
云计算管理平台	2	OpenStack
容器	N	Docker 1.11
SDN 控制器	1	NOXRepo
虚拟机	M	Ubuntu 14.04
虚拟交换机	2	Open vSwitch 2.3.0

容器, 每个 Docker 容器里, 我们都包装了服务器和客户端托架, 每一个虚拟机中的 Docker 的配置信息是不同的, 通过虚拟机中一个我们设计的控制中心, 他们之间可以进行等价切换。同时我们安装了一个 SDN 控制器和 Open vSwitch, SDN 控制器通过 OpenFlow 协议给 Open vSwitch 下发一些规则。

5.2 实验结果

5.2.1 可行性和有效性测试

当客户端通过 http 访问服务器端 Web Server, 在到达对方服务器的过程中, 客户端虚拟机中的 Docker 控制中心会将请求发送到一台 Docker 容器的客户端托架中, 客户端托架将变迁后的 URL 继续往下发送, 通过 NOXRepo 下发的流表, 强制通过服务器端的 Open vSwitch 然后进入服务器端任意一台 Docker 容器当中的服务器端托架进行逆运算, 得到正常的 URL 发送给 Web Server。图 8 显示了客户端可以正常的运行, URL 也按要求进行变迁得到变迁后的 URL, 每一次变迁完成后客户端的 Docker 进行一次等价切换。

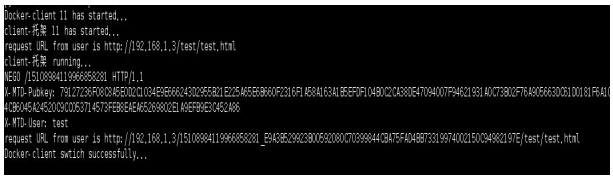


图 8 多层次 URL 变迁客户端托架的运行结果

服务器端托架接到客户端变迁后的 URL, 在服务器端的已经起的 Docker 容器中进行反运算, 最后的结果如图 9 所示。

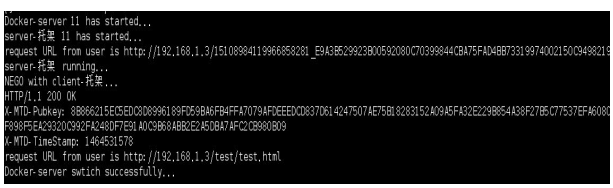


图 9 多层次 URL 变迁服务器端托架的运行结果

以上说明了虚拟自蜕变主动防御网络 URL 变迁的可行性, 下面我们通过一个经典的 IIS6 漏洞去测试其有效性。如图 10 所示, 我们在 Web Server 上安装一个包含经典 IIS6 漏洞的 Web Server, 在客户端托架上, 我们安装一个 Metasploit 工具, 并放置 Perl 脚本以绕过验证。

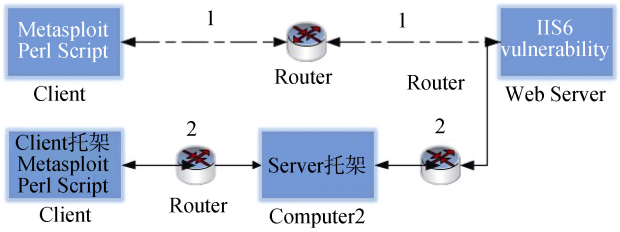


图 10 验证 URL 变迁有效性模型

客户端通过 Metasploit 工具扫描 Web Server, 然后运行 Perl 脚本去绕过验证得到 Web Server 上的资源。首先, 我们在没有利用我们提出的方法的路线 1 测试, 测试如图 11 显示, 我们可以发现一个漏洞在 Web Server 上, 利用它可以得到我们想要的资源, 如图 12 所示。然后, 我们在利用我们提出的方法上路线 2 进行测试, 测试的结果如图 13 所示, 扫描不到想要的资源, 也就是说, 我们提出的虚拟自脱变网络 URL 变迁有效。

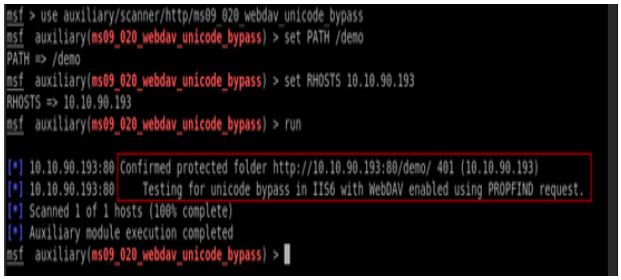


图 11 扫描没有利用虚拟自蜕变主动防御网络下的 Web Server

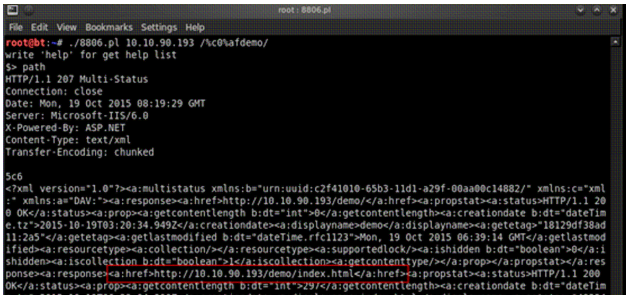


图 12 运行 Perl 脚本绕过验证

通过以上的简单实验, 我们可以验证我们所提出的多层次 URL 变迁功能的可行性和有效性。

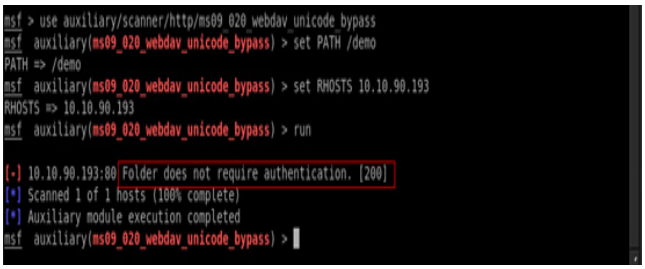


图 13　扫描利用虚拟自脱变网络下的 Web Server

5.2.2　性能测试

将已有的程序，放入到安全容器中，并进行 URL 变迁，在这个过程中，都增加了系统的开销，带来了性能的损失。对于网络访问而言，我们最关心的是网络时延的问题，所以，我们不考虑过程中的内存开销等问题，只研究我们所提出的方法带来的网络时延问题。下面我们设计一个对比实验，对多层次网络所带来的性能损失进行一个分析和评价。实验的架构图如图 14，图 15 所示。Client 端，我们使用普通的浏览器, Web Server 我们使用了 Tomcat 7。为了只测网络时延，我们所获取的 test.html 文件，是一个只包含一句话的网页。

图 14　正常的访问访问模型

图 15　经过多层次虚拟 URL 架构的访问模型

因为在同一个子网内做的实验，因此，我们可以假设实验不受其他因素影响，只是受我们所加的架构带来的处理时延影响。

表 2　性能测试实验结果

方式	实验次数	请求文件	时间 (ms)	平均每次访问时间 (ms)
正常访问	100	test.html	2040	20.4
URL 变迁架构	100	test.html	2507	25.07
正常访问	10000	test.html	202816	20.2816
URL 变迁架构	10000	test.html	246474	24.6474

通过计算我们发现，虚拟多层次 URL 自蜕变主动防御网络所带来的开销为 20%左右，相比于之前的不加多层次的网络架构的 16.9%[2]，增加了一些，这个是由于我们所增加的 Docker 层和 SDN 控制器的影响。

6　展望

主动防御技术，例如移动目标防御技术(MTD, Moving Target Defence)，包括指令集随机化技术，地址空间随机化技术，网络随机化，软件应用层随机化等等作为一个新兴的热点技术，近几年得到了快速的发展，但大都是单点技术为主，而托架的提出就是为解决单点主动防御技术不够系统化，解决不能从体系上支撑主动防御的问题，本文所论述的托架其实质是个开放式的可变可控的体系架构平台，通过装配组合形成容器，并以容器包裹应用软件作为基本运行单元，本文所描述的虚拟自蜕变主动防御网络框架的 URL 的变迁，仅仅是托架一种最简单的应用，通过容器间相互嵌套组合或关联组合，可以形成各种复杂应用，满足应用多样性需求，同时通过对容器内部及托架出入口进行态势感知，通过体系内嵌软件定义网络，实现容器间业务流程及数据流程的柔性再造，最后辅以可信技术等，最终形成一个支撑主动防御技术组合运行的柔性体系框架，这样的框架可以非常容易的集成当前众多主动防御技术思想，包括单点防御技术，组合技术思想(如异构冗余，主动重构，动态化，随机化等)等，融合提升成为体系化防御，具备纵深防御能力，大大提升了系统的安全性。

我们提出的托架作为基本的体系架构，可以兼容现有的硬件体系结构，不仅可以简化作为单机系统(包括网络设备，工控，嵌入式系统)的结构防护体系，在已有设备环境下，通过叠加性操作，使得原有系统具备主动防御的特性，也可以作为适用于集群系统以及云环境，同时也可以融入到处理器芯片，和处理器 FPGA 化，或者同构/异构众核芯片相结合，使得系统体系在设计之初就融入了系统级安全框架的理念，从硬件底层开始具备主动防御的能力。

目前，托架整体框架已初步完成，但体系部分细节还需要进一步完善优化，比如容器切换时，原提供服务的服务连续性如何保持问题；随着攻击越来越智能，如何提升容器内嵌的各类感知器的有效感知能力；随着主动防御技术的蓬勃发展，如何提供更灵活的开放式接口以便纳入最新的防御技术；托架通过容器嵌套技术及容器间关联技术形成不同的应用，但如何使容器更细粒度化及更层次化，以达到更加柔性的变化等等，这些问题还需要进一步研究优化解决。

当前我国正在信息化建设，相比发达国家，我国在信息产业基础方面还处于落后状态，“被后门”的威胁更大，而网络空间安全威胁的根本原因是普

遍存在的漏洞和后门, 造成网络空间“易攻难守”的基本态势, 有效的网络空间安全防御已成为最为紧迫的网络空间战略问题, 而主动防御技术正是改变这一状态的有效利器, 随着各类主动防御技术及防御体系支撑技术(如托架)的进一步发展, 主动防御技术体系将逐步走向实用化, 将更有效防备目前各类“有毒带菌”的软件未知漏洞/木马, 必将在网络安全防护中得到广泛应用。

参考文献

[1] NITRD, “National Cyber Leap Year Summit 2009 co-chairs’ report, networking and information technology research and development,” National Office for the Federal Networking and Information Technology Research and Development Program, Tech. Rep., Sep. 2009.

[2] D. Kewley, R. Fink, J. Lowry, and M. Dean, “Dynamic approaches to thwart adversary intelligence gathering,” Proceedings of the DARPA Information Survivability Conference& Exposition, pp. 176–185, 2001.

[3] J. Michalski, C. Price, E. Stanton, E. L. Chua, K. Seah, W. Y. Heng, and T. C. Pheng, “Final report for the networksecurity mechanisms utilizing network address translation LDRD project,” Sandia National Laboratories, Tech. Rep.Technical Report SAND2002-3613, November 2002.

[4] S. Antonatos, P. Akritidis, E. Markatos, and K. Anagnostakis, “Defending against hitlist worms using network address space randomization,” Comput. Netw., vol. 51, no. 12, pp. 3471–3490, Aug. 2007.

[5] M. D. Compton, “Improving the quality of service and security of military networks with a network tasking order process,” Ph.D. dissertation, Air Force Institute of Technology, 2009.

[6] E. Al-Shaer, “Toward network configuration randomization for moving target defense,” in Moving Target Defense, ser. Advances in Information Security, S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds. Springer New York, 2011, vol. 54, pp. 153–159.

[7] PaX Team. PaX Homepage. http://pax.grsecurity.net/, 2000.

[8] Elena Gabriela Barrantes, David Ackley, Stephanie Forrest, Trek Palmer, Darko Stefanovic and Dino Dai Zovi. “Intrusion Detection: Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks”. In 10th ACM Conference on Computer and Communications Security (CCS), 2003.

[9] Cristian Cadar, Periklis Akritidis, Manuel Costa, Jean-Phillipe Martin and Miguel Castro, “Data Randomization. Technical Report TR-120-2008”, Microsoft Research, 2008.

[10] Schmerl B, Camara J, Moreno G, et al. “Architecture-Based Self-Adaptation for Moving Target Defense” CMU-ISR-14-109[J]. 2014.

[11] Seungwon Shin, Zhaoyan Xu, Guofei Gu. "CloudRand: Building Heterogeneous and Moving-target Port Interfaces for Networked Systems." Technical Report, Department of Computer Science & Engineering, Texas A&M University, 2011,

[12] Deloach S A, Ou X, Zhuang R, et al. Model-driven, Moving-Target Defense for Enterprise Network Security[M]// Models@run.time. 2014: 137-161.

[13] Zhuang R, Deloach S A, Ou X. A model for analyzing the effect of moving target defenses on enterprise networks[C]// Cyber and Information Security Research Conference. ACM, 2014:73-76.

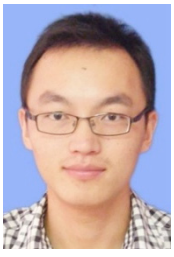
[14] Peng W, Li F, Huang C T, et al. A moving-target defense strategy for Cloud-based services with heterogeneous and dynamic attack surfaces[C]// IEEE International Conference on Communications. IEEE, 2014:804-809.

[15] David Evans, Anh Nguyen-Tuong, John Knight, Effectiveness of Moving Target Defenses, 2011.

吴承荣 于 2011 年在复旦大学计算机应用技术专业获得博士学位。现任复旦大学计算机科学技术学院副教授, 研究领域为网络与信息安全。研究兴趣包括: 云计算、大数据。Email: cwu@fudan.edu.cn

金嵩林 于 2014 年在华东理工大学大学计算机科学与技术专业获得学士学位。现在复旦大学计算机应用技术专业攻读硕士学位。研究领域为移动目标防御。研究兴趣包括: 信息安全、SDN。Email: hljin14@fudan.edu.cn

严明 于 2008 年在复旦大学软件工程专业获得硕士学位。现在复旦大学计算机应用专业攻读博士学位/现任复旦大学网络信息安全审计与监控工程研究中心工程师。研究领域为高速网络、云计算、网络安全。研究兴趣包括: 主动防御机制、高速网络数据处理。Email: myan@fudan.edu.cn



刘巍 于 2014 年在河南大学计算机科学与技术专业获得学士学位。现在复旦大学计算机应用技术专业攻读硕士学位。研究领域为拟态安全。研究兴趣包括: 网络安全、虚拟网络。Email: 14210240047@fudan.edu.cn

张世永 复旦大学计算机科学技术学院教授。研究领域为计算机网络、信息安全、数据通信、协议测试、电子商务等。Email：szhang@fudan.edu.cn

曾剑平 于 2006 年在厦门大学凝聚态物理专业获得博士学位。现任复旦大学副教授。研究领域为内容安全、大数据安全、社交媒体挖掘与应用。研究兴趣包括：互联网内容分析技术、网络用户行为分析、金融类社交媒体挖掘技术等。Email：zjp@fudan.edu.cn