## 论文拟解决的问题

### 研究背景

容器技术的优势与广泛应用

1. 生产环境中的部署优势：容器编排工具（如Docker、Kubernetes）的出现，极大地方便了容器化应用的部署、扩展和管理，使得容器在生产环境中的应用日益增多，云厂商也纷纷提供容器服务（如Amazon Fargate、Microsoft Azure Kubernetes Service）。

2. 资源受限平台的吸引力：作为一种轻量级的操作系统级虚拟化技术，容器机制对资源的需求相对较低，在资源受限的移动平台上具有很大优势，促使多种基于容器的Bring Your Own Device（BYOD）解决方案被提出和部署。

容器安全问题的凸显

1. 内核共享带来的风险：所有容器共享同一Linux内核，一旦内核被攻破，容器机制提供的隔离将完全失效，这成为容器进一步推广应用的主要障碍。

2. 现有研究的局限性：尽管已有部分研究从系统架构或设计原理等层面评估容器安全，但由于缺乏基于真实exploit的测量研究，难以准确反映容器在现实世界中的安全性。同时，漏洞数据库中针对容器的exploit数量较少，也给全面评估容器机制的安全性带来挑战。

### 问题

论文围绕Linux容器安全展开研究，主要拟解决的问题包括：

1. 缺乏真实攻击场景下容器安全的系统评估

容器技术在工业界应用广泛，但现有研究多从系统架构或设计原理层面评估容器安全，缺乏使用真实漏洞利用（exploit）对容器安全进行测量研究，难以确切知晓容器在实际中的安全性。

由于漏洞数据库中针对容器的exploit数量有限，难以基于此对容器机制进行有说服力的安全评估。

2. 容器机制对内部程序的安全增强效果不明确

不清楚容器机制在实际应用中是否能有效增强内部程序的安全性，抵御外部攻击。

3. 容器隔离机制对Linux内核保护能力的评估

需确定容器机制在保护Linux内核方面的隔离强度，尤其是面对特权提升攻击时的有效性。

4. 内核安全机制在防止特权提升攻击中的作用及关系不明

要深入分析内核安全机制（如Capability、Seccomp、MAC等）在防止特权提升攻击中的作用，以及它们与容器隔离机制（如Namespace、Cgroup）之间的关系，明确各机制对抵御此类攻击的贡献程度。

探究这些安全机制之间相互依赖、相互影响的关系对保护能力的影响，例如不当配置是否会削弱其防护能力。

5. 缺乏针对特权提升攻击的有效防御机制

在发现现有容器机制存在安全漏洞，特别是特权提升攻击可能突破容器隔离的情况下，需要提出有效的防御机制来应对这些攻击，保障容器安全。

## 论文提出的核心方法

论文中提出的核心方法主要围绕Linux容器安全评估与防御展开，具体包括以下几个方面：

**攻击数据集构建与分类**

1. exploit收集：基于容器主要受Linux内核集成安全机制保护这一观察，从Exploit-db收集可能对容器平台有效的exploit。先获取各类别中针对Linux平台的最新100个exploit（共400个），再筛选出可能在容器平台有效的223个exploit，收集其相关信息。

2. 攻击分类方法：提出二维攻击分类法，从攻击后果（敏感信息泄露、远程控制、拒绝服务、特权提升）和影响范围（web应用层、服务器层、库层、内核层）两个维度对收集的exploit进行分类，便于后续安全分析。

**容器安全评估实验**

1. 实验设置：通过比较exploit在原始Linux平台和容器平台上的执行结果来评估容器机制安全性。针对攻击用户空间程序的exploit，在容器外运行exploit攻击容器内的易受攻击程序；针对攻击Linux内核的exploit，在容器内执行exploit攻击部署的易受攻击内核。实验采用支持多种安全机制的Docker 17.09.1-ce构建容器，并根据exploit需求选择合适Linux发行版。

2. 结果分析

总体效果评估：从223个exploit中筛选出88个典型exploit进行测试，发现56.82%的exploit在容器平台默认配置下仍有效，表明容器对内部程序安全增强有限。

不同类型攻击分析

敏感信息泄露：“web应用层”和“服务器层”的此类攻击可通过构造HTTP请求或响应获取信息且未被容器机制阻止，“内核层”攻击因需特定能力而失败。

远程控制：容器机制无法阻止通过注入、构造请求或响应等方式的远程控制攻击。

拒绝服务：多数服务器和库的此类攻击未被容器机制阻止，原因是Docker默认未通过Cgroup机制限制容器资源使用；部分攻击可导致内核恐慌或应用崩溃，少数“内核层”攻击因Namespace隔离、能力不足和Seccomp禁止的API而被阻止。

特权提升：主要因Docker默认给予容器内进程的能力受限而失败，多数攻击只能获得14种能力而非全部38种（ROOT特权）。

3. 特权提升攻击深度分析

安全机制作用分析

CPU保护机制：KASLR和SMAP&SMEP通常可被容器内攻击绕过，如通过dmesg命令获取内核基地址绕过KASLR，利用漏洞覆盖内核数据结构指针或寻找可利用gadgets来禁用SMAP&SMEP。

容器机制：Namespace和Cgroup机制可阻止约21.62%的特权提升攻击，如mnt Namespace可阻止修改与超级用户账户相关文件的攻击，Cgroup机制可阻止访问特定设备的攻击。

内核安全机制：Capability、Seccomp和MAC等机制在默认配置下可阻止约86.49%的攻击，如某些攻击因需特定能力而被Capability机制阻止，部分系统调用因Seccomp策略被禁止而使攻击失败，某些攻击因MAC机制阻止挂载文件系统而失败。

安全机制间关系探讨：各安全机制并非独立工作，相互之间存在影响，如Namespace和Capability政策配置不当可能使对方失效，Seccomp和MAC机制配置不当也可能影响防御效果，某些攻击需Namespace和Cgroup机制协同配置才能被阻止。

**针对特权提升攻击的防御机制**

1. 攻击模型提取：分析仍能在容器平台有效提升特权的11个exploit，发现它们遵循一个4步攻击模型，包括绕过KASLR机制、利用内核漏洞覆盖内核函数指针、禁用SMEP&SMAP机制、调用commit\_creds()获取ROOT特权。

2. 防御系统设计：鉴于容器内进程通常无需ROOT特权，通过修改commit\_creds()函数实现防御。在更新进程凭据前，检查调用进程是否在容器内，若在容器内且为特权提升操作，则阻止更新。具体通过比较进程的nsproxy字段与init\_nsproxy判断进程位置，比较commit\_creds()输入参数和当前进程real\_cred判断是否为特权提升操作。

3. 防御效果与性能评估：实验表明，在为容器内进程启用“CAP\_NET\_ADMIN”能力的情况下，部署防御系统后，11个exploit均无法实现特权提升，证明防御系统有效。同时，通过UnixBench工具评估，防御系统引入的开销可忽略不计。

## 论文贡献总结

该论文在Linux容器安全研究方面做出了多方面的重要贡献，具体总结如下：

1. 构建攻击数据集与分类体系

手动创建了包含223个针对容器解决方案exploit的数据集，为后续研究提供了丰富的数据基础。

提出二维攻击分类法，从攻击后果（敏感信息泄露、远程控制、拒绝服务、特权提升）和影响范围（web应用层、服务器层、库层、内核层）两个维度对exploit进行分类，有助于深入理解容器面临的安全威胁。

2. 系统评估容器安全

通过在原始Linux平台和容器平台手动执行88个典型exploit，评估现有容器安全机制的有效性，发现超过一半（56.82%）的exploit在容器平台默认配置下仍有效，明确了容器在实际应用中的安全状况。

3. 深入分析特权提升攻击

详细分析了特权提升攻击，特别是内核安全机制（Capability、Seccomp、MAC）、容器机制（Namespace、Cgroup）和CPU保护机制（KASLR、SMAP、SMEP）在防止特权提升攻击中的作用。

揭示了各安全机制之间相互依赖、相互影响的关系，以及不当配置可能削弱防护能力的问题，为优化容器安全配置提供了理论依据。

4. 提出特权提升攻击防御方案

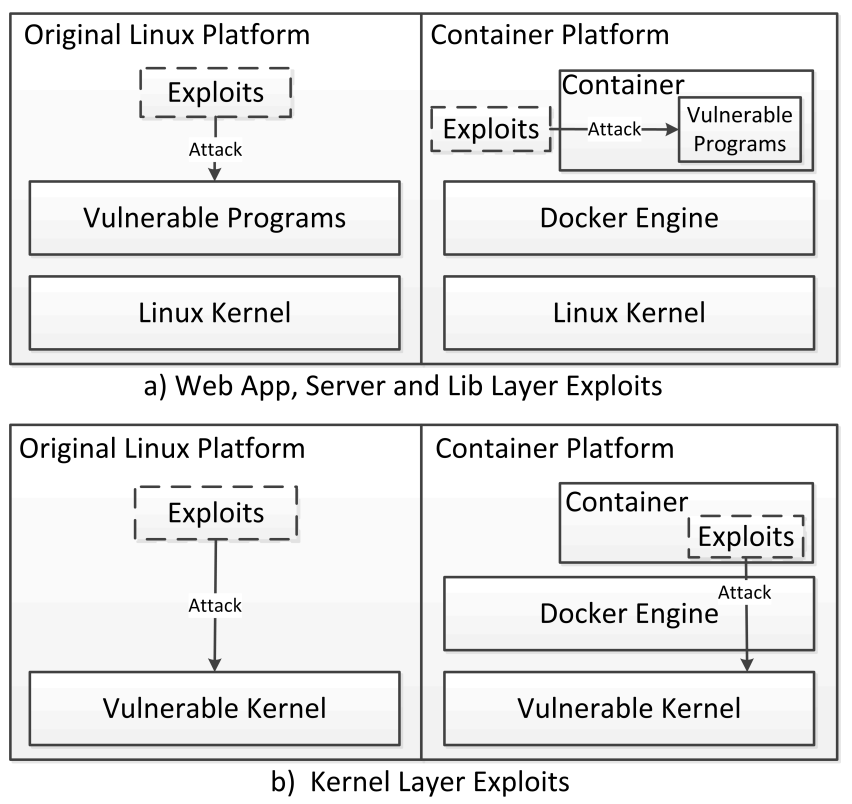
提取出内核特权提升攻击的通用4步模型，包括绕过KASLR、利用内核漏洞覆盖指针、禁用SMEP&SMAP、调用commit\_creds()获取ROOT特权。

基于攻击模型，提出通过修改commit\_creds()函数来阻止容器内特权提升攻击的防御机制，实验证明该机制能有效阻止11个相关exploit，且开销可忽略不计，为保障容器安全提供了有效的解决方案。

## 论文实验设计总结与分析

该论文的实验设计全面且系统，主要目的是评估Linux容器机制的安全性，特别是针对各种类型攻击的抵御能力，以及分析不同安全机制在其中的作用。实验设计的关键步骤和要点如下：

**实验设置**



实验设置框架

1. 平台选择与配置

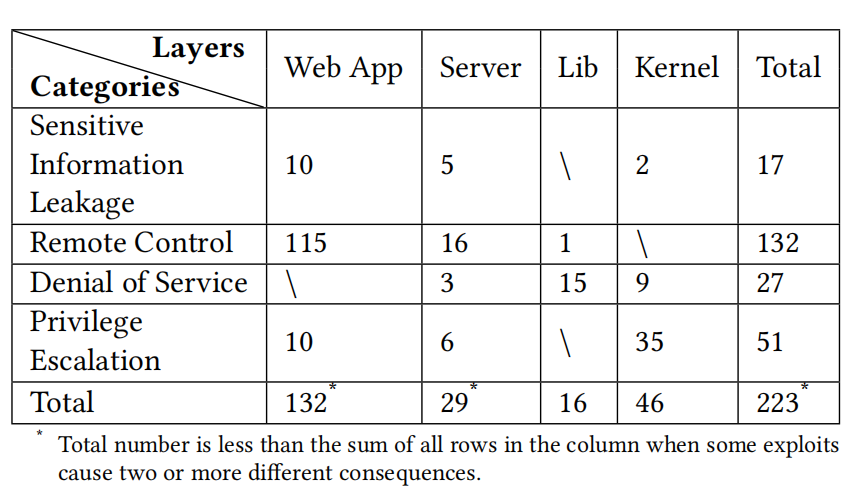
容器平台：使用Docker 17.09.1-ce构建容器，该版本支持Linux内核安全机制（如Seccomp、Capability和MAC）。对于一些需要较低版本Linux发行版的exploit，手动安装Docker 17.09.1-ce以确保实验环境的一致性。

Linux发行版选择：当一个exploit在多个Linux发行版上有效时，优先选择支持更多Linux安全机制的发行版。例如，对于某个exploit在Ubuntu 14.04和15.10上均有效，但只有Ubuntu 14.04上安装的Docker支持Seccomp机制，此时选择Ubuntu 14.04进行实验。

2. exploit选择

从收集的223个可能对容器平台有效的exploit中筛选出88个用于实验。首先排除在原始Linux平台上失败的41个exploit，失败原因主要包括目标旧版本程序不可用或exploit代码执行失败。

对于同一层（除特权提升exploit外）且攻击原理相似的exploit，仅选择一个作为代表，以避免冗余，最终确定了88个具有代表性的exploit进行实验评估。



Exploit Dataset

**实验方法**

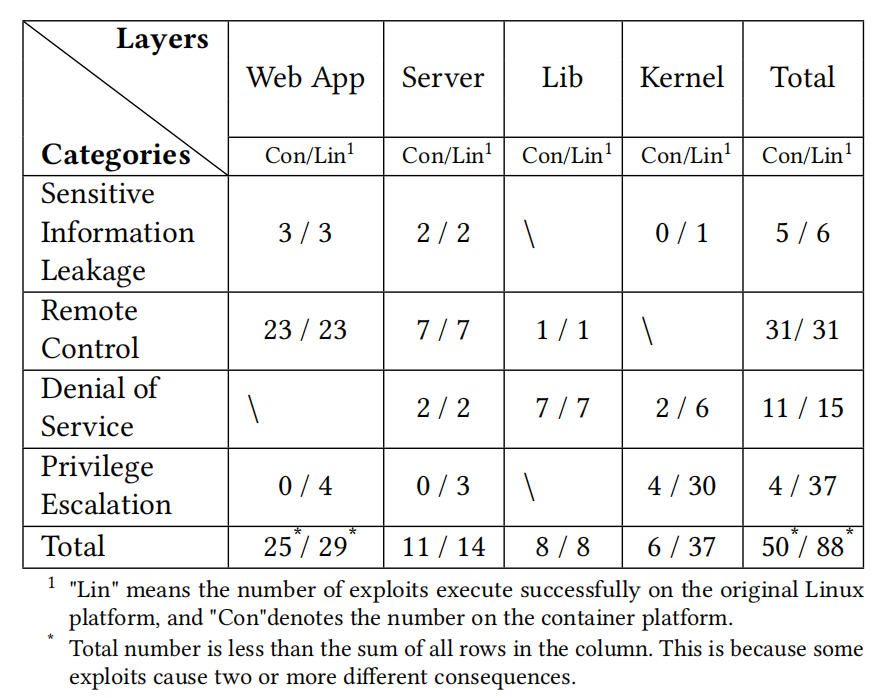
1. 针对不同类型exploit的测试方法

攻击用户空间程序的exploit（web应用层、服务器层和库层）：将易受攻击的程序部署在容器内，从容器外（控制主机）运行exploit，以此评估容器是否为内部程序提供了额外的安全增强。

攻击Linux内核的exploit（内核层）：在容器内执行exploit攻击部署的易受攻击内核，从而评估容器对Linux内核的隔离保护能力。

2. 对比分析：通过对比exploit在原始Linux平台和容器平台上的执行结果，评估容器机制对各类攻击的防御效果。

**实验结果分析**



安全评估结果概述

1. 总体结果评估：在88个exploit中，50个（56.82%）在容器平台的默认配置下仍然有效，表明容器在抵御外部攻击方面存在一定局限性，对内部程序的安全增强效果并不理想。

2. 不同类型攻击结果分析

敏感信息泄露：“web应用层”和“服务器层”的此类exploit通过构造HTTP请求或响应获取敏感信息，未被容器机制有效阻止；“内核层”exploit因需特定能力（如CAP\_SYS\_ADMIN）而失败，该能力在容器内默认被移除。

远程控制：容器机制未能阻止通过注入（如XSS、SQL注入等）和构造请求/响应（如CSRF）的远程控制攻击。

拒绝服务：多数针对服务器和库的此类攻击未被容器机制阻止，原因是Docker默认未有效利用Cgroup机制限制容器资源使用。在手动限制资源后，部分攻击被成功阻止。此外，部分“内核层”攻击因Namespace隔离、能力不足和Seccomp禁止的API而被阻止。

特权提升：主要由于Docker默认给予容器内进程的能力受限，多数此类攻击只能获得14种能力而非ROOT特权（38种能力）。

3. 特权提升攻击深度分析

安全机制作用分析

CPU保护机制：KASLR和SMAP&SMEP通常可被容器内攻击绕过，攻击者可通过多种方式（如利用dmesg命令获取内核基地址、利用漏洞覆盖内核数据结构指针等）实现。

容器机制：Namespace和Cgroup机制可阻止约21.62%的特权提升攻击，例如通过隔离文件系统或限制设备访问来阻止相关攻击。

内核安全机制：Capability、Seccomp和MAC等机制在默认配置下可阻止约86.49%的攻击，但各机制之间存在相互影响，不当配置可能降低防护能力。

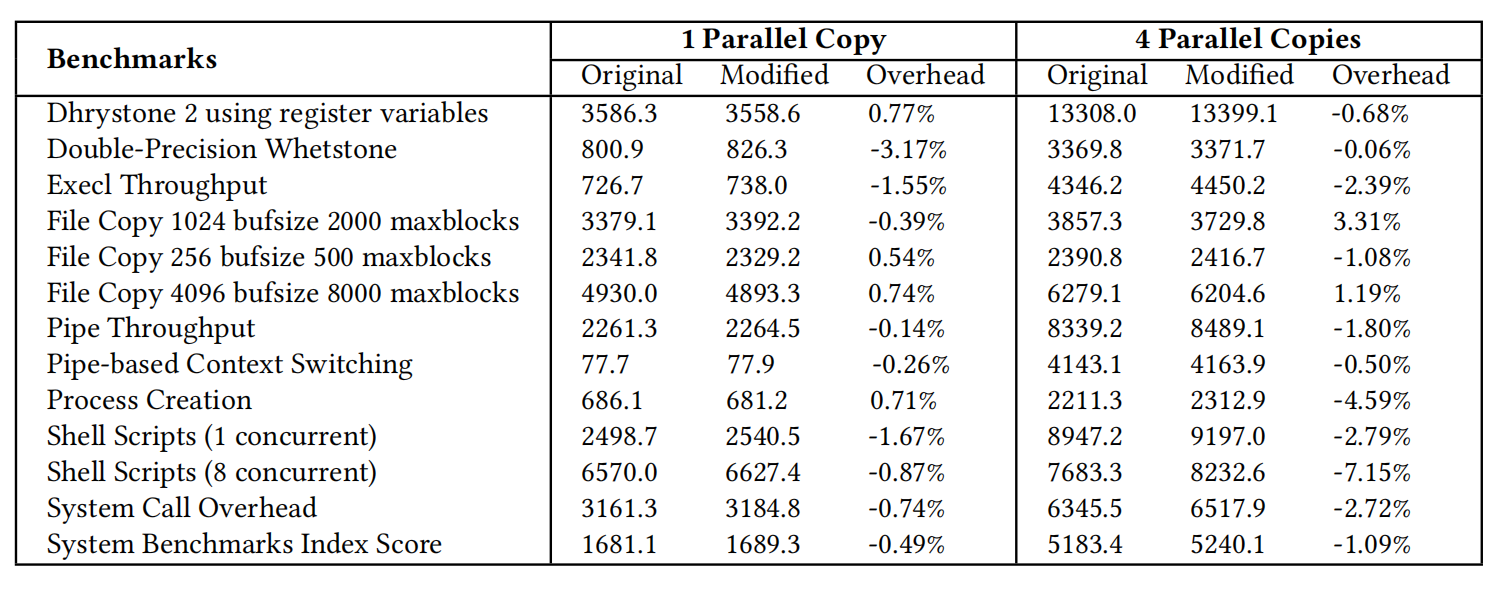
安全机制间关系探讨：各安全机制并非独立工作，如Namespace和Capability政策配置不当可能相互影响防御效果，Seccomp和MAC机制配置不当也可能导致攻击突破防御，某些攻击需Namespace和Cgroup机制协同配置才能被有效阻止。

**防御机制实验**

1. 防御系统设计：基于对特权提升攻击模型的分析，提出通过修改commit\_creds()函数来阻止特权提升攻击的防御机制。在更新进程凭据前，检查调用进程是否在容器内，若在容器内且为特权提升操作，则阻止更新。

2. 防御效果验证：在为容器内进程启用“CAP\_NET\_ADMIN”能力的情况下，测试防御系统对11个可成功实现特权提升的exploit的防御效果。实验结果表明，部署防御系统后，这些exploit均无法实现特权提升，证明了防御系统的有效性。

3. 性能评估：使用UnixBench工具在特定配置的虚拟机（VMWare虚拟机器，Ubuntu 16.04 LTS AMD 64 OS，四核2.80GHz CPU和2GB内存）上评估防御系统的性能开销。结果显示，防御系统引入的开销可忽略不计，对系统性能影响较小。



防御系性能

## 论文阅读心得

在阅读了Xin Lin等人撰写的《Linux容器安全的测量研究：攻击与对策》这篇论文后，我对Linux容器安全有了更深入且系统的认识，以下从几个方面说说我的心得体会。

**研究的深度与创新性**

论文在Linux容器安全领域进行了全面且深入的研究，其创新性体现在多个方面。作者通过收集大量真实的exploit并构建二维分类体系，从实际攻击角度对容器安全进行评估，填补了以往研究缺乏真实场景测量的空白。在分析特权提升攻击时，深入探讨了各安全机制间的复杂关系，这为后续研究提供了新的视角。尤其是针对发现的问题提出的防御机制，简单而有效，对保障容器安全具有重要意义。

**实验设计的严谨性**

实验设计是论文的一大亮点。作者精心规划了实验设置，包括选择合适的容器平台（Docker 17.09.1-ce）及Linux发行版，确保了实验环境的代表性和一致性。在exploit选择上，经过严格筛选，排除无效和冗余的exploit，保证了实验结果的准确性和可靠性。通过对比exploit在原始Linux平台和容器平台的执行结果，清晰地呈现了容器安全机制的实际效果，为后续分析提供了坚实的数据基础。

**安全机制的深入剖析**

论文对容器涉及的多种安全机制进行了详细分析，使我深刻认识到它们在容器安全中的重要作用及相互关系。例如，内核安全机制（Capability、Seccomp、MAC）在防止特权提升攻击方面发挥着关键作用，但它们之间的相互依赖和影响也可能导致“短板效应”。容器机制（Namespace、Cgroup）在资源隔离和部分攻击防御上有一定成效，但默认配置下的Cgroup机制在防范拒绝服务攻击方面存在不足。CPU保护机制（KASLR、SMAP、SMEP）虽为第一道防线，但易被攻击者绕过。这些分析有助于理解容器安全的复杂性，为安全配置和防护策略的制定提供了重要参考。

**对实际应用的指导意义**

研究成果对容器技术在实际生产环境中的应用具有重要的指导价值。企业在使用容器部署应用时，应充分认识到容器安全机制的局限性，尤其是特权提升攻击带来的风险。基于论文的研究，可采取相应措施，如合理配置各安全机制参数，避免因不当配置削弱防护能力；关注内核漏洞，及时更新内核以防止攻击者利用漏洞突破容器隔离；采用有效的防御机制，如论文中提出的针对commit\_creds()函数的改进措施，增强容器对特权提升攻击的抵御能力。

**研究的局限性与展望**

尽管论文做出了诸多贡献，但也存在一定局限性。例如，提出的防御机制虽能有效应对当前的11种特权提升攻击，但无法涵盖所有潜在的类似攻击。正如作者在讨论部分所述，攻击者可能通过直接修改内核数据或利用其他内核函数发起攻击，未来需要在Linux内核之外探索更全面的防御解决方案，如基于虚拟机监控器或硬件的保护机制。这也为后续研究指明了方向，期待在容器安全领域能有更多深入的研究成果出现，以应对日益复杂的安全挑战。

总体而言，这篇论文为Linux容器安全研究提供了丰富的信息和深刻的见解，对相关领域的研究人员、安全从业者以及容器技术的使用者都具有重要的参考价值。