



基于信创环境的网络安全测试验证靶场平台 技术报告

参赛单位：武汉大学国家网络安全学院

参赛人员：钱一铭 杜泓波 赵攀 秦天杰 关皓匀
张须彭 何秉坤 李奕璇 钱嘉乐 潘鸿远

指导老师：张立强 文贝西

目录

1 概述	1
1.1 研究背景	1
1.2 国内外研究背景	1
1.3 知识背景	3
1.3.1 网络靶场与虚拟化仿真	3
1.3.2 资源隔离机制	5
1.3.3 网络仿真	6
1.3.4 漏洞挖掘	8
2 研究要求	11
2.1 研究目标	11
2.2 研究内容	11
2.3 技术指标	11
2.4 考核方式	12
3 系统功能概述	13
3.1 网络靶场搭建	13
3.2 系统安全排查	14
3.3 web 服务漏洞	15
3.3.1 背景介绍	15
3.3.2 我们的工作	15
3.4 用户态程序漏洞挖掘	17
3.4.1 研究背景	17
3.4.2 AFL	17
3.4.3 我们的工作	18
3.5 内核态漏洞挖掘	18
3.5.1 内核漏洞	18
3.5.2 Syzkaller	19
3.5.3 我们的工作	20
3.6 容器逃逸风险检测	21
3.6.1 docker 技术与 namespace	21
3.6.2 docker 逃逸	22
3.6.3 我们的工作	23
3.7 展示系统设计	23
4 技术方案	25
4.1 网络验证靶场设计	25
4.2 主机信息扫描与收集	26
4.3 网络资产测绘	26

4.4	web 服务漏洞	27
4.5	内核漏洞测评	28
4.5.1	基于版本的内核 cve 初探	28
4.5.2	内核 cve 漏洞库构建	28
4.5.3	内核 poc 验证	29
4.5.4	可疑内核模块的识别与测试	29
4.6	用户态程序漏洞挖掘	30
4.6.1	机器学习模型辅助逆向	30
4.6.2	大语言模型构建输入	31
4.6.3	fuzzer 的选择	32
4.7	容器逃逸风险评估	33
4.7.1	容器权限检查	33
4.7.2	漏洞数据库构建	33
4.8	系统评估标准建模	34
4.8.1	测评结果统一化	34
4.8.2	指标评估框架	35
4.9	系统自动化防护技术实现	36
5	主要成果和创新点	37
5.1	信创环境搭建与系统部署	37
5.1.1	网络拓扑结构	37
5.1.2	环境中靶机情况介绍	37
5.1.3	环境中网络设备介绍	38
5.1.4	验证靶场搭建	39
5.2	安全总结报告	40
5.3	防护成效	41
5.4	创新点总结	42
6	预期应用与总结	47
6.1	预期应用	47
6.2	总结	47
6.2.1	存在问题	47
6.2.2	未来工作建议	47
7	附件说明	49
7.1	源代码结构说明	49
7.2	系统部署说明	50

1 概述

内容提要

研究背景

网络安全

新创环境

验证靶场

1.1 研究背景

党的十八大以来，以习近平同志为核心的党中央高度重视网络安全工作，特别在目前日趋复杂的背景下，深刻认识和有力防范网络安全风险，切实维护网络空间安全，已成为事关全局的重大课题。习近平总书记对此多次发表重要讲话、作出重要指示批示，指出“网络安全为人民，网络安全靠人民，维护网络安全是全社会共同责任，需要政府、企业、社会组织、广大网民共同参与，共筑网络安全防线”。

在经济社会日常运行中，为避免因网络攻防演练和安全测试对业务系统造成的负面影响，要求基于国产操作系统等信创环境，利用虚拟仿真技术搭建网络安全测试验证环境，实现安全配置核查、安全策略评估、网络安全设备接入验证等风险评估分析功能。

在此背景下，网络测试验证靶场作为网络安全测试和评估的重要平台，具有重要的现实意义。传统的网络靶场主要依赖于国外软硬件产品和技术，无法完全满足信创环境下的信息安全需求。基于信创环境的网络测试验证靶场，则是针对这一现状所提出的新型解决方案，旨在构建一个完全自主可控的网络安全测试平台，全面验证和评估信创产品的安全性、稳定性和性能。



图 1: 网络安全

1.2 国内外研究背景

随着国家对信息技术自主创新的重视，信创环境建设在国内得到了政策和资金的强力支持。例如，《网络安全法》《密码法》等法律法规为信创产业的发展提供了制度保障。

许多省市出台了相关政策，鼓励本地企业和机构进行信创项目的开发和应用，推动形成安全可靠的信息技术生态体系。国内科研机构和高校在信创环境下的网络安全研究方面投入了大量资源。例如，中国科学院、清华大学、北京大学等机构在信创环境中的漏洞挖掘、攻击检测和防御技术方面取得了显著进展。

安全企业如启明星辰、天融信等积极参与信创环境的建设，开发了许多适用于信创环境的安全产品和解决方案。各大网络安全公司和科研机构建立了多个信创环境的网络安全靶场，用于模拟真实的攻击和防御场景。例如，国家互联网应急中心（CNCERT）和国家计算机网络与信息安全管理中心（CNITSEC）等机构已经建设了多个网络安全测试验证平台。这些靶场被用于安全产品的验证、网络安全培训以及攻防演练，帮助提升了整体网络安全防护能力。

此外，随着人工智能、云计算、大数据等新兴技术的发展，信创环境的网络安全研究也逐渐向智能化和自动化方向转变。国内外研究者纷纷探索如何利用这些新技术来增强信创环境的安全防护能力。例如，通过机器学习算法分析网络流量，自动检测异常行为；利用大数据技术进行安全态势感知，实时监控和预警安全威胁。这些研究不仅提升了信创环境的安全水平，也为未来的网络安全技术发展提供了新的思路和方向。



图 2: 国内知名安全企业

当前，依托于虚拟化技术实现网络靶场的 CTF (Capture The Flag) 竞赛在全球范围内蓬勃发展，成为网络安全领域的一项重要赛事。CTF 竞赛通过设置各种网络安全挑战，如漏洞挖掘、逆向工程、密码破解、取证分析等，旨在考察参赛者的综合技术和问题解决能力。比赛通常分为 Jeopardy (解题模式) 和 Attack-Defense (攻防模式) 两种形式，前者注重解答一系列独立的题目，后者则模拟真实网络环境中的攻防对抗。CTF 竞赛不仅为参赛者提供了一个展示和提升技能的平台，也促进了全球网络安全技术的发展和交流。

AWD (Attack With Defense) 竞赛作为一种新兴的网络安全比赛形式，其需要更加复杂的网络虚拟环境，这对网络靶场的构建的技术要求也与日俱增，该类赛事近年来也逐渐受到关注。AWD 竞赛将参赛者分为多个团队，要求他们在维护自身系统安全的同时，对其他团队发起攻击。比赛强调攻防兼备，参赛者需要在确保自我系统不被攻破的前提下，通过发现和利用对手系统中的漏洞获取积分。AWD 竞赛不仅考验参赛者的技术能力，还需要他们具备战略思维和团队协作精神。随着网络安全威胁的日益复杂，AWD 竞赛提供了一个模拟真实攻防场景的环境，帮助参赛者积累实战经验，提升综合防御能力。



图 3: 常见网络攻防赛事

国外在信创环境下的网络安全研究起步较早，形成了较为成熟的技术体系。欧美国家在网络安全技术、标准和框架方面具有较强的创新能力和市场主导权。例如，美国的 NIST（国家标准与技术研究院）发布了多项关于网络安全的标准和指南，欧洲也通过 ENISA（欧洲网络和信息安全局）推动网络安全技术和标准的研究与应用。

国外许多研究机构和企业已经建立了先进的网络安全靶场，用于模拟复杂的网络攻击和防御场景。例如，美国的 Cyber Range 和以色列的 CyberGym 等。

这些靶场被广泛用于政府、军队、企业和教育机构的网络安全培训、产品测试和攻防演练，帮助用户提升网络安全防护能力。国外在信创环境的研究中注重国际合作与交流，通过联合研究和项目合作，共同提升网络安全技术水平。例如，欧盟的 Horizon 2020 计划中有多个项目涉及网络安全靶场的研究与应用。跨国企业和科研机构之间的合作也非常活跃，通过技术共享和资源整合，共同应对复杂的网络安全挑战。



图 4：国际网络安全形势图

1.3 知识背景

1.3.1 网络靶场与虚拟化仿真

网络靶场是一种模拟真实网络环境的测试平台，通常用于网络安全训练、攻防演练、漏洞测试等多种应用场景。在网络靶场中，安全研究人员可以搭建并测试各种网络架构、攻击手段和防御策略，而不必担心对真实生产环境造成影响。网络靶场的关键在于其能够提供一个真实感极强的网络环境，使得训练和测试具有高保真度。

虚拟化技术在构建网络靶场中扮演着至关重要的角色。虚拟化技术通过在物理硬件上创建多个虚拟机（VM），每个虚拟机可以独立运行操作系统和应用程序，从而有效利用物理资源，提高系统的灵活性和可管理性。

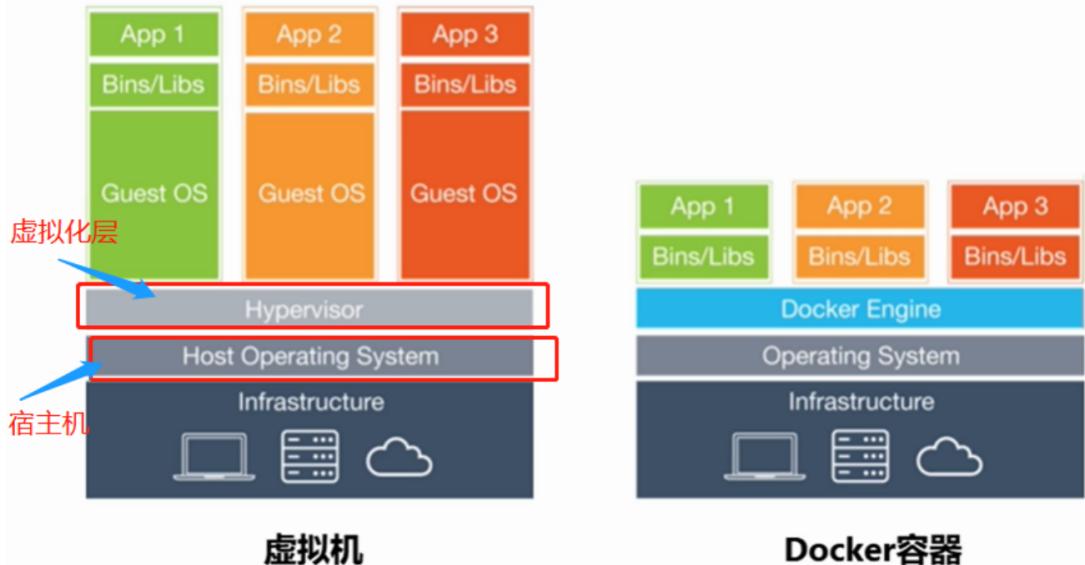


图 5: 虚拟机与容器示意图

在网络靶场中，虚拟化技术主要体现在以下几个方面：

资源隔离与管理：虚拟化技术允许多个虚拟机在同一物理服务器上运行，这不仅节约了硬件资源，还实现了不同实验环境的资源隔离。通过虚拟化管理平台，管理员可以轻松创建、删除和配置虚拟机，快速搭建不同的网络实验环境。

快速部署与回滚：利用虚拟化技术，网络靶场可以实现虚拟机的快速部署与回滚。例如，在进行网络攻防演练时，管理员可以迅速部署一组虚拟机模拟受攻击的目标系统，并在演练结束后快速恢复到初始状态，方便重复测试和训练。

灵活的网络拓扑：虚拟化技术支持灵活的网络配置，管理员可以根据需求设计和调整虚拟网络的拓扑结构。这使得网络靶场能够模拟各种复杂的网络环境，包括局域网、广域网、云计算环境等，满足不同的测试需求。

环境可控性：通过虚拟化技术，网络靶场的所有虚拟机和网络配置都在可控范围内。管理员可以精确控制网络流量、攻击行为和防御措施，从而在实验过程中获得更精确的数据和结果。

成本效益：虚拟化技术降低了网络靶场的建设和维护成本。相比传统的物理设备，虚拟化技术所需的硬件投资较少，且易于扩展和管理，极大地提高了资源利用率和经济效益。

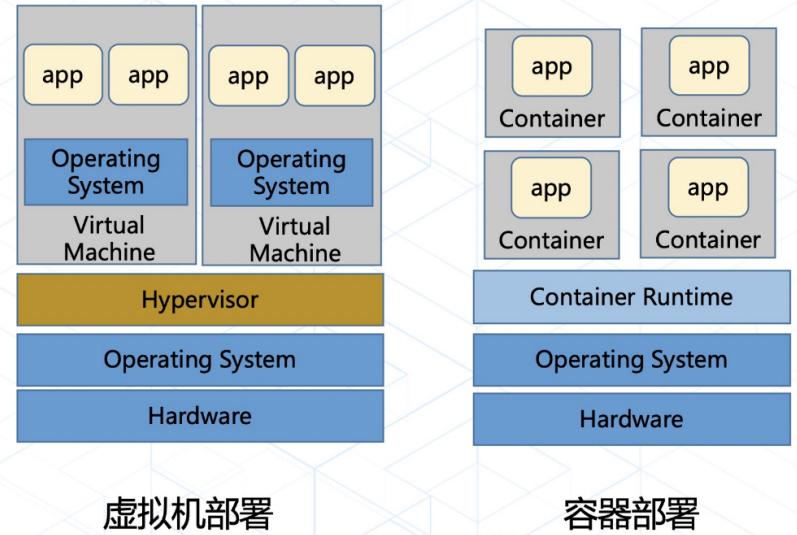


图 6: 虚拟机与容器的区别对比图

1.3.2 资源隔离机制

虚拟机中的资源隔离原理：虚拟机通过虚拟化技术，在单一物理硬件上运行多个独立的操作系统实例。主要的资源隔离机制包括：

硬件虚拟化：虚拟机管理程序（Hypervisor）或虚拟机监控器（VMM, Virtual Machine Monitor）负责将物理硬件资源（如 CPU、内存、存储、网络）虚拟化，并将其分配给各个虚拟机。常见的 Hypervisor 包括 VMware ESXi、Microsoft Hyper-V 和 KVM 等。

CPU 虚拟化：Hypervisor 创建虚拟 CPU (vCPU) 并分配给每个虚拟机，确保它们在物理 CPU 上的执行是独立的。硬件辅助虚拟化（如 Intel VT-x 和 AMD-V）进一步增强了这一隔离。

内存虚拟化：Hypervisor 分配独立的内存空间给每个虚拟机，并使用内存管理单元（MMU, Memory Management Unit）来隔离不同虚拟机的内存访问，防止内存泄露和非法访问。

I/O 虚拟化：Hypervisor 虚拟化物理 I/O 设备（如硬盘、网络接口），并通过设备虚拟化技术（如虚拟网络接口、虚拟硬盘文件）为虚拟机提供独立的 I/O 通道。

安全隔离：Hypervisor 在硬件和操作系统之间提供了一层额外的隔离，确保每个虚拟机的操作系统和应用程序互不干扰，提升了安全性。

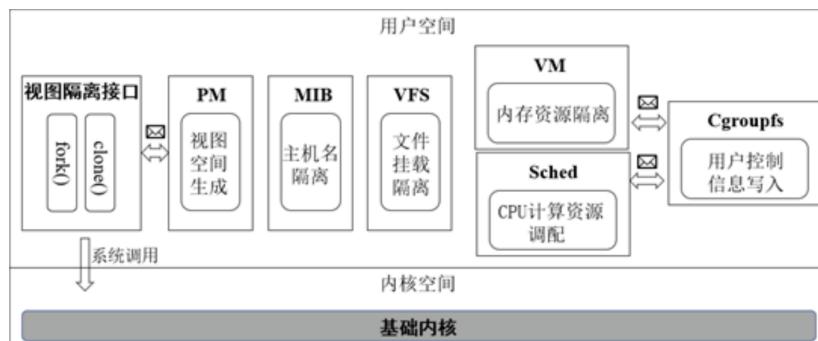


图 7: 资源隔离示意图

虚拟容器通过操作系统级虚拟化，在共享同一个操作系统内核的情况下，运行多个独立的用户空间实例。主要的资源隔离机制包括：

命名空间 (Namespaces): 命名空间是 Linux 内核提供的一种隔离机制，用于将全局系统资源（如进程 ID、主机名、用户 ID、文件系统、网络、IPC 等）划分为独立的区域。常见的命名空间包括：

PID 命名空间：隔离进程 ID
Net 命名空间：隔离网络设备、IP 地址等
Mount 命名空间：隔离文件系统挂载点
UTS 命名空间：隔离主机名和域名
IPC 命名空间：隔离进程间通信资源
User 命名空间：隔离用户和用户组
ID 控制组 (Cgroups): 控制组是一种内核功能，用于限制、隔离和管理系统资源（如 CPU、内存、块设备 I/O、网络带宽等）的使用。通过 cgroups，可以确保每个容器的资源使用互不干扰，防止资源争用。

文件系统隔离：容器使用联合文件系统（如 OverlayFS）来实现文件系统的隔离。每个容器有自己的文件系统视图，与宿主机和其他容器的文件系统独立。

安全增强 (Security Enhancements): 容器利用 Linux 安全模块（如 SELinux、AppArmor）和能力机制 (Capabilities) 来进一步增强安全隔离，限制容器内进程的权限，防止恶意操作。

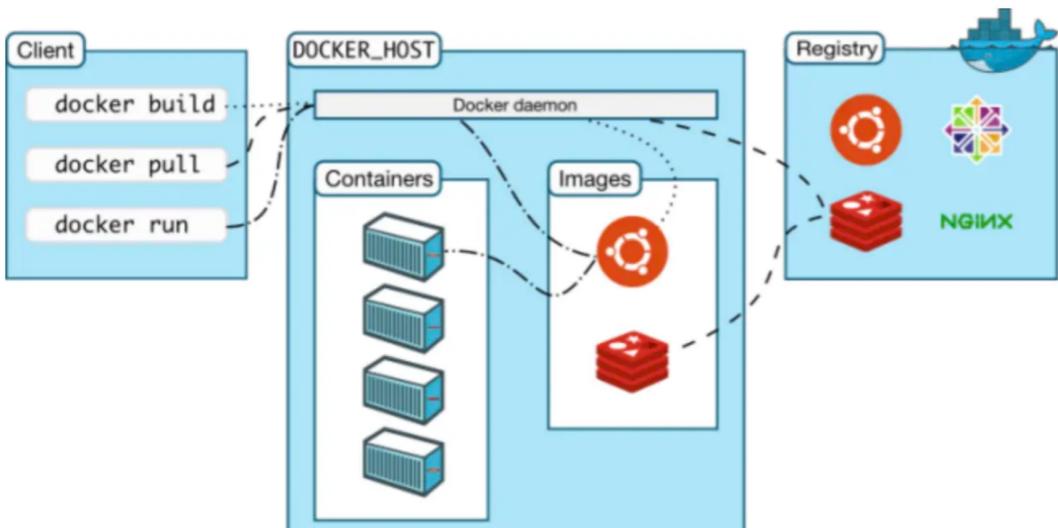


图 8: docker 架构解析图

1.3.3 网络仿真

在当前的虚拟化技术中，网络的仿真主要通过虚拟网络接口和虚拟交换机等机制实现。虚拟网络仿真旨在为虚拟机（VM）和容器（Container）提供隔离且灵活的网络连接，确保虚拟化环境中的网络性能和安全性。

虚拟网络接口

虚拟网络接口 (Virtual Network Interface, VNIC) 是网络仿真的基础组件。每个虚拟机或容器都会分配一个或多个 VNIC，这些接口与物理网络接口类似，能够发送和接收网络数据包。VNIC 通常使用以下技术进行仿真：

虚拟网桥 (Virtual Bridge): 虚拟网桥类似于传统的网络交换机，用于连接多个 VNIC，实现虚拟机或容器之间的网络通信。Linux 中的 bridge 工具和 Open vSwitch (OVS) 是常用的虚拟网桥实现。

TAP/TUN 设备: TAP (Ethernet Tap) 和 TUN (Network Tunnel) 设备是 Linux 内核提供的虚拟网络设备，TAP 用于仿真二层（数据链路层）网络设备，TUN 用于仿真三层（网络层）网络设备。这些设备能够在用户空间和内核空间之间转发数据包。

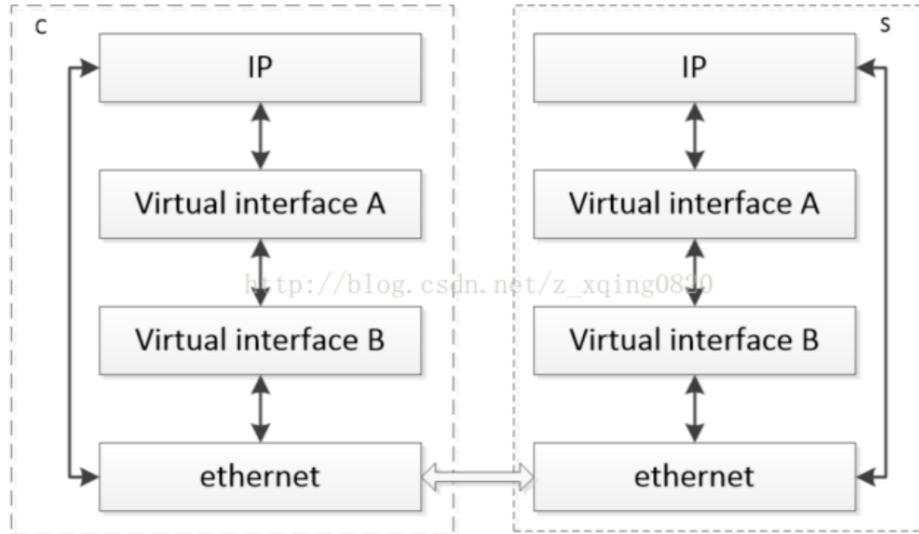


图 9: 虚拟网络接口原理图

虚拟交换机

虚拟交换机（Virtual Switch）在虚拟化网络中扮演关键角色，负责在虚拟机或容器之间转发数据包，并与物理网络连接。虚拟交换机的主要功能包括：

数据包转发：根据 MAC 地址或 IP 地址表，将数据包转发到正确的虚拟端口，确保虚拟机或容器之间的通信。

网络隔离：通过 VLAN (Virtual LAN) 和 VXLAN (Virtual Extensible LAN) 等技术实现网络隔离，防止不同租户或应用之间的网络流量干扰。

流量控制：支持流量控制和带宽管理，确保网络资源的公平分配和性能优化。

Open vSwitch (OVS) 是目前广泛使用的虚拟交换机实现，支持丰富的网络功能和灵活的配置。

虚拟路由器和虚拟防火墙

除了虚拟交换机，虚拟路由器和虚拟防火墙也是虚拟化网络仿真中的重要组件。虚拟路由器负责在不同子网或 VLAN 之间转发数据包，虚拟防火墙则用于实现网络访问控制和安全策略。常见的虚拟路由器和防火墙解决方案包括 VyOS、pfSense 等。

软件定义网络 (SDN)

软件定义网络 (Software Defined Networking, SDN) 在虚拟化网络中得到了广泛应用。SDN 通过将网络控制平面与数据平面分离，实现网络资源的集中管理和动态配置。SDN 控制器可以根据业务需求动态调整虚拟网络的拓扑结构、路由策略和安全规则，显著提高了网络的灵活性和可管理性。OpenFlow 是 SDN 常用的协议标准。

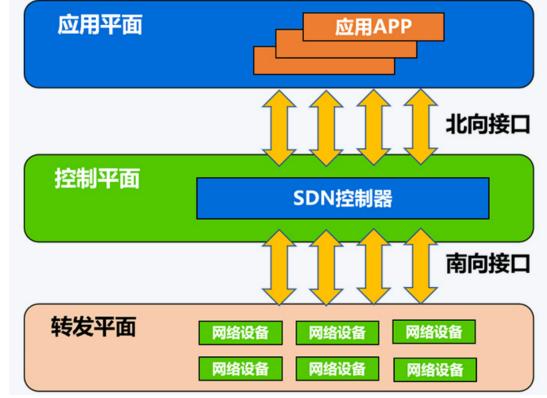


图 10: 软件定义网络原理图

网络功能虚拟化 (NFV)

网络功能虚拟化 (Network Functions Virtualization, NFV) 进一步推动了网络仿真技术的发展。NFV 将传统的网络设备功能（如路由器、防火墙、负载均衡等）虚拟化为软件模块，运行在标准的虚拟机或容器中。NFV 架构包括虚拟网络功能 (VNF)、NFV 管理与编排 (NFV MANO) 等组件，通过标准接口和协议实现网络功能的灵活部署和动态调整。

1.3.4 漏洞挖掘

模糊测试

模糊测试的核心思想是，根据一定的规则，自动或半自动生成的随机数据，然后将产生的数据输入到程序中，并监视程序是否有异常出现，以发现可能的程序错误，如内存泄漏、系统崩溃、未处理的异常等。当一个模糊测试生成器开始启动并运行后，它将自己寻找漏洞，并不需要人工干预，非常有助于发现传统测试方法或手动审计无法检测到的缺陷。模糊测试包括几个基本的测试步骤：确定被测系统-> 给定输入-> 生成测试用例-> 灌入用例进行测试-> 监控目标程序情况-> 输出崩溃日志。

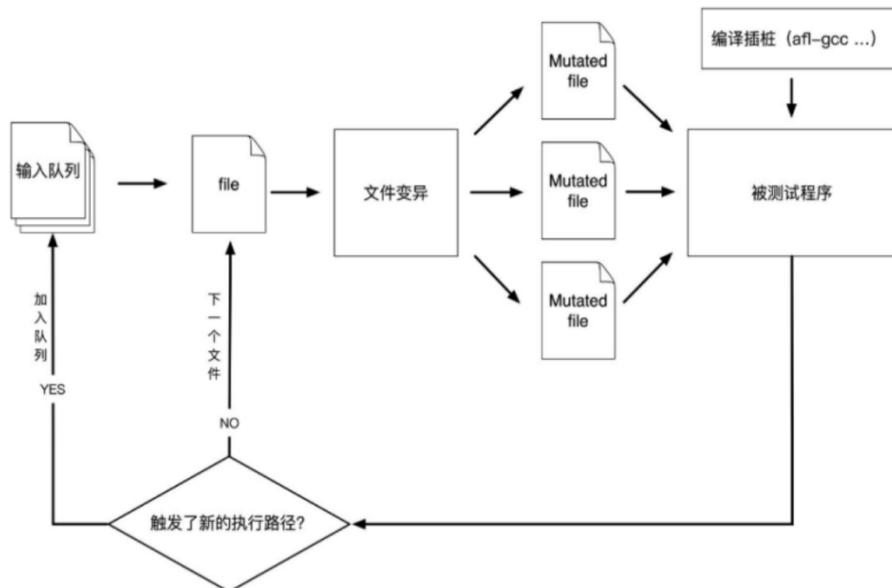


图 11: afl 流程图

变异策略

当前制约模糊测试效果最大因素就是生成输入的质量，而输入的质量又可以通过种子的变异策略来得到提升，我们总结了当前学术界常见的模糊测试变异策略：

基于覆盖率的指导变异：

覆盖率反馈（Coverage Feedback）策略是模糊测试中最广泛使用的策略之一。通过收集目标程序在处理输入时的代码覆盖率信息，指导模糊测试的输入生成过程。覆盖率反馈策略的基本思路是：初始阶段提供一组种子输入，模糊测试工具在执行这些输入时监控并记录程序的代码覆盖率。根据覆盖率信息，选择那些能够触发新的代码路径的输入进行优先变异，以此产生新的测试用例。这样，测试工具能够不断扩展代码覆盖范围，逐步探索程序的不同部分，从而发现潜在的缺陷和漏洞。

基于程序状态的指导变异：

程序状态反馈（Program State Feedback）策略通过监控程序在执行过程中的状态变化来指导变异过程。具体方法包括收集程序执行时的内存状态、寄存器值、数据流信息等，并将这些信息作为反馈，用于优化输入生成。通过分析这些状态变化，可以识别出哪些输入导致了异常行为，如崩溃、挂起等，从而针对这些输入进行进一步变异和测试。这种策略能够更精准地定位和触发潜在漏洞，提高测试的深度和广度。

基于变异的fuzz



基于生成的fuzz



图 12: 常见 fuzz 生成规则示意图

基于语义的指导变异：

语义变异（Semantic Mutation）策略基于对目标程序输入的语义理解，进行有针对性的变异。例如，在测试网络协议时，可以根据协议的格式和语法规则，生成合法但边界条件不同的输入数据。这种策略通过对输入的语义理解，生成更具针对性的测试用例，从而提高漏洞发现的效率和准确性。语义变异策略通常结合静态分析技术，提前分析输入的语法和语义结构，为模糊测试提供指导。

基于历史数据的指导变异：

历史数据反馈（Historical Data Feedback）策略利用之前测试中收集到的失败案例和成功变异记录，指导当前的变异过程。通过分析历史数据，可以识别出哪些变异策略更有可能触发漏洞，并对这些策略进行优先选择和优化。这种策略通过积累和利用测试经验，不断改进变异过程，提高测试效率。

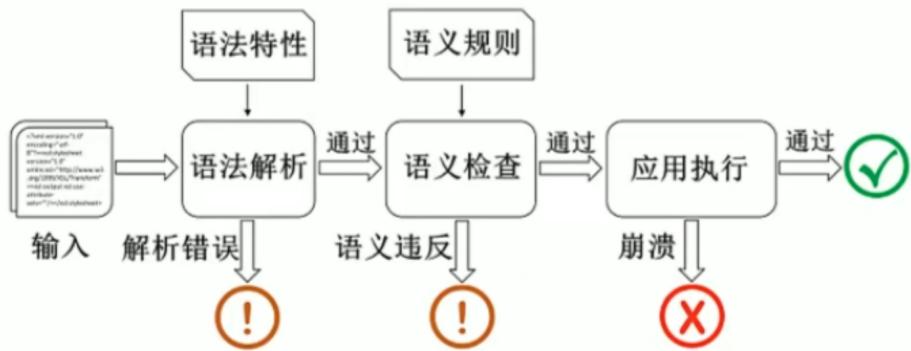


图 13: 基于语法语义指导 fuzz 变异示意图

2 研究要求

内容提要

研究目标

技术指标

研究内容

考核方式

2.1 研究目标

本项目的研究目标是设计一个基于新创环境的网络安全测试验证靶场，该靶场能够对目标网络拓扑环境进行全真模拟，在使用虚拟化的技术之后，得到的仿真环境应该与原目标网络环境等价。同时，该网络靶场应具有漏洞挖掘的功能，能够对目标网络环境中潜在的各方面的漏洞进行挖掘，相应方向包括但不限于二进制程序模糊测试、docker 等容器的逃逸、操作系统内核提权、web 应用注入风险、拒绝服务攻击、弱密钥与弱口令等等。

在对目标网络环境中漏洞挖掘得到相应的数据之后，靶场应该具有一定的自动分析能力，结合当前比较新兴的 AI 技术，对得到的数据以及结合网络中公开的漏洞信息、CVSS 评价指标等进行综合分析，对整个系统的安全性进行评价，并对各漏洞对整个系统的威胁得到一个综合评估，给出漏洞修复的优先级建议。

最后系统应具备一定的自我修复功能，这需要一定的专家知识经验，对系统中常见的经典漏洞进行修复，之后靶场要对整个系统进行再次评级，观察漏洞修复工具的功效。

2.2 研究内容

首先基于 virsh 工具和 docker 容器虚拟化技术实现靶场的搭建，并完成对目标网络环境的模拟，其中包括这 ssh 和 vnc 等多种连接技术，力求使靶场环境轻量、可扩展性高；

接下来在漏洞挖掘方面，聚焦于利用 AI 模型辅助模糊测试、爬虫技术建立海量 cve 数据集、静态与动态扫描结合、xray 与 nuclei poc 等工具的集成使用等方面的技术，力求实现全方位的漏洞扫描工作，在深度、广度、效率等方面取得行业优势。

在系统测评方面，为了解决 CVSS 目前存在的诸多缺陷，尽可能的优化评估标准以适应日益复杂的目标网络拓扑环境，我们将评估问题抽象成一个而数据建模/表征有效性的问题，并以此构建模型对问题进行求解。

最后，结合专家经验知识，尝试构建自动修复系统，对目标网络环境中的漏洞进行修复。

2.3 技术指标

本项目的技术指标如下：

- 靶场对目标网络环境模拟的仿真程度不低于 90%
- 漏洞挖掘的覆盖率不低于 90%
- 漏洞挖掘的效率不低于市面上 85% 的开源工具
- 漏洞挖掘涉及范围包含当前所有开源工具所能涉及到的范围
- 系统测评的准确性和可靠性要高，其适应复杂网络拓扑的能力要优于当前市面上 90% 的开源测评工具
 - 至少支持 10 种以上经典漏洞的修复能力
 - cve 数据集至少包含 2000 个 cve 的详细信息

2.4 考核方式

自行构建网络拓扑结构，并将相应的网路拓扑图输入到靶场环境中，查看相应配置是否正确，并分析靶场模拟效果是否与真是的目标网络环境等价；之后启动靶场的漏洞挖掘功能，对目标网络中的所存在的漏洞进行挖掘，并将结果呈现，分析漏洞挖掘的覆盖率以及时间效率；然后将漏洞挖掘交给测评系统对系统整体进行评估，并对评估结果进行分析，将评估结果的准确性和有效性作为考核指标之一；最后根据评测结果得到漏洞修复优先级，对目标网络环境中的漏洞进行修复，并再次进行测评，对比前后两次的测评效果，作为考核对网络安全设备修复能力的指标。

3 系统功能概述

内容提要

网络靶场搭建

全覆盖漏洞挖掘

扩展性与轻量级

AI 赋能系统评估

3.1 网络靶场搭建

为了对目标网络环境进行良好的测试，需要搭建可以导入目标网络环境、解析其网络拓扑并最终实现虚拟仿真的网络靶场。该工作目前已经在学术界和工业界有了许多比较有价值和意义的工作，网络上也已经出现了许多开源工具，例如 Terraform、Ansible 和 Docker Compose 等可以实现自动化靶场的部署和配置，但是，目前这些工具大多都是为了实现 CTF、AWD 等网络攻防比赛，仅仅是便于在靶场中部署漏洞，并没有在靶场内部实现漏洞的自动探测和评估以及最终的修复，而我们的项目基于本次大赛主题，创新性地设计出了一款在能够自动部署的基础上实现自动化探测多方面漏洞、系统性评估并有策略地进行修复的网络安全测试验证靶场。



图 14: 常见虚拟化仿真工具与平台

此外，当前市面上的靶场仅仅适用于特定的 CTF、AWD 竞赛所需的小型网络环境，甚至多为单主机。而随着当前企业以及网络渗透竞赛对网络拓扑环境复杂性的要求日益提高，需要测试的目标网络环境也变得越来越复杂，所以，网络测试验证靶场应该能够承担模拟仿真日益复杂的网络拓扑环境以及相关设备，为此，在我们团队的项目中，我们对于虚拟机和虚拟容器相关的网络拓扑构造问题，通过智能化识别并解析输入靶场的网络拓扑结果，构建相应的图结果，并将其建模、转化为图可达问题，最终智能化编写 xml、Dockerfile 相关文件，实现网络连通性等价仿真的效果。

我们的网络靶场是用 Python 语言实现的，它分为两大组成部分：基于虚拟机镜像的网络靶场和基于 docker 容器的虚拟靶场。其中，对于虚拟机靶场部分，我们使用了 libvirt 实现了从虚拟机的 xml 文件编写、虚拟机构建、网络配置及连接，以及远程测试到最后的销毁全过程的自动化，并将测试结果写成 json 文件回传靶场主机。而对于测试 docker 容器安全性的这一部分，我们同样实现了全自动化，此外，由于 docker 的 cp 命令可以十分方便的实现文件的传递，因此我们可以十分灵活地将我们用于漏洞测试的静态链接文件实现主客机之间传递，并从客户机得到漏洞分析结果。

说明

总结来说，我们的网络测试验证靶场有以下几个优点：

1. 能够全自动构建目标网络环境，并能通过智能化解析网络拓扑结构，实现对网络拓扑的模拟仿真；
2. 支持虚拟机和 docker 多种镜像；
3. 十分轻量：使用 VNC 和 ssh，测试文件静态链接编译后向目标测试主机传递，测试完毕后回传结果文件；
4. 可扩展性强：所有测试文件、库都只存在于靶场主机，所有被测试客户机不需要安装，即插即用；
5. 兼容性好：适配各种操作系统，完全适配国产化；

3.2 系统安全排查

在系统安全的管理和检测中，权限设置的不当和保护措施的不严格都可能引发潜在的安全漏洞。此外，在攻击过程中，攻击者往往会留下各种痕迹，提示系统的安全状态。因此，在对系统进行进一步的安全分析之前，首先需要对系统进行全面的信息扫描和数据收集。这一步骤对于挖掘和探测系统中的漏洞，以及对整个系统进行有效的安全评估至关重要。

我们的工作基于一个开源项目，对其进行改写和扩展，以更好地适用于本次项目的需求。这一扩展不仅包括对项目功能的优化，还涵盖了系统安全检查的多个关键领域。

```
total 212K
drwxr-xr-x 147 root root 12K Jul 25 11:00 ..
drwxr-xr-x 2 root root 4.0K Jul 18 02:25 .
-rw xr-xr-x 1 root root 2.9K Nov 13 2023 docker
-rw xr-xr-x 1 root root 5.5K Jun 14 2023 mysql
-rw xr-xr-x 1 root root 2.9K Apr 13 2023 apport
-rw xr-xr-x 1 root root 1.9K Mar 16 2023 open-vm-tools
-rw xr-xr-x 1 root root 4.0K Jan 11 2023 ssh
-rw xr-xr-x 1 root root 3.0K Apr 15 2022 gdm3
-rw xr-xr-x 1 root root 985 Apr 15 2022 grub-common
-rw xr-xr-x 1 root root 2.0K Apr 6 2022 cups-browsed
-rw xr-xr-x 1 root root 2.9K Mar 24 2022 bluetooth
-rw xr-xr-x 1 root root 6.8K Mar 8 2022 udev
-rw xr-xr-x 1 root root 959 Feb 25 2022 procps
-rw xr-xr-x 1 root root 469 Feb 24 2022 pulseaudio-enable-autospawn
```

图 15: 系统文件扫描运行图

具体来说，我们的系统安全检查工具提供了以下功能：

基础配置检查：审计系统的核心配置文件，确保其符合最佳安全实践。

网络流量监控：分析和捕获系统的网络数据包，以识别异常流量和潜在的网络威胁。

任务计划审核：检查系统中的计划任务配置，发现未授权或异常的任务设置。

环境变量审查：分析系统和用户环境变量配置，防止不安全的变量设置。

用户信息审计：检查用户账户和权限设置，确保没有异常或过度权限的用户。

服务配置检查：评估系统服务的配置和状态，确认只有必要的服务在运行。

bash 脚本分析：扫描系统中的 bash 脚本文件，检测是否存在潜在的恶意脚本。

恶意文件检测：利用文件扫描工具识别系统中的恶意软件或不明文件。

内核 Rootkit 检测：检查内核模块和配置，以发现可能存在的 Rootkit。

SSH 配置审核：审查 SSH 服务的配置文件，确保安全设置符合最佳实践。

Webshell 识别：检测 Web 服务器和应用程序中的 Webshell，防止远程控制。

挖矿活动检查：扫描系统中的文件和进程，发现潜在的挖矿软件。

供应链安全检查：评估系统中的软件包和依赖项，确保其来源可信且未被篡改。

服务器风险评估：综合评估服务器的整体安全状态，识别和修复潜在的风险。

通过对这些功能的全面检查，我们能够更准确地识别和评估系统中的安全风险。我们基于 GitHub 项目进行的改写和扩展，不仅提升了工具的功能性，还确保了其在本次项目中的有效应用。这些检查和扫描将为系统的安全管理提供强有力的支持，帮助我们全面了解和保护系统的安全性。

说明

这里给出我们参考的项目链接：

<https://github.com/al0ne/LinuxCheck/blob/master/README.md>

3.3 web 服务漏洞

3.3.1 背景介绍

随着互联网的快速发展，Web 应用程序已经成为信息交流、业务处理和服务提供的重要平台。然而，Web 应用程序的普及也带来了诸多安全隐患。攻击者可以通过多种方式对 Web 应用进行攻击，例如 SQL 注入、跨站脚本攻击（XSS）、跨站请求伪造（CSRF）、文件上传漏洞等。这些攻击可能导致敏感数据泄露、系统瘫痪、服务中断，甚至是经济损失和声誉受损。因此，Web 安全已成为信息安全领域的一个重要研究方向。

无论是攻防演练的靶场，还是实际生产环境中的网络集群，其外围机往往都部署 web 应用以提供服务，换言之，攻破 web 服务可能是攻入系统的第一步，其重要性可见一斑。如今市面上也有许多成熟的自动攻击工具，例如由 Mazin Ahmed 与 Khaled Farah 开发的漏洞检测与利用工具 Shennina，由长亭科技推出的漏洞检测工具 xray 等，也有许多可用的 poc 数据库如长亭 xray 工具附属的 xpoc，poc 收集项目 NucleiTP 等。除此之外，由于我们可以在靶机上执行命令，所以也可以静态扫描代码评估可能存在的漏洞。



图 16: web 安全知识体系图

3.3.2 我们的工作

我们的漏洞检测系统将对靶场进行全面的扫描，充分收集主机的资产并进行网络资产测绘。在收集服务信息后，将会匹配漏洞库，查询对应服务与版本是否存在已知的 CVE 以及现成的 poc。

如果存在 CVE 与 poc，则系统将自动使用 poc 进行验证，验证成功即代表该服务或版本真实存在漏洞。对于部分存在的 SQL 查询接口和模板注入类接口，我们同样会使用模拟测试方法，验证这些接口是否存在潜在的安全漏洞。

在这一过程中，我们的漏洞检测系统不仅仅局限于表面信息的收集和初步分析，而是深入挖掘主机和网络资产的细节信息，确保所有可能的漏洞点都得到充分关注。通过精细化的扫描和资产收集，系统可以识别出各类设备和服务的版本信息，进一步提高漏洞匹配的准确性。特别是对于一些常见但危害严重的漏洞类型，例如 SQL 注入和模板注入，系统会进行多次、反复的验证，确保检测结果的可靠性和准确性。

此外，在漏洞验证的过程中，我们不仅考虑了漏洞存在的可能性，还评估了其利用的可行性和潜在的影响。通过对漏洞利用的成功与否进行严格的验证，系统能够提供更具实用性的安全建议，帮助用户在实际环境中采取有效的防护措施。同时，对于每一次的漏洞扫描和验证，系统都会生成详尽的报告，报告中包含了漏洞的详细描述、影响分析以及修复建议。这些报告不仅为系统管理员提供了及时的漏洞信息，还为后续的安全策略制定和实施提供了重要依据。通过这些综合措施，我们的漏洞检测系统在保障网络和主机安全方面发挥了关键作用，有效提升了整体的安全防护能力。

在性能提升方面，考虑到当前漏洞测试工具普遍存在效率低、等待时间长的缺点，我们充分发挥 go 语言在多线程方面的优势，在漏洞测试系统中使用多线程优化以提高执行效率。通过对我们自行搭建的目标系统中的部分 web 服务进行测试发现，我们设计的工具可以在 10 秒钟内测试超过 300 个 poc。同时由于获取了具体服务指纹，可以遴选具体使用的 poc，大大优化了执行效率，同时结合静态扫描扩展了漏洞扫描的广度与精度。

http	80	poc-yaml-shopxo-cnvd-2021-15822	False
http	80	poc-yaml-showdoc-default-password	False
http	80	poc-yaml-skywalking-cve-2020-9483-sqli	False
http	80	poc-yaml-solarwinds-cve-2020-10148	False
http	80	poc-yaml-shiziyu-cms-apicontroller-sqli	False
http	80	poc-yaml-solr-cve-2019-0193	False
http	80	poc-yaml-solr-fileread	False
http	80	poc-yaml-solr-velocity-template-rce	False
http	80	poc-yaml-sonarqube-cve-2020-27986-unauth	False
http	80	poc-yaml-sonicwall-ssl-vpn-rce	False
http	80	poc-yaml-showdoc-uploadfile	False
http	80	poc-yaml-spark-api-unauth	False
http	80	poc-yaml-spon-ip-intercom-file-read	False
http	80	poc-yaml-spring-cloud-cve-2020-5405	False
http	80	poc-yaml-spon-ip-intercom-ping-rce	False
http	80	poc-yaml-spring-cloud-cve-2020-5410	False
http	80	poc-yaml-spark-webui-unauth	False
http	80	poc-yaml-spring-cve-2016-4977	False
http	80	poc-yaml-springboot-env-unauth	False
http	80	poc-yaml-springcloud-cve-2019-3799	False
http	80	poc-yaml-supervisord-cve-2017-11610	False
http	80	poc-yaml-tamronos-iptv-rce	False
http	80	poc-yaml-telecom-gateway-default-password	False
http	80	poc-yaml-terramaster-cve-2020-15568	False
http	80	poc-yaml-terramaster-tos-rce-cve-2020-28188	False
http	80	poc-yaml-thinkadmin-v6-readfile	False
http	80	poc-yaml-tensorboard-unauth	False
http	80	poc-yaml-tianqing-info-leak	False
http	80	poc-yaml-thinkcmf-lfi	False
http	80	poc-yaml-tomcat-cve-2018-11759	False
http	80	poc-yaml-tomcat-cve-2017-12615-rce	False
http	80	poc-yaml-tongda-meeting-unauthorized-access	False
http	80	poc-yaml-thinkphp-v6-file-write	False
http	80	poc-yaml-tongda-user-session-disclosure	False
http	80	poc-yaml-tpshop-sqli	False
http	80	poc-yaml-thinkcmf-write-shell	False
http	80	poc-yaml-tvt-nvms-1000-file-read-cve-2019-20085	False
http	80	poc-yaml-typecho-rce	False
http	80	poc-yaml-ueditor-cnvd-2017-20077-file-upload	False
http	80	poc-yaml-uwsgi-cve-2018-7490	False

图 17: web 漏洞挖掘运行图

3.4 用户态程序漏洞挖掘

3.4.1 研究背景

近年来，随着机器学习技术的迅猛发展，其在自然语言处理、自动化测试以及代码分析等领域的应用日益广泛。传统的人工智能技术，如支持向量机、遗传算法和推理引擎，也逐渐在代码分析中得到应用，以防止系统部署前可能引入的漏洞。根据 Cybersecurity Ventures 的报告，全球每年产生的代码量已达到 1110 亿行。通过在系统部署前运用自动化机制检测漏洞，开发团队能够将更多资源投入到功能开发和性能优化中。虽然机器学习和人工智能技术在安全漏洞检测方面已经取得显著成果，但为了进一步提升检测的精度和效率，仍需不断探索和研究新的应用领域。

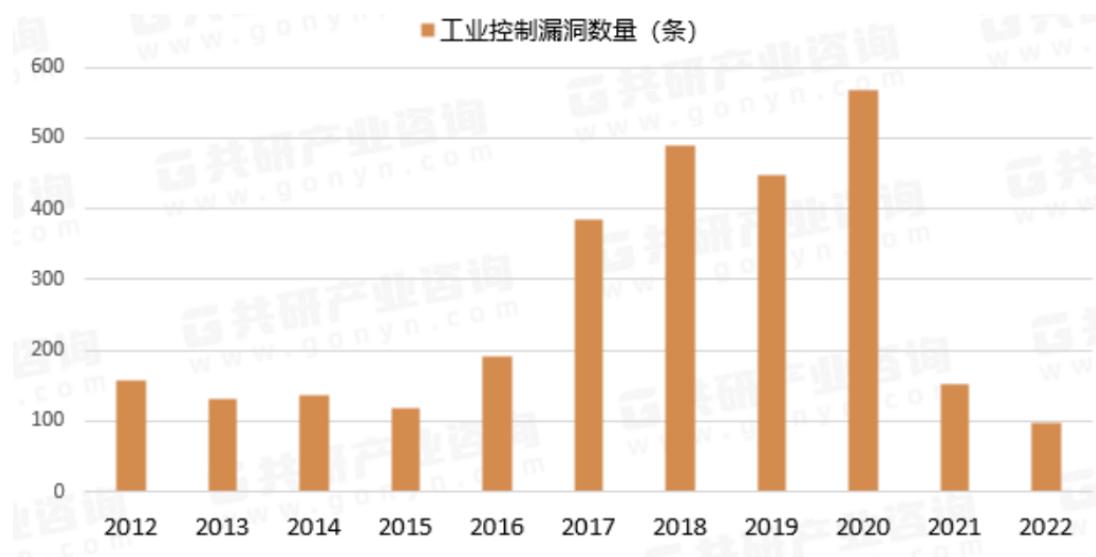


图 18: 2012-2022 年工业控制漏洞数量走势图

自然语言处理的进步为研究人员提供了自动化处理漏洞描述和产品规范的新机遇。在 2017 年，CVE 每周记录近 300 个漏洞，这给系统管理员带来了巨大的压力，他们需要对这些漏洞进行评估、优先处理和修补。Bozorgi 等人利用机器学习技术对 CVE 漏洞标签和描述进行分析，并生成单一评分，用于综合评估漏洞的可利用性和严重性。这一评分系统帮助系统管理员更有效地优先处理和修补新发现的漏洞，从而提高整体系统的安全性。

由清华大学团队设计的用于辅助逆向分析的工具 MLM，是一种利用机器学习技术来辅助恶意软件分析和逆向工程的工具。它通过训练机器学习模型来自动识别和分类恶意软件，从而帮助安全分析人员更快速地进行恶意软件分析和逆向工程。MLM 的设计目标是提高恶意软件分析的效率和准确性。

3.4.2 AFL

AFL (American Fuzzy Lop) 是一款广泛使用的开源模糊测试工具，专为发现软件中的安全漏洞而设计。它通过自动化生成大量随机输入并监测程序的运行状态，帮助用户在短时间内识别出可能存在的崩溃点和内存错误。AFL 的独特之处在于其高效的覆盖率引导策略，使其能够探索更多代码路径，从而提高漏洞发现的概率。

AFL 的工作原理基于插桩技术和遗传算法。首先，AFL 会对目标程序进行插桩，以收集运行时的覆盖率信息。然后，利用遗传算法对输入样本进行变异和交叉，以生成新的测试用例。每个新

生成的测试用例都会被执行并监测其覆盖情况。通过不断优化输入样本，AFL 能够快速定位程序中的潜在漏洞，并提供详细的崩溃报告，便于开发者进行修复和调试。这种高效的漏洞检测方式，使得 AFL 成为众多安全研究人员和开发团队的首选工具。

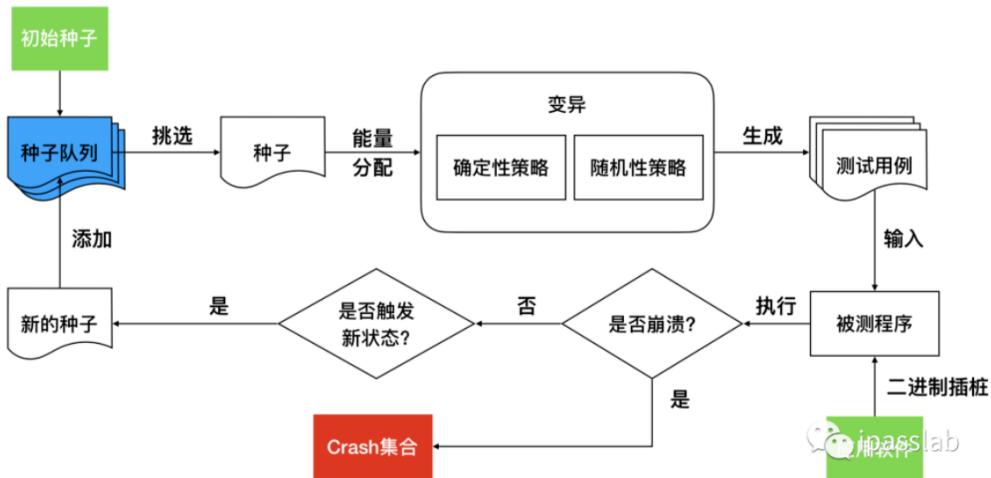


图 19: AFL 的 CGF 主要流程示意图

3.4.3 我们的工作

基于以上信息，在工程实践上，我们创新性地提出了一种首先基于现有机器学习模型、以数据流为导向进行辅助逆向，进而采用大语言模型辅助构造输入的新型模糊测试方法，具体实现方法见 4.6. 通过具体的实践证明，我们的模糊测试方法能够更好的理解程序的语义结构，对若干个我们部署在目标靶场中的二进制程序进行测试，发现对于所有目标程序，我们能够用更短的时间触发更多的 poc（使用普通的 afl 共发现了 138 个 poc，用时 12 小时；而使用我们集成的工具则只需要使用 10.5 个小时发现了 154 个漏洞）。

3.5 内核态漏洞挖掘

3.5.1 内核漏洞

内核漏洞是指存在于操作系统内核中的安全漏洞。这些漏洞可能会被攻击者利用来提升权限、执行任意代码、获取敏感信息或造成系统崩溃等。内核作为操作系统的核心部分，负责管理硬件资源、提供底层服务，任何内核漏洞都可能对整个系统的安全性和稳定性产生严重影响。常见的内核漏洞类型有如下几种。

权限提升漏洞：利用内核漏洞，攻击者可以从低权限用户提升到高权限用户（如 root 用户）。Dirty COW (CVE-2016-5195)，这是一个内存子系统中的漏洞，允许攻击者通过竞争条件提升权限。

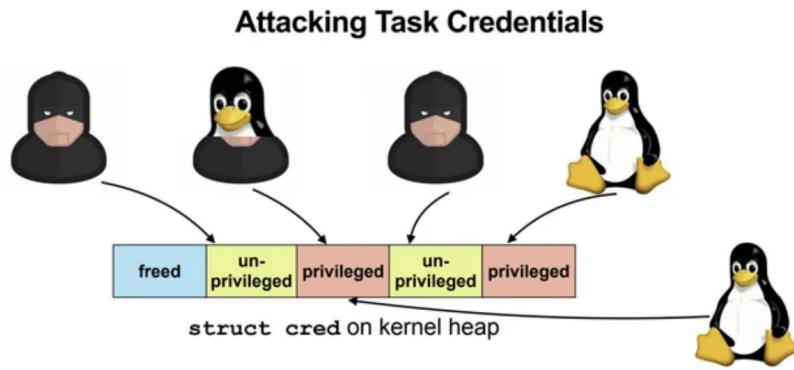


图 20: 内核提权原理图

内存泄露: 内核中的内存处理错误导致敏感数据泄露。Heartbleed (尽管主要影响的是 OpenSSL 库, 但其对内存处理的漏洞也可以被视为与内核相关)。

缓冲区溢出: 内核中的缓冲区溢出漏洞允许攻击者执行任意代码或导致系统崩溃。CVE-2017-6074, 一个存在于 Linux 内核 DCCP 模块中的双重释放漏洞, 允许攻击者执行任意代码。

竞争条件: 内核在处理多个进程或线程时出现的同步问题, 可以导致意外行为或安全漏洞。CVE-2014-3153 (Towelroot), 这是一个影响多个 Linux 内核版本的竞争条件漏洞, 允许本地用户提升权限。

信息泄露: 内核中的信息泄露漏洞可能使攻击者获得内核地址空间布局或其他敏感信息, 从而更容易利用其他漏洞。CVE-2013-6282, Android 内核中的一个漏洞, 允许应用程序读取物理内存。

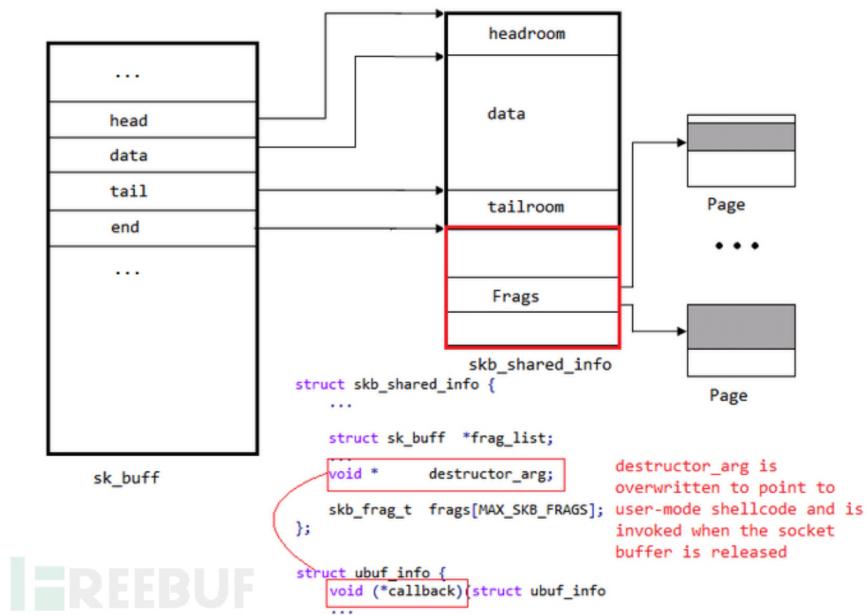


图 21: 内核分析示意图

3.5.2 Syzkaller

通过一定策略随机生成足够多的系统调用来对内核模块进行模糊测试已经成为当前研究内核安全的重要方法, 为此我们进行了相关调研, 在了解当前行业内主流工作 [10] 之后, 也对我们的系

统功能进行了进一步的扩充，使之能够通过调用一个内核 fuzzer 来提升其对内核模块漏洞挖掘的能力。

Syzkaller 是一个专为内核开发和测试设计的模糊测试 (fuzzing) 工具。它主要用于发现操作系统内核中的安全漏洞和稳定性问题。Syzkaller 的工作原理是自动生成大量随机和边界情况的输入数据，并将这些数据输入到内核中，以触发潜在的漏洞或异常行为。

Syzkaller 通过生成和执行大量随机系统调用序列，对内核进行模糊测试，旨在发现内核中的崩溃、挂起和安全漏洞。其使用覆盖率引导的模糊测试技术，通过收集和分析代码覆盖率信息，不断改进生成的测试输入，以触及更多的代码路径，增加发现漏洞的概率。它能支持并行执行多个测试实例，从而提高测试效率和覆盖范围。还具有自动化崩溃分析功能，能够自动收集和分析内核崩溃日志，帮助开发者定位和修复问题。

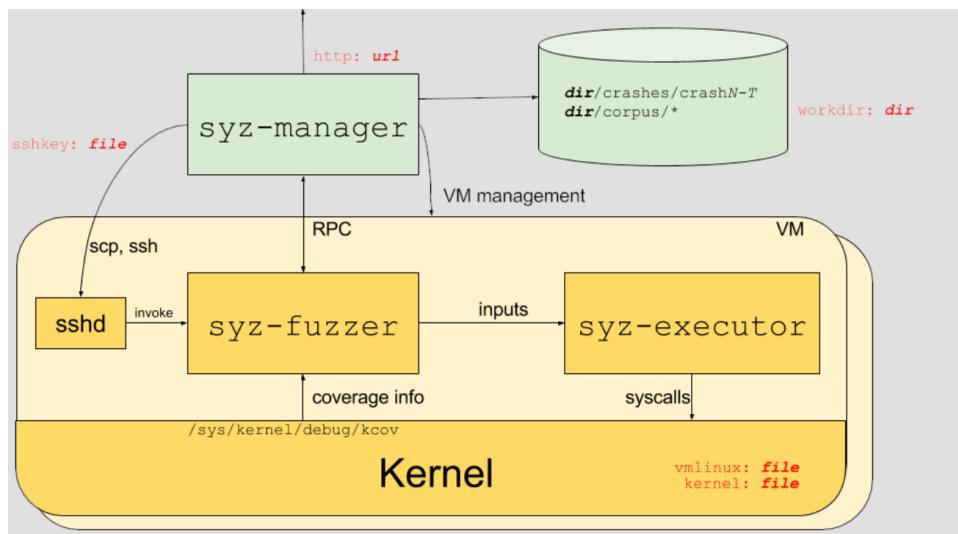


图 22: syz-fuzz 示意图

3.5.3 我们的工作

我们通过自行编写 Python 脚本，采用爬虫技术从网站上获取海量的 cve 信息以及相应的 poc、exp 或者源码，以此来构建我们的 cve 验证数据集合，以具体实现中，我们使用了 Scrapy 框架来实现爬虫。收集完数据集后，通过比较当前内核版本和 cve 的适用版本，判断当前内核版本是否会有对应 cve 的漏洞。如果有就可以尝试用现有的 cve 进行攻击。通过攻击结果分析当前内核环境的漏洞数量、漏洞类型、漏洞危险等级，进而对当前环境整体评估。

通过 lsmod 命令和常见内核模块过滤表，过滤出当前内核中的可疑模块。通过对可疑模块的分析，判断该可疑模块是否有缓冲区溢出、竞态条件、权限提升、信息泄露、未授权访问、内存泄漏等常见漏洞类型。

通过 Syzkaller 工具进一步挖掘探索出当前内核中存在的其他漏洞。Syzkaller 自动生成和执行测试用例，减少了人工编写测试用例的工作量。它能够持续运行测试，自动化检测内核中的问题，提高测试覆盖率和效率。它使用覆盖率引导的模糊测试技术，通过收集代码覆盖率信息，不断改进生成的测试用例，以触及更多的代码路径，从而发现更多潜在问题。通过 Syzkaller 工具进一步对内核层面的漏洞进行挖掘和分析，得到更好的评估结果。

说明

我们的爬虫使用了 Scrapy 框架，涵盖了从 2022 年开始至今的 700 个内核 cve，下面简单展示一下我们爬虫的界面。

```
Run: get ×
  cvesource/cvecollections (Stars: 0)

  Searching for CVE-2023-4208...
  Found 7 repositories for CVE-2023-4208:
    - nomi-sec/PoC-in-GitHub (Stars: 6281)
    - alphaSeclab/sec-tool-list (Stars: 581)
    - khulnasoft-lab/awesome-security (Stars: 24)
    - DarkFunct/TK-CVE-Repo (Stars: 8)
    - rxerium/stars (Stars: 1)
    - vexari/CheatSheet (Stars: 1)
    - hshivhare67/Kernel_4.1.15_CVE-2023-4206_CVE-2023-4207_CVE-2023-4208 (Stars: 0)

  Searching for CVE-2023-4272...
  Found 7 repositories for CVE-2023-4272:
    - RENANZG/My-Forensics (Stars: 89)
    - cometkim/awesome-list (Stars: 20)
    - sunlei/awesome-stars (Stars: 4)
    - vexari/CheatSheet (Stars: 1)
    - deepalgithub/productDisplay (Stars: 0)
    - tfriedel/awesome-stars (Stars: 0)
    - LulzSecToolkit/Lulz4Life (Stars: 0)

  Searching for CVE-2023-4569...
  Found 5 repositories for CVE-2023-4569:
    - bcoles/kasld (Stars: 405)
    - vexari/CheatSheet (Stars: 1)
    - dhalubiec/baw-project (Stars: 0)
    - zerosorai/Update-Office-2013 (Stars: 0)
    - hiyorijl/all-my-repo-starts (Stars: 0)

  Git Run Python Packages TODO Python Console Problems Terminal Services
  Localized PyCharm 2022.3 is available // Switch and restart // Don't ask again (moments ago)
```

图 23: 爬虫过程示意图

3.6 容器逃逸风险检测

3.6.1 docker 技术与 namespace

Docker 容器通过利用内核的 namespace 特性实现了进程隔离和资源管理。Linux 内核的 namespace 提供了一种将全局系统资源隔离到各自独立的视图中的机制，使得在一个 namespace 中运行的进程看不到或干涉其他 namespace 中的进程。Docker 容器主要利用了以下几种 namespace：

PID namespace (进程 ID 命名空间)：隔离进程 ID，使得容器中的进程拥有独立的进程 ID 空间。容器内的进程 ID 从 1 开始，外部看不到容器内部的进程 ID。

NET namespace (网络命名空间)：提供独立的网络栈，包括网络接口、路由表、端口等。每个容器可以有自己的虚拟网络接口、IP 地址和端口。

IPC namespace (进程间通信命名空间)：隔离系统的进程间通信资源，如信号量、消息队列和共享内存。容器内部的进程只能访问其所在 namespace 中的 IPC 资源。

MNT namespace (挂载命名空间)：隔离挂载点，使得容器拥有独立的文件系统视图。容器可以拥有自己的挂载点，不影响宿主机和其他容器的文件系统。

UTS namespace (UTS 命名空间)：隔离主机名和域名。容器可以设置自己的主机名和域名，与宿主机和其他容器无关。

USER namespace (用户命名空间): 提供用户和组 ID 的隔离。容器可以映射和使用不同的用户 ID 和组 ID，使得容器内的 root 用户在宿主机上不具备 root 权限，增强安全性。

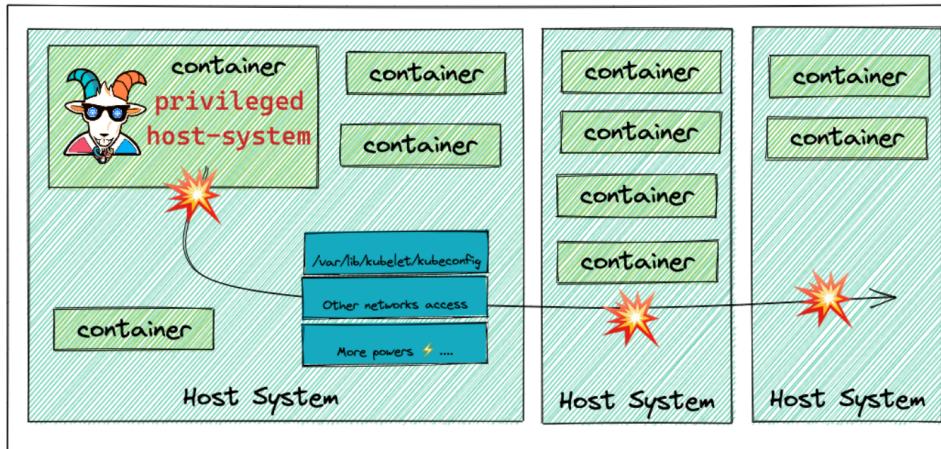


图 24: 容器权限示意图

3.6.2 docker 逃逸

Docker 容器逃逸是指攻击者通过某种方式突破容器的隔离机制，获得对宿主机或其他容器的访问权限。容器逃逸与内核的关系密切，因为 Docker 依赖于 Linux 内核的 namespace 和 cgroup 特性来实现容器的隔离和资源管理。以下是容器逃逸的几种常见方式及其与内核的关系：

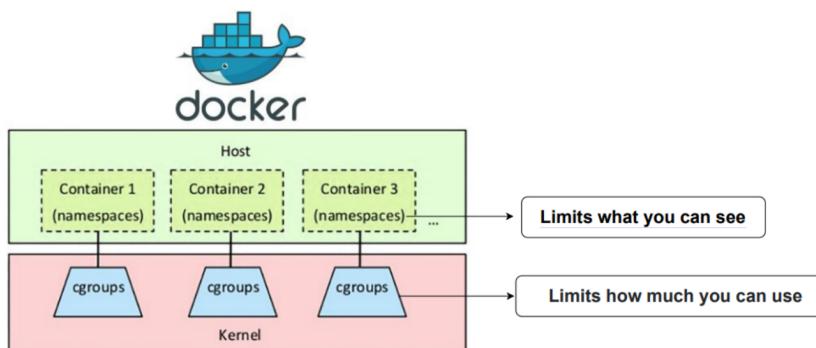


图 25: docker 逃逸

内核漏洞：

内核漏洞是最常见的容器逃逸途径之一。攻击者可以利用内核漏洞提升权限，从而突破容器的隔离。例如，通过利用某些内核特权漏洞，攻击者可以在容器内执行任意代码，进而控制宿主机。示例：Dirty COW (CVE-2016-5195) 是一个知名的内核漏洞，允许攻击者通过竞争条件获得写权限，从而提升权限。

配置不当：

不安全的 Docker 配置可能导致容器逃逸。例如，将宿主机的敏感目录挂载到容器中、使用特权模式运行容器、禁用安全选项（如 seccomp、AppArmor、SELinux）等，都可能增加容器逃逸的风险。示例：如果容器以 -privileged 选项运行，它将拥有宿主机的大部分特权，攻击者可以利用这些特权访问和修改宿主机的资源。

CAP_SYS_ADMIN 权限滥用：

CAP_SYS_ADMIN 是一个非常强大的权限，赋予进程广泛的系统管理能力。如果容器内的进程拥有 CAP_SYS_ADMIN 权限，攻击者可以利用该权限执行各种操作，可能导致容器逃逸。示例：利用 CAP_SYS_ADMIN 权限，攻击者可以加载内核模块、挂载文件系统、更改网络设置等，从而突破隔离。

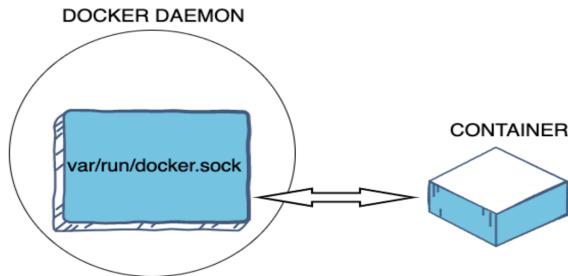


图 26: docker daemon

共享内核资源的滥用：

容器共享宿主机内核，因此某些内核资源（如 /proc 文件系统、/sys 文件系统）可能暴露敏感信息。攻击者可以利用这些信息发起针对内核的攻击，进而逃逸。示例：通过访问 /proc 文件系统，攻击者可以获取宿主机的进程信息，利用这些信息进一步发起攻击。漏洞利用和逃逸工具：

攻击者可能使用专门的漏洞利用工具和逃逸工具（如 nsjail、libseccomp-bpf）来突破容器的隔离。示例：nsjail 是一个用于测试和逃逸容器的工具，攻击者可以利用它在不同的 namespace 之间切换，从而逃逸。

3.6.3 我们的工作

当前的漏洞测评工具大多基于 web 安全和网络资产测绘，以及一些软件漏洞等问题。例如，开源工具 OSV-Scanner 聚焦于软件扫描，找出影响软件的依赖项和漏洞；而 Sqlmap 则主要聚焦于数据库的漏洞扫描，同时适合挖掘各种注入类漏洞。在这里我们创新性地将针对 docker 等容器逃逸的安全性检测模块加入到一个系统的安全测评工具中，并在实现了在 docker 外进行主机隔离测试，从而提供了一些基于内核 cve 的崩溃性逃逸测试，大大增加了测试的广度。具体技术细节详见 4.7 .

3.7 展示系统设计

此外，我们还针对系统扫描测绘得到的结果，以及评估模块对系统做出的整体分析评价，使用 HTML、CSS 和 JavaScript 来创建一个简单的数据展示页面。通过可视化网页，用户可以清晰地看到整个系统中各类漏洞的总数目以及占比，我们也通过下面几个板块清晰地展示了系统中网络资产重要性的评估以及实时漏洞挖掘报告，在展示大屏的右边则体现的是我们靶场对整个目标系统安全性的评分，以及近期检测的评分变化趋势。如下图所示的是一个我们系统的漏洞挖掘实时报告：

序号	本地危险评分	修复优先级	综合危险评分
105	65.96	95.62	41.33
106	35.36	84.33	11.33
107	43.96	86.95	10.57
108	5.62	18.62	1.33

图 27: 漏洞挖掘实时报告

这有助于提高用户对当前信创环境中潜在安全漏洞的认识和理解，增强对其重要性的认知。同时，当我们能够看到数据和趋势时，更容易认识到目标网络环境中的脆弱点并对后续的防御策略提供思路。

下面展示我们实现的一个对本地系统测评结果的可视化前端网页界面：



图 28: 前端整体界面

4 技术方案

内容提要

网络渗透

容器逃逸预防

内核提权检测

AI 辅助模糊测试

4.1 网络验证靶场设计

尽管当前在漏洞测评领域已经有相当多的团队设计出了许多小有成效的工具，但是他们的整体结构基本都是基于单主机，聚焦于某一类漏洞进行测评 [1]。然而，随着信创环境中网络拓扑的日益复杂，人们对网络环境的整体安全性的要求也日益提高，所以以往的基于单主机的测评工具在面对日渐复杂的网络环境时往往显得力不从心 [5]。

针对网络拓扑结构多样的特点，我们创新性地提出了一种基于 C/S 结构的多主机漏洞测评的网络靶场。整个靶场基于 virsh 和 docker 虚拟化技术实现整个网络拓扑的模拟，在将整个网络拓扑模拟出来之后，通过 ssh 和 VNC 来保证靶场对各主机的控制。下面展示一下我们前端中关于靶场整体信息的展示界面：



图 29: 靶场信息前端展示图

在正式漏洞检测时，首先由 Server 端扫描整个网络，获取目标机的架构等信息，从而向所有目标机上传 Client。再由每个 Client 端进行本地的安全评估，总结出报告返回 Server 端统一分析。这样一来我们的测评系统就具有了良好的兼容性和可扩展性，可以灵活自由地向网络拓扑中添加或者删除一个任意操作系统的节点，而整个测评系统仍然可以不受任何影响地正常工作。



图 30: SSH 标识

4.2 主机信息扫描与收集

通过运行改写后的 Go 项目，我们对系统进行了全面的扫描，充分收集了主机的信息和系统状态。这些信息包括系统配置、网络流量、用户账户、服务状态等关键数据。为了便于进一步分析和报告，我们将所有扫描结果整理并写入了一个 Markdown 文件中。

下面展示我们的部分系统核查结果：

```
731 ## 用户信息检查
866 ### 登录信息 lastlog
867     ||| openvpt
909 saned          **Never logged in**
910 colord          **Never logged in**
911 geoclue         **Never logged in**
912 pulse           **Never logged in**
913 gnome-initial-setup
914 hplip           **Never logged in**
915 gdm             **Never logged in**
916 zzzp            pts/2    192.168.76.1 Fri May 31 14:34:10 +0800 2024
917 mysql           **Never logged in**
918 sshd            **Never logged in**
919 libvirt-qemu
920 libvirt-dnsmasq
921 swtpm           **Never logged in**
922
923 --
```

图 31: 主机扫描结果图

4.3 网络资产测绘

对于网络资产测绘，我们的系统将会首先获取正在监听端口的服务，再调用 nmap 进行指纹识别获取具体的服务信息。在网络安全中，全面了解并掌握网络资产情况是至关重要的，因此我们的系统设计了一套高效、精准的资产测绘流程。

在实际操作中，我们的系统会通过监听网络流量来识别当前活跃的服务，并利用 nmap 工具对这些服务进行深度扫描。nmap 作为一款强大的网络扫描工具，能够准确地检测出网络中的各种服务信息，包括但不限于服务的类型、版本、开放端口等。通过这些信息，系统可以构建出一张详尽的网络资产图谱，为后续的安全防护提供基础数据支持。

对于部分 web 相关的资产识别，我们的靶场系统会进行更具体的指纹识别，判断是否为记录有 POC 的 CMS 或其他服务。这一步的具体目的是为了准确识别出网络中运行的具体应用程序，尤其是那些常见的 CMS 和其他关键服务。通过指纹识别，系统可以确定这些应用的具体版本和配置，从而进一步评估其安全性。

在指纹识别的过程中，系统会对比已知的 poc 漏洞库，判断当前服务是否存在已知漏洞。这一

过程不仅能发现潜在的安全风险，还能为系统管理员提供具体的漏洞利用方法，帮助他们及时修复安全隐患。

此外，我们还会进行路径扫描、弱口令扫描检测是否存在信息泄露以及弱口令。路径扫描旨在识别网络中存在的所有路径和节点，确保网络拓扑的完整性和准确性。弱口令扫描则是通过尝试常见的弱口令来检测系统是否存在弱口令漏洞，这是一种常见且有效的安全测试方法。通过这些扫描，系统能够全面评估网络资产的安全状况，并提供相应的安全加固建议。



图 32: 网络资产测绘结果图

4.4 web 服务漏洞

通过上文所述的信息收集，系统将会获得详细的 web 服务记录，以此查询漏洞库中可用 poc。如果存在可用 poc，则会进入测试环节，在 server 上执行 poc（动态测试）同时在 client 执行扫描任务（静态扫描），最后统计真实漏洞数。我们的系统兼容 xpoc 与 Nuclei 的大量 poc，同时还可以自行添加 poc，保留了可扩展性。poc 验证使用 cel-go 模板引擎，支持多种运算符与函数，包括随机数，各类编码等，大大简化了 poc 编写过程。此前也有大量静态扫描工具，诸如 php 后门扫描、静态代码审计等。我们的系统也有类似功能，可以扫描 web 路径，扫描源码中可能存在的漏洞。在验证的环节中，我们发挥 go 并发型优势，使用线程池管理验证任务，大大加快了执行效率。

提示

cel-go 是 Google 发布的 CEL 的 Go 语言实现，提供了高效的表达式评估能力。它常用于需要动态规则评估、策略执行、过滤和选择等场景。以下是 cel-go 的一些主要特性：

轻量级：cel-go 库轻量且易于集成。

性能高效：设计上考虑了高效评估，适合大规模数据处理。

类型安全：在表达式编写和评估过程中提供了类型检查，减少运行时错误。

易于扩展：支持自定义函数和操作符，方便扩展功能。

4.5 内核漏洞测评

4.5.1 基于版本的内核 cve 初探

在我们的工具中，基于 uname -r 命令获取当前内核版本信息，在./kernel/cve 目录下存放着很多 cve-xxx-xxx 文件夹。通过 cve-xxx-xxx 目录下的 version 文件，可以判断当前内核版本是否匹配 CVE 漏洞的版本需求，如果有漏洞可以尝试现有的 CVE，则选取已有的 EXP 进行攻击。通过攻击结果分析当前内核环境的漏洞数量、漏洞类型、漏洞危险等级，进而对当前环境整体评估。

在这一过程中，我们的工具采用系统化的方法对内核版本信息进行精准提取，通过对内核版本与 CVE 数据库中的已知漏洞版本需求，确保漏洞匹配过程的准确性和高效性。工具中存储的 CVE 目录结构设计合理，文件命名规范，使得系统能够快速定位并分析相关漏洞信息。

```
[+] Extracted version: 5.8.0
[+] check cve for given version
[+] Entering dir: ./kernel/cve/
%!(EXTRA string=)
```

图 33: 内核版本识别结果

在实际应用中，通过读取 cve-xxx-xxx 目录下的 version 文件，工具可以高效地判断当前内核版本是否存在已知的安全漏洞。一旦匹配成功，系统将进一步调用现有的 EXP（利用代码）进行模拟攻击测试，以验证漏洞的实际存在性和可利用性。这个步骤不仅仅是验证漏洞的存在，更是对漏洞利用成功率和潜在影响的评估。

攻击测试的结果将被详细记录和分析，结果文件中包括但不限于漏洞数量、类型、严重性及其对系统的潜在影响。这些信息为我们系统中后续的指标测评模块提供了全面的安全态势感知，使之可以更好地理解当前内核环境的安全状况。最终，基于这些详尽的评估结果，可以制定更为精准的安全加固措施，从而有效提升系统整体的安全防护水平。

```
[+] Checking cve: cve-2021-43267
[+] Checking cve: cve-2022-0847
[+] Version is in the range of cve-2022-0847
%!(EXTRA string=)
[+] hijacking suid binary..
[+] dropping suid shell..
[+] restoring suid binary..
```

图 34: 内核漏洞检测

4.5.2 内核 cve 漏洞库构建

我们自行编写 Python 脚本，采用爬虫技术从网站上获取海量的 cve 信息以及相应的 poc、exp 或者源码，以此来构建我们的 cve 验证数据集合，以下是我们选取的几个比较权威完善的提供 cve 及其相关信息网站：

- NVD (National Vulnerability Database)
- <https://nvd.nist.gov/>
- CVE Details: <https://www.cvedetails.com/>
- Mitre CVE Database: <https://cve.mitre.org/>

具体实现中，我们使用了 Scrapy 框架来实现爬虫。Scrapy 是一款功能强大的开源爬虫框架，具有高度的可扩展性和灵活性，非常适合复杂的网页抓取任务。

我们的爬虫由以下几个主要部分组成：

- **数据收集模块**: 负责从各个数据源中提取 CVE 信息。
- **数据清洗模块**: 对收集到的数据进行预处理，去除冗余信息，规范数据格式。
- **数据存储模块**: 将清洗后的数据存入数据库，便于后续的查询和分析。

本模块通过爬虫技术实现了大规模的 CVE 数据收集，为内核潜在 cve 的发现提供了坚实的数据基础。爬虫技术的应用不仅提高了数据收集的效率，还拓展了数据的覆盖范围，对整个项目的后续进展具有重要意义。

说明

由于我们爬虫构建的漏洞数据集极其庞大，所以提交作品时只提供了一小部分以供展示，如需更多 cve 数据，请联系我们。

4.5.3 内核 poc 验证

对符合漏洞条件的内核版本，我们运行之前构造好的 cve 合集中的 poc 进行测试，我们在 exp 中于提权之后通过写 root 权限的文件来用于批量检测各脚本是否成功进行内核提权；另一方面，我们也在提权后自动进行一些诸如“whoami”、“id”之类的命令，便于我们的中间效果展示。

下图是一个成功提权的 exp 运行过程中的运行命令“whoami”的效果图：

```
[+] hijacking suid binary..  
[+] dropping suid shell..  
[+] restoring suid binary..  
[+] popping root shell.. (dont forget to clean up /tmp/sh ;))  
# whoami  
root  
#
```

图 35: 内核 poc 验证

注意

我们的内核 poc 测试中的脚本对当前内核存在一定的破坏性测试，所以在运行前请确保对重要数据进行备份，以免 poc 崩溃导致系统重启数据丢失。

4.5.4 可疑内核模块的识别与测试

我们通过“lsmod”命令，可以获取当前内核中的所有模块，下面是我们直接通过 lsmod 命令得到的内核模块示例：

Module	Size	Used by
vulnerable_module	12288	0
isofs	61440	2
nf_conntrack_netlink	57344	0
xfrm_user	61440	1
xfrm_algo	20480	1 xfrm_user

图 36: 本机 LKM 示意图

之后我们通过在项目中制作过滤表，通过数据搜集整理常见的内核模块，然后制作过滤器，过滤系统中自带的内核模块，就可以得到可疑的模块。下面是我们运行工具发现当前环境中可疑内核模块的示例：

```
## Rootkit检查
### lsmod 可疑模块
```shell
vulnerable_module 12288 0
```
```

```

图 37: 可疑 LKM 示意图

## 4.6 用户态程序漏洞挖掘

首先，我们利用 nmap 工具进行端口扫描，并过滤掉常见的服务名字后发现可疑的部署在开放端口的服务，然后定位对应二进制文件的路径；之后首先将定位的二进制文件输入到机器学习模型中进行初步逆向分析，然后将逆向分析的结果交给大语言模型进行分析，得到一个更符合程序预期的输入格式，并以此作为种子交给 fuzzer 进行模糊测试，最后得到若干 poc 并进行分析。

### 4.6.1 机器学习模型辅助逆向

我们结合当前学术界及工业界已有的技术思路 [4]，实现了一个基于 Python 和 Capstone 库实现的二进制文件分析模块，该模块能够对 X86 架构的 64 位二进制文件进行反汇编，并进一步提供静态分析功能。该工具的核心功能是反汇编，通过 Capstone 库的 Cs 类实例化，指定 CPU 架构 (CS\_ARCH\_X86) 和模式 (CS\_MODE\_64)，对二进制文件中的机器码进行解析，生成汇编指令。工具以 0x1000 为起始地址，逐条反汇编指令，并收集这些指令到列表中。同时，工具还计算了指令的总数和二进制文件的总字节大小，为后续分析提供了基础数据。

进一步地，该模块实现了指令频率分析，通过统计每种指令出现的频率，可以对程序的行为模式有一个初步的了解。此外，该模块还提供了将反汇编结果输出到文件的功能，这不仅方便了结果的保存和后续分析，也使得结果的查看更为直观。

在指令类型统计方面，我们的思路是通过分析不同指令的类型，识别程序中使用的不同指令集，这有助于深入理解程序的执行逻辑。高级格式化输出功能则进一步优化了反汇编结果的展示方式，使得每条指令的地址、助记符和操作数信息更加清晰易读。

除了基础的反汇编和统计功能，我们还尝试实现了函数调用识别和安全漏洞检测。函数调用识别通过查找包含“call”关键字的指令，列出所有可能的函数调用点，这对于理解程序的控制流程至关重要。安全漏洞检测则通过检查包含“jmp”关键字的指令，识别潜在的代码执行漏洞，这对于提高软件安全性具有重要意义。

最后，我们通过一个示例展示了如何使用这些功能对一个具体的二进制文件进行分析，并打印出包括指令、函数调用和潜在漏洞在内的分析结果。我们相信这种集成式的分析流程不仅提高了分析效率，也为后续大语言模型提供了一个全面、深入理解二进制文件的途径。

```
0x1000: jg 0x1047
0x1002: add r8b, byte ptr [rcx]
0x1006: add dword ptr [rax], eax
0x1008: add byte ptr [rax], al
0x100a: add byte ptr [rax], al
0x100c: add byte ptr [rax], al
0x100e: add byte ptr [rax], al
0x1010: add eax, dword ptr [rax]
0x1012: add byte ptr ds:[rcx], al
0x1015: add byte ptr [rax], al
0x1017: add byte ptr [rax], ah
0x1019: adc al, byte ptr [rax]
0x101b: add byte ptr [rax], al
0x101d: add byte ptr [rax], al
0x101f: add byte ptr [rax], al
0x1022: add byte ptr [rax], al
0x1024: add byte ptr [rax], al
0x1026: add byte ptr [rax], al
```

图 38: 机器学习模型辅助逆向结果示意图

总体而言，该工具以其强大的反汇编能力和丰富的静态分析功能，为二进制文件分析领域提供了一个实用的研究和应用平台。随着软件安全形势的日益严峻，此类工具的开发和完善将对提高软件安全性、促进计算机科学的发展起到积极的推动作用。

### 提示

为了简化流程和满足用户的多样化需求，我们项目中用于 fuzz 的两个人工智能模型都是可选项，需要指定相关选项，且在默认情况下不展示中间结果，如果需要中间结果，请进行进一步配置。

#### 4.6.2 大语言模型构建输入

在得到了机器学习模型得到了辅助逆向的结果之后，我们利用大语言模型（OpenAI 的 GPT）辅助模糊测试，用以生成更加符合要求的输入种子，以显著提升 Fuzz 的效率和覆盖率。当前学术界对于此类技术已经有了很多尝试和应用，例如，加利福尼亚大学的 Davis 团队的 LLAMAFUZZ 使用大语言模型增强灰盒模糊测试，以及同样属于该团队的另一项工作，利用大语言模型和崩溃重用挖掘嵌入式系统中的漏洞 [3]。

我们的团队在此基础之上做了更进一步的工作，我们在项目中引入了基于一个深度学习模型的神经网络 [9]，在具备了生成随机二进制数据文件功能的同时，我们增加了特征工程步骤，以便考虑当前操作类型及其相邻的操作类型；此外，为了提高模型的性能，我们使用交叉验证来评估模型性能，并输出详细的评估结果。最后我们，改进了错误处理逻辑，确保在解析二进制数据时能处理异常情况，并提供了更多的控制台输出信息，以便更好地跟踪训练和预测过程。

下面是我们用于模型训练的部分代码：

```

将二进制数据解析为特征向量
def parse_binary_data(data):
 feature_vectors = []
 offset = 0
 struct_format = 'I' # 假设每个操作类型是一个4字节的无符号整数
 struct_size = struct.calcsize(struct_format)

 while offset < len(data):
 try:
 (operation,) = struct.unpack_from(struct_format, data, offset)
 feature_vectors.append([operation])
 offset += struct_size
 except struct.error as e:
 console.print(f"[red]Error unpacking data at offset {offset}: {e}[/red]")
 break

 return np.array(feature_vectors) # 将数据转化为适合模型输入的格式

```

图 39: 大语言模型部分代码

```

特征工程: 为每个操作类型增加更多特征
def feature_engineering(feature_vectors):
 features = []
 for i, operation in enumerate(feature_vectors):
 # 当前操作类型、上一个操作类型、下一个操作类型
 current_op = operation[0]
 previous_op = feature_vectors[i-1][0] if i > 0 else 0
 next_op = feature_vectors[i+1][0] if i < len(feature_vectors) - 1 else 0
 features.append([current_op, previous_op, next_op])
 return np.array(features)

```

图 40: 大语言模型部分代码

#### 4.6.3 fuzz 的选择

当前业界有很多团队将 AFL 作为他们的 fuzz，AFL 作为一款开源的模糊测试工具，以其高效的漏洞检测能力和广泛的应用范围而著称，接下来，我们将探讨选择 AFL 的原因及其在本工作中的具体应用。

首先，AFL 的核心优势在于其自动化和高效的测试能力。AFL 通过生成随机输入数据来测试目标程序，并通过覆盖率引导的方式发现潜在漏洞。这种方法能够自动发现程序中的内存泄漏、崩溃和其他安全漏洞，极大地提高了漏洞检测的效率。对于科研人员来说，AFL 能够显著减少手动测试的工作量，使他们能够更加专注于漏洞分析和修复工作。

其次，AFL 具有强大的兼容性和扩展性。AFL 支持多种平台和编程语言，包括 C、C++、Java 等，这使得它在各种科研项目中都能得到广泛应用。此外，AFL 还可以与其他工具和框架结合使用，例如 QEMU、libFuzzer 等，进一步增强了其功能和适用范围。这种灵活性使得科研人员可以根据具体需求调整和优化测试流程，以达到最佳的测试效果。

此外，AFL 的开源性质和活跃的社区支持也是选择它的重要原因。AFL 的开源代码使得科研人员可以自由地研究其实现原理，并根据需要进行定制和改进。同时，活跃的社区支持确保了 AFL 的持续更新和完善，为用户提供了丰富的资源和帮助。

最后，我们也成功的将前文所述的两个人工智能模型与 AFL 进行衔接，对多个目标程序进行模糊测试，发现衔接效果良好，自动化程度极高，相较于其他模糊测试其，我们能够在更短的时间之内发现更多的漏洞。



图 41: AFL

## 4.7 容器逃逸风险评估

为了实现对 docker 等容器的逃逸风险的检测，我们首先针对目标环境中 docker 容器进行扫描，然后一方面查看 docker 容器的权限设置，另一方面依据当前内核版本作为参考进行已有 cve 的探测，如果当前版本符合漏洞条件，则调用我们通过爬虫技术自行构建的漏洞数据库中的 poc 来验证漏洞的存在性。

### 4.7.1 容器权限检查

在这里我们首先通过读取 /proc/self/status 文件并检查其中的某些字段来判断是否为特权模，然后检查当前容器是否挂载了 docker socket 文件，接着检查当前容器是否挂载了 procfs 文件系统，再检查当前容器是否挂载了根目录 (/) 文件系统，再检查当前容器是否启用了 Docker 远程 API，再检查当前容器是否挂载了主机的 /var/log 目录。

检查完上述内容后，先检查当前容器是否拥有 CAP\_DAC\_READ\_SEARCH 权限，然后检查当前容器是否拥有 CAP\_SYS\_ADMIN 权限，最后检查当前容器是否拥有 CAP\_SYS\_PTRACE 权限。以上所有内容检查完后，将进行常见 cve 的检查。

### 4.7.2 漏洞数据库构建

这里我们同样使用了 Scrapy 框架对一下几个网站进行爬虫，获得关于 docker 等容器逃逸的海量数据集，并编写脚本对其进行统一格式的处理：

- <https://wiki.teamssix.com/cloudnative/docker/docker-escape-vulnerability-summary.html>
- <https://docs.docker.com/security/security-announcements/>
- <https://snyk.io/learn/docker-security/top-5-vulnerabilities/>
- <https://docs.docker.com/security/security-announcements/>

#### 说明

同样由于我们爬虫构建的漏洞数据集极其庞大，所以提交作品时只提供了一小部分以供展示，如需更多 cve 数据，请联系我们。

## 4.8 系统评估标准建模

### 4.8.1 测评结果统一化

我们在靶机中进行漏洞检测，每一个模块的检测结果都将生成相应的 json 文件通过 vnc、ssh 或者 docker 的 cp 命令回传到靶场中，最后靶场对所有的探测结果进行统一整合，得到的结果大致如下所示：

```
{
 "host1": {
 "kernel": {
 "cve-version": [3, 4, 3.3, 4.2, 4, 4, 5.5],
 "cve-poc": [7.1, 8, 8],
 "cve-root": [9, 9.1],
 "vul-LKM": [2.2, 2.3, 2.3],
 "vul-LKM-poc": [5.5, 5, 5.8, 6.1, 6]
 },
 "docker": {
 "privilege": [3, 3.4, 4, 2.8, 2.9],
 "cve-version": [3, 4, 3.8, 4.2, 4, 4, 3.1],
 "cve-poc": [6.6, 6.7, 7],
 "cve-escape": [8, 8.1, 6.9]
 },
 "userspace": {
 "fuzz-stack": [6, 6.6, 7, 4.9, 5, 5.5, 6.2, 7, 7.3, 6.8, 4.9, 5, 5.5, 6.9],
 "fuzz-heap": [6.1, 6.7, 5.5, 4.9, 5.4, 4.8, 7.2, 8.1, 8, 6, 5.6],
 "fuzz-fmtstr": [6.8, 6.9, 7.1, 7.0, 6.9, 6.9, 6.9],
 "aar-aaw": [7.8, 7.8, 7.9, 7.2, 8.1],
 "get-shell": [8.5, 8.6, 8.8, 8.1]
 },
 "weak password": [5, 8], //包含服务、ssh等弱口令
 "web": {
 "ssrf": [7, 6.9, 8], //可以ssrf，即可以通过http访问其他主机
 "rce": [10], //执行命令
 "sql": [6.7, 9.3], //sql注入，获取sql内容
 "leak": [6] //各种泄露，源码、配置文件等
 }
 },
 "host2": {
 ...
 }
}
```

图 42: 漏洞结果 json 文件示意图

#### 提示

数组内为 cvss 评分，无 cvss 可能需要自行评分，数组长度为漏洞个数

docker 与 host 采取同样格式

可能一个漏洞可以同属多类，例如 ssh 弱口令可加入 rce，rce 一般也可以 ssrf

## 说明

对上述 json 做一个简单的说明：

[cve-version] 表示匹配的漏洞版本数量

[cve-poc] 表示成功 poc 数量

[cve-root] 表示成功提权数量

[vul-LKM] 表示可疑 LKM 数量

[vul-LKM-poc] 表示可疑 LKM 中 fuzz 出来的漏洞数量

[privilege] 表示包含的敏感特权数

[cve-version] 表示匹配的漏洞版本数量

[cve-poc] 表示成功 poc 数量

[cve-escape] 表示成功逃逸数量

[fuzz-stack] 表示栈溢出漏洞

[fuzz-heap] 表示堆区内存漏洞

[fuzz-fmtstr] 表示格式化字符串漏洞

[aar-aaw] 表示其中能够造成任意地址读写的漏洞

[get-shell] 表示能够获得 shell 的漏洞

[weak password] 表示包含服务、ssh 等弱口令

[ssrf] 表示可以 ssrf，即可以通过 http 访问其他主机

[rce] 表示可以远程执行命令

[sqli] 表示 sql 注入，获取 sql 内容

[leak] 表示各种泄露，源码、配置文件等

### 4.8.2 指标评估框架

通用漏洞评分系统 CVSS 是一个广泛应用的标准化的框架，用于衡量计算机系统和软件中漏洞的严重性。然而 CVSS 也存在一定的局限性：CVSS 仅考虑了单步攻击风险，同时 CVSS 指标更多面向漏洞的严重性而非安全风险。此外，常见的漏洞公共目录如国家脆弱性数据库（NVD）通常使用基本度量组作为漏洞的严重性指标，这忽略了漏洞的上下文信息，包括时间和环境度量，因此在实际的应用中无法有效考虑漏洞的优先级。

在提出的系统框架中，针对由多个主机组成的网络拓扑环境，我们创新性地提出了一种基于人工智能的系统量化评估策略。该策略由两个阶段组成：资产和安全风险发现和系统安全评估。在资产和安全风险发现阶段，我们首先使用端口扫描、漏洞检测等工具对整个环境进行全局扫描，得到包含漏洞信息、主机拓扑、资产信息的系统风险文档；在系统安全评估阶段，我们利用人工智能强大的模式挖掘和信息整合能力对系统风险文档进行分析并最终得到包括量化指标、修复建议在内的系统风险报告。具体来说，我们通过注意力机制捕捉单个主机上不同漏洞间的隐式关系和漏洞对资产的关注情况，同时通过图神经网络的方法对主机拓扑进行显式建模，从而考虑主机间漏洞的相互影响。

我们建立了一种基于 CVSS 的多主机多漏洞系统风险定量评估方法，该方法考虑了主机和资产的重要性、漏洞的隐式关系、系统中的主机拓扑、单个漏洞的易受攻击性以及影响范围等多个系统安全相关的关键要素，而非单纯参考漏洞的严重性指标。

## 指标评估框架

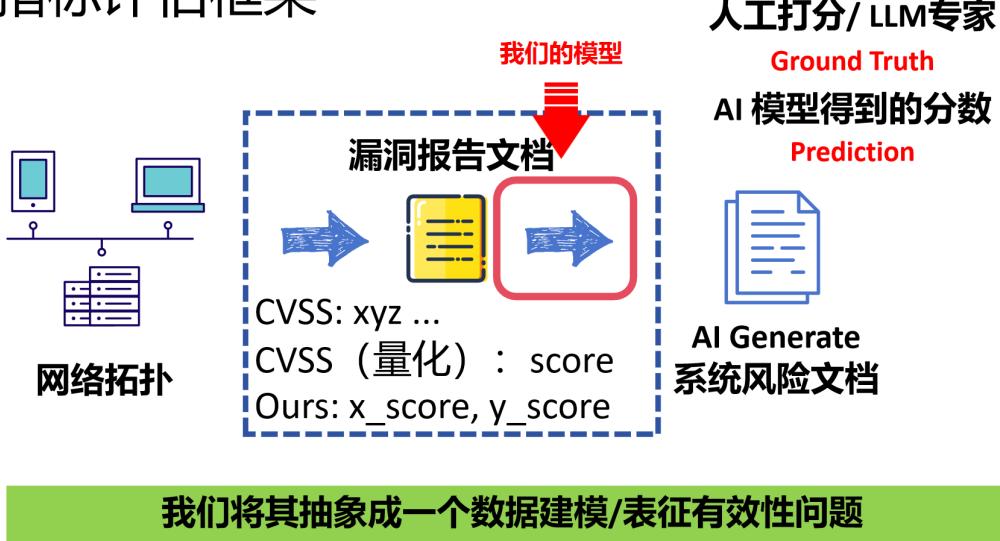


图 43: 指标评估框架

### 4.9 系统自动化防护技术实现

基于前文所述的指标评估框架，对整个系统安全性的整体评估，同时还可以得到综合考量漏洞之间的相互依赖、以及具体网络拓扑结构和对应主机网络资产价值等因素，得到了关于系统漏洞重要性的排序，接下来我们初步实现了一个自动修复系统，该系统首先以漏洞对整个系统的威胁性为依据，优先修复威胁程度大的漏洞，以期在最短的时间之内将系统所收到的威胁降到最低。

具体修复部分我们初步实现了一个包含内核提权、docker 逃逸、普通用户程序以及 web 应用的自动化修复方案。首先，我们对系统整体进行版本升级，以解决内核以及 docker 高风险版本面临的问题；其次，我们在我们的辅助逆向的机器学习模型中增加高危漏洞识别的功能，以辅助识别堆栈溢出、uaf、格式化字符串等常见二进制安全漏洞；值得注意的是，我们团队针对以往内核提权及 docker 逃逸攻击脚本的总结，发现加大针对结构体成员读写的检查可以大大限制脚本提权或者逃逸的成功率，因此我们借鉴了现在科研界比较创新的混合别名识别的方法，对部分脚本的执行初步实现了混合别名的执行，以防止重要的内核结构体的值被篡改；最后，针对 web 漏洞我们通过修复对扫描到的文件进行修复，逐步提高服务的安全性与可靠性。

# 5 主要成果和创新点

## 内容提要

- 靶场实现全真模拟
- 良好的漏洞挖掘效果
- 多维度测评
- 极高的商业价值

## 5.1 信创环境搭建与系统部署

为了验证我们靶场的功能，我们自行搭建了一个极为复杂的目标网络环境，其中共包含了 18 台主机，4 个路由器以及若干必要的交换机；我们将其构成一个复杂的网络拓扑，且之间存在不同主机之间的不同连通性，尽可能模拟当前复杂的网络环境。

### 5.1.1 网络拓扑结构

下图是我们自行搭建的网络拓扑图：

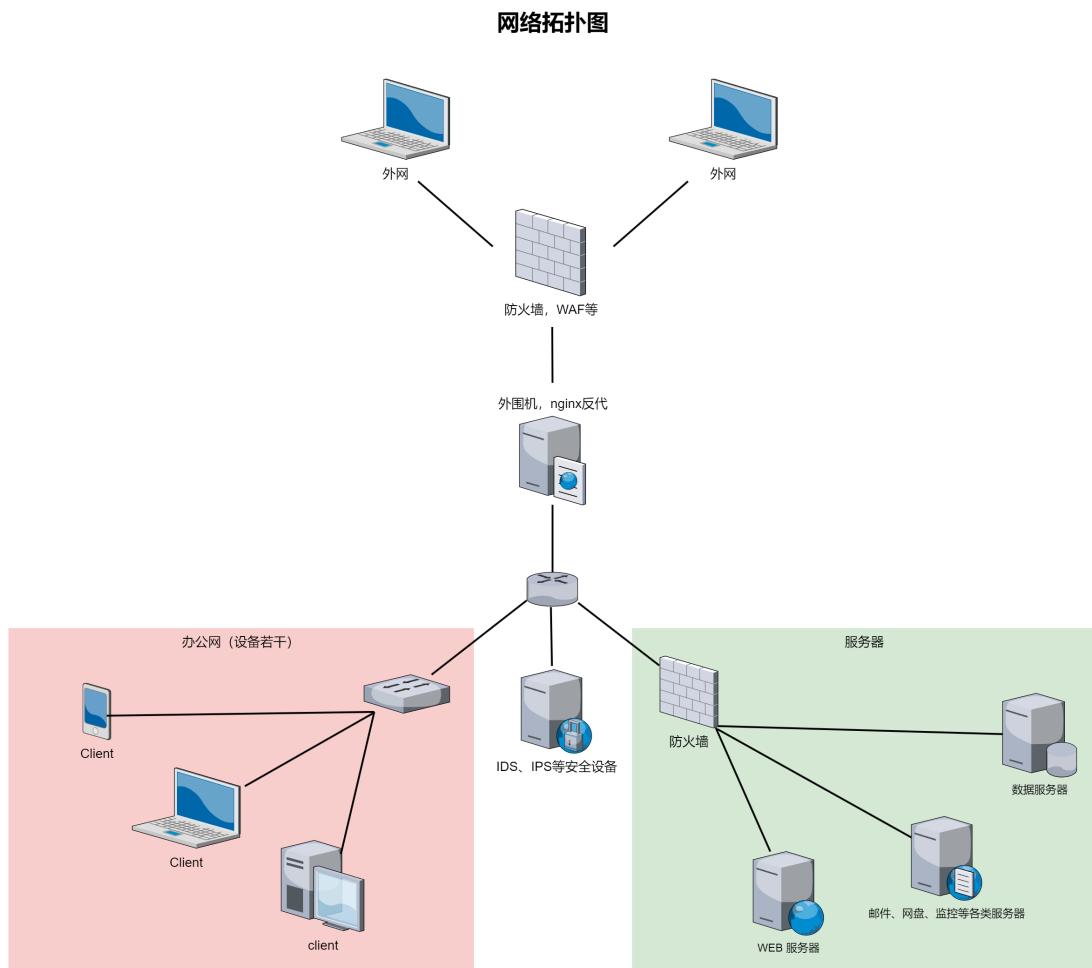


图 44: 模拟靶场网络拓扑图

### 5.1.2 环境中靶机情况介绍

在我们网络环境的 18 台主机中，其中有 11 台是由虚拟机镜像给出，另外的 7 台则是由 docker 镜像给出，并通过配置虚拟网络连通性的方法模拟了其中的路由器和交换机对网络拓扑中主机连

通的影响。

在我们的 11 台虚拟机中，由 5 台是 Linux 操作系统、3 台式 Windows 操作系统，3 台是麒麟操作系统，下面将他们列举如下：

- 主机 1: Ubuntu 22.04 LTS
- 主机 2: Ubuntu 20.04 LTS
- 主机 3: CentOS 7
- 主机 4: Windows 10 Pro
- 主机 5: Debian 10 (Buster)
- 主机 6: Windows Server 2022
- 主机 7: CentOS 8
- 主机 8: Windows 10 Enterprise
- 主机 9: 中标麒麟 (Kylin)
- 主机 10: 银河麒麟 (NeoKylin)
- 主机 11: UOS 服务器版

同时我们在 docker 中部署了大量的服务以满足对整个网络系统的复杂性的要求，下面简要列举额 docker 中部署的服务：

- Nginx
- Apache HTTP Server
- Node.js
- PHP
- Django
- Flask
- Ruby on Rails
- WordPress
- Ghost
- Jenkins
- Redis

### 说明

为了模拟 web 服务的安全漏洞，我们在一定程度上改写某些应用，或者下载一些存在已知漏洞的服务，使之在不同深度包含各种漏洞，便于后续的漏洞检测测试效果；同时，为了模拟二进制安全漏洞，我们在上述 18 台主机中共部署了 52 个带有若干漏洞的二进制程序于开放端口之上以供连接检测。由于部署漏洞较多，这里就不一一列举了。

### 5.1.3 环境中网络设备介绍

我们模拟实现了一个复杂的模拟拓扑网络，由一台外围机连接外网，接收 web 请求并使用 nginx 反代访问内网 web 服务器提供服务，内网存在服务器网与办公网，存在邮件服务器、网盘服务器、数据库服务器等服务器。办公网中有若干办公设备，包括个人电脑、手机、打印机等设备。

同时，在核心交换机存在 IDS 入侵检测系统，IPS 入侵预防系统等安全设备，外围机有 waf 防火墙，服务器间也存在防火墙。waf 与 IPS 使用软件模拟，自动审查流量，过滤黑名单关键词，同时监测 IDS 是否报警，由此评判接入安全设备的效果。

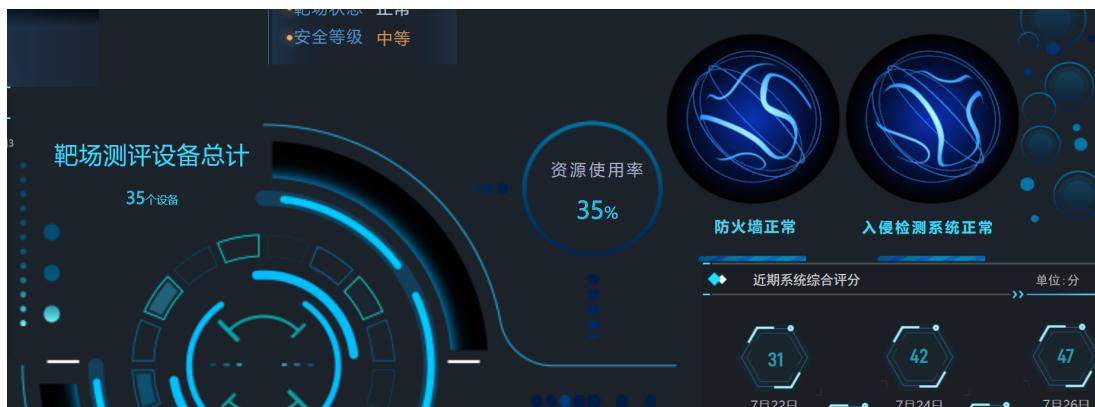


图 45: 靶场中设备监控图

#### 5.1.4 验证靶场搭建

我们基于 virsh 和 docker 的虚拟计划技术，实现了网络安全测试验证靶场的搭建，靶场能够有效识别网络拓扑结构，并对其进行虚拟仿真搭建，且我们的仿真靶场实现了全自动搭建，只需要准备好相应的镜像文件、磁盘文件以及网络拓扑就可以一键启动，且具有良好的可扩展性，下面简单展示一个我们靶场的运行截图：

```
内核测试完成，正在保存结果：
Progress: |██████████| 100.0% Complete
准备进行web测试：
Progress: |██████████| 100.0% Complete
标准输出：
Start web checking...
checking target cve-2021-41773
checking target cve-2021-41773-mysql
checking target sql
Target: cve-2021-41773(172.17.0.2)
+-----+-----+-----+-----+
| SERVICE | PORT | VERSION | AVAILABLE POC | IS SUCCESS |
+-----+-----+-----+-----+
| Apache | 80 | 2.4.49 | apache-httpd-cve-2021-41773-path-traversal | True |
| Apache | 80 | 2.4.49 | apache-httpd-cve-2021-41773-rce | False |
| Docker | | | container-escape-check.sh | False |
+-----+-----+-----+-----+
Target: cve-2021-41773-mysql(172.17.0.3)
```

图 46: 靶场运行图

同时，我们的靶场系统还能够全自动地将多个虚拟机进行从定义、创建、开启、扫描到最后关闭以及销毁的全部工作，并在具体的执行过程中展示相关的虚拟机的状态以及 VNC 端口信息，下面简单展示一个靶场中创建虚拟机仿真环境的界面：

```

Domain 'vm8' started

Id Name State

9 vm1 running
10 vm2 running
11 vm3 running
12 vm4 running
13 vm5 running
14 vm6 running
15 vm7 running
16 vm8 running

检查虚拟机vnc端口：
Progress: |██████████| 100.0% Complete
:0

```

图 47: 虚拟机环境仿真运行图

此外，对于基于 docker 容器的目标网络环境在我们的靶场系统中也得到了很好的支持，我们的测试结果表明，我们的靶场系统对 docker 虚拟环境的搭建也同样实现了自动化，可以在将指定镜像文件和配置文件放入到指定目录下之后实现自动环境搭建、自动漏洞测试、自动生成漏洞报告，并在测试完成得到返回结果之后自动将所创建的 docker 容器关闭以及销毁。下面简单展示一下我们的与运行效果图：

```

启动docker镜像：
0eb6ce20e536b3dbd54beabf025b7951789c0c698a6aaab669a752c70c751508
a7236a3764be3a6dad2a85f1af92a1f9047948d31848039dc9ee4c9d408875f1
d1eea18e436c64eeb1b2d2efacc56fc26554d67ea54f643d99f61071b55de7f
05dbe0666ca5ca402fe99ddc0d639409c05400f16f30fa3af77e58275f535bc9
3baef478b231816433a9699e83699aab2128bfd5acac71e225528594c6488c67
0f5safe4cb0f317be19ec6e25790204ba7a05846d94807de41f8a86b8869a219
0c9a3c681d915843b3b993cf6be21cf0ebbf8f79d25a106c3ce54ce9e9f7a18d

ps:
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
0c9a3c681d91 wasm:v1.0 "/bin/sh -c 'socat T..." Less than a second ago Up Less than a second nif
ty_black wasm:v1.0 "/bin/sh -c 'socat T..." Less than a second ago Up Less than a second fri
endly_wu wasm:v1.0 "/bin/sh -c 'socat T..." 1 second ago Up Less than a second gra
3baef478b2318 wasm:v1.0 "/bin/sh -c 'socat T..." 1 second ago Up 1 second str
cious_euler wasm:v1.0 "/bin/sh -c 'socat T..." 1 second ago Up 1 second lau
d1eea18e436 wasm:v1.0 "/bin/sh -c 'socat T..." 2 seconds ago Up 1 second ghing_fermi

目标网络环境拓扑仿真搭建：
Progress: |██████████| 100.0% Complete
搭建成功！

```

图 48: docker 环境仿真运行图

## 5.2 安全总结报告

我们在自行搭建的网络拓扑环境中部署了我们自主研发的检测系统，并对相关服务端口进行了扫描。结果显示，我们共发现了 110 个 Web 服务漏洞，以及 104 个可供外部连接的基于二进制程序的服务。我们对这些可疑的二进制程序进行了模糊测试，经过 8.5 个小时的测试，共发现 y 处崩溃。漏洞分析模块对这些崩溃进行深入分析后，得到 98 个可能造成拒绝服务的 PoC 以及 52 个可能导致内存任意地址读写的漏洞。随后，我们将这些漏洞进行建模分析，评估出 33 个可能导致远程用户态提权的漏洞。

此外，内核安全模块对主机系统环境进行了安全检测，发现有 5 台主机的内核版本与超过 10

个内核 CVE 所需环境匹配成功，属于高危版本，需及时修复。同时，我们对所有可能的 CVE 进行了更深入的 PoC 验证，其中 CVE-2022-0847 Dirty Pipe) 在多台 Linux 主机上成功实现了提权。此外，诸如 CVE-2022-27666、CVE-2022-32250 等漏洞在多台主机上成功导致了主机瘫痪。

这些结果表明，我们的检测系统在识别和评估网络环境中的安全风险方面具有显著效果，为提升系统安全性提供了可靠的技术支持和数据基础。

下面是我们团队对自行搭建的网络拓扑导入靶场之后进行的一次全面漏洞测试得到的漏洞汇总表格：

漏洞类型	参考评级	总数目	备注
fuzz-stack	5	48	普通栈溢出
fuzz-heap	5	35	普通堆漏洞
fuzz-fmtstr	5	28	格式化字符串漏洞
aar-aaw	7	52	任意地址读写
get-shell	9	33	可以获得 shell 的二进制漏洞
docker-previige	4	11	docker 容器存在特权
docker-cve-version	5	7	dockers 存在匹配 cve 的版本
docker-poc	8	21	已有 exp 可以造成 poc
docker-escape	9	4	成功 docker 逃逸
kernel-cve-version	5	21	内核属于危险版本
kernel-cve-poc	8	11	已有 exp 可以造成 poc
kernel-cve-root	10	3	可以成功内核提权
kernel-vul-LKM	5	14	存在可疑 LKM
kernel-LKM-poc	8	3	可疑 LKM 能够造成 poc
weak-passwd	8	9	存在弱口令
web-ssrf	8	22	可以 ssrf，即可以通过 http 访问其他主机
web-rce	8	13	远程执行命令
web-sqli	8	37	存在 SQL 注入
web-leak	8	38	web 层的信息泄露

表 1：漏洞挖掘结果

#### 提示

该表只展示了我们一部分的漏洞挖掘报告，具体详细的报告请关注我们的附件报告文档.pdf.

### 5.3 防护成效

我们在系统中部署了我们的防御模块，对前面挖掘到的漏洞进行分类，并尝试实行自动化 patch，对系统中出现的诸如 SQL 注入、缓冲区溢出、格式字符串等等经典漏洞进行 patch，使得系统的安全性得到了一定程度的提高，此项工作仍然较大程度上依赖于专家知识的储备，会在后续工作中得到急需的巩固和提升。



图 49: 靶机评分示意图

## 5.4 创新点总结

### 实现完全国产化适配

随着国家对信息安全的重视，政府出台了一系列政策鼓励和要求关键领域的信息技术产品进行国产化。目的是减少对国外技术的依赖，保障国家信息安全。例如在《网络安全法》中明确提出加强自主可控技术的应用，提升关键信息基础设施的安全保障能力。此外，推动国产化适配可以促进国内相关产业的发展，带动国产软硬件技术的进步和市场竞争力的提升。这不仅有助于提升国家科技创新能力，还有助于国内企业在国际市场上的竞争力。

因此我们在开发部署我们的项目时充分考虑到了国产化适配的问题，对我们的采取了多样化兼容的思想，首先，操作系统方面，我们的项目适配了麒麟 V10 操作系统，提升了系统的兼容性和稳定性，满足了国产操作系统的要求。其次，在硬件方面，我们一部分主机使用了国产中央处理器，确保了核心计算部件的自主可控，降低了对国外芯片的依赖。最后，在数据库方面，我们在部分主机中部署了国产数据库 GDMBASE，进一步保障了数据管理和存储的安全性。



图 50: 国产化软硬件

### 更加轻量及更好的可扩展性

我们整个靶场都是基于一个 C/S 结构的设计理念，让靶机作为一个服务器，而所有通过虚拟化技术搭建起来的靶机则都是客户端，我们仅通过发送必要的、预先编译好的静态链接文件来实现远程测试，这极大程度上保证了整个测试系统的轻量，同时我们的网络拓扑结构解析模块十分便捷，基本上实现了即插即用的效果，确保了靶场系统的可扩展性和兼容性。



图 51: 轻量级与可扩展性

### 测评维度广

我们的项目中综合考虑了 web 服务漏洞、内核提权漏洞、容器逃逸漏洞、缓冲区溢出漏洞的检测，并配合各种基于命令行的系统权限检查以及全覆盖的网络资产测绘技术，对目标网络系统实现全方位的安全检查，力求不放过每一个漏洞。



图 52: 测评维度总结图

### AI 辅助逆向与 fuzz 极大提高效率

我们使用机器学习模型辅助对二进制程序的逆向，然后将更贴近源代码的结果交给大语言模型进行输入格式的解析，并得到更加符合预期的输入样本，最后将 AFL 作为 fuzzer 进行模糊测试。

经过实际测试，对于相同的测试目标，我们的 fuzzer 能够在更短的时间内测试出更多的 poc，例如我们选取了一组测试程序，使用普通的 afl 共发现了 138 个 poc，用时 12 小时；而使用我们集成的工具则只需要使用 10.5 个小时发现了 154 个漏洞。



图 53: 大语言模型

#### 爬虫获得海量的 cve 数据集

我们基于 sorapy 框架，自行编写爬虫脚本，在各大知名 cve 网站中获取了海量的 cve 相关信息，以此来构建我们针对内核提权漏洞和 docker 等容器逃逸的漏洞数据集，并通过逐个 cve 的 exp 运行检测 poc 的存在性，在漏洞检测模块中起到了良好的效果。



图 54: Scrapy

#### 良好的漏洞测评效果

我们的漏洞挖掘各模块在实际运行中达到了良好的预期效果，其中，我们共发现了 110 个 Web 服务漏洞，以及 104 个可供外部连接的基于二进制程序的服务。我们对这些可疑的二进制程序进行了模糊测试，共发现 y 处崩溃。poc 分析模块对这些崩溃进行深入分析后，得到 98 个可能造成拒绝服务的 PoC 以及 52 个可能导致内存任意地址读写的漏洞。此外，内核安全模块对主机系统环境进行了安全检测，发现有 5 台主机的内核版本与超过 10 个内核 CVE 所需环境匹配成功，属于高危版本，需及时修复。同时，我们对所有可能的 CVE 进行了更深入的 PoC 验证，其中 CVE-2022-0847 (Dirty Pipe) 在多台 Linux 主机上成功实现了提权。此外，诸如 CVE-2022-27666、CVE-2022-32250 等漏洞在多台主机上成功导致了主机瘫痪。



图 55: 漏洞挖掘实时报告前端展示图

## 创新性地将人工智能技术加入到指标评估框架中

我们针对 CVSS 仅考虑了单步攻击风险、其指标更多面向漏洞的严重性而非安全风险的局限性，创新性地提出了一种基于人工智能的系统量化评估策略，通过注意力机制捕捉单个主机上不同漏洞间的隐式关系和漏洞对资产的关注情况，同时通过图神经网络的方法对主机拓扑进行显式建模，从而考虑主机间漏洞的相互影响。为此，我们建立了一种基于 CVSS 的多主机多漏洞系统风险定量评估方法，该方法考虑了主机和资产的重要性、漏洞的隐式关系、系统中的主机拓扑、单个漏洞的易受攻击性以及影响范围等多个系统安全相关的关键要素，而非单纯参考漏洞的严重性指标。



图 56: 人工智能技术辅助指标评估

## 对各类设备有着良好的防护成效

我们的自动化系统防护初步取得了不错的成效，我们的工具根据系统威胁性优先级进行漏洞修复，在修复了所发现的 45% 的漏洞之后即可减轻约 70% 威胁。



图 57: 防护成效示意图

## 前端展示呈现极佳体验

为了更好的展示我们的漏洞挖掘以及系统测评成果，我们团队专门设计了前端展示系统，通过前端展示系统，能够将我们得到的结果更加直观的展现给用户，同时友好的前端界面可以使用户更加便捷的进行操作，从而极大程度上避免了因为专相关业知识缺乏而导致的使用困难，大大提升了用户的体验效果，具有极佳的商业价值。

整体前端展示界面在前文中已经有所呈现，下面展示一下我们前端展示系统中的漏洞查询界面：



图 58: 前端部分界面示意图

# 6 预期应用与总结

## 内容提要

虚拟化仿真

极高的商业价值

多维度漏洞挖掘

系统漏洞修复建议

## 6.1 预期应用

### 支持目标网络环境的虚拟化仿真

靶场能够对目标网络拓扑环境进行全真模拟，在使用虚拟化的技术之后，得到的仿真环境应该与原目标网络环境等价。

### 支持目标网络环境的漏洞挖掘

网络靶场应具有漏洞挖掘的功能，能够对目标网络环境中潜在的各方面的漏洞进行挖掘，相应方向包括但不限于二进制程序模糊测试、docker 等容器的逃逸、操作系统内核提权、web 应用注入风险、拒绝服务攻击、弱密钥与弱口令等等。

### 对系统安全进行评估以及预警

靶场应该具有一定的自动分析能力，结合当前比较新兴的 AI 技术，对得到的数据以及结合网络中公开的漏洞信息、CVSS 评价指标等进行综合分析，对整个系统的安全性进行评价，并对各漏洞对整个系统的威胁得到一个综合评估，在系统收到威胁时及时给用户预警，避免由于网络攻击而导致的财产损失。

### 给系统漏洞修复顺序给出建议

靶场能够根据系统评估结果，给出漏洞修复顺序的建议，以期能够在最短的时间之内将系统面临的网络攻击威胁降到最低，具有极高的商业价值。

## 6.2 总结

### 6.2.1 存在问题

在漏洞挖掘方面：在辅助模糊测试的前期逆向模块种所使用的机器学习模型中，模型缺少对数据预处理和特征选择的详细考虑，在一定程度上影响其性能和准确性。此外，大语言模型对于某些比较少见的序列化的程序交互处理能力还有待提升，

在系统漏洞自动修复方面：由于我们对该模块的准备时间较短，专家知识数据集仍然有待扩充，所以存在许多漏洞类型的误判和修复方向的错误，在后续造成了一定的误报。

### 6.2.2 未来工作建议

为了提升漏洞挖掘和系统漏洞自动修复的效果，我们在未来的工作中将采取以下措施：

数据预处理和特征选择：我们将深入研究数据预处理和特征选择方法，优化数据的处理流程，确保特征工程的科学性和合理性，从而提高模型的性能和准确性。

增强序列化处理能力：针对模型在处理少见序列和程序交互上的能力不足，我们计划引入更先进的序列化处理技术，并通过增加模型复杂性和多样性来提升其在复杂情景下的表现。

**扩充专家知识数据库：**为了减少因专家知识不足导致的误判和修复方向错误，我们将继续扩充专家知识数据库，完善知识库内容，确保在面对各种类型漏洞时有充分的参考依据。

**延长准备时间，强化测试：**我们将适当延长模块准备和测试时间，确保每个模块经过充分的验证和优化，避免因时间紧张而导致的错误和疏漏。

**跨领域合作：**进一步加强与其他研究团队的合作，借鉴其他领域的先进技术和经验，提升模型的综合能力和鲁棒性。通过以上改进措施，我们有信心显著提升漏洞挖掘和自动修复系统的整体性能，为网络安全领域的发展贡献更多的力量。

## 7 附件说明

### 内容提要

网络靶场文件

系统部署

漏洞挖掘 go 项目

相关文件说明

### 7.1 源代码结构说明

我们整个项目的源代码主要由两个部分组成：靶场搭建模块和漏洞挖掘模块，其中靶场搭建模块是基于 Python 编写的，而漏洞挖掘模块则是基于 go 语言编写的，并通过编译生成静态链接文件，并提供给靶场后续使用。

下面首先介绍我们网络测试验证靶场相关的附件：

- boot.sh：用于启动靶场以及打印相关提示；
- vm.py：用于针对虚拟机仿真等操作；
- docker.py：用于实现 docker 容器的仿真搭建；
- net\_parse.py：用于解析网络拓扑；
- net.py：用于实现虚拟环境中网络的仿真；
- show.py：展示效果方面的辅助函数；

#### 提示

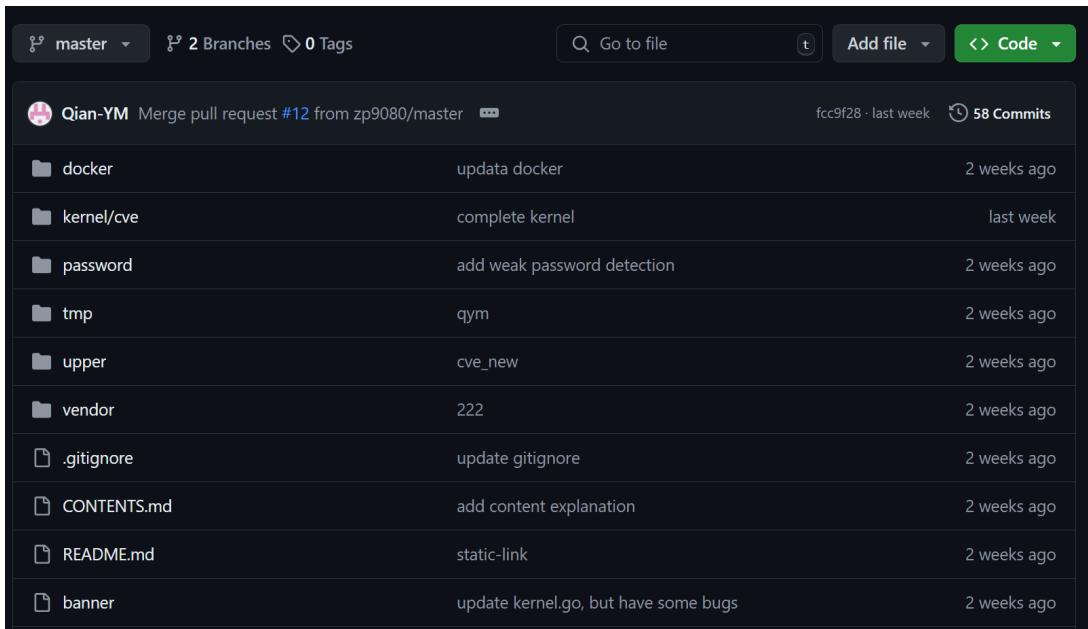
上述文件为我们的靶场部署文件，在使用之前用户需要将虚拟机的光盘映像文件放入到 iso 目录下，将磁盘文件放入到 qcow2 目录下；对于 docker 容器仿真功能，用户在使用前需要将 docker 镜像放置在 docker 目录下，将 docker 容器相关的版本信息按照指定格式放入到 docker\_info 目录下。

下面是我们漏洞测试项目文件说明：

- main.go：整个项目的主文件；
- kernel.go：内核测试模块；
- docker.go：dockers 虚拟容器逃逸测试模块；
- fuzz.go：二进制漏洞挖掘模块；
- network.go：网络安全测试模块；
- password.go：弱口令、弱密钥漏洞检测模块；
- service.go：用于检测网络服务中漏洞的模块；
- file.go：文件扫描模块；
- process.go：用于系统安全性检查与扫描；
- base.go：用于基础配置检查；
- bash.go：用于进行 bash 配置检查；
- kernel/cve/：该目录中存放我们搜集的 kernel cve 数据集；

## 说明

除此之外，还有一些相关的辅助性文件，在这里就不一一介绍了，我们的项目部署在 GitHub 上，欢迎关注我们的项目：<https://github.com/Qian-YM/Vulnerability-Assessment/tree/Refactor>



A screenshot of a GitHub repository page. At the top, it shows 'master' branch, '2 Branches', '0 Tags', and search bar options 'Go to file', 'Add file', and 'Code'. Below the header, there's a summary of a merge pull request from 'zp9080/master' to 'master' by 'Qian-YM' with '58 Commits' made 'last week'. The main area lists 13 commits with details: 'updata docker' (2 weeks ago), 'complete kernel' (last week), 'add weak password detection' (2 weeks ago), 'qym' (2 weeks ago), 'cve\_new' (2 weeks ago), '222' (2 weeks ago), 'update gitignore' (2 weeks ago), 'add content explanation' (2 weeks ago), 'static-link' (2 weeks ago), and 'update kernel.go, but have some bugs' (2 weeks ago).

Commit	Message	Date
updata docker		2 weeks ago
complete kernel		last week
add weak password detection		2 weeks ago
qym		2 weeks ago
cve_new		2 weeks ago
222		2 weeks ago
update gitignore		2 weeks ago
add content explanation		2 weeks ago
static-link		2 weeks ago
update kernel.go, but have some bugs		2 weeks ago

图 59: 项目仓库

## 7.2 系统部署说明

首先请解压我们项目两个模块的源代码，按照您的需求编译出您所需的测试文件，并将其放置到靶场搭建模块的指定路径之下，（具体编译方法请关注我们的仓库）。之后请解压我们源代码中的靶场搭建模块，将您的虚拟机镜像文件（或者是您的 docker 镜像文件）和相应的 vmdk 文件（如果是 docker 则不需要）放置到指定路径之下（具体路径我们已经在前面说明过了，这里就不再赘述），然后在 root 权限下运行脚本 boot.sh 即可启动靶场。

当您看到如下界面时说明靶场已经成功启动：

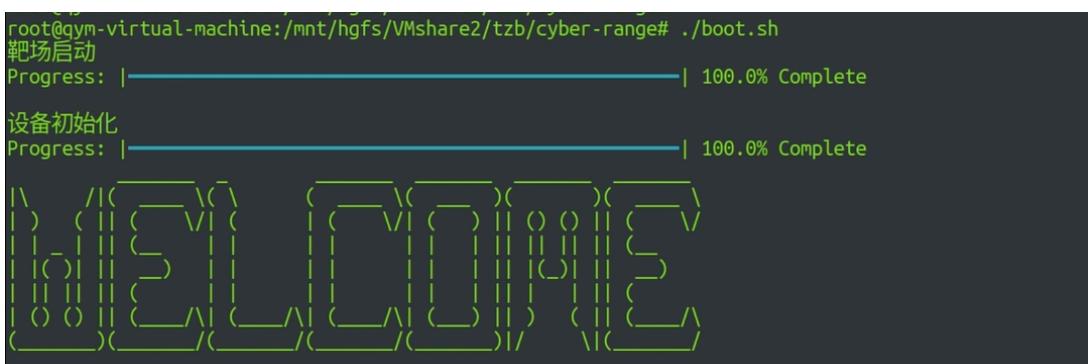


图 60: 靶场启动场景图

此外如果您想单独运行基于虚拟机的环境搭建也可以直接运行我们的 vm.py 脚本，得到如下界面即可说明成功运行：

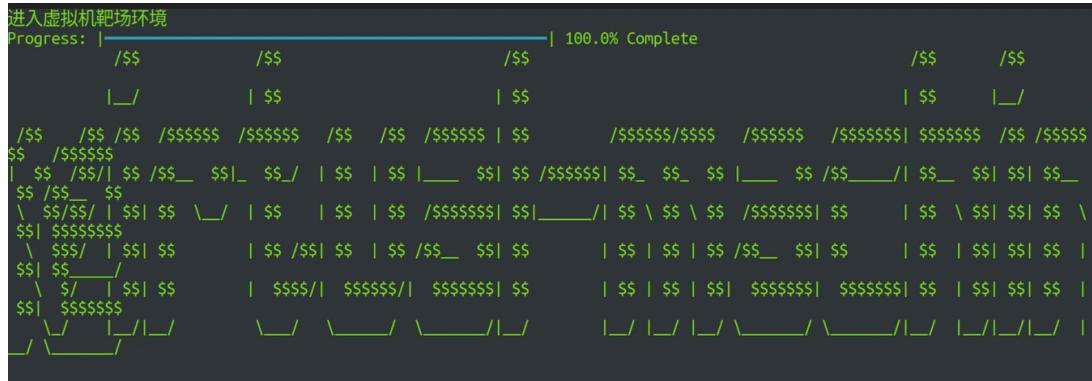


图 61: 虚拟机模拟运行图

同理，您也可以同样直接运行我们的 docker.py 脚本来仅实现模拟 docker 虚拟环境搭建：

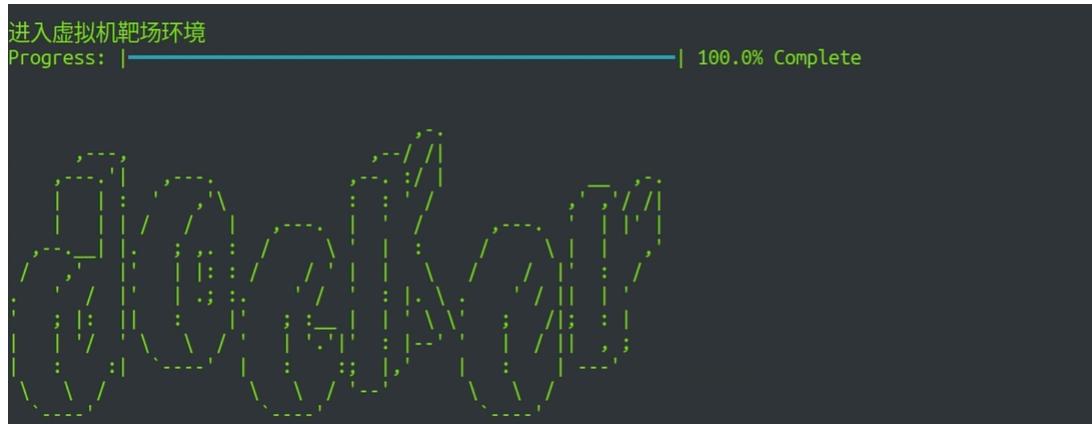


图 62: docker 模拟运行图

### 提示

为了不过多占用您的存储空间，我们的靶场会在漏洞测试完成之后会删除掉模拟过程中所创建的虚拟机以及 docker 容器，所以请您在使用我们的靶场之前对 virsh 启动的虚拟机以及重要的 docker 容器做好备份，以免误删。

当您看到如下界面时说明所创建的全部虚拟机都已经被销毁：

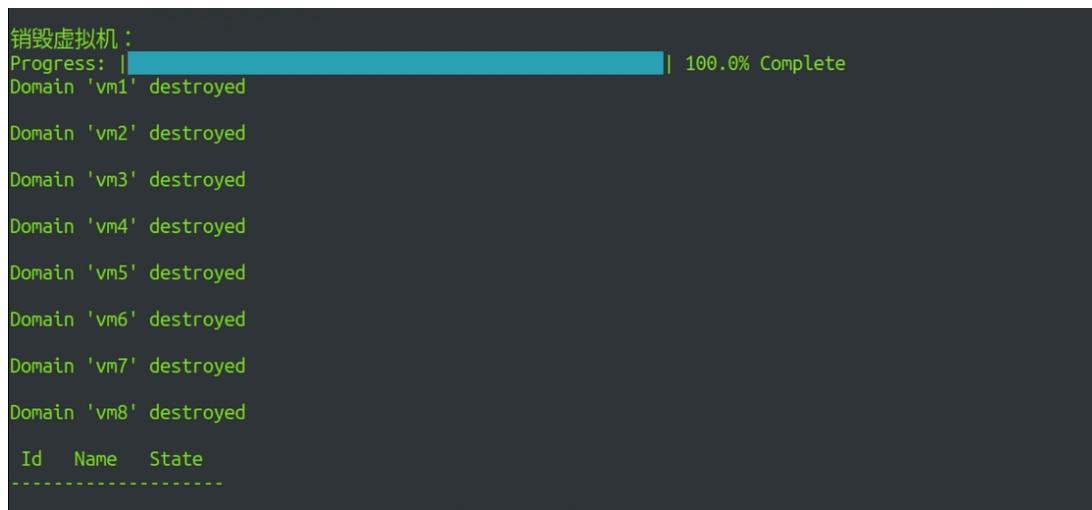


图 63: 虚拟机销毁图

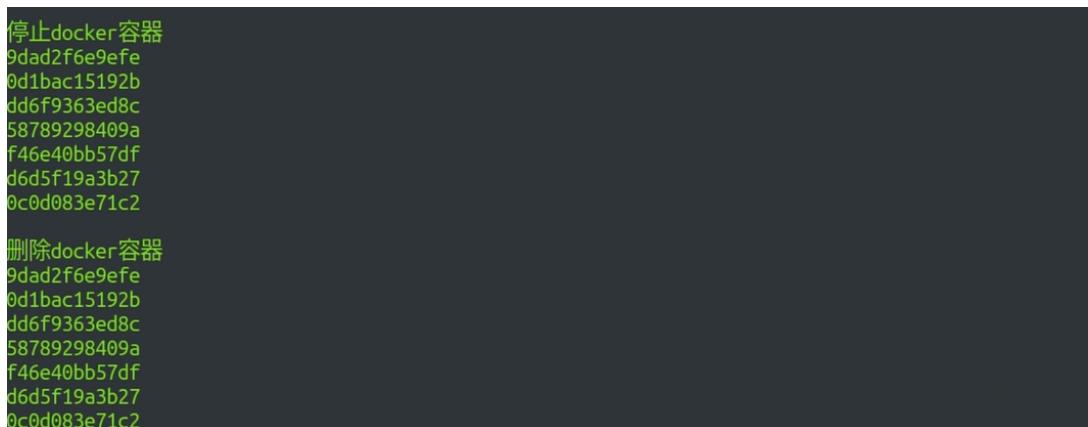
当您看到如下界面时，说明虚拟机已经成功解除定义：



```
虚拟机解除定义
Progress: |██████████| 100.0% Complete
Domain 'vm1' has been undefined
Domain 'vm2' has been undefined
Domain 'vm3' has been undefined
Domain 'vm4' has been undefined
Domain 'vm5' has been undefined
Domain 'vm6' has been undefined
Domain 'vm7' has been undefined
Domain 'vm8' has been undefined
```

图 64: 虚拟机解除定义界面图

如果您看到如下界面，则是 docker 容器关闭并成功删除：



```
停止docker容器
9dad2f6e9efe
0d1bac15192b
dd6f9363ed8c
58789298409a
f46e40bb57df
d6d5f19a3b27
0c0d083e71c2

删除docker容器
9dad2f6e9efe
0d1bac15192b
dd6f9363ed8c
58789298409a
f46e40bb57df
d6d5f19a3b27
0c0d083e71c2
```

图 65: docker 容器关闭并销毁界面图

**提示**

关于 docker 的相关信息需要用 json 格式，请按照下图格式准备。



```
{
 "name": "iot",
 "version": "v1.0"
}
```

图 66: docker\_info

## 参考文献

- [1] Yuan Xiao, Cai Zhi-Ping, Liu Shu-Hao, et al. Large scale network emulation software and its key technologies. Computer Technology and Development, 2014, 24(9): 9-12 (in Chinese) (袁啸, 蔡志平, 刘书豪等. 大规模网络仿真软件及其仿真关键技术. 计算机技术与发展, 2014, 24(9): 9-12).
- [2] 杨鑫, 张超, 李贺, 等. 基于系统调用依赖的 Linux 内核模糊测试技术研究 [J]. 网络安全技术与应用, 2019 (11): 13-16. YANG X, ZHANG C, LI H, et al. Research on Linux kernel fuzzing technology based on system call dependency[J]. Network Security Technology & Application, 2019 (11): 13-16.
- [3] Asmita, Yaroslav Oliynyk, Michael Scott, Ryan Tsang, Chongzhou Fang, Houman Homayoun. "Fuzzing BusyBox: Leveraging LLM and Crash Reuse for Embedded Bug Unearthing".
- [4] Deng J., Dong W., Socher R., Li LJ., Li K., Li FF. ImageNet: A large-scale hierarchical image database. In: Proc. of the 2009 Conf. on Computer Vision and Pattern Recognition. Miami: IEEE, 2009. 248–255. [doi: 10.1109/CVPR.2009.5206848]
- [5] Salopok D., Vasic V., Zec M., et al. A network testbed for commercial telecommunications product testing. Proceedings of the 22nd International Conference on Software, Telecommunications and Computer Networks. Split, Croatia, 2015: 372-377.
- [6] Li L, Cifuentes C, Keynes N. Boosting the performance of flow-sensitive points-to analysis using value flow. In: Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering. ACM Press, 2011. 343-353. [doi: 10.1145/2025113.2025160]
- [7] Li L, Cifuentes C, Keynes N. Precise and scalable context-sensitive pointer analysis via value flow graph. ACM SIGPLAN Notices, 2013, 48(11): 85-96. [doi: 10.1145/2555670.2466483]
- [8] Bataille J., Riera J. F., Escalona E., et al. On the implementation of NFV over an OpenFlow infrastructure: routing function virtualization. Proceedings of the IEEE Software Defined Networks for Future Networks and Services. Trento, Italy, 2013: 1-6.
- [9] Microsoft Security Development Lifecycle, Verification Phase. [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307418\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307418(v=msdn.10))
- [10] ZHANG K, XIAO X, ZHU X, et al. Path transitions tell more: optimizing fuzzing schedules via runtime program states[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 1658-1668.
- [11] WANG D, ZHANG Z, ZHANG H, et al. SyzVegas: beating kernel fuzzing odds with reinforcement learning[C]//Proceedings of 30th USENIX Security Symposium (USENIX Security 21). 2021: 2741-2758.

- [12] TAN Z, LU H. A systemic review of kernel fuzzing[C]//Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies. 2020: 283-289.
- [13] Attariyan M, Flinn J. Automating configuration troubleshooting with dynamic information flow analysis. In: Proc. of the Usenix Conf. on Operating Systems Design and Implementation. USENIX Association, 2010. 1-11. <https://www.usenix.org/conference/osdi10/automating-configuration-troubleshooting-dynamic-information-flow-analysis>
- [14] Google chromium security. 2021. <https://www.chromium.org/Home/chromium-security/bugs>
- [15] Cui W, Peinado M, Cha SK, Fratantonio Y, Kemerlis VP. RETracer: Triaging crashes by reverse execution from partial memory dumps. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 820-831. [doi: 10.1145/2884781.2884844]