

**《****通用工具操作实训》**

**实训报告**

|  |  |
| --- | --- |
| **姓 名:** | **胡佳珏** |
| **学 号:** | **2423030009** |
| **专 业­­­­­­­­­­­­­­­­­­­:** | **网络空间安全** |
| **班 级:** | **网安2401** |
| **指导教师:** | **王海东** |
| **职 称­­­­­­­­­­­­­­­­­­­:** |  |

**前沿交叉学院**

**2025年5月**

**《通用工具操作实训》评审表**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | **胡佳珏** | | 学院 | 前沿交叉学院 | | | | 学 号 | | | **2423030009** | | | |
| 专业班级 | | | **网安2401** | | | |
| 实训内容 | **Git，Docker，GDB** | | | | | | | | | | | | | |
|
| 评  审  意  见 | 项目 | | 具体要求 | | | 单项分值 | 分项评分分数参考区间 | | | | | | | 得分 |
| A | | B | C | | D | E |
| 平时表现 | | 学习态度认真，自主学习相关方法、总结归纳相关技术的能力优秀。积极与团队成员合作，积极参与讨论分析，团队关系融洽，能按时完成团队分配的工作。明确团队角色和分工，积极与团队成员合作，团队工作进展顺利。 | | | 20 | 18 | | 16 | 14 | | 12 | <12 |  |
| 实操成绩 | | Git 版本控制（15 分）：基础操作（5 分）分支管理（5 分）远程协作（5 分）；  Docker（15 分） ：镜像操作（5 分），容器管理（5 分），数据与编排（5 分）；  攻防工具选修（20 分，任选1项完成）。 | | | 50 | 45 | | 40 | 35 | | 30 | <30 |  |
| 文档质量 | | 文档规范，结构合理，文献工作量饱满，格式正确。 | | | 10 | 9 | | 8 | 7 | | 6 | <6 |  |
| 实验原理正确，步骤完整，实验结果充分，结论有效。方案合理可行，效果好，有创新意识。工作量饱满，完成质量高。 | | | 20 | 18 | | 16 | 14 | | 12 | <12 |  |
| 评审成绩：  □优秀（100-90） □良好（89-80） □中等（79-70） □及格（69-60） □不及格（<60） | | | | | | | | | | | | | |
| 指导教师签名 | |  | | 职称 |  | 时间 | | | 年 月 日 | | | | | |

目录

[**摘要** 4](#_Toc202384994)

[第1章 绪论 5](#_Toc202384995)

[1.1 实训背景与意义 5](#_Toc202384996)

[1.2 实训内容 5](#_Toc202384997)

[第2章 Git 5](#_Toc202384998)

[2.1基本原理 5](#_Toc202384999)

[2.2实验步骤 5](#_Toc202385000)

[2.3实验结果与分析 6](#_Toc202385001)

[第3章Docker 6](#_Toc202385002)

[3.1基本原理 6](#_Toc202385003)

[3.2实验步骤 6](#_Toc202385004)

3.3实验结果与分析…………………………………………………………………………..6

[第4章GDB 7](#_Toc202385005)

[4.1基本原理 7](#_Toc202385006)

[4.2实验步骤 7](#_Toc202385007)

[4.3实验结果与分析 7](#_Toc202385008)

[第5章总结 7](#_Toc202385009)

[参考文献 8](#_Toc202385010)

**摘要**

本实训报告主要介绍了Git、Docker和GDB三种通用工具的操作和应用。

Git 是一款分布式版本控制系统，用于代码版本管理和协作开发。实训内容包括Git基础操作、分支管理和远程协作，旨在让学生掌握代码版本控制的全流程，理解现代软件开发的工程化规范。

Docker 是一种容器化技术，用于创建、部署和管理容器。实训内容包括Docker基础操作、镜像管理、数据卷管理和网络管理，旨在让学生掌握容器化部署的流程，理解容器化技术在快速部署、环境一致性方面的优势。

GDB 是一款强大的程序调试工具，用于定位程序错误和分析程序逻辑。实训内容包括GDB基础操作、调试控制、程序状态检查和高级调试功能，旨在让学生掌握程序动态调试的技巧，培养软件漏洞定位与代码逻辑还原能力。

关键词：Git，Docker，GDB等。

# 第1章 绪论

## 1.1 实训背景与意义

1.1.1实训背景

网络安全实战化需求升级：工具链成为安全能力核心：现代网络安全工作（如漏洞分析、渗透测试、应急响应）高度依赖工具链的熟练使用，而Git、DDB、GDB等工具是开发、调试、协作的基础支撑；企业用人需求变化：企业招聘安全工程师时，普遍要求掌握版本控制（Git）、容器化（Docker）、逆向调试（GDB）等技能，传统课程体系却常忽视这些“通用工具”的教学。

技术发展与攻防场景演变：Dev SecOps的普及：安全左移（Shift Left）趋势下，安全人员需嵌入开发流程，Git（代码管理）、Docker（环境隔离）成为必备技能。

漏洞分析复杂化：二进制漏洞利用（如堆溢出）依赖GDB等调试工具，而物联网、云原生安全进一步要求容器（Docker）环境下的分析能力;教学与行业实践的脱节:学生常面临“懂原理但不会操作”的困境（如理解缓冲区溢出原理但无法用GDB动态调试），亟需通过实训填补工具操作短板;开源协作成为安全研究主流（如GitHub漏洞PoC共享），但学生缺乏Git协作开发的经验。

1.1.2实训意义

该课程通过聚焦Git、Docker、GDB等“通用但关键”的工具，解决网络安全领域“重理论轻实操”的痛点，使得学生拥有工具链思维，将离散工具整合为完整工作流；实战能力，直接应对漏洞分析、云安全、逆向工程等现实场景；职业竞争力：满足企业对“即插即用”型安全人才的需求。

这些工具不仅是技术手段，更是现代安全工程师的“生存技能”，其意义远超工具本身，而是构建安全工程化能力的基石。

## 1.2 实训内容

1.2.1掌握Git的文件状态管理与分支协作、Docker的镜像构建与容器编排，完成从代码版本控制到应用容器化部署的全流程实践，理解现代软件开发的工程化规范。

1.2.2程序调试与逆向分析：通过GDB掌握程序动态调试技巧，结合IDA Pro完成二进制的静态分析与动态追踪，培养软件漏洞定位与代码逻辑还原能力。

# 第2章 Git

## 2.1基本原理

2.1.1分布式版本控制：每个开发者都拥有完整的代码仓库，可以在本地进行版本控制和提交。

2.1.2快速高效：Git使用哈希函数来存储文件和版本信息，因此操作快速高效。

2.1.3强大的分支管理：Git的分支概念十分灵活，可以轻松创建、合并和管理分支，支持同时存在多个独立开发线。

## 2.2实验步骤

2.2.1Git基础操作，创建git账户和仓库，如下图所示；利用’git init’命令初始化本地仓库：利用‘git add’命令添加文件到暂存区；提交文件到本地仓库，使用‘git commit’命令，撰写提交信息；使用‘git log’指令查看提交历史。

2.2.2分支管理操作：创建分支，使用‘git branch’命令；切换分支，使用‘git checkout’命令；合并分支，使用‘git merge’命令；删除分支，使用‘git branch -d’命令。

2.2.3远程仓库操作：添加远程仓库地址，使用‘git remote add’命令；从远程仓库克隆代码，使用‘git clone’命令：将本地更改推送到远程仓库，使用‘git push’命令：从远程仓库拉取最新代码，使用‘git pull’命令。

标签管理：创建标签，用于标记重要版本，如发布版；查看标签列表，使用‘git tag’命令；推送标签到远程仓库，使用‘git push—tags’命令。

## 2.3实验结果与分析

分布式特性：本地仓库完整独立，支持离线操作（如提交、分支管理）。

高效性：哈希算法和增量存储使操作快速（如 `git status` 响应速度显著优于集中式工具）。

分支灵活性：多分支并行开发未引发性能下降，合并流程清晰。

# 第3章Docker

## 3.1基本原理

Docker 的原理是使用 Linux 内核和内核功能（例如 Cgroup 和命名空间）来分隔进程，以便各进程相互独立运行。这种独立性使得容器可以独立运行多种进程、多个应用，同时保持各个独立系统的安全性。

## 3.2实验步骤

3.2.1Docker基础操作：安装Docker并确保其正常运行；使用‘docker version’命令查看；拉取镜像，使用‘docker pull’命令；运行容器，使用‘docker run’命令，指定镜像和端口映射等参数；查看容器状态，使用‘docker ps’命令；停止、启动和删除容器，使用‘docker stop’、‘docker start’和‘docker rm’命令。

3.2.2Docker镜像管理：查看本地镜像列表，使用‘images docker’命令；删除镜像，使用‘docker

rmi’命令；构建自定义镜像，编写Docker file并使用‘docker build’命令；打标签并推送镜像到Docker Hub，使用‘docker tag’和‘docker push’命令。

3.2.3Docker数据卷管理：创建和挂载数据卷，确保容器数据的持久化；查看和删除数据卷，使用‘docker volume is’和‘docker volume rm’命令

3.2.4Dockers网络管理：创建自定义网格，使用‘docker network create’命令；查看网络列表，使用‘docker network is’命令；将容器连接到指定网络，实现容器间的通信。

**3.3实验结果与分析**

轻量化：容器启动速度（<1 秒）远快于虚拟机。

隔离性：进程、文件系统、网络命名空间有效隔离，互不干扰。

可移植性：镜像构建后可在任意 Docker 环境中运行。

实验证实 Docker 在快速部署、环境一致性方面的核心价值，适合微服务与 CI/CD 场景，但需注意资源管理和安全配置。

# 第4章GDB

## 4.1基本原理

GDB的原理是通过ptrace系统调用来控制和监控被调试程序的执行过程。GDB利用编译器生成的调试信息，如源代码和符号表，来定位程序的位置和变量的值。GDB还可以通过读取和修改被调试程序的内存地址，来查看和改变变量的值。GDB的print命令或p命令可以用来显示变量的值。

## 4.2实验步骤

4.2.1 GDB基础操作：安装GDB，在Linux系统中安装GDB调试工具，确保其正常运行：启动GDB，使用‘GDB’命令启动调试器，后面可以跟要调试的可执行文件名；查看版本，使用‘gdb –version’命令查看当前安装的GDB版本信息;加载程序,在GDB中使用’file’命令加载需要调试的可执行程序;运行程序,使用‘run’命令（或简写‘r’）开始执行被调试的程序。

4.2.2 调试控制操作：设置断点，使用`break`命令（或简写`b`）在指定函数或行号处设置断点，使用`info breakpoints`查看所有断点信息；执行控制，使用`next`（或`n`）单步执行（不进入函数），使用`step`（或`s`），单步执行（进入函数），使用`continue`（或`c`）继续执行直到下一个断点；中断执行，在程序运行时按Ctrl+C可中断程序执行。

4.2.3程序状态检查：查看变量，使用`print`命令（或简写`p`）查看变量值，使用`display`命令持续显示变量值；查看堆栈，使用`backtrace`（或`bt`）查看函数调用堆栈，使用`frame`命令切换堆栈帧；查看寄存器，使用`info registers`查看CPU寄存器状态；查看内存，使用‘x’命令检查指定内存地址的内容。

4.2.4 高级调试功能：条件断点，使用`break ... if condition`设置条件断点；观察点，使用`watch`命令设置数据观察点，当变量值改变时中断；修改值，使用`set variable`命令在调试过程中修改变量值；多线程调试，使用`info threads`查看所有线程，使用`thread`命令切换调试线程。

4.5 结束调试：退出GDB，使用`quit`命令（或简写`q`）退出调试器；保存断点，可使用`save breakpoints`命令将断点信息保存到文件；加载断点，使用`source`命令加载之前保存的断点配置。

## 4.3实验结果与分析

精准调试:相比 `printf` 调试法，GDB 能更高效地定位程序错误，减少试错时间。

动态调整：修改变量、跳转执行等特性使调试过程更灵活，适用于复杂逻辑的测试。

GDB 作为强大的命令行调试工具，在程序错误诊断、内存分析和多线程调试方面具有不可替代的优势。尽管其学习成本较高，但对于系统开发和性能关键型应用的调试仍是首选工具。结合 `-g` 编译选项和脚本化调试（`.gdbinit`），可进一步提升调试效率。

# 第5章总结

通过本次《通用工具操作实训》的学习和实践，我对 Git、Docker 和 GDB 这三个重要的工具有了更深入的理解和掌握。以下是我对本次实训的总结和体会：

5.1 Git 版本控制

5.1.1 Git 的分布式版本控制特性极大地提高了代码管理的效率和灵活性，让我能够轻松进行代码的版本回溯、分支管理和远程协作。

5.1.2通过学习 Git 的基础操作、分支管理和远程仓库操作，我掌握了 Git 的核心功能，能够有效地管理代码版本，并与团队成员进行协作开发。

5.2 Docker 容器化

5.2.1 Docker 的轻量级虚拟化技术为应用部署提供了极大的便利，容器化的应用可以轻松地在不同环境中运行，保证了应用环境的一致性。

5.2.2通过学习 Docker 的基础操作、镜像管理、数据卷管理和网络管理，我掌握了 Docker 的核心功能，能够构建和运行容器化的应用，并进行环境配置和数据管理。

5.3 GDB 程序调试

5.3.1GDB的强大调试功能让我能够深入理解程序的运行过程，有效地定位和解决程序中的错误。

5.3.2通过学习GDB 的基础操作、调试控制、程序状态检查和高级调试功能，我掌握了 GDB 的核心功能，能够进行程序的单步执行、断点设置、变量查看和内存分析。

5.4实训意义

本次实训让我将 Git、Docker 和 GDB 这三个工具整合到一起，形成了一套完整的工具链，能够有效地支持软件开发、部署和调试的各个环节。通过实训，希望这些工具的使用方法能被我掌握，希望我能够将这些工具灵活地运用到实际项目中，提高开发效率和代码质量。在未来的学习和工作中，我将继续深入学习和实践 Git、Docker 和 GDB，并探索其他相关的工具和技术，不断提升自己的软件开发和调试能力。

# 参考文献

[1]《 Git 权威指南》，人民邮电出版社，孙振鹏编著；

[2]《 Docker 容器与容器云》，机械工业出版社，马全一、韩冀中著；

[3]《 GDB 调试权威指南》，电子工业出版社，张健编著；

[4]《软件调试（卷1)》，电子工业出版社，张银奎编著；

[5]《逆向工程与软件安全》，科学出版社，钱思宇、戴林编著．