

Министерство науки и высшего образования Российской Федерации
Федеральное государственное образовательное учреждение высшего
образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ
И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)
Кафедра комплексной информационной безопасности
электронно-вычислительных систем (КИБЭВС)

УТВЕРЖДАЮ

Заведующий каф. КИБЭВС

_____ А.А. Шелупанов

«____» _____ 2018г.

РАЗРАБОТКА И РАЗВИТИЕ СИСТЕМЫ ДЛЯ ОБУЧЕНИЯ И ПРОВЕДЕНИЯ
ПРАКТИК ПО ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ
Отчет по групповому проектному обучению
Проект КИБЭВС-1808

Ответственный исполнитель
студент гр.725

_____ Д.Г. Дудкин

«____» _____ 2018г.

Научный руководитель
младший научный сотрудник каф.
КИБЭВС

_____ Д.С. Никифоров

«____» _____ 2018г.

Томск 2018

РЕФЕРАТ

Групповая проектная работа, 42 страницы, 16 рисунков, 11 источников, 3 приложения.

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ, FREEHACKQUEST, АВТОМАТИЗИРОВАННАЯ СИСТЕМА, КОНТЕЙНЕРИЗАЦИЯ, LXD, C++, LATEX, GNU DEBUGGER, PLANTUML

Объектом разработки является система для обучения и проведения практических занятий по информационной безопасности.

Цель работы – разработка платформы для проведения практических занятий по информационной безопасности в формате тестирования на проникновение внутри среды, моделирующей работу информационной инфраструктуры организации.

В текущем семестре была продолжена работа с контейнерами и в результате были реализованы асинхронные операции для работы с контейнерами, развернуты сервисы внутри контейнера, настроен порт для контейнера, а также исправлены ошибки с подсчетом рейтинга участников и изменением имен пользователей, сформирован учебный материал и разработано практическое задание «Smash The Stack – IO» для FreeHackQuest, изучена система компьютерной верстки документов \LaTeX , с использованием которой подготовлены следующие разделы документации по системе: общее представление о системе, структура системы, описание стека технологий и процессов системы.

В качестве инструментария для выполнения данной работы были использованы: язык программирования C++ с библиотекой STL, \LaTeX , PlantUml, Gnu Debugger.

Список исполнителей

Дудкин Данил Геннадьевич, гр. 725, каф. КИБЭВС

Косенко Екатерина Игоревна, гр. 716, каф. КИБЭВС

Михайлова Александра Андреевна, гр. 725, каф. КИБЭВС

Ушев Сергей Дмитриевич, гр. 725, каф. КИБЭВС

Министерство науки и образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Групповое проектное обучение

УТВЕРЖДАЮ

Зав. кафедрой КИБЭВС
Шелупанов Александр Александрович

«__» _____ 20__ г.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
на выполнение проекта № КИБЭВС-1808

1. Основание для выполнения проекта: приказ № 208ст от 24.01.2018.
2. Наименование проекта: разработка и развитие системы для обучения и проведения практик по информационной безопасности.
3. Цель проекта: разработать платформу для проведения практик по информационной безопасности в формате тестирования на проникновение внутри среды, моделирующей работу информационной инфраструктуры организации.
4. Основные задачи проекта на этапах реализации: реализация развертывания сервиса внутри контейнера, настройки подсети-порта для контейнера и импортирования/экспортирования задач и настроек сервиса, исправление ряда существующих ошибок, доработка и улучшение системы, добавление возможности оценки квестов, разработка документации по системе.
5. Научная новизна проекта: генерация компьютерной сети для каждого участника (либо команды) и устройств в сети, выполняющих какую-либо работу и обменивающихся информацией друг с другом. Идея состоит в том, чтобы смоделировать реальные сети организаций и предоставить участнику доступ к ней либо возможность получения доступа к сети. После чего участнику предлагается попытка повысить свои привилегии в сети, получить доступ к устройствам, получить доступ к информации или получить контроль над сетью.
6. Планируемый срок реализации: планируемый срок создания системы 2 года.
7. Целевая аудитория (потребители): пользователи, заинтересованные в получении практических навыков в сфере информационной безопасности.

8. Заинтересованные стороны: Факультет безопасности.
9. Источники финансирования и материального обеспечения: разработка ведется за счет личных средств группы.
10. Ожидаемый результат (полученный товар, услуга): в результате работы над данным проектом будет создана система, позволяющая создавать и эксплуатировать компьютерные сети с целью оттачивания навыков проведения тестов на проникновение.
11. Руководитель проекта: Никифоров Д.С., МНС.
12. Ответственный исполнитель проекта: Дудкин Данил Геннадьевич гр. 725.
13. Члены проектной группы:

Дудкин Данил Геннадьевич гр. 725
 Косенко Екатерина Игоревна гр. 716
 Михайлова Александра Андреевна гр. 725
 Ушев Сергей Дмитриевич гр. 725
14. Место выполнения проекта: ауд. 805.
15. Календарный план выполнения проекта

Таблица 1 – Состав и содержание работ по созданию (развитию) объекта разработки и ввода в эксплуатацию

№ этапа	Наименование этапа	Содержание работы	Сроки выполнения		Ожидаемый результат этапа
			Начало	Окончание	
1	Подготовительный	Развертывание тестового сервиса внутри контейнера Исправление ошибок и доработка системы Изучение набора компьютерной верстки LaTeX	06.09.2018	04.10.2018	Работоспособный тестовый сервис внутри контейнера Исправленные ошибки Навыки работы в LaTeX
2	Начало реализации	Настройка подсети - порта для контейнера Исправление ошибок и доработка системы Начало подготовки документации по системе	04.10.2018	15.11.2018	Настроенная подсеть - порт для контейнера Исправленные ошибки Разделы документации «Введение» и «Общее представление о системе»

Продолжение таблицы 1

№ этапа	Наименование этапа	Содержание работы	Сроки выполнения		Ожидаемый результат этапа
			Начало	Окончание	
3	Завершение реализации	Асинхронная обработка операций с контейнером Разработка заданий для сайта Завершение подготовки документации по системе	15.11.2018	06.12.2018	Реализация асинхронной обработки операций с контейнером Задания для сайта Раздел документации «Описание отдельных компонентов системы и логики»
4	Отчетность	Написание отчетной документации по проделанной работе	06.12.2018	20.12.2018	Отчет по проделанной работе

« ____ » _____ 20 ____ г.

Руководитель проекта:

_____ Никифоров Д.С.

« ____ » _____ 20 ____ г.

Члены проектной группы:

_____ Дудкин Данил Геннадьевич

_____ Косенко Екатерина Игоревна

_____ Михайлова Александра Андреевна

_____ Ушев Сергей Дмитриевич

« ____ » _____ 20 ____ г.

Содержание

Введение	8
1 Задачи текущего семестра	9
2 Описание использованных инструментов	10
2.1 CMake	10
2.2 STL	10
2.3 LXD	11
2.4 LaTeX	11
2.5 PlantUML	12
2.6 GNU Debugger	13
3 Общее представление о системе	15
3.1 Описание стека технологий системы	16
3.2 LXD	17
4 Работа с контейнерами	19
4.1 Развертывание тестового сервиса внутри контейнера	19
4.2 Асинхронная обработка запроса операций с контейнером	22
4.3 Настройка подсети-порта для контейнера	24
5 Разработка учебного материала для курса ТиМП	25
5.1 Формирование лекции для курса ТиМП	25
5.2 Разработка практического задания для FNQ	28
6 Результаты работы	29
Заключение	30
Список сокращений, обозначений, терминов и определений	32
Список использованных источников	33
Приложение А (Справочное) Статья для Интернет-конференции ГПО	34
Приложение Б (Справочное) Статья для Интернет-конференции ГПО	38
Приложение В (обязательное) Компакт-диск	42

Введение

FreeHackQuest (FNQ) – это платформа для обучения и проведения соревнований по компьютерной безопасности. FNQ включает в себя учебник, различные задачи для решения, связанные с администрированием, криптографией, компьютерно-криминалистической экспертизой, стеганографией и многими другими направлениями информационной безопасности.

Ключевая концепция проведения обучения и соревнований заключается в генерации компьютерной сети, состоящей из сервисов и устройств, выполняющих какую-либо работу и обменивающихся информацией друг с другом. Идея состоит в том, чтобы смоделировать реальные сети предполагаемых организаций и предоставить участникам соревнований доступ (или возможность получения доступа) к данной сети. Затем каждому обучающемуся или игроку предлагается получить доступ к устройствам и информации или контроль над сетью.

Актуальность нашего проекта определяется колоссальным ростом скорости развития информационных технологий и постоянной потребностью в практической подготовке специалистов в сфере информационной безопасности.

Целью данного проекта является разработка платформы для проведения практически занятий по информационной безопасности в формате тестирования на проникновение внутри среды, моделирующей работу информационной инфраструктуры организации.

1 Задачи текущего семестра

- реализация асинхронной обработки операций с контейнером;
- развертывание сервисов внутри контейнера, настройка порта для контейнера;
- изучение набора компьютерной верстки документов \LaTeX , и дальнейшее использование в рамках проекта для подготовки документации: общее представление о системе, структура системы, описание стека технологий и процессов системы;
- исправление имеющихся ошибок подсчета рейтинга участников, изменения имен пользователей;
- написание учебного материала и разработка практических заданий для FHQ.

2 Описание использованных инструментов

При выполнении данной проектной работы использовались следующие инструменты: кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода CMake, стандартная библиотека шаблонов C++ STL, контейнерный гипервизор LXD, система подготовки (верстки) документов L^AT_EX, инструмент для создания диаграмм PlantUML.

2.1 CMake

CMake – это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода.

CMake является по-настоящему кроссплатформенным генератором проектов, позволяющим создавать единые описания проектов для Linux и других Unix-систем (включая Mac OS X) и Windows. Кроме того, CMake стремится максимально использовать фирменные средства генерации сборочных файлов, обладает интеллектуальной системой поиска инструментов сборки и библиотек на конкретной платформе и автоматического конфигурирования. Благодаря этому, система CMake сама устанавливает многие параметры сборочных файлов, которые в других системах управления сборкой приходится устанавливать вручную [1].

2.2 STL

STL (Standard Template Library) – набор согласованных между собой обобщенных алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

Механизм шаблонов встроен в компилятор C++, чтобы дать возможность программистам делать свой код короче за счет обобщенного программирования. Среди существующих стандартных библиотек, реализующих эти механизмы, STL является самой эффективной библиотекой C++ на сегодняшний день [2].

2.3 LXD

LXD (Linux Container Daemon) – диспетчер контейнеров нового поколения, обеспечивающий опыт работы, схожий с виртуальными машинами, используя вместо них Linux-контейнеры.

LXD базируется на API LXC (Application Programming Interface Linux Containers) и ее привязках для различных языков программирования, но при этом инструменты и шаблоны заменены новыми. К числу преимуществ LXD можно отнести его относительную простоту, безопасность по умолчанию, возможность работы с контейнерными образами при создании контейнера (а не с шаблонами) и оптимизированная поддержка состояний контейнера checkpoint и restore для живой миграции [3].

2.4 LaTeX

LaTeX – это собирательное название для системы подготовки (верстки) документов. Она включает набор инструментов, которые из текстовых файлов, записанных с использованием специального языка разметки формируют готовые к печати документы (как правило в формате PDF). Собственно, TeX – это низкоуровневый язык разметки и программирования который лежит в основе этой системы [4].

Особенности LaTeX:

- удобное средство для написания технических отчетов и различной документации;
- стиль, шрифты, оформление таблиц, рисунков и т.д. согласованы во всём документе;
- большие документы можно разбивать на несколько файлов и работать с ними отдельно, в том числе с использованием систем управления версиями;
- легко создаются алфавитные указатели, библиографические списки, сноски, ссылки;
- удобно включать такие вставки, как исходный код, математические формулы.

2.5 PlantUML

PlantUML – это инструмент с открытым исходным кодом, позволяющий пользователям создавать диаграммы UML с обычного текстового языка. Язык PlantUML является примером специфического для приложения языка. Он использует программное обеспечение Graphviz для выкладки своих диаграмм.

В сравнении с другими известными инструментами для создания диаграмм такими, как Microsoft Visio, Rational Rose, можно выделить следующие преимущества использования PlantUML:

- создание диаграмм в виде текста (пример диаграммы приведен ниже на рисунке 1, ее графическое отображение на рисунке 12), соответственно в любом текстовом редакторе можно произвести рефакторинг, что очень удобно, также текстовый формат дает возможность легко организовывать групповую работу и осуществлять отслеживание изменений в системе контроля версий;
- на создание диаграмм требуется меньше времени, чем при создании в любых визуальных редакторах;
- гибкий подход к нотации UML – поддерживая все основные ее элементы, он дает пользователю множество возможностей для их свободного использования в диаграмме, например, очень удобной является поддержка использования вики-разметки в содержимом элементов;
- не позволяет создавать больших диаграмм, но это не совсем ограничение – он распределяет элементы самостоятельно и, естественно, при большом количестве элементов результат начинает быть сомнительным, и лучше диаграмму разделить или уменьшить количество несущественных деталей;
- диаграммы PlantUML можно генерировать автоматически [5].

```

@startuml "Диаграмма последовательности для создания контейнера"
skinparam Monochrome true
actor admin as "Администратор"

admin ->> FHQ as "FreeHackQuest" : \tFHQ API\nСоздание контейнера
FHQ ->> LXD : \tLXD REST\nСоздание контейнера
alt Успешно
    LXD ->> FHQ : \tLXD REST\nОперация началась
else Ошибка
    LXD ->> FHQ : \tLXD REST\nОперация не может\nначаться
end
alt Успешно
    FHQ ->> admin : \tFHQ API\nОперация началась
else Ошибка
    FHQ ->> admin : \tFHQ API\nОперация не может\nначаться
end
alt Успешно
    LXD ->> FHQ : \tLXD REST\nКонтейнер создан
else Ошибка
    LXD ->> FHQ : \tLXD REST\nОперация завершилась\nс ошибкой
end
alt Успешно
    FHQ ->> admin : \tFHQ API\nКонтейнер создан
else Ошибка
    FHQ ->> admin : \tFHQ API\nОперация завершилась\nс ошибкой
end
@enduml

```

Рисунок 1 – Текст диаграммы

2.6 GNU Debugger

GNU Debugger (GDB) – переносимый отладчик проекта GNU, способный работать на многих UNIX-подобных системах и производить отладку программ, написанных на различных распространенных компилируемых языках программирования, включая языки Си и C++. GDB является свободным программным обеспечением, распространяемым по лицензии GPL (GNU General Public License) [6].

Данный отладчик предлагает широкий список функций, позволяющих контролировать ход выполнения программы:

- пошаговая отладка с возможностью задать точку остановки программы в требуемом месте или по достижению определенного условия;
- исследование и изменение значений внутренних переменных программы;

- вызов внутренних функций программы независимо от ее обычного поведения;
- предоставление информации о том, что произошло в момент, когда программа остановилась.

3 Общее представление о системе

FreeHackQuest – это платформа для обучения, проведения практических занятий и соревнований по компьютерной безопасности в формате тестирования на проникновение внутри среды, моделирующей работу информационной инфраструктуры организации. FHQ включает в себя учебник, различные задачи для решения, связанные с администрированием, криптографией, компьютерно-криминалистической экспертизой, стеганографией и многими другими направлениями информационной безопасности.

FreeHackQuest представляет собой клиент–серверную многофункциональную систему с подсистемами и состоит из следующих основных компонентов:

- сервер – отвечает за обработку запросов со стороны клиента;
- LXD (сервер виртуальных машин) – предназначен для размещения контейнеров, необходимых, для изолированного выполнения сервисов с уязвимостями;
- MySQL (сервер баз данных) – отвечает за хранение и предоставление информации;
- клиент – отвечает за формирование запросов подсистеме сервера и представление ответов со стороны сервера;
- административный клиент – отвечает за управление системой.

Структура системы представлена на рисунке 2.

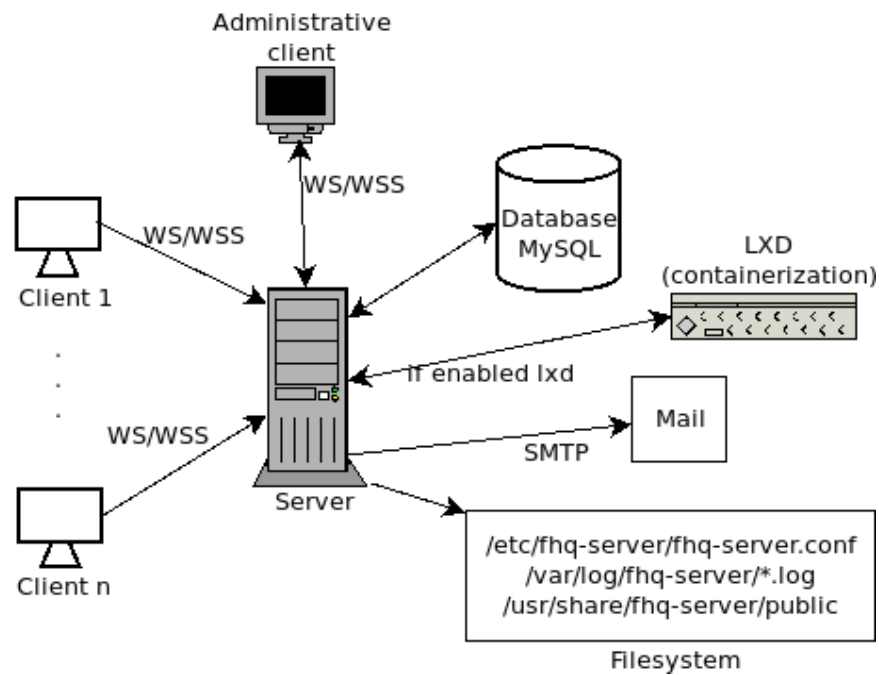


Рисунок 2 – Структура системы

3.1 Описание стека технологий системы

Архитектура клиент-сервер определяет лишь общие принципы взаимодействия между компьютерами, детали взаимодействия определяют различные протоколы. Данная концепция нам говорит, что нужно разделять машины в сети на клиентские, которые делают запрос, и на серверные, которые отвечают на него. При этом взаимодействие всегда начинается клиент, а правила, по которым происходит взаимодействие описывает протокол.

За взаимодействие между сервером и клиентами, в том числе с административным клиентом, отвечает протокол WebSocket (WS/WSS). Данный протокол связи поверх TCP-соединения предназначен для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

MySQL как СУБД, если рассматривать её в рамках клиент-серверной архитектуры, является прежде всего программой-сервером, который может получать от различных клиентов, задачи по работе с данными (посредством SQL-запросов).

Этими клиентами могут быть:

- собственная-программа клиент работающая в командной строке;

- скрипт, написанный на каком-нибудь языке программирования, например, на PHP (так называемый «запрос из приложения»);
- программа для работы с данными в графическом интерфейсе.

Взаимодействие с e-mail осуществляется посредством SMTP-протокола – протокола для исходящей связи по электронной почте. Простой протокол передачи почты (SMTP), используется для связи с удаленным сервером и последующей отправки сообщений с локального клиента на удаленный сервер, и в конечном итоге на сервер получателя сообщений. SMTP используется исключительно для отправки сообщений.

Если выполняется условие `if enabled lxd`, то сервер обращается к серверу виртуальных машин (LXD).

3.2 LXD

Система виртуализации необходима для изолированного выполнения сервисов с уязвимостями, так как участники могут скомпрометировать машину, на которой развернута моделируемая инфраструктура организации. Система виртуализации должна обладать сетевым управлением, такое требование позволяет размещать контейнеры на отдельной от backend сервера машине. Например, у хостинг-провайдеров можно арендовать необходимые вычислительные мощности на время проведения практики и обучения.

В качестве системы виртуализации выбран LXD (LXC). LXD это то, что называется легковизор. Ядром LXD является демон, который предлагает API REST для управления контейнерами подобно виртуальным машинам. Чтобы создавать, управлять и мониторить множество уязвимых сервисов с определенными настройками сети прямо в административной странице FreeHackQuest, появилась необходимость интеграции FreeHackQuest с LXD.

Система оркестрации, предназначенная для развертывания сервисов с уязвимостями и запуска ботов, имитирующих поведение пользователей позволяет отправлять HTTP GET, POST, PUT, DELETE запросы серверу LXD, а также создавать, получать, запускать, останавливать и удалять контейнеры,

узнавать информацию о контейнере.

Система оркестрации является частью подсистемы сервера FreeHackQuest и имеет структуру взаимодействия, показанную на рисунке 3.

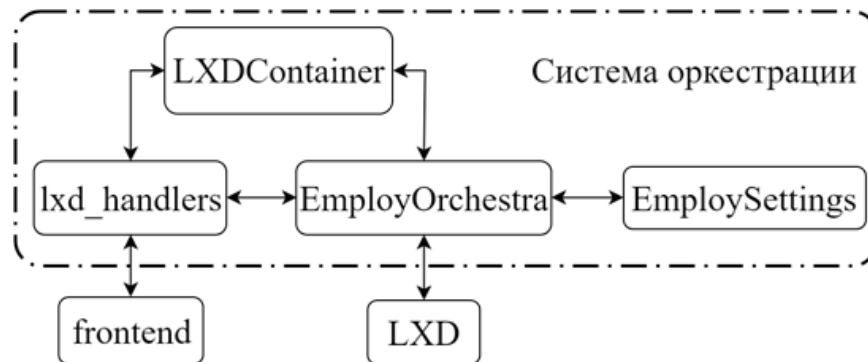


Рисунок 3 – Структура взаимодействия системы оркестрации

4 Работа с контейнерами

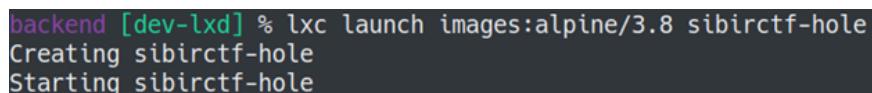
4.1 Развертывание тестового сервиса внутри контейнера

В ходе работы по проекту развернут сервис The Hole, взятый с соревнований по информационной безопасности SibirCTF-2018. Под развертыванием сервиса понимается его настройка и запуск внутри контейнера.

Для образа контейнера была выбрана операционная система Alpine 3.8. Она была выбрана благодаря небольшому размеру образа и ориентированности на безопасность. Размер чистого образа Alpine с чистого старта 8 Мбайт оперативной памяти в отличие от Ubuntu 16.04 с размером 170 Мбайт. Достигается это за счет минимально необходимого набора системных и прикладных программ. Все дополнительные необходимые программы можно установить с помощью встроенного пакетного менеджера apk.

Последовательность действий по развертыванию контейнера:

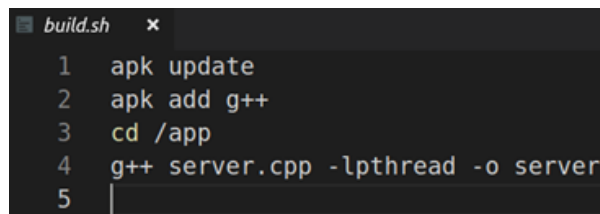
- скачивание образа и создание контейнера с помощью команды `lxc launch images:alpine/3.8 sibirctf-hole` (рис. 4);



```
backend [dev-lxd] % lxc launch images:alpine/3.8 sibirctf-hole
Creating sibirctf-hole
Starting sibirctf-hole
```

Рисунок 4 – Результат команды `lxc launch`

- подготовка сервиса к загрузке в контейнер;
- создание сценария командной строки по установке необходимых зависимостей системных и прикладных программ и сборки сервиса: сценарий командной строки по сборке сервиса состоит из установки зависимостей и компиляции исходных кодов сервиса на C++ (рис. 5), сценарий запуска сервиса может состоять из запуска необходимых программ внутри контейнера, но данный сервис не зависит от других программ, поэтому необходимо только выполнить запуск сервиса (рис. 6);

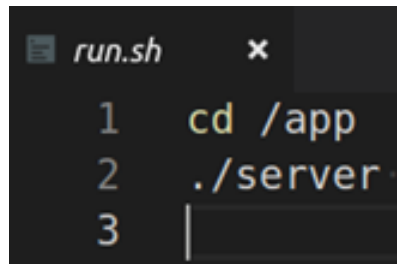


```

1  apk update
2  apk add g++
3  cd /app
4  g++ server.cpp -lpthread -o server
5

```

Рисунок 5 – Сценарий сборки сервиса



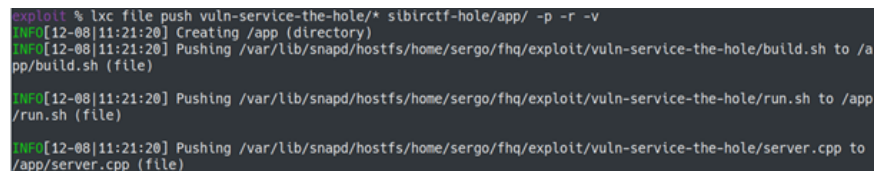
```

1  cd /app
2  ./server
3

```

Рисунок 6 – Сценарий запуска сервиса

— загрузка подготовленного сервиса в контейнер с помощью команды `lxc file push vuln-service-the-hole/* vuln-hole/app/ -p -r -v` (рис. 7);



```

exploit % lxc file push vuln-service-the-hole/* sibirctf-hole/app/ -p -r -v
[INFO][12-08|11:21:20] Creating /app (directory)
[INFO][12-08|11:21:20] Pushing /var/lib/snapd/hostfs/home/sergo/fhq/exploit/vuln-service-the-hole/build.sh to /a
pp/build.sh (file)
[INFO][12-08|11:21:20] Pushing /var/lib/snapd/hostfs/home/sergo/fhq/exploit/vuln-service-the-hole/run.sh to /app
/run.sh (file)
[INFO][12-08|11:21:20] Pushing /var/lib/snapd/hostfs/home/sergo/fhq/exploit/vuln-service-the-hole/server.cpp to
/app/server.cpp (file)

```

Рисунок 7 – Загрузка подготовленного сервиса

— запуск сборки сервиса с помощью команды `lxc exec sibirctf-hole sh /app/build.sh` (рис. 8);

```
backend [dev-lxd] % lxc exec sibirctf-hole sh /app/build.sh
fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.8/community/x86_64/APKINDEX.tar.gz
v3.8.1-142-g179c136670 [http://dl-cdn.alpinelinux.org/alpine/v3.8/main]
v3.8.1-142-g179c136670 [http://dl-cdn.alpinelinux.org/alpine/v3.8/community]
OK: 9550 distinct packages available
(1/14) Installing libgcc (6.4.0-r9)
(2/14) Installing libstdc++ (6.4.0-r9)
(3/14) Installing binutils (2.30-r5)
(4/14) Installing gmp (6.1.2-r1)
(5/14) Installing isl (0.18-r0)
(6/14) Installing libgomp (6.4.0-r9)
(7/14) Installing libatomic (6.4.0-r9)
(8/14) Installing pkgconf (1.5.3-r0)
(9/14) Installing mpfr3 (3.1.5-r1)
(10/14) Installing mpc1 (1.0.3-r1)
(11/14) Installing gcc (6.4.0-r9)
(12/14) Installing musl-dev (1.1.19-r10)
(13/14) Installing libc-dev (0.7.1-r0)
(14/14) Installing g++ (6.4.0-r9)
Executing busybox-1.28.4-r1.trigger
OK: 162 MiB in 32 packages
```

Рисунок 8 – Сборка сервиса

– открытие TCP-порта 5003 для доступа к сервису с помощью команды `lxc config device add sibirctf-hole port5003 proxy listen=tcp:0.0.0.0:5003 connect=tcp:localhost:5003` (рис. 9);

```
exploit % lxc config device add sibirctf-hole port5003 proxy listen=tcp:0.0.0.0:5003 connect=tcp:localhost:5003
Device port5003 added to sibirctf-hole
```

Рисунок 9 – Открытие порта 5003

– запуск сервиса внутри контейнера с помощью `lxc exec sibirctf-hole sh /app/run.sh` (рис. 10);

```
backend [dev-lxd] % lxc exec sibirctf-hole sh /app/run.sh
SYSTEM: Warning, 'users' file not found! New file will be created.
SYSTEM: Server started.
SYSTEM: Messenger started.
```

Рисунок 10 – Запуск сервиса

– проверка доступа к сервису с локальной машины с помощью `nc localhost 5003` (рис. 11);

```

exploit % nc localhost 5003
Welcome to Kronbash ltd overpowered and oversecured chat-server
TTTTT H H EEEEE H H 000 L EEEEE
T H H E H H 00 00 L E
T H H H H H EEEEE H H 0 0 L EEEEE
T H H E H H 00 00 L E
T H H EEEEE H H 000 LLLLL EEEEE
To call help send /HELP

```

Рисунок 11 – Проверка доступа к сервису

4.2 Асинхронная обработка запроса операций с контейнером

В ходе работы над проектом добавлена возможность асинхронно обрабатывать запросы по созданию, запуску, остановке и удалению контейнеров. Необходимо было обрабатывать асинхронные команды LXD, не блокируя основной поток выполнения backend-сервера FreeHackQuest. Рассматривались разные подходы к реализации такие, как `feature`, `promise`, `task` из стандартной библиотеки C++ или сопрограммы из Boost. Но выбран шаблон проектирования пул потоков (Thread Pool) из-за его простоты и достаточности для решения поставленной задачи. Он заранее запускает нужное количество потоков, которые выполняют задачи, поступающие в очередь задач.

В ходе работы над задачей возникла проблема отправки сообщения frontend части клиентской стороны – WebSocket сервер не позволял отправлять сообщение из другого потока. Данную проблему удалось решить с помощью концепции сигналов и слотов. Из потока выполнения задачи в основной поток отправлялся сигнал с сообщением нужному клиенту и в основном потоке слот обрабатывал поступивший сигнал. Тем самым удалось обойти ограничение отправки WebSocket сообщений только из основного потока выполнения.

Наглядное представление асинхронной обработки запроса операций с контейнером, а конкретной операция создания контейнера, можно увидеть в виде UML диаграммы последовательности на рисунке 12.

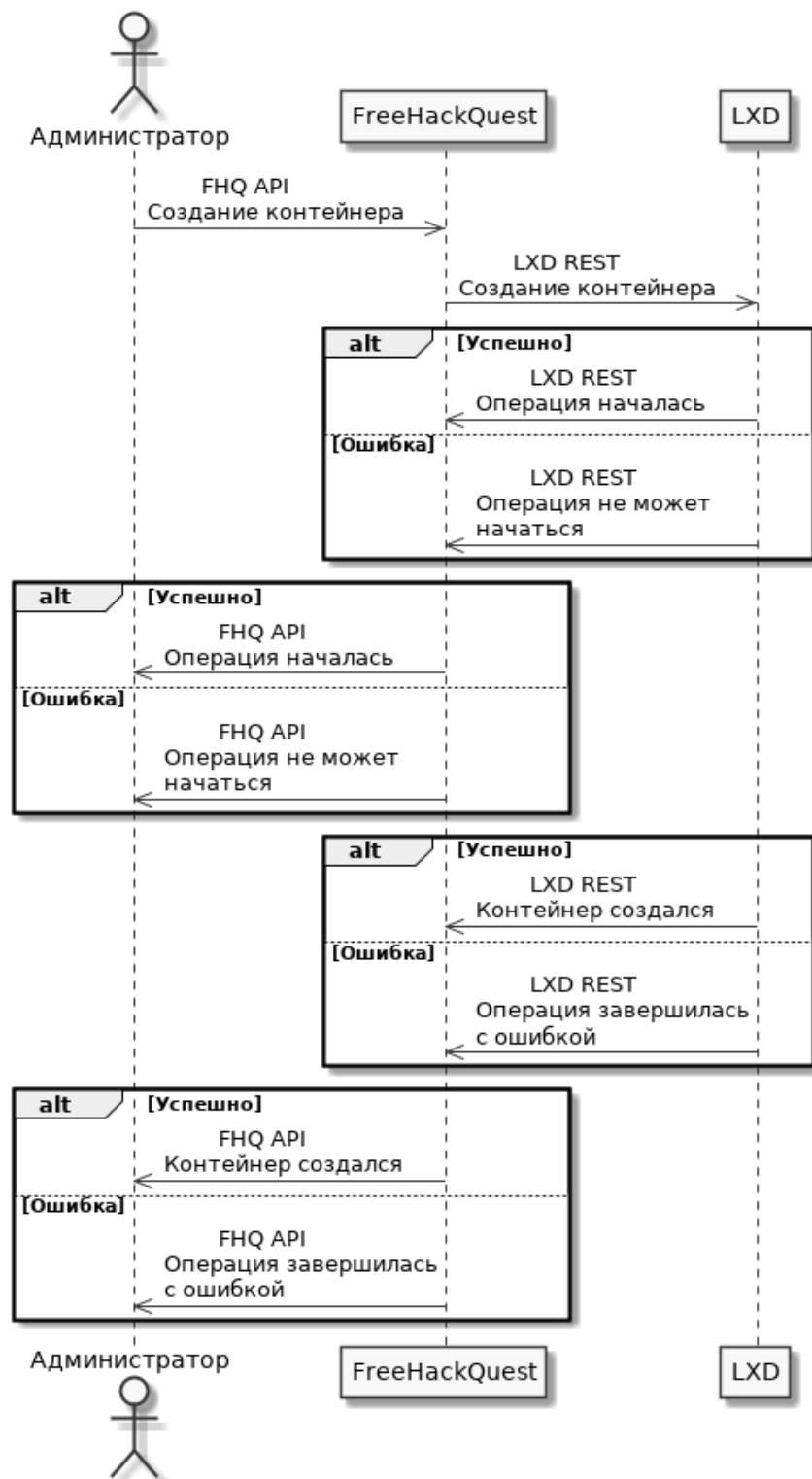
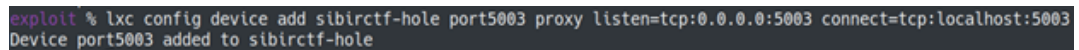


Рисунок 12 – Асинхронное создание контейнера

4.3 Настройка подсети-порта для контейнера

Порты можно настраивать с помощью добавления к контейнеру устройства типа прокси (проху). Устройство типа прокси позволяет пересылать сетевые соединения между хостом и контейнером. Это позволяет перенаправлять трафик, попадающий на один из адресов хоста, на адрес внутри контейнера. Возможные типы подключения: TCP с TCP, UNIX socket с UNIX socket, TCP с UNIX socket, UDP с UDP и так далее.

Открытие TCP порта 5003 для доступа к сервису с помощью команды `lxc config device add sibirctf-hole port5003 proxy listen=tcp:0.0.0.0:5003 connect=tcp:localhost:5003` (рис. 13).



```
exploit % lxc config device add sibirctf-hole port5003 proxy listen=tcp:0.0.0.0:5003 connect=tcp:localhost:5003
Device port5003 added to sibirctf-hole
```

Рисунок 13 – Открытие порта 5003

Настроить подсеть контейнеров можно с помощью группы команд `lxc network`. Создать отдельную сеть можно с помощью команды `lxc network create <имя_сети>`. Прикрепить созданную сеть к контейнеру можно с помощью команды `lxc network attach <имя_сети> <имя_контейнера> <имя_устройства> <имя_интерфейса>`.

Настроить использование NAT можно с помощью `lxc network set <имя_сети> ipv4.nat true`.

Задать IPv4 адрес статически можно с помощью команды `lxc config device set sibirctf-hole eth0 ipv4.address 10.105.26.88`.

После перезагрузки контейнера сетевые адреса обновятся, а команда `lxc network list-leases sibirctf` покажет наличие записи о статическом адресе для контейнера.

5 Разработка учебного материала для курса ТиМП

Достаточно очевидным является тот факт, что студенты, занимающиеся программированием, обучаясь на специальности «Информационная безопасность», должны уметь разрабатывать свои программы, не допуская каких-либо уязвимостей в своем коде. Вполне логично было бы обучать студентов аспектам безопасности программирования на предмете «Технологии и методы программирования» (ТиМП). К сожалению, текущий курс ТиМП не охватывает подобные проблемы, поэтому было принято решение о том, что он нуждается в развитии.

Платформа FNQ включает в себя специальную систему Classbook, позволяющую хранить и предоставлять учебную информацию. Эта система поможет обучить студентов основным существующим уязвимостям, встречающимся в программном обеспечении, и поможет им избегать их при разработке собственных приложений.

Также FNQ послужит площадкой для выполнения практических заданий по темам, связанным с уязвимостями программного обеспечения, и позволит на практике закрепить полученные в ходе прохождения курса знания.

В качестве темы для первых тестовых лекции и практического занятия была выбрана уязвимость «Переполнение стека». Ее подтверждает тот факт, что уязвимости, относящиеся к переполнению различных сегментов памяти, обнаруживаются специалистами регулярно. Проиллюстрировать этот довод могут две уязвимости, включенные в реестр CVE (Common Vulnerabilities and Exposures – список общих уязвимостей, подверженных воздействию извне) за последнее время, а именно CVE-2015-7547 [7] и CVE-2018-10731 [8].

5.1 Формирование лекции для курса ТиМП

Для подготовки материала к новой лекции курса ТиМП была взята и переработана информация с англоязычного ресурса Техасской группы технической безопасности RaiderSec [9].

Для демонстрации уязвимости, связанной с переполнением стека,

используется программа третьего уровня игры «Smash The Stack – IO» [10], призванной показать на практике механизмы работы уязвимостей. Исходный код программы представлен на рисунке 14.

```
#include <stdio.h>
#include <string.h>

void good()
{
    puts("Win.");
    execl("/bin/sh", "sh", NULL);
}
void bad()
{
    printf("I'm so sorry, you're at %p and you want to be at %p\n", bad, good);
}

int main(int argc, char **argv, char **envp)
{
    void (*functionpointer)(void) = bad;
    char buffer[50];

    if(argc != 2 || strlen(argv[1]) < 4)
        return 0;

    memcpy(buffer, argv[1], strlen(argv[1]));
    memset(buffer, 0, strlen(argv[1]) - 4);

    printf("This is exciting we're going to %p\n", functionpointer);
    functionpointer();

    return 0;
}
```

Рисунок 14 – Исходный код программы, демонстрирующей уязвимость

Допустим, злоумышленнику интересен некий абстрактный флаг, скрытый в программе в функции `good`. Суть эксплойта (программы или кода, использующего уязвимость с целью извлечения выгоды) заключается в том, что при передаче в буфер более 50 символов произойдет его переполнение, и часть данных перезапишется поверх адреса функции `bad`.

Таким образом, возникает возможность изменить адрес функции, и, когда код продолжит свое исполнение, в месте предполагаемого вызова функции `bad` будет вызвана функция `good`, следовательно, злоумышленник получит доступ к секретным данным.

Чтобы наглядно продемонстрировать расположение функций и их аргументов в адресном пространстве памяти, код программы был дизассемблирован посредством отладчика GDB. Далее на основе полученных данных было составлено схематическое расположение функций и их аргументов по конкретным адресам в различных сегментах памяти на протяжении всего времени работы программы (рис. 15).

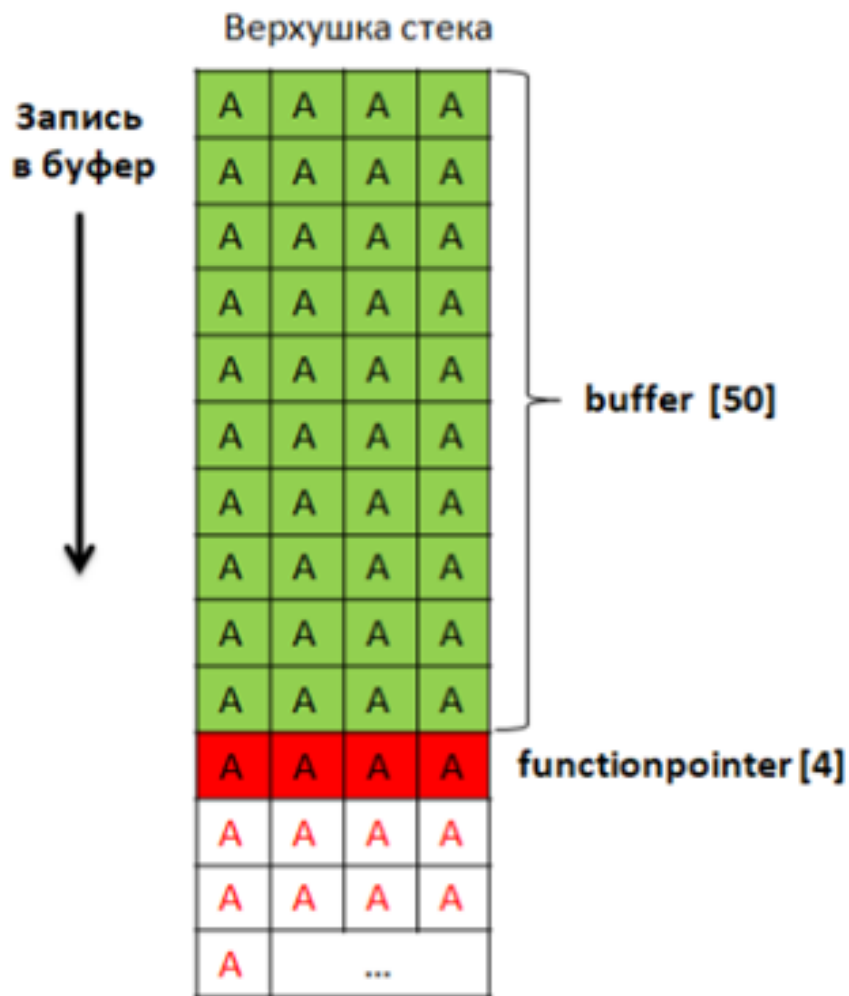


Рисунок 15 – Схематичное представление памяти стека

На основе полученных данных было разработано краткое методическое пособие, подробно описывающее на низком уровне работу программы задачи «Smash The Stack – IO». Дополнительно была оформлена презентация, содержащая наглядную демонстрацию отладки программы с использованием GDB с комментариями.

В скором времени методическое пособие, а также презентация будут загружены в ресурс FHQ в систему Classbook для доступа студентов и всех желающих.

5.2 Разработка практического задания для FNQ

Для того, чтобы обучающиеся на курсе ТиМП студенты могли на практике закрепить знания, полученные в ходе изучения была разработана программа, содержащая в себе уязвимость переполнения стека. Код программы представлен на рисунке 16.

```
1  #include <iostream>
2  #include <fstream>
3  #include <unistd.h>
4  #include <cstring>
5  using namespace std;
6
7  int main(){
8
9      char path[100]="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin";
10     char name[10];
11     cout <<"What is your name?";
12     cin >> name;
13     cout << "Here is your flag, " << name << "...";
14     setenv("PATH",path,1);
15     system("cat flag.txt");
16     return 0;
17 }
```

Рисунок 16 – Исходный код программы для практического задания

Данная программа базируется на принципах, схожих с принципами, описанными в методическом пособии и презентации, и также повержена уязвимости переполнения стека.

В скором времени на платформе FNQ появится новый раздел, в который будет помещена данная программа, и студенты или каждый желающий смогут проверить свои знания и попытаться заполучить скрытый флаг.

6 Результаты работы

В результате работы в рамках данного семестра:

- реализована асинхронная обработка операций с контейнером;
- развернуты сервисы внутри контейнера, настроен порт для контейнера;
- изучена система верстки документов \LaTeX , которая в рамках проекта применялась для подготовки документации: общее представление о системе, описание отдельных компонентов системы и логики;
 - исправлены имеющиеся ошибки подсчета рейтинга участников, изменения имен пользователей;
 - написание учебного материала и разработка практических заданий для FNQ, связанных с «Smash The Stack»;
 - написаны статьи для «Интернет-конференции ГПО» (Приложения А, Б).

Заключение

По истечению сроков, отведенных на выполнение групповой проектной работы, большинство запланированных на семестр задач были выполнены. Проведена работа с контейнерами (реализована асинхронная обработка операций для работы с контейнерами, развернуты сервисы внутри контейнера, настроен порт для контейнера), исправлены ошибки с подсчетом рейтинга участников и изменением имен пользователей, написан учебный материал и разработано практическое задание «Smash The Stack – IO» для FreeHackQuest, изучена система компьютерной верстки документов \LaTeX , с использованием которой подготовлены следующие разделы документации по системе: общее представление о системе, структура системы, описание стека технологий и процессов системы.

В следующем семестре планируется продолжить работу над разработкой платформы для проведения практик по информационной безопасности в формате тестирования на проникновение внутри среды, моделирующей работу информационной инфраструктуры организации.

Будущие результаты ГПО рекомендуется использовать при проведении практик по обучению тестированию на проникновение, а также при подготовке специалистов по информационной безопасности. Разрабатываемая система не только поможет устраивать практики по тестированию на проникновение, но и облегчит работу организаторов соревнований CTF, сделав процесс развертывания и настройки всех подсистем более простым и комфортным.

Также в ходе выполнения групповой проектной работы были освоены компетенции, необходимые для получаемой специальности, а именно:

- ОПК-8: способность к освоению новых образцов программных, технических средств и информационных технологий;
- ПК-6: способность проводить анализ, предлагать и обосновывать выбор решений по обеспечению эффективного применения автоматизированных систем в сфере профессиональной деятельности;
- ПК-24: способность обеспечить эффективное применение

информационно-технологических ресурсов автоматизированной системы с учетом требований информационной безопасности.

Отчет по результатам группового проектного обучения выполнен в соответствии с образовательным стандартом ТУСУР [11].

Список сокращений, обозначений, терминов и определений

API LXC – Application Programming Interface Linux Containers;

FHQ – FreeHackQuest;

GDB – GNU Debugger;

GPL – GNU General Public License;

LXD – Linux Container Daemon;

STL – Standard Template Library;

UML – Unified Modeling Language;

ТиМП – Технологии и методы программирования.

Список использованных источников

- 1 C++ Программирование – Система CMake [Электронный ресурс]. – Режим доступа: <http://procplusplus.blogspot.ru/2011/06/cmake-1.html> (дата обращения: 29.11.2018).
- 2 Tproger – STL: стандартная библиотека шаблонов C++ [Электронный ресурс]. – Режим доступа: <https://tproger.ru/articles/stl-cpp/> (дата обращения: 29.11.2018).
- 3 Linux Containers – LXD introduction [Электронный ресурс]. – Режим доступа: <https://linuxcontainers.org/lxd/> (дата обращения: 29.11.2018).
- 4 An introduction to LaTeX [Электронный ресурс]. – Режим доступа: <https://www.latex-project.org/about/> (дата обращения: 5.12.2018).
- 5 PlantUML – все, что нужно бизнес-аналитику для создания диаграмм в программной документации [Электронный ресурс]. – Режим доступа: <https://habr.com/post/416077/> (дата обращения: 5.12.2018).
- 6 GDB: The GNU Project Debugger [Электронный ресурс]. – Режим доступа: <http://www.gnu.org/software/gdb/> (дата обращения: 6.12.2018).
- 7 Разбор эксплойта уязвимости CVE-2015-7547 [Электронный ресурс]. – Режим доступа: <https://cyberleninka.ru/article/n/razbor-eksployta-uyazvimosti-cve-2015-7547> (дата обращения: 7.12.18).
- 8 PT-2018-14: Переполнение буфера в PHOENIX CONTACT FL SWITCH [Электронный ресурс]. – Режим доступа: <https://www.securitylab.ru/lab/PT-2018-14> (дата обращения: 7.12.18).
- 9 Smash the Stack IO Level 3 Writeup [Электронный ресурс]. – Режим доступа: <http://raidersec.blogspot.com/2012/10/smash-stack-io-level-3-writeup.html> (дата обращения: 8.12.2018).
- 10 Smash The Stack Wargaming Network [Электронный ресурс]. – Режим доступа: <http://smashthestack.org/wargames.html> (дата обращения: 9.12.2018).
- 11 Чернышев А.А., Кормилин В.А. ОС ТУСУР 01-2013 Образовательный стандарт ВУЗа, 2013. – 53 с.

Приложение А

(Справочное)

Статья для Интернет-конференции ГПО

ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ КОНТЕЙНЕРИЗАЦИИ В СФЕРЕ
ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ*А.А. Михайлова, С.Д. Ушев, студенты каф. КИБЭВС**Научный руководитель: Д.С. Никифоров, мл.н.с. каф. КИБЭВС**г. Томск, ТУСУР, 725_maa@fb.tusur.ru**Проект ГПО КИБЭВС-1808 – Разработка и развитие системы для обучения и проведения практик по информационной безопасности*

В данной статье рассматривается технология контейнеризации, ее применение в рамках текущего проекта, преимущества и недостатки.

Ключевые слова: виртуализация, контейнеризация, Linux-контейнер, LXD.

Основная идея проекта – обучение участников практических занятий по информационной безопасности посредством участия в тестировании на проникновение имитированной инфраструктуры организации.

В рамках проекта было необходимо решить задачу запуска уязвимых сервисов и приложений, с помощью которых участникам необходимо проникнуть в сеть организации и получить доступ к информации ограниченного доступа, которая в рамках практик называется флагом. В ходе практики участник будет искать уязвимости в имитированной инфраструктуре организации, помогающие получить контроль над одним из сервисов. Из этого вытекает риск, что участник сможет получить контроль не только над сервисами, но и над системой проведения практик по информационной безопасности. В связи с чем было решено использовать виртуализацию. Если участник и сможет получить контроль, то это будет виртуальная машина, а не реальная система.

Рассматривались такие типы виртуализации, как полная виртуализация, паравиртуализация и виртуализация на уровне операционной системы. В итоге

была выбрана виртуализация на уровне операционной системы по причине ненужности виртуализации аппаратной платформы. Именно отсутствие расходов на эмуляцию оборудования и работы виртуального программного обеспечения на реальном оборудовании позволяет получить высокую производительность.

Технология контейнеризации – это метод, позволяющий запускать приложение и необходимый ему минимум системных библиотек изолированно от других процессов [1]. Контейнеризация является виртуализацией на уровне операционной системы, то есть изоляция приложения осуществляется с помощью средств операционной системы.

Технология контейнеризации начала свое развитие и получила широкое применение в системах GNU/Linux, поэтому стоит обозначить термин Linux-контейнер или просто контейнер.

Linux-контейнер – это набор процессов, изолированный от остальной операционной системы и запускаемый с отдельного образа операционной системы, который содержит все файлы, необходимые для их работы. Образ содержит все зависимости приложения и поэтому может легко переноситься из среды разработки в среду тестирования, а затем в промышленную среду [2].

Для запуска и управления контейнерами выбран системный менеджер Linux-контейнеров Linux Container Daemon (LXD).

К преимуществам использования контейнеров относятся:

- быстрый запуск: запускается как процесс, а не виртуальная машина;
- высокая производительность и малые задержки;
- эффективное использование памяти [3];
- единая среда исполнения для разработки, тестирования и эксплуатации.

Недостаток заключается в новых угрозах безопасности для хостовой операционной системы. Под хостовой операционной системой (хост) понимается система, на которой происходит выполнение контейнеров, а также запуск и управление виртуальными (гостевыми) операционными системами.

Можно выделить следующие угрозы безопасности для хостовой системы при использовании контейнеризации:

— одно общее ядро хостовой операционной системы для всех контейнеров, эксплуатация уязвимости которого в одном из контейнеров ведет к компрометации хоста;

- увеличение поверхности атаки;
- доступ к файловой системе хоста из контейнера;
- запуск привилегированных контейнеров: существуют эксплойты, позволяющие выходить из таких контейнеров и получать полный контроль над хостом;
- некорректное разграничение прав доступа на хосте.

Средства операционной системы с ядром Linux, позволяющие изолировать выполнение процессов:

- kernel namespaces – пространства имен позволяют изолировать пользователей, сетевые интерфейсы, межпроцессное взаимодействие и файловую систему;
- AppArmor и SELinux для мандатного разграничения доступа;
- seccomp политики – это фильтры безопасного вычисления, которые позволяют разрешать или запрещать системные вызовы для контейнера;
- chroots вместе с pivot_root перемещает корневую файловую систему в подготовленный образ операционной системы;
- kernel capabilities – разрешения ядра позволяют выдавать контейнеру только необходимые разрешения на использование возможностей ядра без предоставления прав суперпользователя;
- cgroups – контрольные группы позволяют выставять квоты на использование аппаратных ресурсов и управлять доступом к устройствам.

ЛИТЕРАТУРА:

1. What is container technology? [Электронный ресурс]. – Режим доступа: <https://www.techradar.com/news/what-is-container-technology> (дата обращения: 8.11.2018).

2. Linux-контейнеры: изоляция как технологический прорыв [Электронный ресурс]. – Режим доступа: <https://habr.com/company/redhatrussia/blog/352052/> (дата обращения: 8.11.2018).

3. LXD crushes KVM in density and speed [Электронный ресурс] - Режим доступа: <https://blog.ubuntu.com/2015/05/18/lxd-crushes-kvm-in-density-and-speed> (дата обращения: 10.11.2018).

Приложение Б

(Справочное)

Статья для Интернет-конференции ГПО

ИЗУЧЕНИЕ УЯЗВИМОСТЕЙ ПАМЯТИ И РАЗРАБОТКА КУРСА ЛЕКЦИЙ И
ПРАКТИЧЕСКИХ ЗАНЯТИЙ ПО ДИСЦИПЛИНЕ «ТЕХНОЛОГИИ И
МЕТОДЫ ПРОГРАММИРОВАНИЯ»

*Е.И. Косенко, студент, Д.Г. Дудкин, студент каф. КИБЭВС**Научный руководитель Д.С.Никифоров, младший научный сотрудник каф.**КИБЭВС, Томск, ТУСУР, nds@csp.tusur.ru**Проект ГПО КИБЭВС-1808 Разработка и развитие системы для обучения и
проведения практик по информационной безопасности*

В данной статье рассматривается одна из наиболее распространенных уязвимостей – переполнение буфера, а также пример кода, предназначенный для демонстрации студентам опасности эксплуатации этой уязвимости в рамках курса «Технологии и методы программирования» (ТиМП).

Ключевые слова: уязвимость, память, переполнение буфера.

Введение. Под термином «уязвимость» понимается недостаток в компьютерной системе, эксплуатация которой приводит к нарушению целостности или некорректной работе системы. Такие недостатки были обнаружены во всех основных операционных системах, включая Microsoft Windows, MacOS, а также UNIX.

Проблема переполнения буфера выходит за рамки обычного нарушения целостности или логики функционирования программы и даже может являться причиной удаленного вторжения в систему с прямым наследованием всех привилегий. Данная уязвимость является самой опасной с точки зрения поставщиков ОС и одной из самых значимых в компьютерной безопасности в целом [1].

Актуальность данной проблемы подтверждает тот факт, что уязвимости, относящиеся к переполнению различных сегментов памяти, обнаруживаются

специалистами регулярно. Проиллюстрировать этот довод могут две уязвимости, включенные в реестр CVE (common vulnerabilities and exposures – список общих уязвимостей, подверженных воздействиям извне) за последнее время, а именно CVE-2015-7547 [2] и CVE-2018-10731 [3].

Новизна подхода заключается в том, что текущий курс ТиМП не охватывает проблему, описанную выше, и нуждается в развитии и частичной доработке.

В ходе работы в рамках ГПО проводится изучение уязвимостей, возникающих при работе с памятью, формирование материалов для проведения занятий по дисциплине ТиМП и подготовка контента для интернет-ресурса freehackquest.com.

Описание уязвимости «buffer overflow». Ее смысл заключается в перезаписи данных за пределами стека при переполнении буфера. Рассмотрим данную уязвимость на примере разбора задания «Smash the stack – level 3», код которого представлен на рисунке 1 [4]. Допустим, злоумышленнику интересен некий абстрактный флаг, скрытый в программе в функции good. Суть эксплойта (программы или кода, использующего уязвимость с целью извлечения выгоды) заключается в том, что при передаче в буфер более 50 символов произойдет его переполнение и часть данных перезапишется поверх адреса функции bad.

Таким образом, возникает возможность изменить адрес функции, и, когда код продолжит свое исполнение, в месте предполагаемого вызова функции bad будет вызвана функция good, следовательно злоумышленник получит доступ к секретным данным.

```

#include <stdio.h>
#include <string.h>

void good()
{
    puts("Win.");
    execl("/bin/sh", "sh", NULL);
}
void bad()
{
    printf("I'm so sorry, you're at %p and you want to be at %p\n", bad, good);
}

int main(int argc, char **argv, char **envp)
{
    void (*functionpointer)(void) = bad;
    char buffer[50];

    if(argc != 2 || strlen(argv[1]) < 4)
        return 0;

    memcpy(buffer, argv[1], strlen(argv[1]));
    memset(buffer, 0, strlen(argv[1]) - 4);

    printf("This is exciting we're going to %p\n", functionpointer);
    functionpointer();

    return 0;
}

```

Рис. 1 – Код, реализующий переполнение буфера

Чтобы наглядно продемонстрировать расположение функций и их аргументов в адресном пространстве памяти, код программы был дизассемблирован посредством отладчика GDB. Далее на основе полученных данных было составлено схематическое расположение функций и их аргументов по конкретным адресам в различных сегментах памяти на протяжении всего времени работы программы.

Результаты исследования позволяют «изнутри» понять, что происходит в памяти компьютера после запуска программы.

Закключение. В ходе работы было проведено изучение уязвимости «buffer overflow» на примере разбора задания «Smash the stack level 3». Данная работа имеет практическую значимость, так как будет использована для организации курса лекций и практических занятий по дисциплине ТиМП, а также сформирует контент для нового раздела интернет-ресурса для проведения соревнований по компьютерной безопасности freehackquest.com. Планируется продолжить работу в данном направлении, написать цикл статей об уязвимостях, возникающих при работе с памятью. Следующим этапом является рассмотрение уязвимости форматной строки.

ЛИТЕРАТУРА:

1. Анализ уязвимости переполнения буфера [Электронный ресурс]. – Режим доступа: <https://moluch.ru/archive/138/38783/> (дата обращения: 02.10.18).
2. Разбор эксплойта уязвимости CVE-2015-7547 [Электронный ресурс]. – Режим доступа: <https://cyberleninka.ru/article/n/razbor-eksployta-uyazvimosti-cve-2015-7547> (дата обращения: 02.10.18).
3. PT-2018-14: Переполнение буфера в PHOENIX CONTACT FL SWITCH [Электронный ресурс] – Режим доступа: <https://www.securitylab.ru/lab/PT-2018-14> (дата обращения: 03.10.18).
4. Smash the stack Level 3 [Электронный ресурс]. – Режим доступа: <https://seshagiriprabhu.wordpress.com/2012/02/07/smash-the-stack-level3/> (дата обращения: 04.10.18).

Приложение В
(Обязательное)
Компакт-диск

Компакт-диск содержит:

- тестовый сервис;
- документация к проекту в формате *.tex;
- практические и теоретические материалы для курса по дисциплине «Технологии и методы программирования»;
- электронную версию пояснительной записки в форматах *.tex и *.pdf.