Практическое задание 6

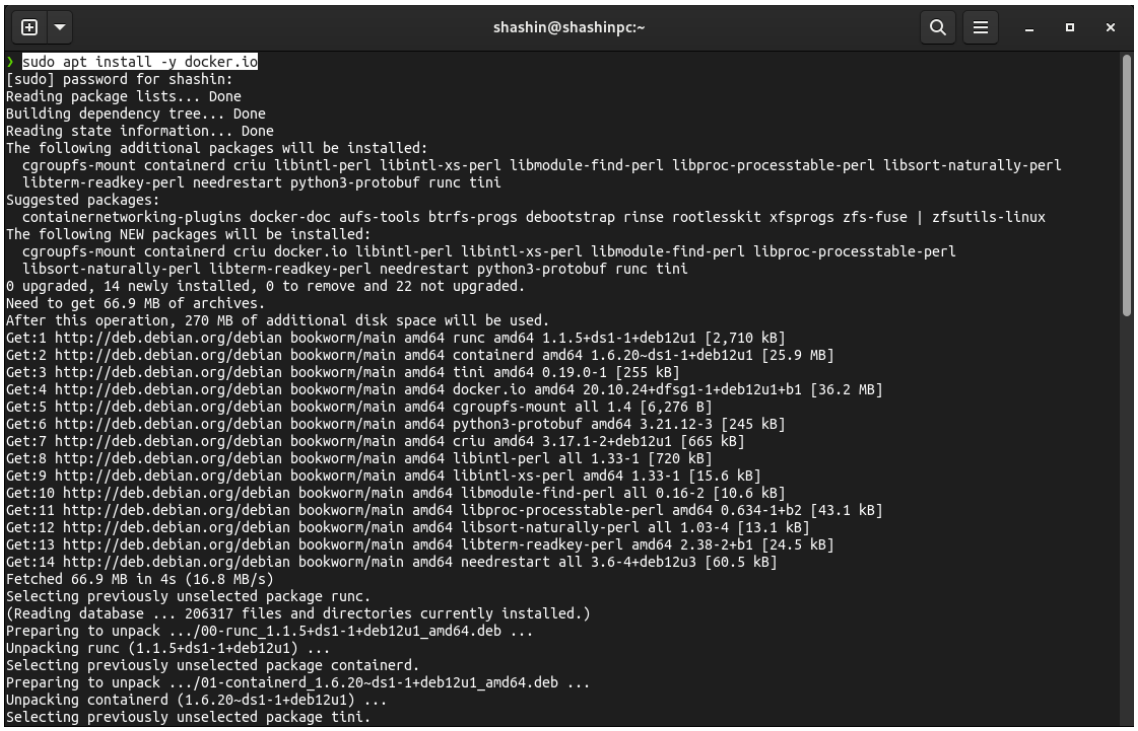
Фаззинг-тестирование приложения

Задачи:

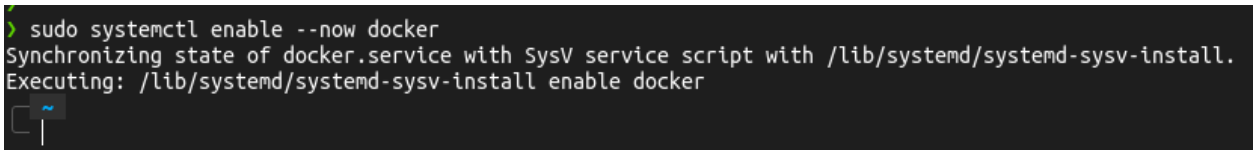
1. Подготовить ВМ с ОС Linux.
2. Запустить в ВМ Docker
3. Выполнить команду `docker pull aflplusplus/aflplusplus`
4. Выбрать фаззинг-цель для тестирования (простое приложение на ‘C’) и запустить фаззинг-тестирование исследуемой программы.
5. Подготовить отчет со скриншотами.

Фаззинг — это метод тестирования программного обеспечения, который используется для обнаружения уязвимостей и ошибок в приложениях. Этот подход включает в себя автоматическую генерацию и отправку случайных или некорректных данных (входных данных) в программу с целью выявления сбоев, исключений или других нежелательных реакций.

1. Устанавливаем Docker



2. Запускаем Docker



3. Добавляем пользователя в группу docker

```
❯ sudo usermod -aG docker $USER
❯ newgrp docker
❯ docker --version
Docker version 20.10.24+dfsg1, build 297e128
❯
```

4. Загрузка docker образа с инструментарием для фаззинга

```
❯ docker pull aflplusplus/aflplusplus
Using default tag: latest
latest: Pulling from aflplusplus/aflplusplus
215ed5a63843: Pull complete
c2afcb461141: Pull complete
3ca02ff3323b: Pull complete
cada57a56dd5: Pull complete
f281e3305537: Pull complete
52b1e6541358: Pull complete
9b0d25773932: Pull complete
0fcef20a7fd1: Pull complete
6561610a9df5: Pull complete
8f974a5e3406: Pull complete
ce657232950d: Pull complete
61b461f082ab: Pull complete
Digest: sha256:23edea458bcc1e5647b19bfba205f21408235972ce6b346c326dbeef179919d6
Status: Downloaded newer image for aflplusplus/aflplusplus:latest
docker.io/aflplusplus/aflplusplus:latest
❯
```

5. Запуск контейнера

```
❯
❯ sudo docker run -ti -v ~/aflt:/src aflplusplus/aflplusplus
[AFL++ dcb7f165c8a9] /AFLplusplus #
```

6. Пишем скрипт на C для запуска фаззинга

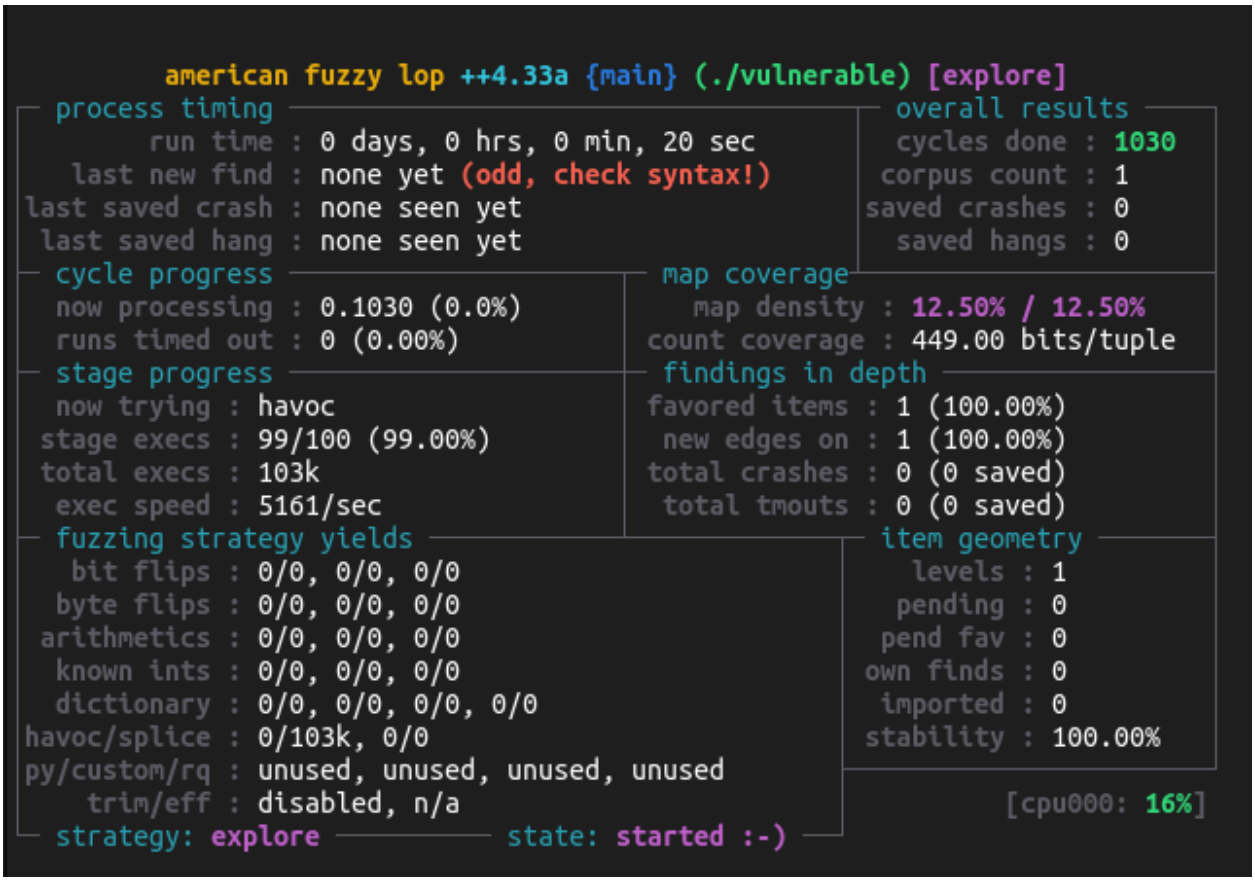
```
 GNU nano 6.2                                          vulnerable.c *
#include <stdio.h>
#include <string.h>

void vulnerable(char *input) {
        char buffer[16];
        strcpy(buffer, input);
}

int main(int argc, char **argv) {
        if(argc > 1) {
                vulnerable(argv[1]);
        }
        return 0;
}
```

7. Запуск

```
[AFL++ dcb7f165c8a9] /src # afl-fuzz input -o output -- ./ valnerable @@
```



В случае краша, система напишет об этом (“saved crashes:”, “total crashes” справа)

```
shashinpc# pwd
/home/shashin/aflt
shashinpc# cd output
shashinpc# ls
main
shashinpc# cd main
shashinpc# ls
cmdline  crashes  fuzz_bitmap  fuzzer_setup  fuzzer_stats  hangs  is_main_node  plot_data  queue  target_hash
shashinpc#
```

Описание компонентов фаззер.

1. queue — место где хранятся входные данные, используемые фаззером для создания новых текстовых вариантов
2. crashes — каталог, в котором хранятся файлы, вызывающие сбои программы во время тестирования
3. hangs — папка, где хранятся тестовые файлы, которые привели к зависанию программы, то есть к длительному выполнению без завершения

4. fuzzer_stats — файлы, содержащие статистические данные о работе фаззера, такие как количество выполненных циклов и покрытие кода
5. plot_data — информация, используемая для построения графиков, например, зависимость покрытия кода от времени
6. target_hash — данные о хэшировании целевых файлов, которые помогают отслеживать уникальность входных данных
7. cmdline — запись команд, используемых для запуска фаззера, включая аргументы командной строки
8. fuzz_bitmap — данные, которые отслеживают покрытие кода, включая битмапы, используемые для определения новых ветвей выполнения
9. fuzzer_setup — конфигурационные файлы или параметры, определяющие настройки работы фаззера

Фаззинг является мощным методом тестирования. Его автоматизация и способность находить ошибки делают его незаменимым инструментов в современном процессе разработки программного обеспечения.