**Выполнил: Андреев Даниил Алексеевич**

**Задание 1.** Дайте ответы на следующие вопросы.

Вопрос 1. Назовите принципы объекто-ориентированного программирования (ООП), прямым образом относящиеся к языку программирования С:

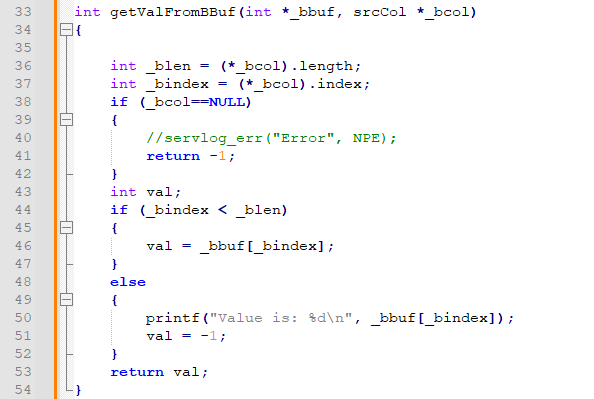
Среди выделеных принципов общепринятыми в ООП являются:

*А. Полиморфизм;  
B. Наследование;   
D. Инкапсуляция;*

Всех их вполне возможно в той или иной степени реализовать в СИ*.*

[*См пример*](https://github.com/Great-Ping/KeysCases/tree/main/Question1.GoshaCar)

Вопрос 2. Представлен участок кода на языке программирования C:



В данном коде присутсвуют дефекты

*А. CWE-476 (36 – 37 строчки);   
D. CWE-125 (50 строка).*

В будущем возможна ошибка связанная с неверным применеием метода, в связи с тем что нет ни описани, ни в названии метода не указано что в случае исключительно ситуации будет возвращен “-1”.

В связи использованием для буфера простой структуры данных, с неизвестной длинной не исключено повторение ошибки CWE-125, либо получение CWE-476, в связи с тем что буффер будет являтся NULL

Примерное решение (Я не знаю причин использования структуры srcCol, поэтому выпилил ее за ненадобностью).

#include <stdio.h>

#include "GoshaCar.h"

typedef struct {

int\* ptr;

int length;

} Buf;

//Вернет значение из буфера либо -1

int getValFromBBufOrError(Buf \_bbuf, int index)

{

if (\_bbuf.ptr == NULL)

{

return -1;

}

if (index < \_bbuf.length && index >= 0)

{

return \_bbuf.ptr[index];

}

else

{

//не уверен что это нужно, ведь это побочный эффект

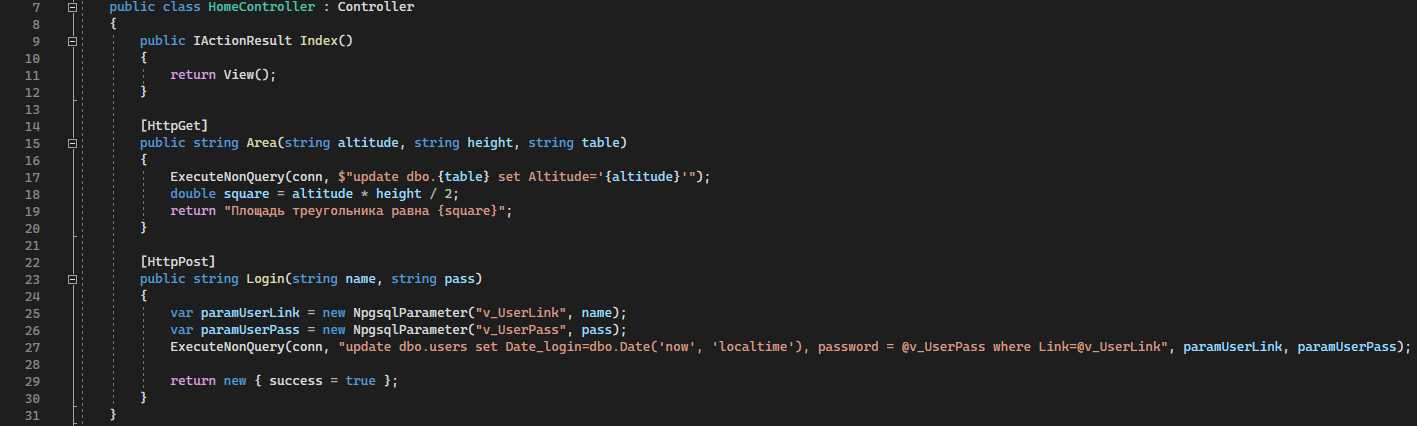
printf("Index out of range(0, %i)\n", \_bbuf.length);

return -1;

}

}

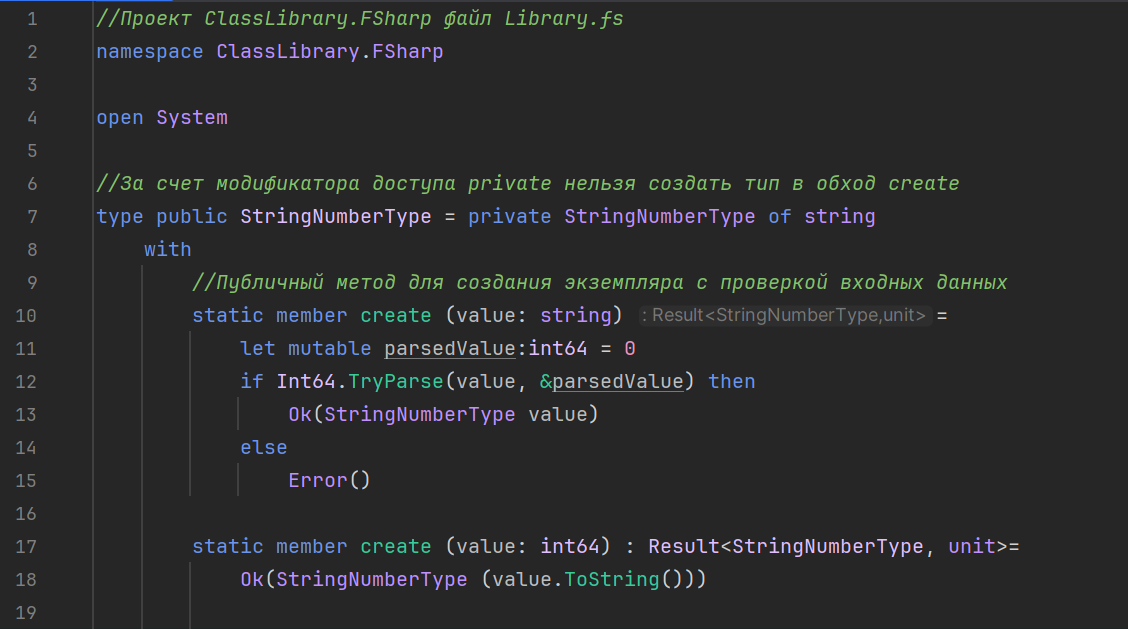
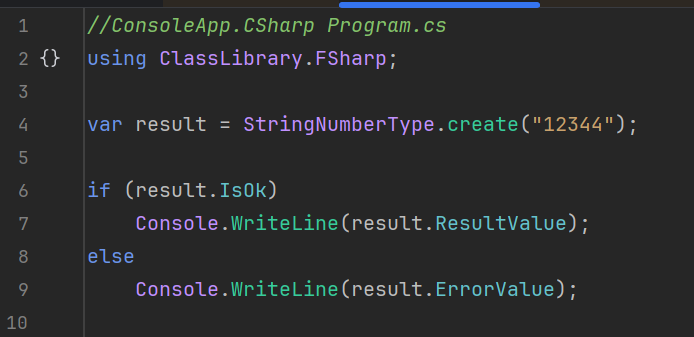
Вопрос № 3. Представлен фрагмент исходного кода, написанный на языке программирования С#. Проанализируйте фрагмент исходного кода и определите, содержится ли в нем какая-либо уязвимость/дефект/баг.



Крайне соминительные структуры данных возвращаемые методами контролера: нет единного стандарта, нет конкретной задачи за которую отвечал бы контроллер. (полагаю на это делается скидка, т.к. это всего лишь простейший пример).

В методе Area возможна SQL инъекция. Я вижу три способа решения подобных проблем, все в конечном итоге сводятся к использованию провалидированых параметров:

1. Учитывая 18 строчку кода, наиболее подходящим решением для данного примера было бы изменить типы параметров метода на double, для table можно было бы завести отдельное перечисление (enum) со всеми допустимыми значениями
2. Использование записи подобной той что в методе логин, в наждежде на наличие экранирования строковых параметров внутри библиотеки.
3. Писать доменную область на F# :). Можно ограничить возможные значения для полей обернув их в специализированый тип для каждого поля свой, данный подход часто встречается в DDD и позволяет убедиться в валидности полей, ограничить возможные операции и значения полей. В родственном для С# языке F# как раз таки имеется очень удобный механизм оборачивания примитивных типов. [Пример](https://github.com/Great-Ping/KeysCases/tree/main/Question3.FSharpInterop):



Данный пример гарантирует исключение SQL инъекции, однако такое возможно не во всех ситуациях, так что лучше еще подстраховаться используя второй пункт.

В sql выражении метода Login содержится ошибка, подозреваю что должен был быть поиск по паролю и имени пользователя , но вместо этого мы каждый раз устанавливаем новое значение для пароля, всем пользователям с одинаковым логином.

Не стоит ли шифровать пароль пользователя, на случай утечки данных, в Microsoft.AspNetCore.Identity уже имеется готовый инструментарий в виде [PasswordHasher<TUser>](https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.aspnetcore.identity.passwordhasher-1?view=aspnetcore-9.0) (по сути реализация PBKDF2)

Так же бы обратил внимание на выделение Методов и Классов средой разработки, в в данной ситуации белый цвет может указывать на то что не был произведен нормальный семантический анализ (возможно вручную отключено отображение ошибок), однако исходя из того что мы видим я бы присмотрелся к следующим моментам: возможно не подключены пространства имен Microsoft.AspNetCore.Mvc и Npgsql, а так же не реализован метод ExcecuteNonQuery.

Вопрос № 4. Выберите флаги безопасности cookie:

*А. Secure*

*C. HttpOnly;*

Вопрос № 5. Укажите существующие разновидности XSS:

*А. DOM-based;*

*B. Reflected;*

*D. Stored;*

Вопрос № 6. Какие существуют способы защиты от SQLi

*А. Параметризованные запросы;*

**Задание 2.** Реализуйте предложенный сценарий (рекомендуется реализовать в ОС Linux Ubuntu версии не ниже 20.04).

Был сформирован следующий докер файл:

FROM ubuntu:20.04

RUN apt-get update && \

    #Компиляторы

    apt-get install -y make gcc && \

    #Зависимости nginx

    apt-get install -y libpcre3-dev zlib1g-dev && \

    #Гит

    apt-get install -y git

#3.1. Выполнить загрузку файлов исходных текстов веб-сервера nginx;

WORKDIR /home

RUN git clone https://github.com/nginx/nginx.git

#3.2. Выполнить компиляцию загруженных файлов исходных текстов nginx;

WORKDIR /home/nginx

RUN ./auto/configure && \

    make && \

    make install

#3.3. Запустить скомпилированный веб-сервер

ENTRYPOINT ["/usr/local/nginx/sbin/nginx"]

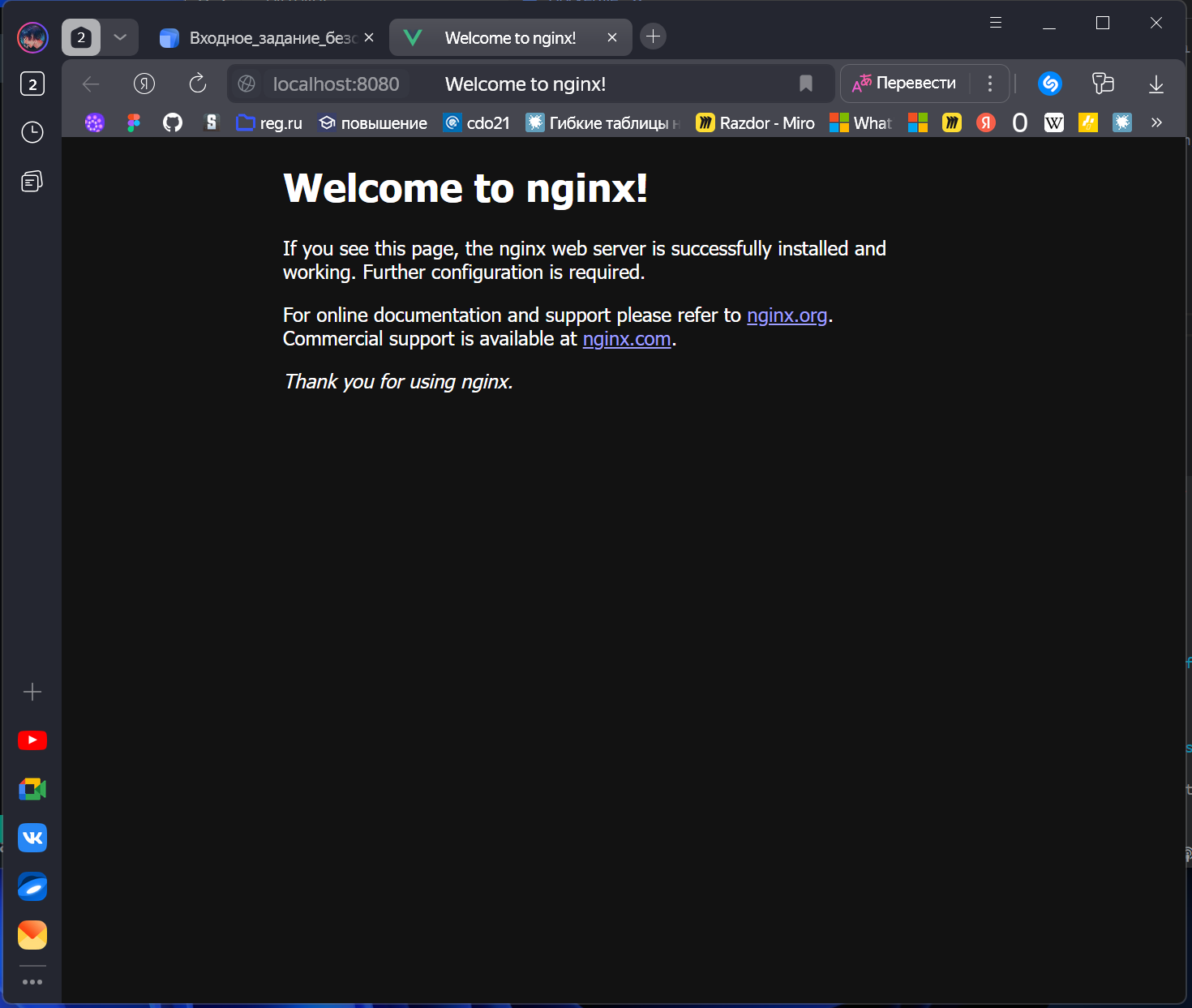
Для сборки изображения использовалась команда:

docker build -t keys-task2 .

Для запуска контейнера:

docker run -it -p 8080:80 --rm keys-task2 -g “daemon off;”

Проверка работы сервера:



Расшифровка использованных команд:

docker build [OPTIONS] PATH - создаёт образ из Dockerfile, параметр -t используется для указания имени и тега изображения, “.” указывает на текущую директорию.

docker run [OPTIONS] IMAGE [COMMAND] [ARG...] – запускает команду в контейнере по созданному ранее образу. Параметр -p позволяет опубликовать порты, -rm указывает на удаление контейнера после завершения контейнра, -it позволяет пехватить ввод вывод терминала в контейнере. Далее идет имя контейнера а за ним параметры для nginx: -g указывает директиву "daemon off;" чтоб nginx не запускался в фоне и контейнер не останавливался.

apt-get update – обновляет базу данных пакетов ubuntu.

apt-get install – устанавливает пакет, флаг -y указывает на автоматический ответ “yes” на все запросы при установке.

git clone – копирует репозиторий

скрипт ./auto/configure – создает makefile

make – сокмпилирует nginx

make install – устанавливает скомпилированный файл

/usr/local/nginx/sbin/nginx – запускает Nginx

Обзор Dockerfile

FROM задает родительский образ ubuntu:20.04

RUN исполняет команды при сборке изображения

WORKDIR задает рабочую директорию

ENTRYPOINT задает точку входа, представляет из себя команду с аргументами которая вызовется при исполнении контейнера.

**Задание 3.** Выполните [тестовый пример](https://github.com/antonio-morales/Fuzzing101/tree/main/Exercise%201) по выявлению уязвимости CVE-2019-13288 в консольной утилите XPDF версии 3.02 с применением фаззера AFL++.

За неимением и не желанием ставить другие виртуальные машины изначально был сформирован следующий dockerfile. В получившемся изображении содержатся все необходимы утилиты для выполнения задания, а именно AFLplusplus, gdb, xpdf-3.02 сбраный с помощью afl.

FROM ubuntu:20.04

RUN apt-get update && \

    apt-get install -y build-essential gcc && \

    apt-get install -y wget tar

WORKDIR /home/fuzzing-xpdf

# Загрузка xpdf-3.02

RUN wget https://dl.xpdfreader.com/old/xpdf-3.02.tar.gz && \

    tar -xvzf xpdf-3.02.tar.gz

# Сборка и установка xpdf

WORKDIR /home/fuzzing-xpdf/xpdf-3.02

RUN ./configure --prefix="/home/fuzzing-xpdf/install/"

RUN make && \

    make install

# Загрузка тестовых pdf

WORKDIR /home/fuzzing-xpdf/pdf-examples

RUN wget https://github.com/mozilla/pdf.js-sample-files/raw/master/helloworld.pdf && \

    # NOT FOUND

    # wget http://www.africau.edu/images/default/sample.pdf && \

    wget https://www.melbpc.org.au/wp-content/uploads/2017/10/small-example-pdf-file.pdf

# Локальная установка AFL++

# Зависимости

RUN DEBIAN\_FRONTEND=noninteractive apt-get install -y build-essential python3-dev automake git flex bison libglib2.0-dev libpixman-1-dev python3-setuptools

RUN apt-get install -y lld-11 llvm-11 llvm-11-dev clang-11 || sudo apt-get install -y lld llvm llvm-dev clang

RUN apt-get install -y gcc-$(gcc --version|head -n1|sed 's/.\* //'|sed 's/\..\*//')-plugin-dev libstdc++-$(gcc --version|head -n1|sed 's/.\* //'|sed 's/\..\*//')-dev

# Загрузка AFL

WORKDIR /home

RUN git clone https://github.com/AFLplusplus/AFLplusplus

# Сборка AFL

WORKDIR /home/AFLplusplus

RUN export LLVM\_CONFIG="llvm-config-11" && \

    make distrib && \

    make install

# Пересборка XPDF с помощью AFL

# Удаление старых файлов

RUN rm -r /home/fuzzing-xpdf/install

WORKDIR /home/fuzzing-xpdf/xpdf-3.02

RUN make clean

# Сборка с помощью afl-clang-fast

RUN export LLVM\_CONFIG="llvm-config-11"

RUN CC=/home/AFLplusplus/afl-clang-fast CXX=/home/AFLplusplus/afl-clang-fast++ ./configure --prefix="/home/fuzzing-xpdf/install/"

RUN make

RUN make install

WORKDIR /home/fuzzing-xpdf/

# Утилита для отладки

RUN apt-get install -y gdb

Изображение было создано с помощью команды:

docker build -t keys-task3 .

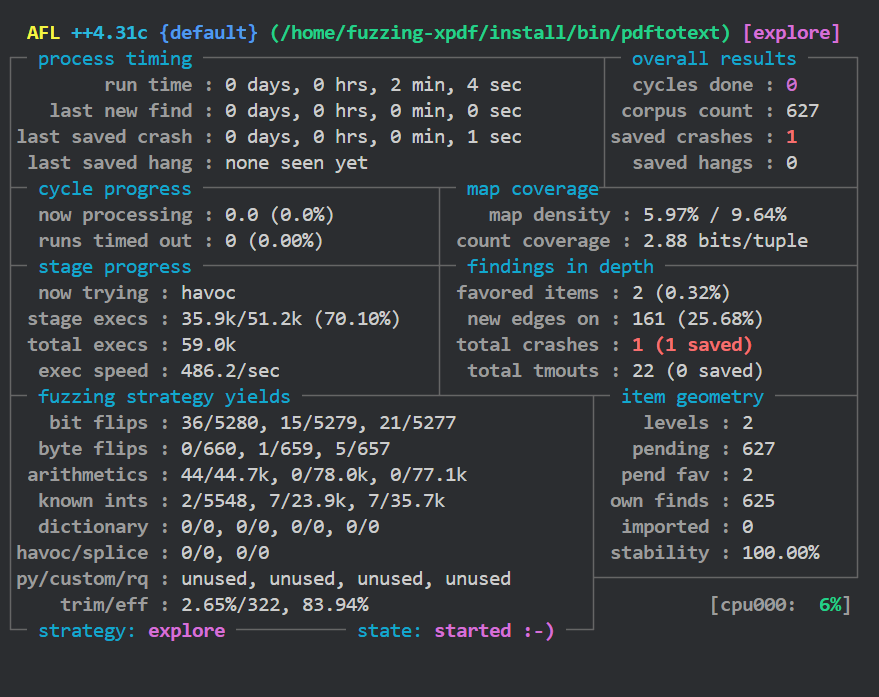
И запущен контейнер:

docker run -it –rm keys-task3

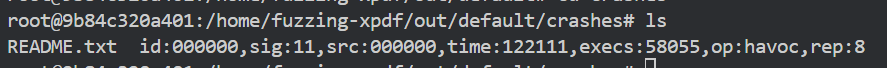
После в контейнере был запущен фаззер с помощью команды

afl-fuzz -i /home/fuzzing-xpdf/pdf-examples/ -o /home/fuzzing-xpdf/out/ -s 123 -- /home/fuzzing-xpdf/install/bin/pdftotext @@ /home/fuzzing-xpdf/output

Спустя две минуты работы была обнаружена первая ошикба:



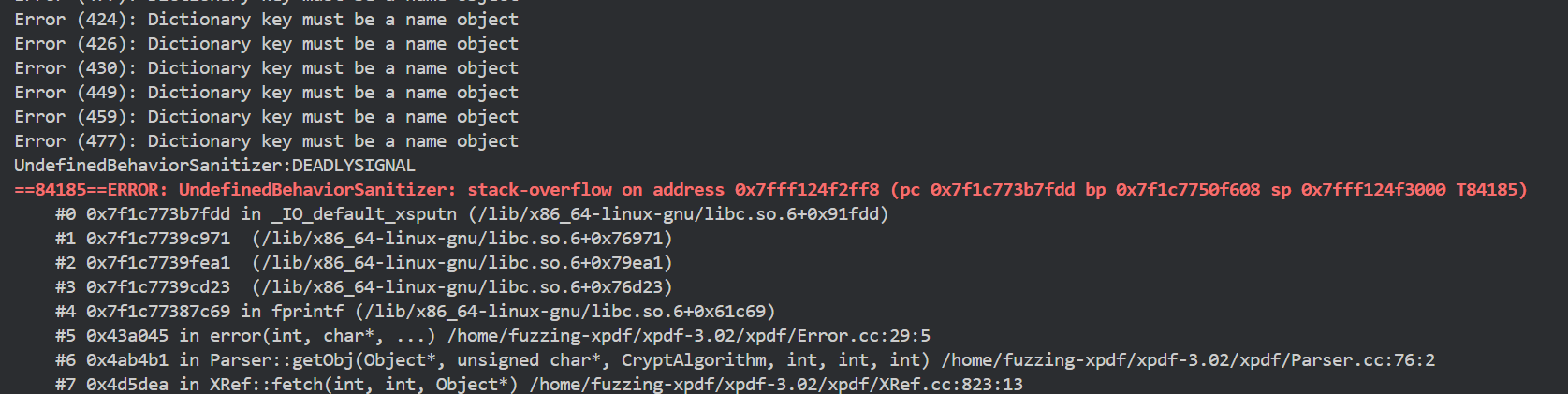
Далее в директории /home/fuzzing-xpdf/out/default/crashes был найден файл соотвествующий сбою:

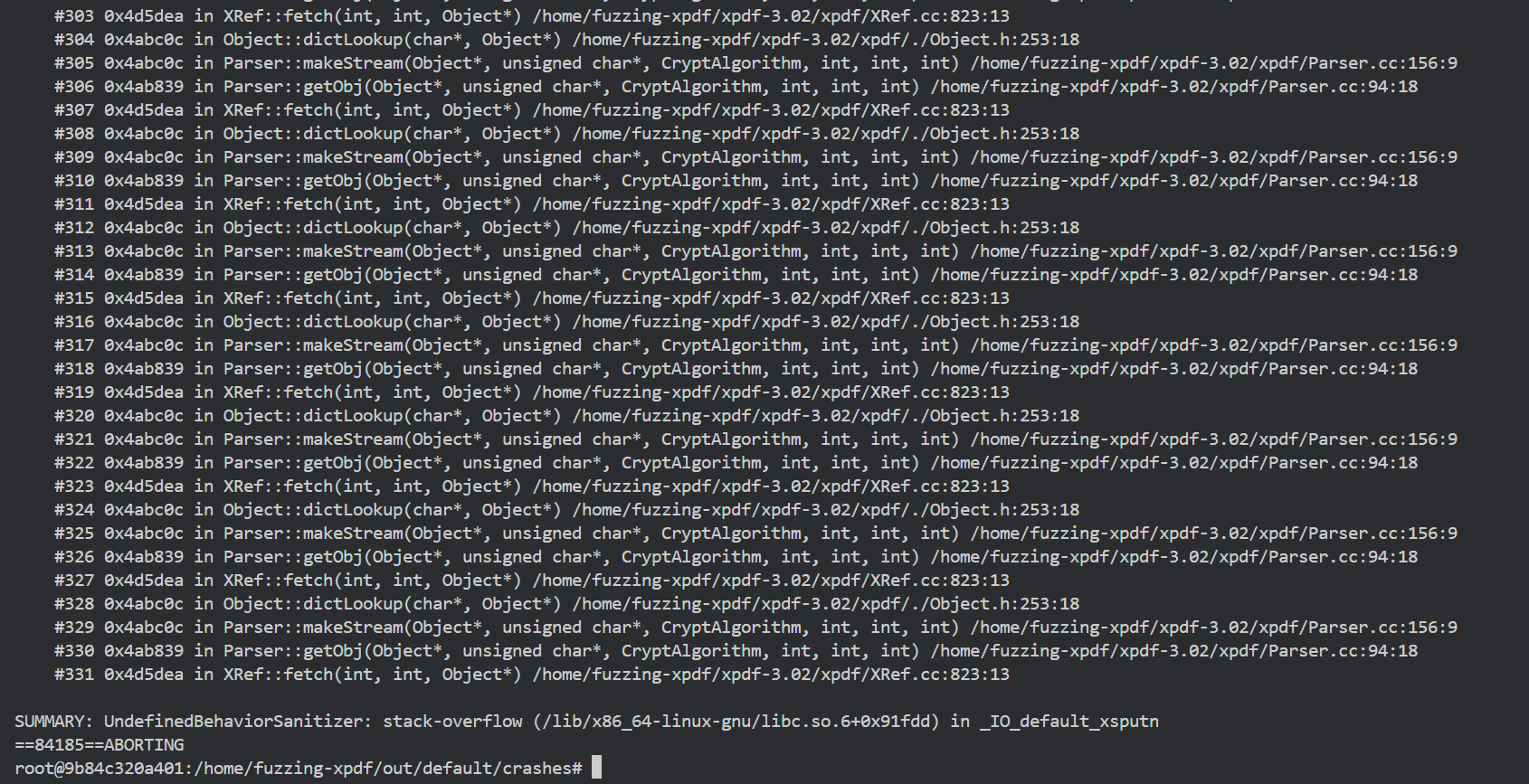


И данный файл был передан в качестве входных параметров для xpdf

/home/fuzzing-xpdf/install/bin/pdftotext '/home/fuzzing-xpdf/out/default/crashes/id:000000,sig:11,src:000000,time:122111,execs:58055,op:havoc,rep:8 ' /home/fuzzing-xpdf/output

После чего был получен следующий вывод, покоторму можно уже сделать предплжения что проблема в бесконечном рекурсивном вызове метода Parser::getObj.





Чтобы убедиться в предположениях была задействована утилита gdb

Прежде всего xpdf был пересобран с отладочной информацией, чтоб можно было получить трассировку стека (хотя у меня она почему то уже была до этого), для этого была задействованы следующие команды:

rm -r /home/fuzzing-xpdf/install

cd /home/fuzzing-xpdf/xpdf-3.02/

make clean

CFLAGS="-g -O0" CXXFLAGS="-g -O0" ./configure --prefix="/home/fuzzing-xpdf/install/"

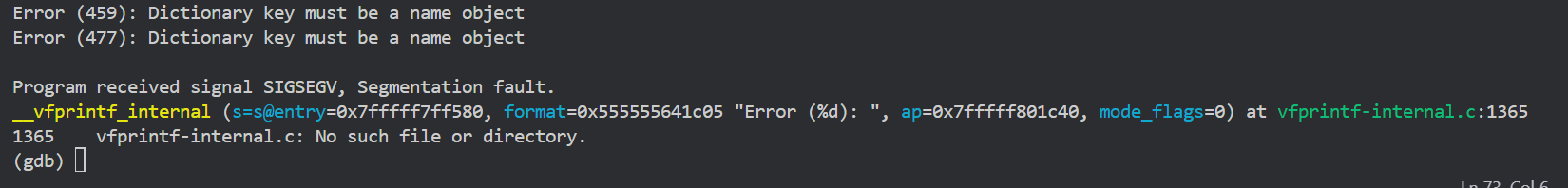
make

make install

И далее была использована сама утилита gdb:

gdb --args /home/fuzzing-xpdf/install/bin/pdftotext /home/fuzzing-xpdf/out/default/crashes/id:000000,sig:11,src:000000,time:122111,execs:58055,op:havoc,rep:8 /home/fuzzing-xpdf/output

После запуска утилиты была введена команда run и получен следующий вывод



После использования команды bt, получили трассировку стека:

