# Цель работы

Изучение типовых ошибок и принципов поиска уязвимостей в программном обеспечении без исходных кодов.

# Задачи работы

В ходе выполнения лабораторной работы необходимо выполнить следующие действия:

1. Изучить основные типы ошибок в программном обеспечении (целочисленное переполнение, отсутствие проверки длины копируемых данных, переполнение буфера и другие).
2. Получить у преподавателя файлы в соответствии со своим вариантом.
3. Реализовать программу, осуществляющую фаззинг формата файла.
4. Реализованная программа должна осуществлять следующие действия:

* осуществлять изменение оригинального файла (однобайтовая замена, замена нескольких байт, дозапись в файл);
* заменять байты на граничные значения (0x00, 0xFF, 0xFFFF, 0xFFFFFF, 0xFFFFFFFF, 0xFFFF/2, 0xFFFF/2+1, 0xFFFF/2-1 и т.д.);
* иметь автоматический режим работы, при котором производится последовательная замена байт в файле;
* находить в файле символы, разделяющие поля (“,:=;”);
* расширять значения полей в файле (дописывать в конец, увеличивать длину строк в файле);
* осуществлять запуск исследуемой программы;
* используя средство динамической бинарной инструментации (DBI) (Intel Pin / DynamoRIO) осуществлять измерение покрытия кода во время фаззинга;
* реализовать режим работы фаззера с обратной связью на основе покрытия кода, основанный на сохранении измененных байт в файле с учетом их влияния на покрытие кода программы;
* обнаруживать возникновение ошибки в исследуемом приложении;
* получать код ошибки и состояние стека и регистров на момент возникновения ошибки;
* логировать в файл информацию о произошедших ошибках и соответствующих им входных параметрах (произведенные замены).

1. Разработать IDC/IDAPython-скрипт, осуществляющий следующие действия:
   * поиск в программе функций ввода данных (fread, fscanf, read, fgets, …);
   * поиск вызовов небезопасных функций (strcpy, sprintf, strncpy, memcpy, memmove, …).
2. Разработанный IDC/IDAPython-скрипт должен выводить следующую информацию:
   * название найденной функции;
   * адрес, откуда вызывается данная функция.
3. Используя разработанную программу для фаззинга формата файлов осуществить поиск в выданной программе уязвимости (целочисленное переполнение, отсутствие проверки длины данных), приводящей к переполнению буфера.
4. Сформировать файл, приводящий к краху исследуемого приложения.
5. Изучить с использованием дизассемблера и отладчика структуру программы и найти участок кода с уязвимостью. Провести исследование найденной уязвимости и условий ее эксплуатации.
6. В отчете необходимо привести следующую информацию:

* исходный код программы, осуществляющей фаззинг формата файла;
* описание методов, применяемых при фаззинге;
* участок кода с найденной уязвимостью (из дизассемблера), с комментариями;
* восстановленный исходный код (на языке С) участка с уязвимостью;
* описание найденной уязвимости (тип ошибки, причины, ограничение при эксплуатации);
* размер буфера, адрес начала буфера, адрес сохраненного на стеке адреса возврата (return address, за буфером);
* формат файла, используемого для эксплуатации, с указанием значения полей, приводящих к краху программы;
* описание уязвимости и механизмов ее эксплуатации;
* исходный код разработанных IDC/IDAPython-скриптов;
* описание возможного способа исправления уязвимости.

# Теоретические сведения

## Фаззинг

В ГОСТ 58142 «Информационная технология. Методы и средства обеспечения безопасности. Детализация анализа уязвимостей программного обеспечения в соответствии с ГОСТ Р ИСО/МЭК 15408 и ГОСТ Р ИСО/МЭК 18045. Часть 2. Тестирование проникновения» есть следующее определение: «фаззинг — это тестирование, использующее как корректные, так и случайные (включая некорректные) входные данные для проверки, устанавливающей, обрабатываются ли должным образом интерфейсом случайные входные данные или возникает ошибочная ситуация (ошибочное условие), указывающая (указывающее) на наличие недостатков при разработке (ошибки исходного кода) или при эксплуатации».

Другими словами фаззинг — это техника автоматизированного тестирования, при которой на вход программе подаются специально подготовленные данные, которые могут привести её к аварийному состоянию или неопределённому поведению.

Базово фаззеры делят на два типа: генеративные и мутационные. Генеративные фаззеры создают данные, которые они отправляют целевому приложению, тогда как мутационные фаззеры берут фрагменты существующих данных и изменяют их.

Также существуют следующие классификации фаззеров:

1. По знанию о входных данных:

* Black-box — ничего не известно о формате входных данных;
* Gray-box — известна неполная информация о формате входных данных;
* White-box — наличие спецификации, известны все сведения о формате данных;

1. По цели, которая будет подвергнута фаззингу:

* Source-based — при наличии исходного кода;
* Binary-based — при отсутствии исходного кода;

1. По наличию обратной реакции от тестируемого приложения:

* Feedback driven;
* Not feedback driven;

1. По операциям, которые будут совершаться над входными данными:

* Генерационные;
* Мутационные;
* Комбинированные.

## Ошибки в программном обеспечении и способы их обнаружения при фаззинге

В программном обеспечении встречаются следующие типовые ошибки:

1. Переполнение буфера (объем данных сохраняется в области памяти, которая слишком мала для их хранения):

* Переполнение стека – если переполнение буфера приводит к перезаписи данных на стеке, что может быть использовано для управления потоком выполнения;
* Переполнение кучи – если переполнение произошло в сегменте кучи исполняемого процесса, были перезаписаны метаданные соседнего блока, примыкающего к области, в которой произошло переполнение, что может быть использовано для записи данных в произвольные области памяти;

1. Целочисленное переполнение происходит при попытке сохранить значение, выходящее за диапазон от -32767 до 32767 в знаковое целое. Процессор отбросит старшие биты, чтобы успешно сохранить значение, что может привести к выделению слишком малого объема памяти и, возможно, к переполнению буфера в будущем;
2. Использование форматных строк происходит при передаче злоумышленником входных данных, которые интерпретируются как спецификаторы формата в некоторых функциях обработки строк. Это может привести к чтению данных различного объема со стека и других областей памяти (%x, %s и др.) или к записи данных в память (например, перезаписи адреса возврата или указателя на функцию при помощи %n);
3. Отсутствие проверки длины копируемых данных приводит к перезаписи памяти за пределами буфера;
4. Неправильно использование рекурсии приводит к переполнению стека вызовов функций.

Для поиска переполнений буфера с помощью фаззинга передаются очень большие объёмы данных целевому приложению в надежде, что они попадут в процедуру, не проверяющую корректно длину данных перед их копированием.

Для поиска целочисленных переполнений фаззеры должны передавать как большие положительные, так и отрицательные (то есть граничные и близкие к ним) значения, провоцирующие переполнения.

Для поиска использования форматных строк во время фаззинга необходимо убедиться, что генерируемые тестовые случаи содержат некоторые из спецификаторов формата, чтобы попытаться задействовать неправильно используемую функцию обработки строк, принимающую на вход этот спецификатор.

# ХОД РАБОТЫ

## Реализация фаззера

При помощи средств Python, DynamoRIO и WinDbg был реализован фаззер. Для работы фаззера необходимо передать путь к целевой программе (например, vuln1.exe), путь к целевому файлу (config). Доступно 3 режима работы:

* Добавление байтов в конец файла: фаззер дописывает «A» в конец файла, по одному байту за шаг;
* Мутация данных на основе обратной связи: изменение данных начинается с маркера начала буфера с данными /start; случайным образом происходит мутация 1-2 байтов по некоторому смещению от начала буфера на значения, представленные в списке boundary\_values (минимальные, максимальные и средние значения для 1-2 байт, значения размером 3 и 4 байта) и выполняется запуск программы для новых данных; при увеличении покрытия программы (изменения в хэше трассировка) мутация фиксируется, тк она привела к новому пути выполнения программы; в случае «падения» программы вся необходимая информация заносится в соответствующую директорию;
* Замена конкретного байта (пользователь указывает позицию) на конкретное значение (пользователь вводит значение).

Для каждого из режимов формируется бэкап целевого файла, чтобы восстанавливать начальные данные по завершении работы программы. В ходе работы программа логирует действия в файл и консоль, что позволяет просмотреть процесс фаззинга (на каждой итерацией сохраняется информация из DynamoRIO, в случае завершения программы с ошибкой используется windbg и информация из stderr и stdout для изучения причины ошибки).

## Реализация скрипта IDAPython

Был написан скрипт IDAPython, реализующий поиск функций, отвечающих за ввод данных и потенциально небезопасных функций:

INPUT\_FUNCTIONS = {

'fgetc', 'fgets', 'fread', 'fscanf', 'getc', 'getchar', 'gets',

'read', 'scanf', 'sscanf', 'vfscanf', 'vscanf', 'vsscanf',

'getpass', 'cin'

}

UNSAFE\_FUNCTIONS = {

# Copying and concatenation

'strcpy', 'strncpy', 'stpcpy', 'lstrcpy', 'StrCpy',

'strcat', 'strncat', 'lstrcat', 'StrCat',

'sprintf', 'vsprintf', 'snprintf', 'vsnprintf',

'memcpy', 'memmove', 'memset', 'gets', 'strecpy', 'streadd', 'strtrns',

# Format strings

'printf', 'fprintf', 'vprintf', 'vfprintf',

# Input/output

'scanf', 'fscanf', 'sscanf', 'vscanf', 'vfscanf', 'vsscanf',

'getpass', 'realpath', 'getwd', 'cin', '\_splitpath', 'makepath'

}

Основная функция поиска в скрипте перебирает все сегменты и инструкции в дизассемблированном коде, ищет инструкции вызова (call, bl, jal) и проверяет, входит ли вызываемая функция в один из списков. Отчет после запуска скрипта содержит все адреса вызовов каждой потенциально небезопасной функции/функции вывода, которая была найдена в программе:

def find\_calls(target\_funcs):

"""

Find calls to functions from target\_funcs (exact name match).

Returns a dict: {function\_name: [call\_addresses]} (only for found functions).

"""

result = {}

for seg\_start in idautils.Segments():

for head in idautils.Heads(seg\_start, idc.get\_segm\_end(seg\_start)):

mnem = idc.print\_insn\_mnem(head)

if mnem in ('call', 'bl', 'jal'): # x86/x64/ARM/MIPS

operand\_addr = idc.get\_operand\_value(head, 0)

if operand\_addr:

callee\_name = get\_func\_name(operand\_addr)

if callee\_name in target\_funcs:

if callee\_name not in result:

result[callee\_name] = []

result[callee\_name].append(head)

return result

## Анализ структуры программы

Результат работы программы при поступлении на вход корректного файла показан на рисунке 1.

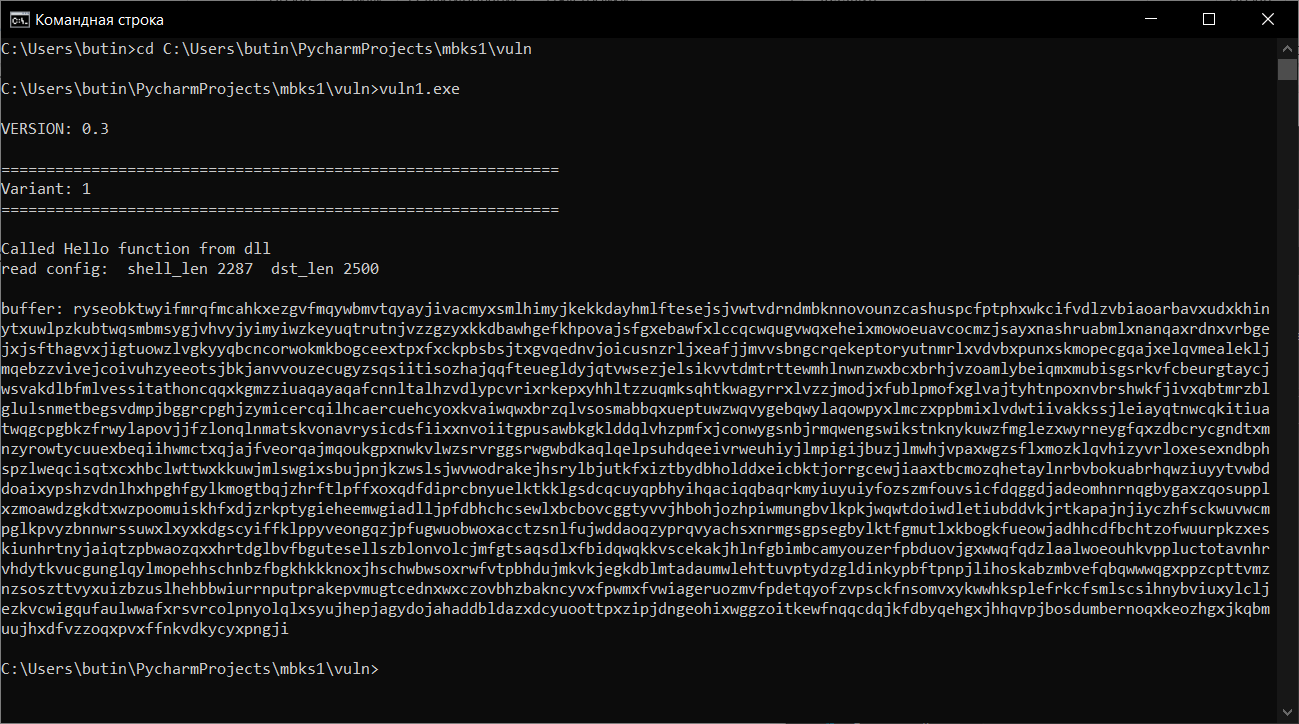


Рисунок 1 - Результат работы программы при поступлении на вход корректного файла

Файл был открыт в IDAPro, были изучены функции, работающие с конфиг-файлом и информацией, полученной из него (read\_config, parse\_string, vuln\_func), псевдокод которых показан на рисунках 2-5.

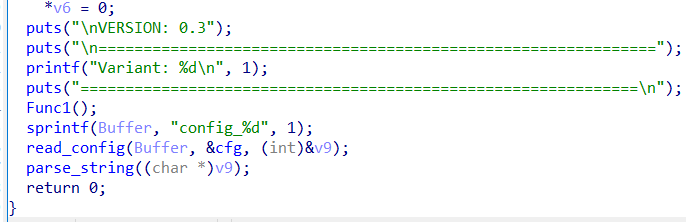


Рисунок 2 – Вызов функций read\_config и parse\_string

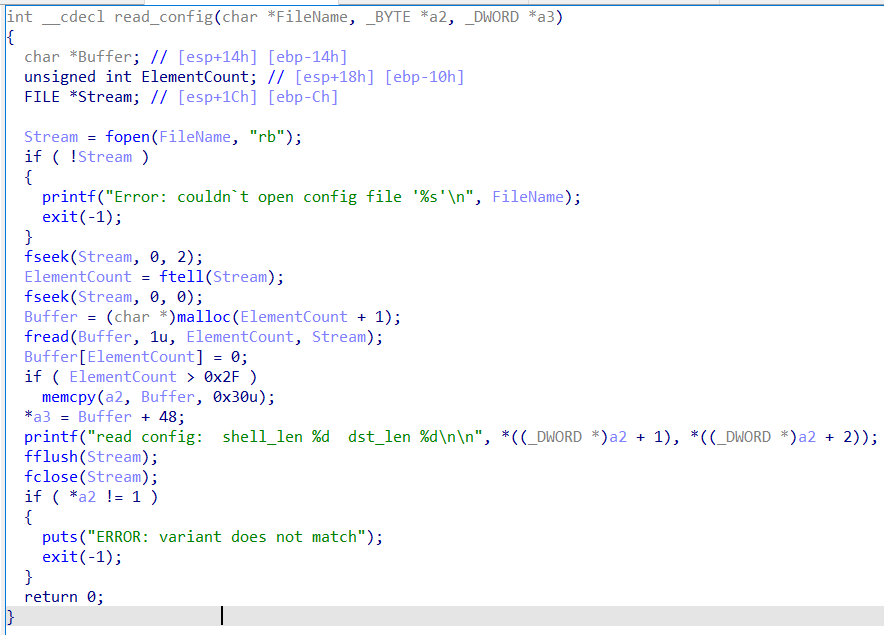


Рисунок 3 – Содержимое функции read\_config

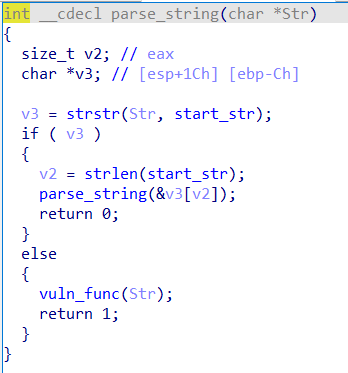


Рисунок 4 – Содержимое функции parse\_string

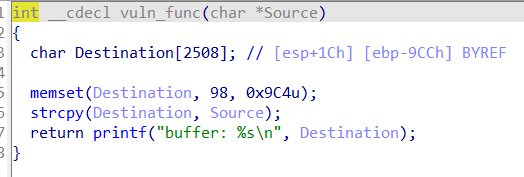


Рисунок 5 – Содержимое функции vuln\_func

Файл «config\_1» открывается на чтение в бннарном режиме, определяется его размер. Согласно размеру файла выделяется буфер (больше на один байт для завершающего нуля). Всё содержимое файла помещается в буфер. Так как размер буфера рассчитывается исходя из размера файла, на данном этапе не будут допущены ошибки типа переполнения буфера. Использование fread, которая не использует форматные строки, не допускает допущение ошибок, связанных с ними.

Выполняется проверка, достаточно ли длины буфера для копирования 48 байт от его начала в переменную (передается указатель на cfg, cfg представляет собой структуру из четырех байт и двух DWORD в сегменте .bss). Если длины будет недостаточно, то копирование выполнено не будет. Так как переменная, которая была передана на вход для этой операции, не инициализируется до момента копирования в неё данных, то такая ситуация может привести к некорректному поведению программы.

Данные, находящиеся с 49 байта буфера и далее, остаются в буфере без проверок и изменений, указатель на начало этих данных помещается в переменную, которая также была подана на вход read\_config и не была инициализирована до этого. В неё в любом случае будет записан адрес Buffer + 48, а от входных данных зависит, находится ли по адресу часть буфера или адрес выходит за пределы буфера.

Файл закрывается. Происходит анализ информации из cfg:

* 1-ый байт: вариант (должен быть 1, иначе выводится сообщение об ошибке и программа завершает работу с кодом -1);
* 5-8 байты: выводится как shell\_len (целое число);
* 9-13 байты: выводится как dst\_len (целое число).

Указатель на 49-ый байт буфера подается как аргумент на вход parse\_string (далее воспринимается как указатель на начало строки Str). Внутри функции объявляются переменная для хранения длины строки (size\_t) и вспомогательный указатель для поиска подстроки.

При помощи функции strstr выполняется поиск подстроки ‘/start’ в строке Str. В случае, если Str является указателем за пределы буфера (короткие данные), strstr может аварийно завершить программу (segmentation fault, access violation – если адрес не принадлежит процессу или не отображен для чтения) или будет считывать «мусор», невалидные данные по адресу, что приведет к непредсказуемому результату, в том числе ложным срабатываниям («найдено»).

Если подстрока была найдена, вычисляется длина найденной подстроки, далее рекурсивно вызывается paprse\_string для указателя на строку, которая начинается сразу после найденной подстроки. После завершения рекурсивного вызова функция возвращает 0.

Если подстрока не найдена, вызывается функция vuln\_func с аргументом Str (то есть в неё будет передана заключительная часть строки, не содержащая ‘/start’). После завершения вызова vuln\_func функция возвращает 1.

Внутри vuln\_func происходит образование буфера Destination размера 2508 байт и заполнение его значением 98 (‘b’). Далее в этот буфер при помощи strcpy копируется строка, пришедшая на вход. Это опасный момент, поскольку такая строка может превышать длину 2507 байт (наибольшая допустимая длина, далее начнется переполнение буфера на стеке) или не содержать завершающего нуля, последствия аналогичные превышению длины.

Далее вызывается printf, выводящая строку Destination. Так как в случае ошибки поведение программы становится неопределенным уже на strcpy (переписывается адрес возврата из функции), то в случае, если strcpy завершилась корректно, printf также завершится корректно.

## Фаззинг vuln1.exe

Каждый из режимов работы фаззера позволяет получить некорректный сценарий для программы.

В режиме 1, при дополнении исходной строки до размера, превышающего размер буфера, происходит падение программы, как показано на рисунке 6.

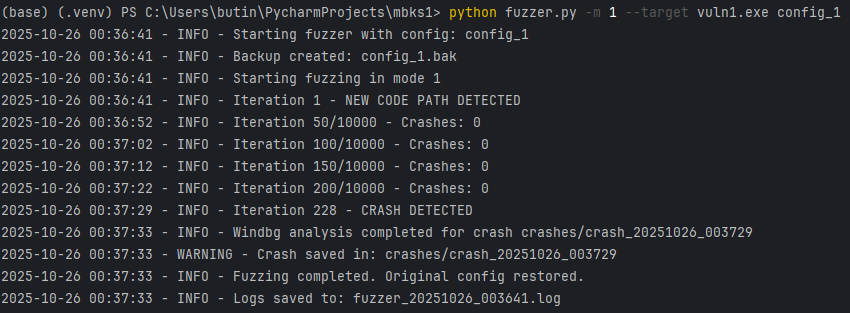


Рисунок 6 – Падение программы при превышении допустимой длины конфигурационного файла

При рассмотрении файлов логов windbg (сохраняются фаззером) можно увидеть следующее:

EXCEPTION\_RECORD: (.exr -*1*)  
.exr -*1*ExceptionAddress: *41414141* ExceptionCode: c0000005 (Access violation)  
 ExceptionFlags: *00000000*NumberParameters: *2* Parameter[*0*]: *00000008* Parameter[*1*]: *41414141*Attempt to execute non-executable address *41414141*FAULTING\_THREAD: 00002ec0  
  
PROCESS\_NAME: *vuln1.exe*EXECUTE\_ADDRESS: *41414141*

Была попытка исполнения адреса 41414141 (этот адрес был сформирован при помощи превышающей допустимую длину строки, дополненной байтами «A» - значением 41), что видно из модифицированного конфигурационного файла, показанного на рисунке 7.

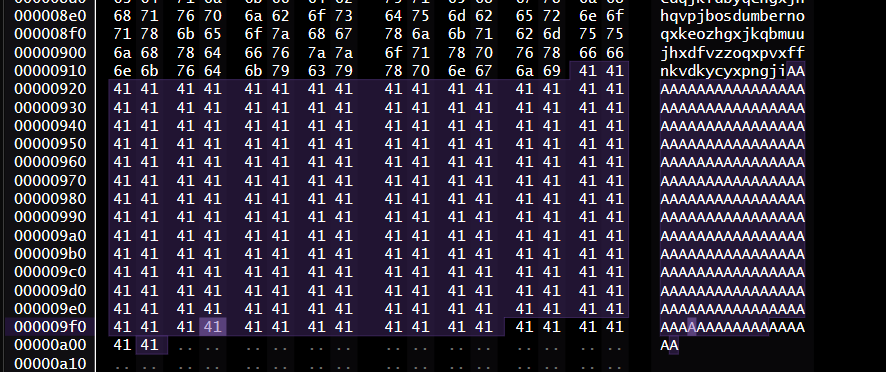


Рисунок 7 – Полученный в ходе фаззинга модифицированный конфигурационный файл

Последовательность исполнения программы до падения:

STACK\_TEXT:   
WARNING: Frame IP not in any known module. Following frames may be wrong.  
009ef904 *41414141 41414141 41414141 41414141 0x41414141*009ef978 007d1420 00684f9e 000008ef 000009c4 *0x41414141*009ef9a8 007d1440 00684f9e 007d9000 000009c4 vuln1!parse\_string+*0x40*009ef9d8 007d1581 00684f98 007dda20 009efafc vuln1!parse\_string+*0x60*009efc08 007d1250 *00000001* 006845f8 006824b8 vuln1!main+*0x13a*009efc34 007d12b5 *00000001 00000000 00000000* vuln1+*0x1250*009efc64 777f82ae 005a5000 046ec0a3 *00000000* vuln1+*0x12b5*009efcc0 777f827e ffffffff 778193ce *00000000* ntdll!\_\_RtlUserThreadStart+*0x2f*009efcd0 *00000000* 007d12a0 005a5000 *00000000* ntdll!\_RtlUserThreadStart+*0x1b*

Из main была вызвана parse\_string, после чего была произведена попытка исполнения по адресу 0x41414141. Также становится известно смещение относительно parse\_string, а именно parse\_string+0x40 – это адрес возврата, на который программа должна была вернуться после выполнения vuln\_func, но по состоянию стека видно, что возврат произошел на 0x41414141. Это подтверждается, если рассмотреть смещения в IDAPro, что показано на рисунке 8.

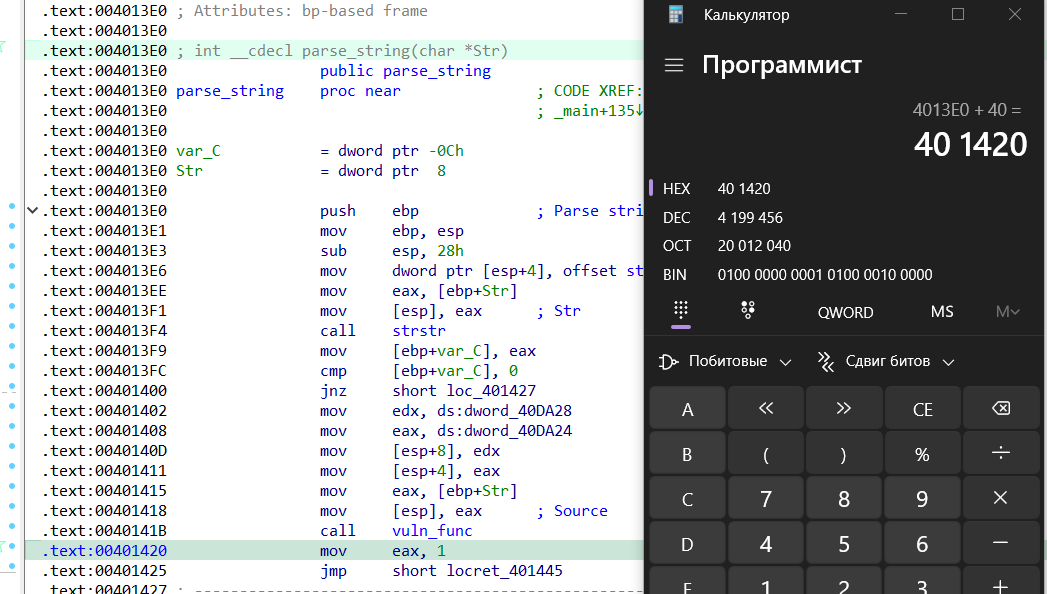


Рисунок 8 – Смещение 0x40 относительно начала parse\_string

Действительно, можно заметить, что в исходном буфере содержится 2279 байтов (считая метку /start), и для превышения допустимого размера буфера достаточно 228 итераций дописывания в конец байта, что показано на рисунке 9.

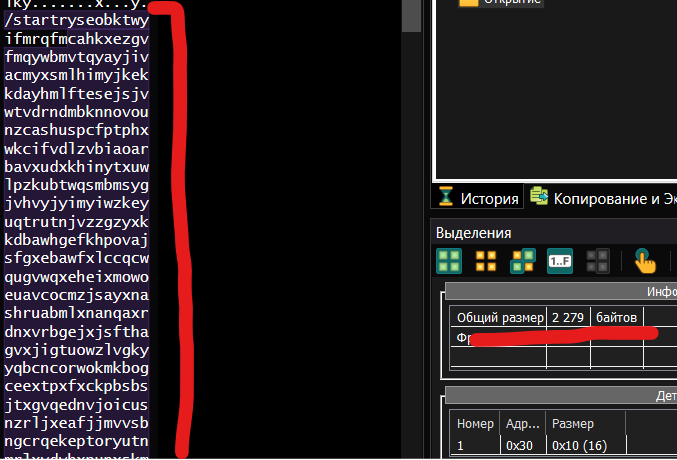


Рисунок 9 – Размер данных в исходном конфигурационном файле

Разработанный IDAPython скрипт также находит адрес уязвимой strcpy. Выходные данные при запуске скрипта:

---------------------------------------------------------------------------------------

Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)]

IDAPython v7.4.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

---------------------------------------------------------------------------------------

INPUT FUNCTION CALLS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Function: fread

Call addresses:

0x401693

UNSAFE FUNCTION CALLS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Function: printf

Call addresses:

0x401506

0x401521

0x4015DA

0x401611

0x4016EB

Function: sprintf

Call addresses:

0x40154E

Function: memset

Call addresses:

0x4015AF

Function: strcpy

Call addresses:

0x4015C4

Function: memcpy

Call addresses:

0x4016C0

0x401FA4

0x401FEE

0x4028E8

0x402A69

0x402C5A

0x4035C9

0x40705D

0x407A23

Function: vfprintf

Call addresses:

0x401F50

Если проверить адрес 0x4015C4, можно увидеть вызов strcpy внутри vuln\_func, как показано на рисунке 10.

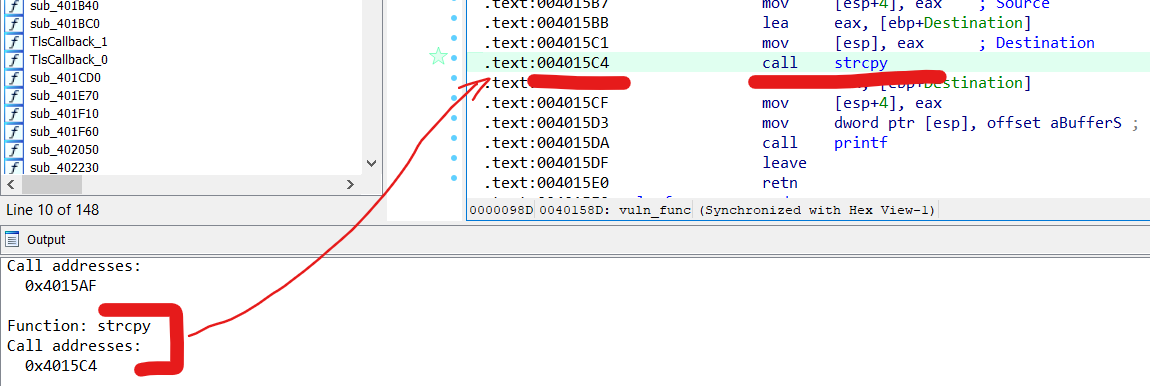


Рисунок 10 – Проверка результата работы IDAPython скрипта

## Дополнительное задание

В качестве дополнительного задания требуется осуществить фаззинг формата файла одного из популярных решений (был выбран 7Zip) с использованием разработанного фаззера.

Формат .zip файла следующий. Первая часть файла содержит файлы с данными в произвольном порядке, вторая часть (так называемый центральный каталог) представляет собой список всех файлов с информацией о каждом, конец центрального каталога содержит сигнатуру и информацию о расположении каталога.

Локальный заголовок файла (в части 1) для каждого файла содержит:

* Сигнатуру 0x04034b50, указывающую на начало файла – 4 байта;
* Версию zip – 2 байта;
* Флаги – 2 байта;
* Метод сжатия – 2 байта;
* Дату/время модификации – 2+2 байта;
* CRC-32 – 4 байта;
* Размеры файла до/после сжатия – 4+4 байта;
* Длина имени файла – 2 байта;
* Длина доп. поля (для расширений формата) – 2 байта;
* Имя файла – длины, заданной выше;
* Дополнительное поле – длины, заданной выше.

Центральный заголовок файла содержит:

* Сигнатуру 0x02014b50, указывающую на начало записи в каталоге – 4 байта;
* Поля из локального заголовка файла;
* Комментарий к файлу – 2 байта;
* Номер диска (для многотомных архивов) – 2 байта;
* Смещение локального заголовка (местонахождение файла в архиве) – 4 байта.

Конец центрального каталога содержит:

* Сигнатуру 0x06054b50, указывающую на конец архива – 4 байта;
* Номера дисков: текущего, с началом центрального каталога (для многотомных архивов) – 2+2 байта;
* Количество файлов (в центральном каталоге на этом диске и в общем в центральном каталоге) – 2+2 байта;
* Размер центрального каталога – 4 байта;
* Смешение каталога (местонахождение центрального каталога в архиве) – 4 байта;
* Длина комментария к архиву – 2 байта;
* Комментарий к архиву.

Эти части можно увидеть, открыв содержимое тестового архива, как показано на рисунке 11. Красным выделены сигнатуры начала и окончания частей файла, синим – название файла, голубым – его содержимое.

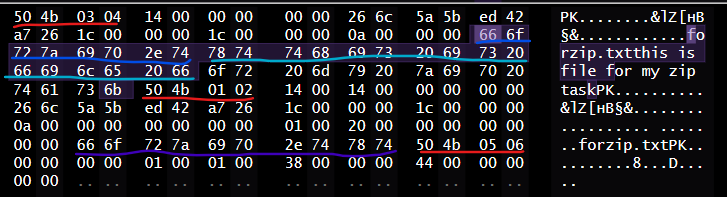


Рисунок 11 – Содержимое .zip файла

Исходный фаззер был усовершенствован для изменения конкретных частей .zip файлов. За основу работы фаззера также берется некоторый исходный .zip-файл, который подвергается мутациям. Мутации выбираются случайным образом и состоят из:

1. Порчи 30 байтов после сигнатуры локального заголовка (поля флагов, заголовков и проч.);
2. Порчи 46 байтов после сигнатуры центрального каталога (записи центрального каталога);
3. Изменение значения байта метода сжатия;
4. Изменение значения CRC-32;
5. Изменение значения полей размеров файлов;
6. Внедрение случайных заголовков;
7. Мутация случайных битов (инверсия);
8. Мутация случайных байтов (граничные значения);
9. Замена последовательности байтов одним значением;
10. Мутация путем сложения/вычитания/умножения случайных байтов на случайное значение.

Фаззер был запущен и нашел 50 случаев падения программы (это сигнал к ранней остановке фаззера, несмотря на число итераций), как показано на рисунке 12.

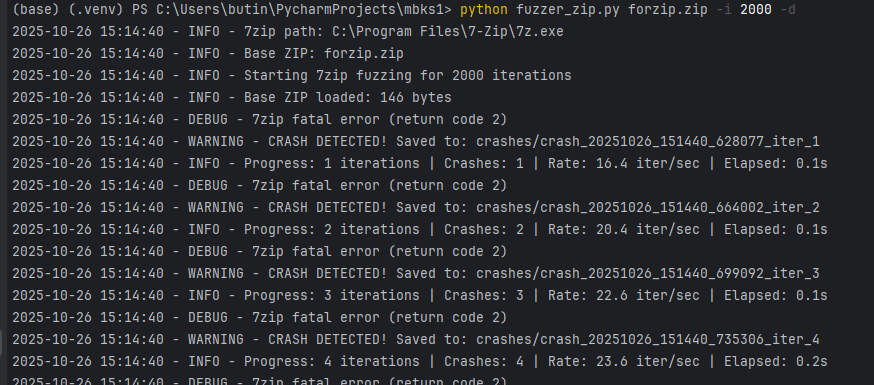


Рисунок 12 – Результаты работы фаззера на 7Zip

При анализе выходных данных фаззера обнаружены ошибки при некорректных входных данных, показанные в таблице 1. Программа не падала из-за исключения, но не была способна корректно обработать входные данные.

Таблица 1 – Найденные ошибки при обработке данных

|  |  |  |
| --- | --- | --- |
| Ошибка | stdout | Изменения в файле |
| Файл не является архивом | === STDOUT ===  7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  Scanning the drive for archives:  1 file, 146 bytes (1 KiB)  Testing archive: fuzzed\_files\fuzz\_000004.zip  Can't open as archive: 1  Files: 0  Size: 0  Compressed: 0  === STDERR ===  ERROR: fuzzed\_files\fuzz\_000004.zip  fuzzed\_files\fuzz\_000004.zip  Open ERROR: Cannot open the file as [zip] archive  ERRORS:  Is not archive  === RETURN CODE: 2 === | Удалены некоторые байты после первой сигнатуры (в т.ч. версия .zip) |
| Проверка CRC не пройдена | === STDOUT ===  7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  Scanning the drive for archives:  1 file, 146 bytes (1 KiB)  Testing archive: fuzzed\_files\fuzz\_000002.zip  --  Path = fuzzed\_files\fuzz\_000002.zip  Type = zip  ERRORS:  Headers Error  Physical Size = 146  Characteristics = Local Central  Sub items Errors: 1  Archives with Errors: 1  Open Errors: 1  Sub items Errors: 1  === STDERR ===  ERRORS:  Headers Error  ERROR: CRC Failed : forzip.txt | Изменение значения CRC-32 |
| Данные после архива | === STDOUT ===  7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  Scanning the drive for archives:  1 file, 179 bytes (1 KiB)  Testing archive: fuzzed\_files\fuzz\_000003.zip  WARNINGS:  There are data after the end of archive  --  Path = fuzzed\_files\fuzz\_000003.zip  Type = zip  ERRORS:  Headers Error  WARNINGS:  There are data after the end of archive  Physical Size = 128  Tail Size = 51  Characteristics = Local  Archives with Errors: 1  Warnings: 1  Open Errors: 1  === STDERR ===  ERRORS:  Headers Error  === RETURN CODE: 2 === | Удаление байтов из конца файла (затронуты байты размера комментария) |
| Неожиданный конец архива | === STDOUT ===  7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  Scanning the drive for archives:  1 file, 146 bytes (1 KiB)  Testing archive: fuzzed\_files\fuzz\_000005.zip  --  Path = fuzzed\_files\fuzz\_000005.zip  Type = zip  ERRORS:  Unexpected end of archive  Physical Size = 146  Characteristics = Local  Sub items Errors: 1  Archives with Errors: 1  Open Errors: 1  Sub items Errors: 1  === STDERR ===  ERRORS:  Unexpected end of archive  ERROR: CRC Failed : forzip.txt  === RETURN CODE: 2 === | Байты размера имеют некорректное значение |
| Ошибка данных | === STDOUT ===  7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  Scanning the drive for archives:  1 file, 146 bytes (1 KiB)  Testing archive: fuzzed\_files\fuzz\_000006.zip  --  Path = fuzzed\_files\fuzz\_000006.zip  Type = zip  ERRORS:  Headers Error  Physical Size = 146  Characteristics = Local Central  Sub items Errors: 1  Archives with Errors: 1  Open Errors: 1  Sub items Errors: 1  === STDERR ===  ERRORS:  Headers Error  ERROR: Data Error : forzip.txt  === RETURN CODE: 2 === | Некорректное имя данных |
| Не подтверждено начало архива | === STDOUT ===  7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  Scanning the drive for archives:  1 file, 162 bytes (1 KiB)  Testing archive: fuzzed\_files\fuzz\_000020.zip  WARNINGS:  There are data after the end of archive  --  Path = fuzzed\_files\fuzz\_000020.zip  Type = zip  ERRORS:  Headers Error  Unconfirmed start of archive  WARNINGS:  There are data after the end of archive  Physical Size = 72  Tail Size = 90  Characteristics = Local  Sub items Errors: 1  Archives with Errors: 1  Warnings: 1  Open Errors: 1  Sub items Errors: 1  === STDERR ===  ERRORS:  Headers Error  Unconfirmed start of archive  ERROR: CRC Failed : forzip.txt  === RETURN CODE: 2 === | Испорчены байты сигнатуры |
| Попытка поместить указатель на файл перед началом файла | === STDOUT ===  7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  Scanning the drive for archives:  1 file, 146 bytes (1 KiB)  Testing archive: fuzzed\_files\fuzz\_000029.zip  Can't open as archive: 1  Files: 0  Size: 0  Compressed: 0  === STDERR ===  ERROR: fuzzed\_files\fuzz\_000029.zip  Cannot open the file as archive  ⪠ 㪠⥫ 䠩 । 砫 䠩.  === RETURN CODE: 2 === | Изменение байта, указывающего на смещение файла |

Подтверждение, что такие ошибки действительно существуют, можно рассмотреть при запуске 7Zip на некорректных данных – программа действительно не может их обработать, хотя работает в штатном режиме, как показано на рисунках 13-15.

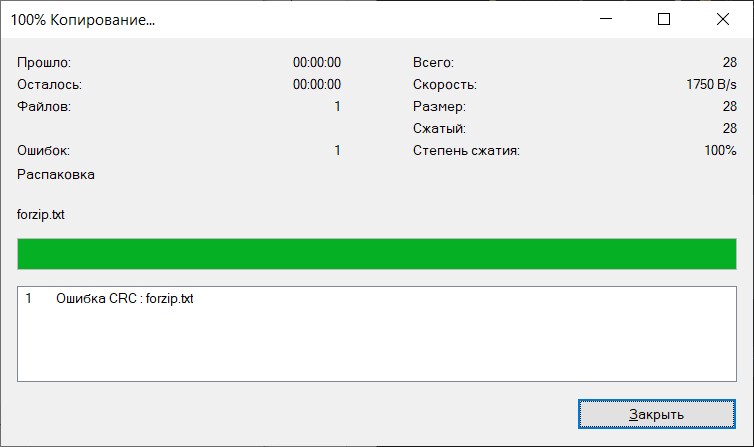


Рисунок 13 – Ошибка в CRC-32

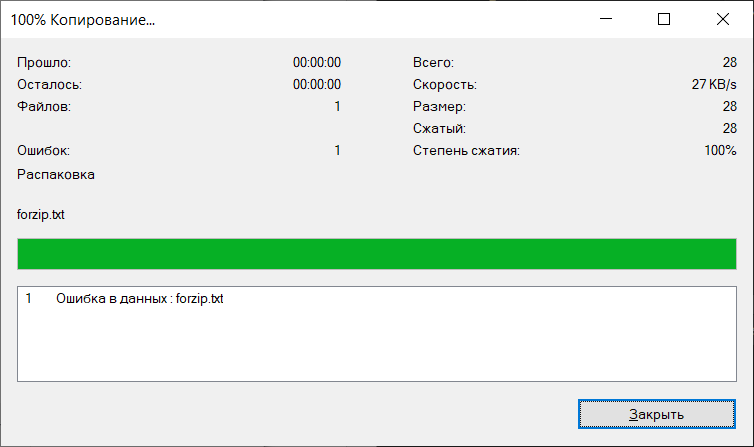


Рисунок 14 – Ошибка в данных

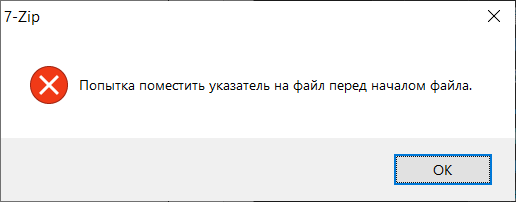


Рисунок 15 – Ошибка в указателе на файл

# Вывод

В ходе лабораторной работы были изучены основные типы ошибок в программном обеспечении, разработан фаззер формата файла для программы без открытого исходного кода и найдено уязвимое к переполнению буфера место в этой программе, а также фаззер формата .zip для программы 7Zip, при помощи которого были получены данные, приводящие к невозможности обработать их программой 7Zip.

Приложение А

Листинг программы «fuzzer.py»

import os  
import re  
import sys  
import logging  
import shutil  
import hashlib  
import random  
import subprocess  
import argparse  
from datetime import datetime  
  
logger = logging.getLogger("AdvancedFuzzer")  
  
  
def setup\_logger(debug=False):  
 logger.setLevel(logging.DEBUG if debug else logging.INFO)  
 formatter = logging.Formatter(  
 '%(asctime)s - %(levelname)s - %(message)s',  
 datefmt='%Y-%m-%d %H:%M:%S'  
 )  
 console\_handler = logging.StreamHandler()  
 console\_handler.setFormatter(formatter)  
 log\_file = f"fuzzer\_{datetime.now().strftime('%Y%m%d\_%H%M%S')}.log"  
 file\_handler = logging.FileHandler(log\_file)  
 file\_handler.setFormatter(formatter)  
 logger.addHandler(console\_handler)  
 logger.addHandler(file\_handler)  
 return log\_file  
  
  
class Fuzzer:  
 def \_\_init\_\_(self, config\_path, debug=False, windbg\_path=None, target\_program=None):  
 self.config\_path = config\_path  
 self.backup\_path = f"{config\_path}.bak"  
 self.original\_hash = None  
 self.debug = debug  
 # self.windbg\_path = windbg\_path or r"C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\windbg.exe"  
 self.windbg\_path = windbg\_path or r"C:\Program Files (x86)\Windows Kits\10\Debuggers\x86\windbg.exe"  
 self.target\_program = target\_program or "vuln1.exe" # По умолчанию vuln1.exe  
 self.setup\_environment()  
 self.crash\_count = 0  
 self.last\_trace\_hash = None  
 self.boundary\_values = [  
 0x00, 0xFF,  
 0xFFFF, 0xFFFF // 2, 0xFFFF // 2 + 1, 0xFFFF // 2 - 1,  
 0xFFFFFF, 0xFFFFFFFF  
 ]  
  
 def setup\_environment(self):  
 if not os.path.exists(self.config\_path):  
 logger.error("Config file not found!")  
 sys.exit(1)  
 shutil.copyfile(self.config\_path, self.backup\_path)  
 self.original\_hash = self.calculate\_hash(self.config\_path)  
 logger.info(f"Backup created: {self.backup\_path}")  
  
 def calculate\_hash(self, file\_path):  
 with open(file\_path, 'rb') as f:  
 return hashlib.md5(f.read()).hexdigest()  
  
 def restore\_config(self):  
 shutil.copyfile(self.backup\_path, self.config\_path)  
 logger.debug("Config restored from backup")  
  
 def parse\_drcov\_trace(self, coverage\_data):  
 trace = []  
 lines = coverage\_data.decode('utf-8', errors='replace').split('\n')  
 for line in lines:  
 if 'module[' in line:  
 parts = line.split(':')  
 if len(parts) < 2:  
 continue  
 module\_part = parts[0].strip()  
 address\_part = parts[1].split(',')[0].strip()  
  
 if not any(module\_part.endswith(f'module[{i}]') for i in [0, 1]):  
 continue  
  
 try:  
 address = int(address\_part, 16)  
 trace.append(address)  
 except ValueError:  
 continue  
 return trace  
  
 def get\_trace\_hash(self, process\_result):  
 if not process\_result:  
 return None  
 trace = self.parse\_drcov\_trace(process\_result.stderr)  
 return hashlib.md5(str(trace).encode()).hexdigest()  
  
 def mutate\_bytes\_feedback(self, data, position):  
 original\_data = bytearray(data)  
 mutated = False  
  
 num\_bytes = random.choice([1, 2])  
 max\_offset = len(data) - position - num\_bytes  
  
 if max\_offset <= 0:  
 return data, False  
  
 offset = random.randint(0, max\_offset)  
 mutation\_value = random.choice(self.boundary\_values)  
  
 try:  
 for i in range(num\_bytes):  
 byte\_offset = position + offset + i  
 if byte\_offset >= len(original\_data):  
 break  
 original\_data[byte\_offset] = (mutation\_value >> (8 \* i)) & 0xFF  
 mutated = True  
 except Exception as e:  
 logger.error(f"Mutation error: {str(e)}")  
  
 return bytes(original\_data), mutated  
  
 def run\_target(self):  
 try:  
 if "7z.exe" in self.target\_program or "Rar.exe" in self.target\_program:  
 drrun\_path = r".\DynamoRIO-Windows-11.90.20196\bin64\drrun.exe"  
 else:  
 drrun\_path = r".\DynamoRIO-Windows-11.90.20196\bin32\drrun.exe"  
  
 dr\_process = subprocess.run(  
 [drrun\_path,  
 "-t", "drcov", "-dump\_text", "-logdir", "coverage", "--", self.target\_program, self.config\_path],  
 stdout=subprocess.PIPE,  
 stderr=subprocess.PIPE,  
 timeout=10  
 )  
 crash\_detected = self.analyze\_execution(dr\_process)  
 self.log\_coverage(dr\_process.stderr)  
 return crash\_detected, dr\_process  
 except subprocess.TimeoutExpired:  
 logger.warning("Process timeout!")  
 return True, None  
 except Exception as e:  
 logger.error(f"Execution error: {str(e)}")  
 return False, None  
  
 def analyze\_execution(self, process\_result):  
 if process\_result is None:  
 return False  
 crash\_signals = ['SEGV', 'ILL', 'ABRT', 'FPE', 'BUS']  
 stdout = process\_result.stdout.decode('utf-8', errors='replace')  
 stderr = process\_result.stderr.decode('utf-8', errors='replace')  
 if process\_result.returncode not in [0, 1]:  
 logger.debug(f"Non-zero exit code: {process\_result.returncode}")  
 return True  
 if any(sig in stderr for sig in crash\_signals):  
 logger.debug("Crash signal detected")  
 return True  
 error\_patterns = [  
 'segmentation-fault', 'access-violation',  
 'buffer-overflow', 'stack-smashing'  
 ]  
 if any(patt in stdout.lower() or patt in stderr.lower() for patt in error\_patterns):  
 logger.debug("Error pattern detected")  
 return True  
 return False  
  
 def log\_coverage(self, coverage\_data):  
 if b"Coverage data" in coverage\_data:  
 logger.info("New code coverage data recorded")  
  
 def save\_crash(self, iteration, data, process\_result):  
 self.crash\_count += 1  
 crash\_dir = f"crashes/crash\_{datetime.now().strftime('%Y%m%d\_%H%M%S')}"  
 os.makedirs(crash\_dir, exist\_ok=True)  
  
 crash\_config = os.path.join(crash\_dir, "crash\_config.bin")  
 with open(crash\_config, 'wb') as f:  
 f.write(data)  
  
 if process\_result:  
 with open(os.path.join(crash\_dir, "execution.log"), 'w') as f:  
 f.write("=== STDOUT ===\n")  
 f.write(process\_result.stdout.decode('utf-8', errors='replace'))  
 f.write("\n=== STDERR ===\n")  
 f.write(process\_result.stderr.decode('utf-8', errors='replace'))  
  
 if self.windbg\_path:  
 if not os.path.isfile(self.windbg\_path):  
 logger.error(f"Windbg executable not found at {self.windbg\_path}")  
 return  
  
 temp\_backup = f"{self.config\_path}.temp.bak"  
 try:  
 shutil.copyfile(self.config\_path, temp\_backup)  
  
 with open(self.config\_path, 'wb') as f:  
 f.write(data)  
  
 dump\_file = os.path.join(crash\_dir, "crash.dmp")  
 log\_file = os.path.join(crash\_dir, "windbg.log")  
  
 windbg\_cmd = [  
 self.windbg\_path,  
 '-c',  
 f'.dump /ma {dump\_file};'  
 f'.logopen {log\_file};'  
 'g;'   
 '!analyze -v;'  
 '.logclose;'  
 'q',  
 'vuln/vuln1.exe'  
 ]  
  
 try:  
 subprocess.run(  
 windbg\_cmd,  
 stdout=subprocess.PIPE,  
 stderr=subprocess.PIPE,  
 timeout=20  
 )  
 logger.info(f"Windbg analysis completed for crash {crash\_dir}")  
 except subprocess.TimeoutExpired:  
 logger.warning("Windbg execution timed out")  
 except Exception as e:  
 logger.error(f"Windbg execution failed: {str(e)}")  
  
 finally:  
 if os.path.exists(temp\_backup):  
 shutil.copyfile(temp\_backup, self.config\_path)  
 os.remove(temp\_backup)  
 logger.debug("Original config restored after Windbg analysis")  
  
 logger.warning(f"Crash saved in: {crash\_dir}")  
  
 def mutate\_append(self, data, step):  
 cleaned = data.rstrip(b'\x00')  
 return cleaned + b'A' \* step  
  
 def mutate\_bytes(self, data, position):  
 mutation = random.choice([  
 lambda d: d[:position] + bytes([0x00] \* len(d[:position])) + d[position:],  
 lambda d: d[:position] + bytes([0xFF] \* len(d[:position])) + d[position:],  
 lambda d: d[:position] + os.urandom(len(d[:position])) + d[position:]  
 ])  
 return mutation(data)  
  
 def replace\_specific\_byte(self, data, byte\_position, new\_byte\_value):  
 if byte\_position < 0 or byte\_position >= len(data):  
 logger.error(f"Invalid byte position: {byte\_position}. File size: {len(data)} bytes")  
 return data, False  
  
 if new\_byte\_value < 0 or new\_byte\_value > 255:  
 logger.error(f"Invalid byte value: {new\_byte\_value}. Must be 0-255")  
 return data, False  
  
 modified\_data = bytearray(data)  
 old\_value = modified\_data[byte\_position]  
 modified\_data[byte\_position] = new\_byte\_value  
  
 logger.info(f"Replaced byte at position {byte\_position}: 0x{old\_value:02X} -> 0x{new\_byte\_value:02X}")  
 return bytes(modified\_data), True  
  
 def run\_mode(self, mode, iterations=10000):  
 logger.info(f"Starting fuzzing in mode {mode}")  
  
 if mode == 3:  
 try:  
 with open(self.config\_path, 'rb') as f:  
 file\_data = f.read()  
 file\_size = len(file\_data)  
  
 print(f"\nФайл {self.config\_path} содержит {file\_size} байт")  
 print("Позиции байтов: от 0 до", file\_size - 1)  
  
 while True:  
 try:  
 byte\_position = int(input("Введите номер байта для замены (0-{}): ".format(file\_size - 1)))  
 if 0 <= byte\_position < file\_size:  
 break  
 else:  
 print(f"Ошибка: позиция должна быть от 0 до {file\_size - 1}")  
 except ValueError:  
 print("Ошибка: введите корректное число")  
  
 while True:  
 try:  
 new\_byte\_value = int(input("Введите новое значение байта (0-255): "))  
 if 0 <= new\_byte\_value <= 255:  
 break  
 else:  
 print("Ошибка: значение должно быть от 0 до 255")  
 except ValueError:  
 print("Ошибка: введите корректное число")  
  
 logger.info(f"Mode 3: Replacing byte at position {byte\_position} with value {new\_byte\_value}")  
  
 except Exception as e:  
 logger.error(f"Error getting user input for mode 3: {str(e)}")  
 return  
  
 try:  
 for i in range(1, iterations + 1):  
 self.restore\_config()  
 with open(self.config\_path, 'rb') as f:  
 original\_data = f.read()  
 if mode == 1:  
 modified\_data = self.mutate\_append(original\_data, i)  
  
 with open(self.config\_path, 'wb') as f:  
 f.write(modified\_data)  
  
 crash\_detected, result = self.run\_target()  
 current\_trace\_hash = self.get\_trace\_hash(result)  
  
 if current\_trace\_hash != self.last\_trace\_hash:  
 logger.info(f"Iteration {i} - NEW CODE PATH DETECTED")  
 self.last\_trace\_hash = current\_trace\_hash  
  
 if crash\_detected:  
 logger.info(f"Iteration {i} - CRASH DETECTED")  
 self.save\_crash(i, modified\_data, result)  
  
 elif mode == 2:  
 pos = original\_data.find(b'/start')  
 if pos == -1:  
 logger.error("'/start' marker not found!")  
 return  
  
 modified\_data, mutated = self.mutate\_bytes\_feedback(original\_data, pos)  
 if not mutated:  
 continue  
  
 with open(self.config\_path, 'wb') as f:  
 f.write(modified\_data)  
  
 crash\_detected, result = self.run\_target()  
 current\_trace\_hash = self.get\_trace\_hash(result)  
  
 if current\_trace\_hash != self.last\_trace\_hash:  
 logger.info(f"Iteration {i} - NEW CODE PATH DETECTED")  
 self.last\_trace\_hash = current\_trace\_hash  
 else:  
 self.restore\_config()  
  
 if crash\_detected:  
 logger.info(f"Iteration {i} - CRASH DETECTED")  
 self.save\_crash(i, modified\_data, result)  
  
 elif mode == 3:  
 modified\_data, success = self.replace\_specific\_byte(original\_data, byte\_position, new\_byte\_value)  
 if not success:  
 logger.error("Failed to replace byte")  
 continue  
  
 new\_config\_path = f"{self.config\_path}.modified"  
 with open(new\_config\_path, 'wb') as f:  
 f.write(modified\_data)  
  
 original\_config\_path = self.config\_path  
 self.config\_path = new\_config\_path  
  
 crash\_detected, result = self.run\_target()  
 current\_trace\_hash = self.get\_trace\_hash(result)  
  
 self.config\_path = original\_config\_path  
  
 if current\_trace\_hash != self.last\_trace\_hash:  
 logger.info(f"Iteration {i} - NEW CODE PATH DETECTED")  
 self.last\_trace\_hash = current\_trace\_hash  
  
 if crash\_detected:  
 logger.info(f"Iteration {i} - CRASH DETECTED")  
 self.save\_crash(i, modified\_data, result)  
  
 logger.info("Mode 3: Single byte replacement completed")  
 logger.info(f"Original config preserved: {original\_config\_path}")  
 logger.info(f"Modified config saved to: {new\_config\_path}")  
 break  
  
 if i % 50 == 0:  
 logger.info(f"Iteration {i}/{iterations} - Crashes: {self.crash\_count}")  
 if mode == 2:  
 logger.debug(f"Current trace hash: {self.last\_trace\_hash}")  
 if self.crash\_count > 0:  
 break  
 finally:  
 if mode != 3:  
 self.restore\_config()  
 logger.info("Fuzzing completed. Original config restored.")  
 else:  
 logger.info("Mode 3 completed. Original config was never modified.")  
  
  
def main():  
 parser = argparse.ArgumentParser(description="Fuzzer")  
 parser.add\_argument("config", help="Path to config file")  
 parser.add\_argument("-m", "--mode", type=int, choices=[1, 2, 3], required=True,  
 help="Fuzzing mode: 1 - Append, 2 - Byte mutation, 3 - Specific byte replacement")  
 parser.add\_argument("-i", "--iterations", type=int, default=10000,  
 help="Number of iterations")  
 parser.add\_argument("-d", "--debug", action="store\_true",  
 help="Enable debug mode")  
 parser.add\_argument("--windbg-path", type=str, default=None,  
 help="Path to Windbg executable") # (optional, for crash analysis)")  
 parser.add\_argument("--target", type=str, default="vuln1.exe",  
 help="Target program to fuzz (e.g., 7z.exe, Rar.exe)")  
  
 args = parser.parse\_args()  
 log\_file = setup\_logger(args.debug)  
 logger.info(f"Starting fuzzer with config: {args.config}")  
  
 try:  
 fuzzer = Fuzzer(args.config, args.debug, args.windbg\_path, args.target)  
 fuzzer.run\_mode(args.mode, args.iterations)  
 except KeyboardInterrupt:  
 logger.info("Fuzzing interrupted by user")  
 except Exception as e:  
 logger.critical(f"Critical error: {str(e)}")  
 finally:  
 logger.info(f"Logs saved to: {log\_file}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 os.makedirs("crashes", exist\_ok=True)  
 os.makedirs("coverage", exist\_ok=True)  
 main()

Приложение B

Листинг программы «fuzzer\_zip.py»

import os  
import sys  
import logging  
import shutil  
import hashlib  
import random  
import subprocess  
import argparse  
import zipfile  
import time  
from datetime import datetime  
  
logger = logging.getLogger("7ZipFuzzer")  
  
  
def setup\_logger(debug=False):  
 *"""Настройка системы логирования"""* logger.setLevel(logging.DEBUG if debug else logging.INFO)  
 formatter = logging.Formatter(  
 '%(asctime)s - %(levelname)s - %(message)s',  
 datefmt='%Y-%m-%d %H:%M:%S'  
 )  
  
 # Консольный вывод  
 console\_handler = logging.StreamHandler()  
 console\_handler.setFormatter(formatter)  
 logger.addHandler(console\_handler)  
  
 # Файловый вывод  
 log\_file = f"7zip\_fuzzer\_{datetime.now().strftime('%Y%m%d\_%H%M%S')}.log"  
 file\_handler = logging.FileHandler(log\_file)  
 file\_handler.setFormatter(formatter)  
 logger.addHandler(file\_handler)  
  
 return log\_file  
  
  
class SevenZipFuzzer:  
 def \_\_init\_\_(self, base\_zip\_path, sevenzip\_path=None, debug=False):  
 self.base\_zip\_path = base\_zip\_path  
 self.sevenzip\_path = sevenzip\_path or self.find\_7zip()  
 self.debug = debug  
 self.crash\_count = 0  
 self.iteration\_count = 0  
 self.start\_time = None  
  
 # Проверки  
 if not os.path.exists(self.base\_zip\_path):  
 logger.error(f"Base ZIP file not found: {self.base\_zip\_path}")  
 sys.exit(1)  
  
 if not self.sevenzip\_path:  
 logger.error("7zip not found! Please specify path with --sevenzip-path")  
 sys.exit(1)  
  
 logger.info(f"7zip path: {self.sevenzip\_path}")  
 logger.info(f"Base ZIP: {self.base\_zip\_path}")  
  
 def find\_7zip(self):  
 *"""Автоматически ищет 7zip в стандартных местах"""* possible\_paths = [  
 r"C:\Program Files\7-Zip\7z.exe",  
 r"C:\Program Files (x86)\7-Zip\7z.exe",  
 r"C:\Program Files\7-Zip\7zG.exe",  
 r"C:\Program Files (x86)\7-Zip\7zG.exe",  
 ]  
  
 for path in possible\_paths:  
 if os.path.exists(path):  
 return path  
 return None  
  
 def create\_base\_zip(self):  
 *"""Создает базовый ZIP файл если он не существует"""* if not os.path.exists(self.base\_zip\_path):  
 logger.info(f"Creating base ZIP file: {self.base\_zip\_path}")  
 with zipfile.ZipFile(self.base\_zip\_path, 'w', zipfile.ZIP\_DEFLATED) as zipf:  
 # Добавляем разнообразные тестовые файлы  
 zipf.writestr("normal\_file.txt", "This is a normal text file for fuzzing" \* 10)  
 zipf.writestr("binary\_data.bin", bytes([i % 256 for i in range(1000)]))  
 zipf.writestr("empty\_file.txt", "")  
 zipf.writestr("folder/nested\_file.txt", "Nested file content")  
 zipf.writestr("very\_long\_name\_" + "x" \* 100 + ".txt", "File with long name")  
  
 logger.info("Base ZIP file created successfully")  
  
 def mutate\_zip\_structure(self, data):  
 *"""Основная функция мутации ZIP структуры"""* if len(data) == 0:  
 return data  
  
 mutated\_data = bytearray(data)  
 mutation\_type = random.randint(1, 10)  
  
 try:  
 if mutation\_type == 1:  
 return self.\_corrupt\_local\_headers(mutated\_data)  
 elif mutation\_type == 2:  
 return self.\_corrupt\_central\_directory(mutated\_data)  
 elif mutation\_type == 3:  
 return self.\_mutate\_compression\_methods(mutated\_data)  
 elif mutation\_type == 4:  
 return self.\_corrupt\_crc\_values(mutated\_data)  
 elif mutation\_type == 5:  
 return self.\_mutate\_file\_sizes(mutated\_data)  
 elif mutation\_type == 6:  
 return self.\_inject\_random\_headers(mutated\_data)  
 elif mutation\_type == 7:  
 return self.\_bit\_flip\_mutation(mutated\_data)  
 elif mutation\_type == 8:  
 return self.\_boundary\_value\_mutation(mutated\_data)  
 elif mutation\_type == 9:  
 return self.\_repeat\_byte\_mutation(mutated\_data)  
 else:  
 return self.\_arithmetic\_mutation(mutated\_data)  
  
 except Exception as e:  
 logger.warning(f"Mutation failed: {e}, using bit flip fallback")  
 return self.\_bit\_flip\_mutation(mutated\_data)  
  
 def \_corrupt\_local\_headers(self, data):  
 *"""Портит локальные заголовки файлов"""* signature = b'\x50\x4B\x03\x04'  
 pos = 0  
  
 while pos < len(data) - 4:  
 if data[pos:pos + 4] == signature:  
 # Мутируем различные поля заголовка  
 header\_size = min(30, len(data) - pos)  
 for i in range(4, header\_size):  
 if random.random() < 0.4:  
 data[pos + i] = random.randint(0, 255)  
 pos += header\_size  
 pos += 1  
  
 return bytes(data)  
  
 def \_corrupt\_central\_directory(self, data):  
 *"""Портит центральный каталог"""* signature = b'\x50\x4B\x01\x02'  
 pos = 0  
  
 while pos < len(data) - 4:  
 if data[pos:pos + 4] == signature:  
 # Мутируем записи центрального каталога  
 header\_size = min(46, len(data) - pos)  
 for i in range(4, header\_size):  
 if random.random() < 0.3:  
 data[pos + i] = random.randint(0, 255)  
 pos += header\_size  
 pos += 1  
  
 return bytes(data)  
  
 def \_mutate\_compression\_methods(self, data):  
 *"""Мутирует методы сжатия"""* local\_sig = b'\x50\x4B\x03\x04'  
 central\_sig = b'\x50\x4B\x01\x02'  
  
 for signature in [local\_sig, central\_sig]:  
 pos = 0  
 while pos < len(data) - 8:  
 if data[pos:pos + 4] == signature:  
 # Метод сжатия на позиции +8  
 if pos + 8 < len(data):  
 # 0 - stored, 8 - deflated, 9 - deflate64, etc.  
 data[pos + 8] = random.choice([0, 1, 8, 9, 12, 14, 96, 99, 255])  
 pos += 1  
  
 return bytes(data)  
  
 def \_corrupt\_crc\_values(self, data):  
 *"""Портит CRC32 значения"""* signatures = [b'\x50\x4B\x03\x04', b'\x50\x4B\x01\x02']  
  
 for signature in signatures:  
 pos = 0  
 while pos < len(data) - 16:  
 if data[pos:pos + 4] == signature:  
 # CRC32 находится по смещению +14 от локального заголовка  
 # и +16 от центрального заголовка  
 crc\_offset = 14 if signature == b'\x50\x4B\x03\x04' else 16  
 if pos + crc\_offset + 4 < len(data):  
 for i in range(4):  
 data[pos + crc\_offset + i] = random.randint(0, 255)  
 pos += 1  
  
 return bytes(data)  
  
 def \_mutate\_file\_sizes(self, data):  
 *"""Мутирует размеры файлов"""* signature = b'\x50\x4B\x03\x04'  
 pos = 0  
  
 while pos < len(data) - 26:  
 if data[pos:pos + 4] == signature:  
 # Сжатый размер (смещение +18) и несжатый размер (смещение +22)  
 for offset in [18, 22]:  
 if pos + offset + 4 < len(data):  
 # Генерируем случайный размер  
 size = random.randint(0, 0xFFFFFFFF)  
 for i in range(4):  
 data[pos + offset + i] = (size >> (i \* 8)) & 0xFF  
 pos += 30  
 pos += 1  
  
 return bytes(data)  
  
 def \_inject\_random\_headers(self, data):  
 *"""Внедряет случайные ZIP заголовки"""* headers = [  
 b'\x50\x4B\x03\x04', # Local file header  
 b'\x50\x4B\x01\x02', # Central directory  
 b'\x50\x4B\x05\x06', # End of central directory  
 b'\x50\x4B\x06\x07', # Zip64 end of central directory  
 b'\x50\x4B\x07\x08', # Zip64 end of central directory locator  
 b'\x50\x4B\x08\x07', # Data descriptor  
 ]  
  
 if len(data) < 5000: # Добавляем заголовки только в небольшие файлы  
 header = random.choice(headers)  
 position = random.randint(0, len(data))  
 extra\_data = bytearray(os.urandom(random.randint(10, 100)))  
 new\_data = bytearray(data)  
 new\_data[position:position] = bytearray(header) + extra\_data  
 return bytes(new\_data)  
  
 return bytes(data)  
  
 def \_bit\_flip\_mutation(self, data):  
 *"""Битовая мутация случайных байтов"""* num\_mutations = random.randint(1, min(100, len(data) // 10))  
  
 for \_ in range(num\_mutations):  
 if len(data) == 0:  
 break  
 pos = random.randint(0, len(data) - 1)  
 bit\_mask = 1 << random.randint(0, 7)  
 data[pos] ^= bit\_mask  
  
 return bytes(data)  
  
 def \_boundary\_value\_mutation(self, data):  
 *"""Мутация граничными значениями"""* boundary\_values = [0x00, 0xFF, 0x7F, 0x80, 0xFFFF, 0xFFFFFFFF]  
  
 if len(data) > 10:  
 num\_mutations = random.randint(1, 20)  
 for \_ in range(num\_mutations):  
 pos = random.randint(0, len(data) - 1)  
 if random.random() < 0.3:  
 # Заменяем на граничное значение (только младший байт)  
 value = random.choice(boundary\_values) & 0xFF  
 data[pos] = value  
 else:  
 # Заменяем последовательность байтов  
 length = random.randint(2, min(8, len(data) - pos))  
 for i in range(length):  
 if pos + i < len(data):  
 value = random.choice(boundary\_values)  
 for j in range(min(4, length - i)):  
 if pos + i + j < len(data):  
 data[pos + i + j] = (value >> (j \* 8)) & 0xFF  
 break  
  
 return bytes(data)  
  
 def \_repeat\_byte\_mutation(self, data):  
 *"""Мутация повторяющимися байтами"""* if len(data) > 20:  
 pattern = random.randint(0, 255)  
 start\_pos = random.randint(0, len(data) - 10)  
 length = random.randint(5, min(50, len(data) - start\_pos))  
  
 for i in range(length):  
 if start\_pos + i < len(data):  
 data[start\_pos + i] = pattern  
  
 return bytes(data)  
  
 def \_arithmetic\_mutation(self, data):  
 *"""Арифметическая мутация"""* if len(data) > 10:  
 pos = random.randint(0, len(data) - 1)  
 operation = random.choice(['add', 'sub', 'mul'])  
 value = random.randint(1, 255)  
  
 if operation == 'add':  
 data[pos] = (data[pos] + value) & 0xFF  
 elif operation == 'sub':  
 data[pos] = (data[pos] - value) & 0xFF  
 elif operation == 'mul':  
 data[pos] = (data[pos] \* value) & 0xFF  
  
 return bytes(data)  
  
 def run\_7zip\_test(self, test\_file\_path):  
 *"""Запускает 7zip для тестирования файла"""* try:  
 # Команда для тестирования архива  
 cmd = [self.sevenzip\_path, "t", test\_file\_path]  
  
 process = subprocess.run(  
 cmd,  
 stdout=subprocess.PIPE,  
 stderr=subprocess.PIPE,  
 timeout=15, # Увеличиваем таймаут для больших файлов  
 creationflags=subprocess.CREATE\_NO\_WINDOW  
 )  
  
 return self.analyze\_7zip\_result(process), process  
  
 except subprocess.TimeoutExpired:  
 logger.warning(f"Timeout testing file: {test\_file\_path}")  
 return True, None  
 except Exception as e:  
 logger.error(f"Error testing file {test\_file\_path}: {e}")  
 return False, None  
  
 def analyze\_7zip\_result(self, process\_result):  
 *"""Анализирует результат выполнения 7zip"""* if process\_result is None:  
 return True # Таймаут считаем за потенциальный сбой  
  
 stdout = process\_result.stdout.decode('utf-8', errors='ignore')  
 stderr = process\_result.stderr.decode('utf-8', errors='ignore')  
  
 # Признаки критических ошибок в 7zip  
 crash\_indicators = [  
 'Exception',  
 'Access violation',  
 'Segmentation fault',  
 'CRASH',  
 'Stack overflow',  
 'Heap corruption',  
 'Fatal error',  
 'Internal error'  
 ]  
  
 # 7zip коды возврата: 0 - OK, 1 - Warning, 2 - Fatal error, 7 - Command line error, 8 - Not enough memory  
 if process\_result.returncode == 2:  
 logger.debug("7zip fatal error (return code 2)")  
 return True  
  
 if process\_result.returncode == 8:  
 logger.debug("7zip memory error (return code 8)")  
 return True  
  
 if any(indicator in stdout for indicator in crash\_indicators) or \  
 any(indicator in stderr for indicator in crash\_indicators):  
 logger.debug("Crash indicator found in output")  
 return True  
  
 return False  
  
 def save\_crash(self, iteration, data, process\_result):  
 *"""Сохраняет информацию о сбое"""* self.crash\_count += 1  
 timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S\_%f')  
 crash\_dir = f"crashes/crash\_{timestamp}\_iter\_{iteration}"  
 os.makedirs(crash\_dir, exist\_ok=True)  
  
 try:  
 # Сохраняем мутированный файл  
 crash\_file = os.path.join(crash\_dir, "crash.zip")  
 with open(crash\_file, 'wb') as f:  
 f.write(data)  
  
 # Сохраняем информацию о выполнении  
 info\_file = os.path.join(crash\_dir, "info.txt")  
 with open(info\_file, 'w', encoding='utf-8') as f:  
 f.write(f"7Zip Fuzzer Crash Report\n")  
 f.write(f"Timestamp: {datetime.now()}\n")  
 f.write(f"Iteration: {iteration}\n")  
 f.write(f"File size: {len(data)} bytes\n")  
 f.write(f"File hash (MD5): {hashlib.md5(data).hexdigest()}\n")  
 f.write(f"7zip path: {self.sevenzip\_path}\n")  
  
 if process\_result:  
 # Сохраняем вывод 7zip  
 output\_file = os.path.join(crash\_dir, "7zip\_output.txt")  
 with open(output\_file, 'w', encoding='utf-8') as f:  
 f.write("=== STDOUT ===\n")  
 f.write(process\_result.stdout.decode('utf-8', errors='ignore'))  
 f.write("\n=== STDERR ===\n")  
 f.write(process\_result.stderr.decode('utf-8', errors='ignore'))  
 f.write(f"\n=== RETURN CODE: {process\_result.returncode} ===\n")  
  
 logger.warning(f"CRASH DETECTED! Saved to: {crash\_dir}")  
  
 except Exception as e:  
 logger.error(f"Error saving crash: {e}")  
  
 def print\_stats(self):  
 *"""Выводит статистику фаззинга"""* if self.start\_time:  
 elapsed = time.time() - self.start\_time  
 rate = self.iteration\_count / elapsed if elapsed > 0 else 0  
 logger.info(f"Progress: {self.iteration\_count} iterations | "  
 f"Crashes: {self.crash\_count} | "  
 f"Rate: {rate:.1f} iter/sec | "  
 f"Elapsed: {elapsed:.1f}s")  
  
 def fuzz(self, iterations=10000, output\_dir="fuzzed\_files"):  
 *"""Основной цикл фаззинга"""* logger.info(f"Starting 7zip fuzzing for {iterations} iterations")  
 self.start\_time = time.time()  
  
 # Создаем директорию для временных файлов  
 os.makedirs(output\_dir, exist\_ok=True)  
  
 # Загружаем базовый ZIP  
 with open(self.base\_zip\_path, 'rb') as f:  
 base\_data = f.read()  
  
 logger.info(f"Base ZIP loaded: {len(base\_data)} bytes")  
  
 try:  
 for i in range(1, iterations + 1):  
 self.iteration\_count = i  
  
 # Создаем мутированные данные  
 mutated\_data = self.mutate\_zip\_structure(base\_data)  
  
 # Сохраняем временный файл  
 temp\_file = os.path.join(output\_dir, f"fuzz\_{i:06d}.zip")  
 with open(temp\_file, 'wb') as f:  
 f.write(mutated\_data)  
  
 # Тестируем с 7zip  
 crash\_detected, process\_result = self.run\_7zip\_test(temp\_file)  
  
 # Сохраняем сбой если обнаружен  
 if crash\_detected:  
 self.save\_crash(i, mutated\_data, process\_result)  
  
 # Очищаем временный файл  
 try:  
 os.remove(temp\_file)  
 except:  
 pass  
  
 # Вывод прогресса  
 if i % 100 == 0 or crash\_detected:  
 self.print\_stats()  
  
 # Ранняя остановка если много сбоев  
 if self.crash\_count >= 50:  
 logger.info("Stopping early due to high crash count")  
 break  
  
 except KeyboardInterrupt:  
 logger.info("Fuzzing interrupted by user")  
 except Exception as e:  
 logger.error(f"Fuzzing error: {e}")  
 finally:  
 # Финальная статистика  
 self.print\_stats()  
 logger.info(f"Fuzzing completed. Total crashes: {self.crash\_count}")  
  
  
def main():  
 parser = argparse.ArgumentParser(  
 description='7Zip ZIP Format Fuzzer',  
 formatter\_class=argparse.RawDescriptionHelpFormatter  
 )  
  
 parser.add\_argument('base\_zip', help='Path to base ZIP file for fuzzing')  
 parser.add\_argument('-i', '--iterations', type=int, default=10000,  
 help='Number of fuzzing iterations (default: 10000)')  
 parser.add\_argument('-7', '--sevenzip-path', type=str,  
 help='Path to 7z.exe (auto-detected if not specified)')  
 parser.add\_argument('-d', '--debug', action='store\_true',  
 help='Enable debug output')  
 parser.add\_argument('-o', '--output-dir', type=str, default='fuzzed\_files',  
 help='Directory for temporary files (default: fuzzed\_files)')  
  
 args = parser.parse\_args()  
  
 # Настройка логирования  
 log\_file = setup\_logger(args.debug)  
  
 # Создаем необходимые директории  
 os.makedirs('crashes', exist\_ok=True)  
  
 try:  
 # Создаем и запускаем фаззер  
 fuzzer = SevenZipFuzzer(  
 base\_zip\_path=args.base\_zip,  
 sevenzip\_path=args.sevenzip\_path,  
 debug=args.debug  
 )  
  
 # Создаем базовый ZIP если нужно  
 fuzzer.create\_base\_zip()  
  
 # Запускаем фаззинг  
 fuzzer.fuzz(  
 iterations=args.iterations,  
 output\_dir=args.output\_dir  
 )  
  
 except Exception as e:  
 logger.critical(f"Fatal error: {e}")  
 return 1  
 finally:  
 logger.info(f"Log file: {log\_file}")  
  
 return 0  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 sys.exit(main())