

Министерство образования и науки Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого

—
Институт компьютерных наук и кибербезопасности

ЛАБОРАТОРНАЯ РАБОТА № 2

**«Изучение принципов поиска уязвимостей в программном обеспечении
без исходных кодов»**

по дисциплине «Модели безопасности компьютерных систем»

Выполнила

студентка гр. 5131001/10302

Куковьякина Д. А.

<подпись>

Преподаватель

Овасапян Т. Д.

<подпись>

Санкт-Петербург

2024

ПОСТАНОВКА ЗАДАЧИ

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

Задачи:

- 1) реализовать программу, осуществляющую фаззинг формата файла;
- 2) разработать IDC/IDAPython-скрипт, осуществляющий в программе поиск функций ввода данных и небезопасные функции;
- 3) осуществить поиск уязвимости в выданной программе.

ХОД РАБОТЫ

Теоретические сведения

Тестирование программного обеспечения (Software Testing) — проверка соответствия реальных и ожидаемых результатов поведения программы.

Цель тестирования — проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи.

Типовые ошибки в программном обеспечении:

1. Целочисленное переполнение (когда результат арифметической операции не умещается в выделенный для него диапазон целых чисел);
2. Отсутствие проверки длины копируемых данных (при копировании данных в буфер не проверяется их размер, что может привести к перезаписи памяти за пределами буфера);
3. Утечки памяти (программное обеспечение выделяет память, но не освобождает её после использования, что приводит к утечкам памяти и увеличению использования ресурсов системы);
4. Деление на 0 (может возникнуть при вычислениях, которые в итоге приводят к делению на 0);
5. Рекурсивные ошибки (неправильное использование рекурсии может привести к переполнению стека вызовов функций).

Традиционная практика тестирования предполагает, что на вход программы подаются самые разные данные: корректные, почти корректные и граничные случаи. Это позволяет убедиться, что код работает в целом правильно и серьёзных ошибок в логике работы нет. Однако при этом качество проверки зависит от качества тестов и их количества. Фиксированное количество тестов не может покрыть все возможные варианты входных данных. Поэтому всегда есть вероятность, что какой-то важный случай был упущен, не вошёл в фиксированный набор тестов, а код,

следовательно, всё-таки содержит ошибку, которую достаточно талантливый хакер сможет проэксплуатировать.

Фаззинг — это техника автоматизированного тестирования, при которой на вход программе подаются специально подготовленные данные, которые могут привести её к аварийному состоянию или неопределённому поведению. Фаззеры по операциям, которые будут совершаться над входными данными, делятся на:

- Mutation-Based Fuzzers: Этот тип фаззера проще всего создать, поскольку он изменяет существующие образцы данных для создания новых тестовых данных;

- Generation-Based Fuzzers: Этот тип фаззера создает новые тестовые данные на основе входной модели. Обычно он разбивает протокол или формат файла на фрагменты, которые затем выстраиваются в допустимом порядке, и эти фрагменты случайным образом распределяются независимо друг от друга.

Также их можно разделить по наличию обратной реакции от тестируемого приложения: feedback driven и not feedback driven.

Feedback-driven fuzzing — это вид фаззинга, при котором фаззер изменяет входные данные так, чтобы их обработка затрагивала как можно больше участков кода программы. Работа таких фаззеров возможна, благодаря их способности реагировать на отклик (feedback) программы. Обычно таким откликом является покрытие кода. Метрики покрытия кода отслеживают суммарное количество выполненных строк кода, базовых блоков, количество сделанных условных переходов. Задача фаззера — генерировать данные, которые приводят к увеличению покрытия кода.

Реализация фаззера

Функционал фаззера состоит из двух основных пунктов:

- Дозапись случайных байт в конец файла;
- Последовательное изменение байт в файле.

При выборе пункта «дозапись» необходимо указать количество тестов. В каждом из тестов дописывается $2^{\text{номер_теста}}$ байт, байты генерируются случайно из диапазона английского алфавита нижнего регистра.

При выборе пункта «замена» указывается последний байт, до которого необходимо произвести замену. Затем вводится сам байт, на который будет заменяться, и количество байт, которые будут заменяться в рамках одного теста.

В рамках каждого теста происходит запуск исполняемого файла и подсчет покрытия кода при помощи динамической бинарной инструментации.

Динамическая бинарная инструментация (Dynamic Binary Instrumentation, DBI) заключается во вставке в бинарный исполняющийся код анализирующих процедур. Основное преимущество данного подхода заключается в том, что нет необходимости в исходном коде анализируемого приложения – работа происходит непосредственно с бинарным файлом.

За вставку дополнительного кода обычно отвечают инструментирующие процедуры, которые вызываются только раз при возникновении необходимого события и модифицируют целевую программу. Добавленный код представляет собой анализирующие процедуры. Эти процедуры отвечают за проведение необходимого анализа, модификации и мониторинга исследуемой программы и вызываются каждый раз при достижении определенного участка кода или возникновения в программе определенного события (создание процесса, возникновения исключения и т. д.).

Для реализации использовалась утилита DynamoRio, а именно её компонент drcov. Она генерирует файл с логами, содержащий следующие данные: информация о загружаемых модулях и таблицу базовых блоков, которая содержит список основных блоков, которые были выполнены при сборе информации о покрытии. Базовым блоком является последовательность инструкций, не содержащая инструкций передачи

управления. Для анализа покрытия было взято эталонное значение количества блоков – количество блоков при выполнении файла с правильными конфигурационными данными. Затем после внесения изменений считывается количество блоков и вычисляется их отношение. В случае если отношение близко к 1 (то есть код прошел столько же и больше блоков), то изменение сохраняется в файле, иначе конфигурационный файл откатывается до предыдущего состояния.

Далее рассмотрим случай выполнения программы с ошибкой. Для отслеживания используется структура `DebugEvent`, а именно её поле `dwDebugEventCode`, которое сообщает о возникновении исключения. Код исключения сохраняется в `.u.Exception.ExceptionRecord.ExceptionCode`, были обработаны основные исключения такие как переполнение, деление на ноль и т.д.

В случае возникновения исключения текущий конфигурационный буфер сохраняется в файл с логами, туда же записывается само исключение. Затем обрабатываются значения регистров (`Wow64GetThreadContext`) и состояние стека (`ReadProcessMemory`).

Состояние регистров и стека могут быть полезны при возникновении ошибок. Выводятся следующие значения:

- `Eax` – используется для арифметических операций и хранения результата;
- `Ebx` – используется как указатель на данные в памяти;
- `Ecx` – используется в циклах;
- `Edx` – используется для хранения данных и расширенных результатов умножения и деления;
- `Esp` – указатель на вершину стека;
- `Ebp` – используется для доступа к параметрам функции и локальным переменным;
- `Esi` – используется как источник данных для операций копирования и сравнения;

- Edi – используется как назначение для операций копирования и сравнения;
- Eflags – регистр флагов (Carry flag – результат больше максимального значения, Zero flag – результат равен 0, Sign flag – в соответствии со знаковым битом результата последней операции, Overflow flag – переполнение, Parity flag – 8 младших разрядов содержат четное число единиц).

Далее представлен пример работы программы (рисунок 1 – 5).

```
[c] <number of last byte> - auto-change bytes
[w] <number of tests> - auto-write bytes to the end
[h] - help
[e] - exit

# w 3

[1 of 3]

Correct.
Calculate coverage...

Coverage: 1.00471

#####

[2 of 3]

Correct.
Calculate coverage...

Coverage: 1.00433

#####

[3 of 3]

Correct.
Calculate coverage...

Coverage: 1.00433

#####
```

Рисунок 1 – Дозапись в файл

04 02	00 f4	97 01	00 00	c4 09	00 00	00 00	00 00	...	Φ-...Д.....
97 01	c4 09	08 00	00 00	b8 6b	58 00	b8 6b	58 00	-.	Д.....ëkX.ëkX.
b3 6b	58 00	00 00	00 00	00 00	78 00	00 00	58 00	ikX.....x...X.	
2f 73	74 61	72 74	6f 6c	63 73	71 73	79 70	74 67	/startolcsqsyp	tg
67 77	65 72	66 6d	62 72	64 64	66 61	75 6a	64 71	gwerfmb	rddfaujdq
6e 61	7a 69	6d 76	75 66	65 78	67 7a	66 70	70 72	nazimvufexgzf	p
6e 72	6b 65	62 74	77 69	67 67	62 68	72 73	71 6a	nrkebtwigg	bhrsqs
7a 6e	6d 64	7a 63	61 70	75 77	69 7a	62 6b	77 6f	znm	dzcapuwizbkwo
71 64	70 77	6e 6b	6e 62	63 6a	77 6d	6a 67	73 79	qdpwnknbcjw	mjgsy
6e 65	64 76	77 65	6c 6e	7a 78	71 69	62 74	65 62	nedv	welnzxiqibteb
74 66	64 73	68 6b	71 78	7a 7a	6f 76	6e 6e	72 6c	tfdshkq	xzzovnnrl
71 72	61 63	71 62	7a 64	6a 6a	73 63	66 6e	72 65	qracqb	zdzjjscfnre
78 61	69 6e	6b 76	67 77	6a 6f	64 77	61 6b	64 6d	xainkvgw	jodwakdm
66 6c	6b 6c	71 69	64 77	62 6e	70 61	6d 73	70 6e	flklqidw	bnpamspn
61 6e	75 64	74 7a	64 63	74 7a	64 67	72 6d	73 79	anudt	zdzctzdgmsy
76 6e	6f 6c	76 61	75 6a	63 74	69 63	6f 6f	73 70	vnolva	uajcticoosp
76 75	64 6f	76 74	69 75	62 6a	64 75	6e 78	68 6c	vudovti	ubjdunxhl
64 78	75 76	71 6b	71 6a	72 63	72 77	6a 77	70 70	dxuvqk	qjrcrwjwpp
6a 6e	79 74	66 73	71 61	6d 71	68 6b	62 73	68 68	jnytf	sqamqhkbs
66 72	68 68	6e 6d	78 78	72 65	78 76	6f 77	64 79	frhhnm	xxrexvowdy
6e 66	6a 69	6f 6b	78 6d	74 70	6b 69	64 73	61 70	nfjiok	xmtpkidsap
6a 71	66 6a	70 67	6e 67	6b 6d	73 64	77 6f	79 76	jqfj	pgngkmsdwoyv
63 6e	69 74	70 65	78 73	6d 75	72 70	71 66	76 68	cnitp	exsmurpqf
78 76	6b 62	64 74	69 74	62 70	71 72	6c 69	66 7a	xvk	bdtitbpqrlifz
6a 70	6e 74	61 6e	64 70	70 65	6b 6c	78 74	6f 77	jpnt	andppeklxtow
6d 6a	71 72	61 69	66 6a	75 7a	7a 77	70 6e	73 6e	mjqra	ifjuzzwpnsn
68 77	6d 6b	79 64	78 6f	63 67	6c 73	70 74	66 67	hwmky	dxocglsp
65 6a	6a 64	68 6b	6e 62	71 6a	78 78	64 77	7a 65	ejjdhk	nbqjxxdwze
66 74	6e 66	78 67	71 72	6a 61	74 79	64	ftnf	xgqrjaty

Рисунок 2 – 7 дописанных байт

```
# c 48
Enter byte: ff
Enter number of bytes to replace during one test: 6
[1 of 8]
Correct.
Calculate coverage...
Coverage: 0.996799
#####
[2 of 8]
Exception: EXCEPTION_ACCESS_VIOLATION
Log file: 4008_log.txt
#####
[3 of 8]
Correct.
Calculate coverage...
Coverage: 1.00377
#####
[4 of 8]
Correct.
Calculate coverage...
Coverage: 1.00433
#####
```

Рисунок 3 – Замена байт заголовка на FF

04	02	00	f4	97	01	00	00	c4	09	00	00	ff	ff	ff	ff	...ф...Д...яяяя
ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	яяяяяяяяяяяяяяяя
ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	яяяяяяяяяяяяяяяя
2f	73	74	61	72	74	6f	6c	63	73	71	73	79	70	74	67	/startolcsqsyp
67	77	65	72	66	6d	62	72	64	64	66	61	75	6a	64	71	gawrfmbrddfaui
6e	61	7a	69	6d	76	75	66	65	78	67	7a	66	70	70	72	nazimvufexgzfpp
6e	72	6b	65	62	74	77	69	67	67	62	68	72	73	71	6a	nrkebtwiggbhrrs
7a	6e	6d	64	7a	63	61	70	75	77	69	7a	62	6b	77	6f	znmzdscapuwiwbk
71	64	70	77	6e	6b	6e	62	63	6a	77	6d	6a	67	73	79	qdpwnknbcjwmjgs
6e	65	64	76	77	65	6c	6e	7a	78	71	69	62	74	65	62	nedvwe1nxbqitbe
74	66	64	73	68	6b	71	78	7a	7a	6f	76	6e	6e	72	6c	tfdshkqxzzovnnr
71	72	61	63	71	62	7a	64	6a	6a	73	63	66	6e	72	65	qracqbzdjjscfnr
78	61	69	6e	6b	76	67	77	6a	6f	64	77	61	6b	64	6d	xainkvvgwjodwak
66	6c	6b	6c	71	69	64	77	62	6e	70	61	6d	73	70	6e	f1klqidwnbpaspr
61	6e	75	64	74	7a	64	63	74	7a	64	67	72	6d	73	79	anudtzdctzdgrms
76	6e	6f	6c	76	61	75	6a	63	74	69	63	6f	6f	73	70	vnolvauijcticoos
76	75	64	6f	76	74	69	75	62	6a	64	75	6e	78	68	6c	vudovtiubjdunxh
64	78	75	76	71	6b	71	6a	72	63	72	77	6a	77	70	70	dxyufqjqrchwjwp
6a	6e	79	74	66	73	71	61	6d	71	68	6b	62	73	68	68	jnutvsgaqmhkbsh
66	72	68	68	6e	6d	78	78	72	65	78	76	6f	77	64	79	frhhnmxxrexvowd
6e	66	6a	69	6f	6b	78	6d	74	70	6b	69	64	73	61	70	nfjiokxmtpkidsa
6a	71	66	6a	70	67	6e	67	6b	6d	73	64	77	6f	79	76	qfjfpjngmkdswoy
63	6e	69	74	70	65	78	73	6d	75	72	70	71	66	76	68	cnitpexsmurpqfwh
78	76	6b	62	64	74	69	74	62	70	71	72	6c	69	66	7a	xvkbdtitbpqrlifz
6a	70	6e	74	61	6e	64	70	70	65	6b	6c	78	74	6f	77	jpntrandppeklxtow
6d	6a	71	72	61	69	66	6a	75	7a	7a	77	70	6e	73	6e	mjqraifjzdwzwnsr
68	77	6d	6b	79	64	78	6f	63	67	6c	73	70	74	66	67	hwmkydxocglsptfg
65	6a	6a	64	68	6b	6e	62	71	6a	78	78	64	77	7a	65	ejjdhknbgqjxxdwze
66	74	6e	66	78	67	71	72	6a	61	74	79	64	ftnfxgqrjatyd

```
4008_log.txt - Блокнот
Файл Правка Формат Вид Справка
Exception: EXCEPTION_ACCESS_VIOLATION

Current configuration file:
0 0-яяяяяя -Д 0 äkX äkX ikX x X /startolcsqsyptgwerfmbrrddfaujdqnazimvufexgzf

Registers:
EAX: 0x00000000
EBX: 0xFFFFFFFF
ECX: 0x3FFFFFFB1F
EDX: 0x00006778
EIP: 0x770E9C79
ESP: 0x009FEC50
EBP: 0x009FF648
EDI: 0x009FFFFE
ESI: 0x00E910C0
EFLAGS: 0x00010212

Stack:
00 00 00 00 00 A0 12 EF 00 D0 F8 9F 00 D0 15 EF 00 7C EC 9F 00 2E 0F 09 00 FF FF FF FF 00 00 00 00
20 3A A7 00 04 00 04 00 04 00 00 00 00 00 00 00 6F 6C 63 73 71 73 79 70 74 67 67 77 65 72 66 6D 62 72 64 64
66 61 75 6A 64 71 6E 61 7A 69 6D 76 75 66 65 78 67 7A 66 70 70 72 6E 72 6B 65 62 74 77 69 67 67 67
62 68 72 73 71 6A 7A 6E 6D 64 7A 63 61 70 75 77 69 7A 62 6B 77 6F 71 64 70 77 6E 6B 6E 62 63 6A 7A
77 6D 6A 67 73 79 6E 65 64 76 77 65 6C 6E 7A 78 71 69 62 74 65 62 74 66 64 73 68 6B 71 78 7A 7A 7A
67 76 6E 6E 72 6C 71 72 61 63 71 62 7A 64 6A 6A 73 63 66 6E 72 65 68 61 69 6E 6B 76 67 77 6A 6F
64 77 61 6B 64 6D 66 6C 6B 6C 71 69 64 77 62 6E 70 61 6D 73 70 6E 61 6E 75 64 7A 7A 64 63 74 7A
64 67 72 6D 73 79 76 6E 6F 6C 76 61 75 6A 63 74 69 63 6F 73 70 76 75 64 6F 76 74 69 75 62 6A
64 75 6E 78 68 6C 64 78 75 76 71 6B 71 6A 72 63 72 77 6A 77 70 70 6A 6E 79 74 66 73 71 61 6D 71
68 6B 62 73 68 68 66 72 68 68 6E 6D 78 78 72 65 78 76 6F 77 64 79 76 6E 66 6A 69 6F 6B 78 6D 74 70
68 69 64 73 61 70 6A 71 66 6A 70 67 6E 67 6B 6D 73 64 77 6F 79 76 63 6E 69 74 70 65 78 73 6D 75
72 70 71 66 76 68 78 76 6B 62 64 74 69 74 62 70 71 72 6C 69 66 7A 6A 70 6E 74 61 6E 64 70 70 65
6B 6C 78 74 6F 77 6D 6A 71 72 61 69 66 6A 75 7A 7A 77 70 6E 73 6E 68 77 6D 6B 79 64 78 6F 63 67
```

Скрипт

- `idautils.Heads(start, end)` – получение списка заголовков (инструкций или данных)
- `ida_ua.ua_mnem(ea)` – предназначена для получения мнемоники инструкции в ассемблерном коде, мнемоника — символическое имя для одной исполняемой инструкции машинного языка
- `print_operand(ea, n)` – получение операнда инструкции, `n` – номер операнда

В результате запуска скрипта был получен следующий вывод (рисунок 6).

```
Analyze...
0x0040154e: call sprintf
0x004015af: call memset
0x004015cb: call strncpy
0x00401676: call malloc
0x0040169a: call fread
0x004016c7: call memcpy
0x00401d6f: call calloc
0x00401e3f: call free
0x00401fa4: call memcpy
0x00401fee: call memcpy
0x0040251d: call malloc
0x004027bf: call free
0x004027cf: call free
0x00402801: call realloc
0x0040289f: call free
0x004028e8: call memcpy
0x00402a69: call memcpy
0x00402b42: call malloc
0x00402c1f: call free
0x00402c34: call free
0x00402c5a: call memcpy
0x00402fd8: call realloc
0x00403009: call free
0x0040315a: call realloc
0x0040327a: call free
0x004035a6: call malloc
0x004035c9: call memcpy
0x00403662: call free
0x004036e6: call free
0x0040599b: call atoi
0x00405a45: call atoi
0x00405ba2: call atoi
0x00405cc9: call atoi
0x00405d3b: call atoi
0x0040705d: call memcpy
0x0040787f: call malloc
0x00407a23: call memcpy
Done|
```

Рисунок 6 – Результат работы скрипта

Поиск уязвимости

Проанализируем функцию, отвечающую за парсинг полученного буфера, уже без заголовка (рисунок 7). В ней происходит рекурсия – проверяется наличие в буфере подстроки /start, если она есть, то parse_string вызывается еще раз с обрезанным буфером. Если же подстрока не найдена, то исполнение переходит к vuln_func, в которую передаются текущий буфер, shell_len, dst_len.



Рисунок 7 – Функция parse_string

На рисунках 8 и 9 представлен ассемблерный код и псевдокод этой функции соответственно. По коду видно, что создается буфер фиксированного размера – 2508. Затем он заполняется символами b (код 98). После чего при помощи `strncpy` происходит копирование из текущего буфера в этот созданный, при этом в качестве размера указывается переданный параметр `dst_len`. Исходя из этого становится ясно, что уязвимость программы связана с функцией `strncpy`.

```

vuln_func      public vuln_func
                proc near                ; CODE XREF: parse_string+3B↑p

Destination    = byte ptr -9CCh
Source         = dword ptr  8
Count          = dword ptr  10h

                push    ebp
                mov     ebp, esp
                sub     esp, 9E8h
                mov     dword ptr [esp+8], 9C4h ; Size
                mov     dword ptr [esp+4], 62h ; 'b' ; Val
                lea     eax, [ebp+Destination]
                mov     [esp], eax        ; void *
                call    memset
                mov     eax, [ebp+Count]
                mov     [esp+8], eax     ; Count
                mov     eax, [ebp+Source]
                mov     [esp+4], eax     ; Source
                lea     eax, [ebp+Destination]
                mov     [esp], eax      ; Destination
                call    strncpy
                lea     eax, [ebp+Destination]
                mov     [esp+4], eax
                mov     dword ptr [esp], offset aBufferS ; "buffer: %s\n"
                call    printf
                leave
                retn
vuln_func      endp

```

Рисунок 8 – Код vuln_func

```

int __cdecl vuln_func(char *Source, int a2, size_t Count)
{
    char Destination[2508]; // [esp+1Ch] [ebp-9CCh] BYREF

    memset(Destination, 98, 0x9C4u);
    strncpy(Destination, Source, Count);
    return printf("buffer: %s\n", Destination);
}

```

Рисунок 9 – Псевдокод vuln_func

Заменим в заголовке dst_len на 2510 (рисунок 10).

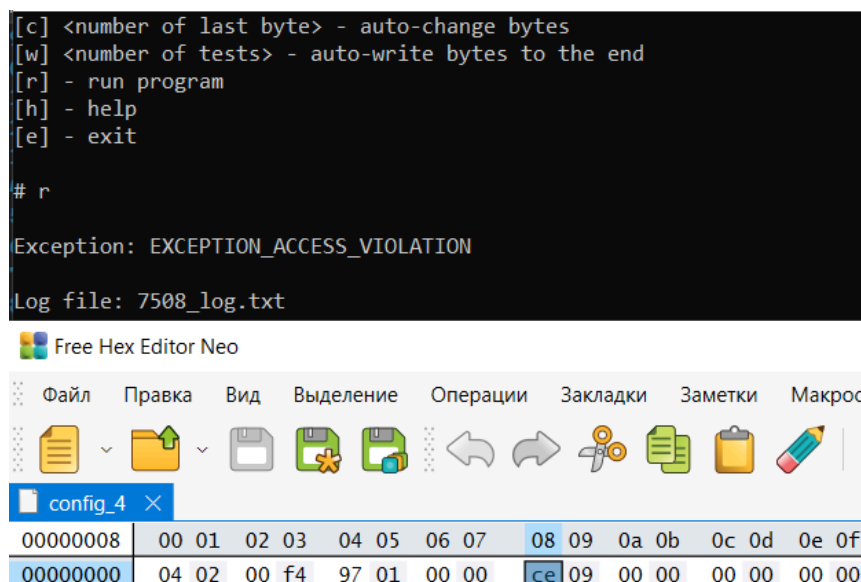


Рисунок 10 – Проверка уязвимости

На рисунке 11 представлено состояние стека и регистров на момент возникновения ошибки, полученное из IDA.

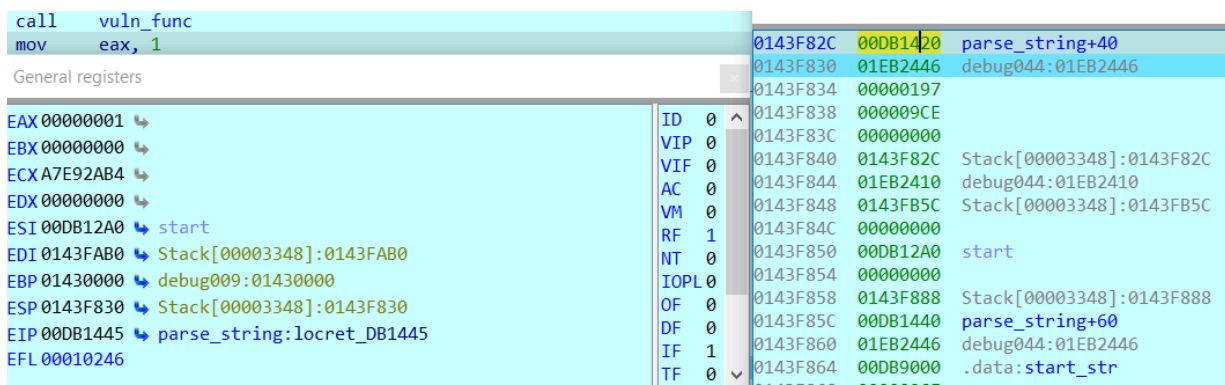


Рисунок 11 – Состояние регистров и стека

Анализ уязвимости

У функции strncpy можно выделить две основные уязвимости:

- не завершает буфер, в который копируется строка, null-символом;
- не осуществляет проверку параметра, переданного в качестве размера.

Рассмотрим два случая: изменение размера src буфера и изменение параметра size.

Исходный буфер всегда завершается нулем, даже если файл нулем не завершается (рисунок 12). В таком случае, если буфер-источник меньше или равен буферу-назначения, то первая уязвимость не возникнет. Однако, если

буфер-источник будет больше буфера-назначения (при этом параметр `size` имеет правильное значение), то символ завершения строки не будет добавлен, что может спровоцировать непредвиденное поведение, например, при вызове функции `printf`, которая выводит строку до встреченного нулевого символа.

```
Buffer = (char *)malloc(ElementCount + 1);
fread(Buffer, 1u, ElementCount, Stream);
Buffer[ElementCount] = 0;
if ( ElementCount > 0x2F )
    memcpy(a2, Buffer, 0x30u);
*a3 = Buffer + 48;
```

Рисунок 12 – Добавление нулевого символа

Для устранения данной уязвимости стоит использовать более безопасные с этой точки зрения функции, например `strncpy` или `snprintf`, которые при копировании строк будут завершать буфер-назначение нулем.

При изменении параметра `size` до значения меньше, чем размер буфера-источника, может возникнуть такая же ситуация (рисунок 13). Тут также произойдет копирование строки без завершающего символа, что приведет к выходу за границы самого буфера при его печати.

[illegible]

Рисунок 13 – Вывод "мусора"

При увеличении значения size произойдет перезапись данных, идущих после буфера-назначения. Данные могут быть заменены как нулевыми

символами (в случае если буфер-источник меньше size), так и значениями из буфера-источника, если позволяет его размер (рисунок 14 и 15). Для устранения данной уязвимости необходимо проверять считанное значение `dst_len` и при необходимости уменьшать его.

```
:00A6FB4E db 62h ; b
:00A6FB4F db 62h ; b
:00A6FB50 db 0A0h
:00A6FB51 db 12h
:00A6FB52 db 5Bh ; [
:00A6FB53 db 0
:00A6FB54 db 0
```

Рисунок 14 – Память до `strncpy`

```
:00A6FB4D db 66h ; f
:00A6FB4E db 62h ; b
:00A6FB4F db 65h ; e
:00A6FB50 db 75h ; u
:00A6FB51 db 6Fh ; o
:00A6FB52 db 73h ; s
:00A6FB53 db 78h ; x
```

Рисунок 15 – Память после `strncpy`

После области памяти, хранящей данный буфер, идут два специальных поля "r" и "s", которые представляют собой обратный адрес и сохраненные регистры (рисунок 16). Поэтому при перезаписи данных полей возникает исключение, так как был затерт адрес возврата.

```

s      db ? ; undefined
r      db ? ; undefined
      db 4 dup(?)
      db 4 dup(?)
```

Рисунок 16 – Специальные поля

ВЫВОД

В ходе работы были изучены типовые ошибки и принципы поиска уязвимостей в программном обеспечении без исходных кодов. Была написана программа на языке C++, осуществляющая автоматический фаззинг формата-файла, и IDAPython-скрипт, осуществляющий поиск небезопасных функций.