Разработка средств защиты информации
Лабораторная 2
Боязитов Вадим
932125

1. Провести эксперимент над примерами с большим объемом данных.

Доступный пул памяти значительно превышает фактически запрашиваемый объем

В рамках файла стресс теста задаем параметры

```
Allocator stress_test_allocator_(1024, 1000000);
```

```
const int kNumBlocks = 100000;
const int kBlockSize = sizeof(TestBlock);
const int kMegabytesToAllocate = kNumBlocks * kBlockSize / (1024 * 1024);
```

Вывод

```
=== ALLOCATOR STRESS TEST PROGRAM ===

Starting allocator stress test...
Attempting to allocate ~97 MB
```

```
Allocated blocks: 90000 (87 MB)
Successfully allocated blocks: 100000 out of 100000
Total allocated memory: 97 MB
Allocation time: 758 ms
Verified blocks: 100, found errors: 0
Starting memory deallocation...
Deallocation time: 3 ms
```

```
===== Allocator statistics after stress test =====
Block size: 1024 bytes
Total blocks: 0
Blocks in use: 0
Allocations: 100000
Deallocations: 100000
```

Результат, все корректно отработало.

Тестирование граничных условий

В рамках файла стресс теста задаем параметры

```
Allocator stress_test_allocator_(1024, 10);
```

```
const int kNumBlocks = 10;
const int kBlockSize = sizeof(TestBlock);
const int kMegabytesToAllocate = kNumBlocks * kBlockSize / (1024 * 1024);
```

Вывод

```
=== ALLOCATOR STRESS TEST PROGRAM ===

Starting allocator stress test...
Attempting to allocate ~0 MB
Allocated blocks: 0 (0 MB)
Successfully allocated blocks: 10 out of 10
Total allocated memory: 0 MB
Allocation time: 0 ms
Verified blocks: 10, found errors: 0
Starting memory deallocation...
Deallocation time: 0 ms
```

```
===== Allocator statistics after stress test =====
Block size: 1024 bytes
Total blocks: 0
Blocks in use: 0
Allocations: 10
Deallocations: 10
```

Результат, все корректно отработало.

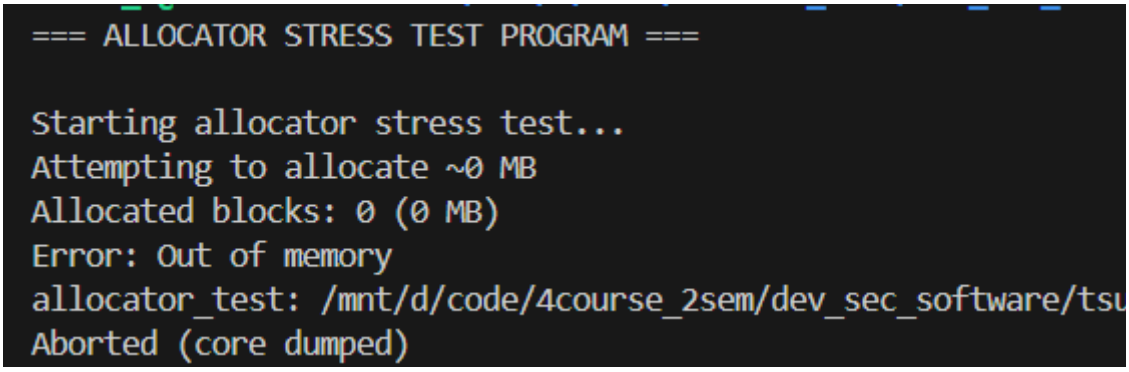
Тест на переполнение

В рамках файла стресс теста задаем параметры

```
Allocator stress_test_allocator_(1024, 100);
```

```
const int kNumBlocks = 150;
const int kBlockSize = sizeof(TestBlock);
const int kMegabytesToAllocate = kNumBlocks * kBlockSize / (1024 * 1024);
```

Вывод



Результат, отловлена ошибка.

2. Обеспечить архитектурную возможность изменения правила выбора переменной ветвления (использовать паттерн "Стратегия") для SAT задачи.

Шаги реализации паттерна

Создать интерфейс стратегий, описывающий этот алгоритм. Он должен быть общим для всех вариантов алгоритма.

```cpp
/*
    Абстрактный класс стратегии ветвления
    который содержит интерфейс стратегии
*/
class BranchingStrategy {
public:
    virtual ~BranchingStrategy() = default;

    virtual int chooseBranchingIndex(BoolInterval** cnf, int cnfSize, BBV& mask) const = 0;
};
```

```cpp
/*
    Стратегия ветвления по столбцам
*/
class ColumnBranchingStrategy : public BranchingStrategy {
public:
    int chooseBranchingIndex(BoolInterval** cnf, int cnfSize, BBV& mask) const override;
};
```

```cpp
/*
    Стратегия ветвления по строкам
*/
class RowBranchingStrategy : public BranchingStrategy {
public:
    int chooseBranchingIndex(BoolInterval** cnf, int cnfSize, BBV& mask) const override;
};
```

Описание реализаций стратегий

Стратегия ветвления по столбцам

Если в столбце мало '-'(отрицаний), значит, переменная чаще встречается в положительной форме, и её выбор может быстрее привести к упрощению формулы.

1. Собрать все индексы переменных, которые не замаскированы (mask[i] == 0)

2. Если таких переменных нет → вернуть -1

3. Для каждой строки (интервала) в CNF:
   Если это первая строка:
      Для каждого доступного столбца:
         если значение равно '-', то записать 1, иначе 0
   Иначе:
      Для каждого доступного столбца:
         если значение равно '-', увеличить счётчик для этого столбца на 1

5. Найти столбец с минимальным количеством '-' и вернуть его индекс

Стратегия ветвления по столбцам

Ветвление по самой короткой строке (с минимальным числом активных переменных) может быстрее привести к упрощению КНФ.

1. Собрать все непустые строки из CNF и посчитать их вес:
 Вес = количество незамаскированных переменных, которые не равны '-'

2. Если нет подходящих строк → вернуть -1

3. Найди строку с минимальным ненулевым весом

4. В этой строке найди первый незамаскированный столбец, значение которого не '-'
      Если такой столбец найден → вернуть его индекс

5. Если метод не помог, то вызвать алгоритм по столбцам

Пример работы с выбором стратегии

```
std::shared_ptr<BranchingStrategy> strategy_ptr_ = std::make_shared<ColumnBranchingStrategy>();

if (argc > 1) {
    file_path_ = argv[1];

    if (argc > 2) {
        std::string strategy_name_ = argv[2];
        if (strategy_name_ == "row") {
            strategy_ptr_ = std::make_shared<RowBranchingStrategy>();
            std::cout << "Using row branching strategy\n";
        }
    }
} else {
    file_path_ = "./SatExamples/Sat_ex11_3.pla";
}
```

3. Исследовать пользовательский проект на уязвимости с помощью любого доступного статического анализатора, например pvs-studio , а также с помощью Valgrind динамического анализа.

Статический анализатор Semgrep

Используются стандартные правила.

```
katet_3@DESKTOP-RBO1289:/mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom$ semgrep scan --lang c++ .

Scanning 24 files (only git-tracked) with:

✓ Semgrep OSS
  ✓ Basic security coverage for first-party code vulnerabilities.
```

```
Scan Summary

Some files were skipped or only partially analyzed.
  Scan was limited to files tracked by git.
  Scan skipped: 4 files matching .semgrepignore patterns
  For a full list of skipped files, run semgrep with the --verbose flag.

(need more rules? `semgrep login` for additional free Semgrep Registry rules)

Ran 56 rules on 20 files: 0 findings.
```

```
katet_3@DESKTOP-RBO1289:/mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom$ semgrep scan --lang c .

Scanning 24 files (only git-tracked) with:

✓ Semgrep OSS
  ✓ Basic security coverage for first-party code vulnerabilities.
```

```
Scan Summary

Some files were skipped or only partially analyzed.
  Scan was limited to files tracked by git.
  Scan skipped: 4 files matching .semgrepignore patterns
  For a full list of skipped files, run semgrep with the --verbose flag.

(need more rules? `semgrep login` for additional free Semgrep Registry rules)

Ran 56 rules on 20 files: 0 findings.
```

Статический анализатор Clang

```
katet_3@DESKTOP-RBO1289:/mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/build$ scan-build make
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static analysis
Consolidate compiler generated dependencies of target SAT_DPLL
[ 70%] Built target SAT_DPLL
[100%] Built target allocator_test
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2025-05-25-200523-937-1' because it contains no reports.
scan-build: No bugs found.
```

```
katet_3@DESKTOP-RBO1289:/mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/build$ scan-build cmake ..
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static analysis
-- Компиляция с использованием пользовательского аллокатора
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/build
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2025-05-25-200611-977-1' because it contains no reports.
scan-build: No bugs found.
```

```
katet_3@DESKTOP-RBO1289:/mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/build$ scan-build clang -c ../main.cpp
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static analysis
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2025-05-25-200829-1022-1' because it contains no reports.
scan-build: No bugs found.
```

```
katet_3@DESKTOP-RBO1289:/mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/build$ scan-build clang -c ../allocator_test.cc
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static analysis
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2025-05-25-200837-1034-1' because it contains no reports.
scan-build: No bugs found.
```

Динамический анализатор Valgrind

```
katet_3@DESKTOP-RBO1289:/mnt/d/code/4course_2sem/dev_sec_software/tsu_devSec
==1120== Memcheck, a memory error detector
==1120== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1120== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==1120== Command: ./SAT_DPLL_custom ../SatExamples/Sat_ex11_3.pla
```

После выполнения смотрим отчет

```
==1120==
Solution found:
001000-0-00-0100--0--001--0--011
Execution time: 142960 µs
==1120==
==1120== HEAP SUMMARY:
==1120==     in use at exit: 52,001 bytes in 1,378 blocks
==1120==   total heap usage: 6,930 allocs, 5,552 frees, 307,874 bytes allocated
==1120==
```

```
==1120== LEAK SUMMARY:
==1120==    definitely lost: 51,841 bytes in 1,338 blocks
==1120==    indirectly lost: 160 bytes in 40 blocks
==1120==      possibly lost: 0 bytes in 0 blocks
==1120==    still reachable: 0 bytes in 0 blocks
==1120==         suppressed: 0 bytes in 0 blocks
==1120==
==1120== For lists of detected and suppressed errors, rerun with: -s
==1120== ERROR SUMMARY: 964 errors from 26 contexts (suppressed: 0 from 0)
```

Разберем одну из заявленных ошибок

```
==1436==    by 0x114B5F: main (in /mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/build/SAT_DPLL_custom)
==1436==  Address 0x4e728d0 is 0 bytes inside a block of size 4 alloc'd
==1436==    at 0x484A2F3: operator new[](unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1436==    by 0x10C20C: BBV::operator=(char const*) (in /mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/b
==1436==    by 0x110FB6: BoolInter                                    lInterval&) (in /mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2
L_custom)
==1436==    by 0x114B5F: main (in /mnt/d/code/4course_2sem/dev_sec_software/tsu_devSecSoftLab2/SAT_DPLL_custom/build/SAT_DPLL_custom)
==1436==
```

Open file in editor (ctrl + click)

Проблема, обнаруженная Valgrind, связана с некорректным управлением памятью в операторе присваивания BBV::operator=(const char*).

```
BBV BBV::operator=(const char * str)
{
    if (str != NULL)//проверяем существует ли строка
    {
        len = strlen(str);//вычисляем размеры для вектора и удаляем старый вектор
        size = (len - 1) / 8 + 1;
        if (vec != NULL)
            delete vec;
        vec = new byte[size];//захватываем память для нового вектора и переписываем в его биты соотв. знач. строки
        if (vec != NULL)
        {
            int i = 0, j = 8, k = 0;
            byte mask = 1;
            vec[0] ^= vec[0];
```

```
216                 }
217             }
218             else
219                 throw 0;
220         }
221         else
222             throw 1;
223         return *this;
224     }
```

Проблемы этого кода:
Используется new byte[size], но освобождается через delete вместо delete[]
Нет проверки на успешное выделение памяти после new
Если new выбрасывает исключение, старые данные уже удалены

Код после изменения будет иметь вид:

```
182
183  BBV BBV::operator=(const char* str) {
184      if (!str) throw std::invalid_argument("Null pointer passed");
185
186      const size_t new_len = strlen(str);
187      const size_t new_size = (new_len + 7) / 8;
188
189      byte* new_vec = new byte[new_size]();  // Инициализация нулям
190
191      try {
192          // Конвертация строки в битовый вектор
193          for (size_t i = 0; i < new_len; ++i) {
194              if (str[i] != '0') {
195                  const size_t byte_idx = i / 8;
196                  const size_t bit_idx = i % 8;
197                  new_vec[byte_idx] |= (1 << bit_idx);
198              }
199          }
200
201          // Освобождаем старые данные
202          delete[] vec;
203
```

```
200
201          // Освобождаем старые данные
202          delete[] vec;
203
204          // Обновляем состояние
205          vec = new_vec;
206          len = new_len;
207          size = new_size;
208
209          return *this;
210      } catch (...) {
211          delete[] new_vec;  // Освобождаем в случае ошибки
212          throw;
213      }
214  }
```

Тестируем

```
==1549== LEAK SUMMARY:
==1549==    definitely lost: 51,841 bytes in 1,338 blocks
==1549==    indirectly lost: 160 bytes in 40 blocks
==1549==      possibly lost: 0 bytes in 0 blocks
==1549==    still reachable: 0 bytes in 0 blocks
==1549==         suppressed: 0 bytes in 0 blocks
==1549==
==1549== For lists of detected and suppressed errors, rerun with: -s
==1549== ERROR SUMMARY: 18 errors from 18 contexts (suppressed: 0 from 0)
```

Результат на глаза, на сколько меньше стало ошибок.