1. Запрос capacity 0 длины

```
int main() {
    istack_t stack;
    istack_request_double(&stack, 0);

    double elem = 0;

    istack_err_t e = istack_push_double(stack, 131);
    e = istack_pop_double(stack, &elem);

    return 0;
}
```

В StackResize происходило расширение с помощью **capacity * 2,** что приводило к тому, что стек не расширялся.

Программа выводила дамп, что стек испорчен, но это неверное поведение.

2. Ошибка в применении static

Защита во многом опиралась на то, что я просто не смогу узнать где в памяти находится стек и не получится его сломать. Автор давал что-то на подобии дескриптора файлов только для стеков. Для этого, Артем **применил static чтобы скрыть переменные, но вся реализация была в header файлах,** которые макроподстановкой переносились в исходный файл с *main()* и все *static* теряли смысл.

Поэтому, ничего не мешало мне найти где и как хранятся стеки.

3. Зависимость checksum от size

```
int main() {
    istack_t stack;
    istack_request_double(&stack, 10);

    istack_push_double(stack, 11);
    istack_push_double(stack, 1);
    istack_push_double(stack, 2);
    istack_push_double(stack, 3);
    istack_push_double(stack, 4);

    istack_container_t_double* stack_container;
    istack_dereference_double(&ISTACK_REFERENCE_LIST_double, stack, &stack_container);

    for (int i = 0; i<4; i++){
        stack_container->impl_list[i]->size = 10000;
        stack_container->impl_list[i]->capacity = 10001;
    }

    istack_push_double(stack, 10 );

    return 0;
}
```

Перед тем как начать расчет *checkSum* по буферу, Артем проверял лишь то, что *capacity > size*.
Поменяв ***size* на 10000, а capacity на 10001, я эту проверку обошел и заставил checkSum считать хэш за пределами выделенной памяти.**
Дамп в данном случае не срабатывал.

4. Я у мамы криптограф (заложено)

```
int main() {
    istack_t stack;
    istack_request_int(&stack, 10);

    istack_push_int(stack, 11);
    istack_push_int(stack, 1);
    istack_push_int(stack, 2);
    istack_push_int(stack, 3);
    istack_push_int(stack, 4);

    istack_container_t_int* stack_container;
    istack_dereference_int(&ISTACK_REFERENCE_LIST_int, stack, &stack_container);


    for(int i = 0; i < ISTACK_IMPL_AMOUNT; i++) {
        stack_container->impl_list[i]->size = 0;
    }


    istack_err_t e = istack_dump_int(stack);          2 ⚠ Unused va
    istack_dump_int(stack);                           Thread 1: s

    return 0;
}
```

```
ArtemStack 〉 Thread 1 〉 0 main

▼ stack_container = (istack_container_t_int *) 0x1006556a0
    impl_pointer_hash = (uint64_t) 10479375047963271326
  ▼ impl_list = (istack_impl_t_int **) 0x10045d000
    ▼ *impl_list = (istack_impl_t_int *) 0x10065e190
        impl_hash = (uint64_t) 11
        size = (size_t) 0
        capacity = (size_t) 10
        min_capacity = (size_t) 10
      ▶ buffer = (int *) 0x10065e1c0
    ▶ [1] = (istack_impl_t_int *) 0x10065e1f0
    ▶ [2] = (istack_impl_t_int *) 0x10065e250
    ▶ [3] = (istack_impl_t_int *) 0x10065e2b0
```

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=
 istack_container_t dump
 - expected value type:    int (4-byte value)
 - address:                0x1006556a0 (valid)
 - validator verdict:      0x0 (ISTACK_OK)
 - entry pointer hash:     0x916e381581f1609e
 - stack amount:           10000
 - stack capacity:         10
 - stack length:           10
 - stack initial capacity: 10
 - corrupted stacks:       0
 - stack items:
=-=-=-=-=-=-=-=-=-=-=-=-=-=
(lldb)
```

Артем решил использовать сомнительную хэш-сумму собственной разработки, у которой основной операцией является **побитовое ИЛИ.** В какой-то момент все **биты хэша становились 1** и все. Хэш больше не работал, можно было делать что хочется.

Я написал программу, которая Брут-форсом ищет collisions для хэш-функции и их оказалось огромное количество уже на первом байте

```
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
FOUND!
```