Взлом стека Александра Дремова

1. Запрос capacity очень большой длины

```c
#define  StackElementDump(FILE, VALUE) {fprintf(FILE, "%g", VALUE);}
#define  StackElementType entry
#include "StackRigid.h"
#include <string.h>

struct entry {
    char massive_array[33];
};

int main(int argc, const char * argv[]) {
    StackRigid_entry* newStack = NewStackRigid_entry(-1, stdout);

    for(int i = 0; i < 1000000; i++) {
        StackPush(&newStack, (entry){0});
    }

    return 0;
}
```

Внутри функции `NewStackRigid_entry` в процессе подсчета количества памяти, необходимого для стека, происходит переполнение. Если архитектура системы подразумевает `size_t` как беззнаковое 64-битное число, а размер элемента массива равен 33 байтам, значение выражения `sizeof(StackRigid_entry) + (capacity - 1) * sizeof(entry)` оказывается равным 6. В результате, последующие вызовы `StackPush` приводят к тому, что стек обращается к неаллоцированной памяти, думая, что его capacity равно 2^{64} - 1.

2. Обход хеширования

```c
#define  StackElementDump(FILE, VALUE) {fprintf(FILE, "%g", VALUE);}
#define  StackElementType int
#include "StackRigid.h"
#include <string.h>

int main(int argc, const char * argv[]) {
    StackRigid_int* newStack = NewStackRigid_int(1, stdout);

    newStack -> size = 130561;
    newStack -> capacity = 130562;

    StackDump(newStack);

    return 0;
}
```

Перед любой операцией со стеком (включая `StackDump`), вызывается валидация стека. Процесс валидации подсчитывает две хеш-суммы: первая включает только служебные поля, такие как размер и количество элементов, а вторая включает в подсчет все элементы стека. Проверка второго хеша запускается только тогда, когда проверка первого хеша проходит успешно. Таким образом валидатор "убеждается", что служебные поля не были повреждены, стек действительно имеет заданный в этом поле размер и значения стека можно безопасно читать, чтобы включать их в второй хеш.

Однако, в качестве метода подсчета хеша используется Adler-32, который выдает достаточно много пересечений на небольших объемах данных. Существует достаточно много комбинаций значений служебных полей стека, при которых этот метод хеширования выдаст одинаковый результат. Один из них указан на скриншоте.

В результате проверка целостности служебных полей стека пройдет успешно и запустится проверка хеша значений, записанных в стек. Поскольку размер стека искажен, функция валидатора начнет считывать неаллоцированную память.

3. Некорректные указатели

```c
#define  StackElementDump(FILE, VALUE) {fprintf(FILE, "%g", VALUE);}
#define  StackElementType int
#include "StackRigid.h"
#include <string.h>

int main(int argc, const char * argv[]) {

    StackPush((StackRigid_int**)1, 0);

    return 0;
}
```

Этот взлом нельзя назвать честным, поскольку функции для проверки правильности указателя нет в стандарте языка C, а её реализации являются некроссплатформенными, однако в коде выше функция-валидатор могла бы сообщить пользвателю, что указатель 1 находится в первой странице памяти, и более того, не выровнен по 8 байтам.