***УДК 004.021; 519.651***

**МЕХАНИЗМЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОГО ДОСТУПА К ПАМЯТИ ПРИ ТРАНСПИЛЯЦИИ С ЯЗЫКА ПРОГРАММИРОВАНИЯ С ПРОИЗВОЛЬНЫМ ДОСТУПОМ**

**Ф.Р. Петренко1**

Кубанский государственный университет

ул. Ставропольская 149, 350040 Краснодар, Россия

**Ключевые слова:** языки программирования, транспайлер, архитектура оперативной памяти компьютера, уязвимости доступа к оперативной памяти компьютера.

**Аннотация**

В данной работе описаны механизмы реализации безопасного доступа к памяти в языках программирования, транспилируемых в языки с произвольным доступом к оперативной памяти компьютера. Методы защиты от уязвимостей произвольного доступа к памяти будут представлены на примере реализации языка Algolite, содержащего урезанный синтаксис языка C++ с набором средств для проведения анализа сложности произвольного алгоритма. Преобразованный синтаксис позволяет обойти различные ошибки памяти.

Проблемы безопасности памяти являются одними из наиболее распространенных в разработке ПО. Некоторое количество современных языков программирования с возможностью обращения к низкоуровневым механизмам доступа к памяти (например, C/C++) позволяют разработчику допускать существования данной типа уязвимости.

При реализации программного комплекса для предоставления сервиса анализа сложности произвольного алгоритма (CAaaS, Complexity Analysis as a Service) был разработан язык Algolite, синтаксис которого основывается на синтаксисе языка C++ и содержит дополнительный функционал, упрощающий процесс анализа классов сложности. В процессе создания траспилятора, переводящего код на Algolite в код на C++, возникли проблемы с обеспечением безопасного выполнения программы на хост-машине.

Язык C++ позволяет беспрепятственно использовать низкоуровневый код для ссылок и указателей, что может привести к обращению к нулевым указателям, созданию висячих указателей или потери указателей. Данные ошибки можно обойти, исключив из синтаксиса языка низкоуровневый доступ к ссылкам и указателям.

Процесс можно рассматривать как некоторый программный объект, исполняемый на процессоре. Его физическое представление отображено на рисунке 1.

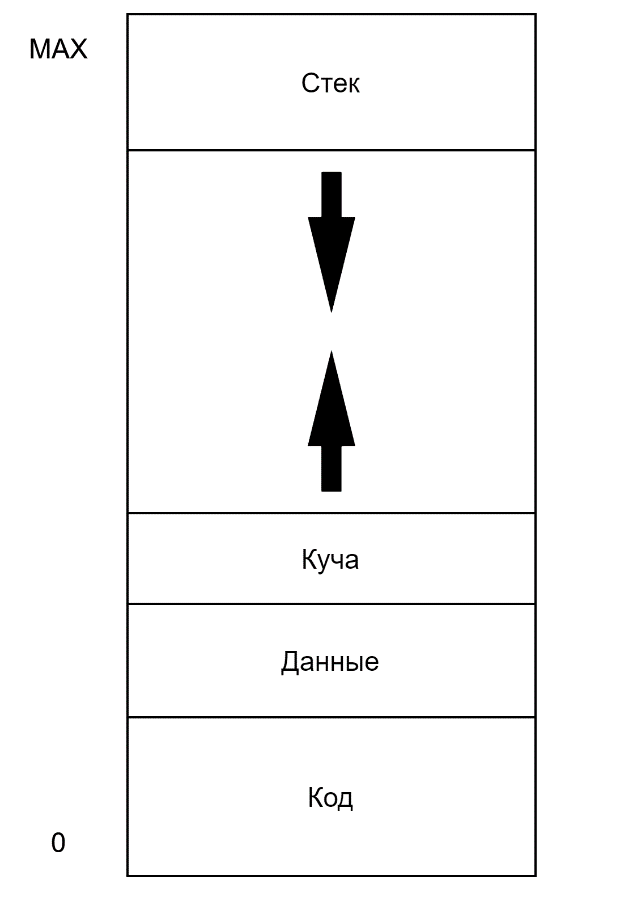


Рисунок 1 – Физическое представление процесса.

Стек – область памяти, где хранится информация о вызываемых функциях, их аргументах и каждой локальной переменной в функции (расположен на максимальном адресе, доступном процессу в оперативной памяти). Куча — это область свободной памяти процесса. Код – область памяти, содержащая инструкции ЦП. Данные – область данных, которые используются машинными командами. Стек и куча являются изменяемыми областями, растущими в противоположных направлениях. Каждый процесс обладает своим адресным пространством.

В языке C++ работа с массивом является еще одним примером произвольного доступа к памяти, так как массивы передаются в функции в качестве указателей и позволяют получить доступ к элементу внутри без дополнительных проверок на границы массива на этапе исполнения. Ввиду таких возможностей программист способен допустить ошибки, связанные с переполнением буфера и чтением за границами буфера, что может привести к нарушению безопасности работы программы или нестабильному поведению процесса исполнения.

Например, пользователь сервиса может использовать хостинг-машину на сервере для несанкционированных целей. Злоумышленник может получить доступ к данным, расположенным выше в стеке (int f(int a[]) { cout << a[-10]; }), которые могут содержать входные переменные, с которыми была запущена программа. Так же он может записать сторонний машинный код в произвольный массив, получить доступ к адресу возврата из функции за пределами массива, после чего заменить его адресом массива. После завершения команды return процесс начнет исполнение стороннего машинного кода, записанного пользователем.

Для уменьшения подобного вида ошибок необходимо отказаться от возможности реализации синтаксиса следующего вида: <аргумент функции> ::= <тип данных> <имя> ['[]'] [{'[' <выражение> ']'}] (представлено в виде РБНФ). Также следует ввести дополнительные ограничения на передачу массивов в функции (например, устранить возможность передачи массива конкретного размера, а сам размер неявно передавать в функцию как дополнительный параметр в транспилированной программе), после чего организовать проверки на выход за границы при любой попытке доступа к элементам массива.

Таким образом, набор методов для обхода уязвимостей, связанных с произвольным доступом к памяти в языке, транспилируемом в язык C++, можно представить следующим образом:

1) исключить из языка низкоуровневый доступ к ссылкам и указателям;

2) добавить ограничения на использования массивов:

2.1) ограничить возможность передачи в функции массивов без размеров;

2.2) ввести механизм проверки границ каждого из массивов при попытке получения доступа к их элементам;

3) внедрить средства статистического анализа.

В результаты введенных механизмов ошибки, связанные с произвольным доступом к памяти в языке C++, могут быть устранены. Результаты сравнения приведены в таблице 1.

Таблица 1 – Сравнение наличия потенциальных ошибок работы

|  |  |  |
| --- | --- | --- |
| Потенциальные ошибки памяти | C++ | Algolite |
| Нулевые указатели | Присутствует | Ограничено синтаксисом |
| Присутствует | Присутствует | Ограничено синтаксисом |
| Потеря указателей | Присутствует | Ограничено синтаксисом |
| Переполнение буфера | Присутствует | Ограничено синтаксисом и проверками на выход за границы |
| Чтение за границами буфера | Присутствует | Ограничено синтаксисом и проверками на выход за границы |

Автор благодарит д.ф.-м.н., профессора А.И. Микова за постановку задачи исследования.

***Библиографический список***

1. Хантер, Р. Проектирование и конструирование компиляторов / Р. Хантер; перевод с английского В. М. Савинкова. – М. : Финансы и статистика, 1984. – 232 с.
2. Мартемьянов, Ю.Ф. Операционные системы. Концепции построения и обеспечения безопасности : Учебное пособие для вузов / Ю.Ф. Мартемьянов, Ал.В. Яковлев, Ан.В. Яковлев. – М. : Горячая Линия - Телеком, 2020. – 332 с. – ISBN 978-5-9912-0677-8.
3. Касперски, К. Техника оптимизации программ. Эффективное использование памяти / К. Касперски. – СПб. : BHV, 2003. – 464 с. – ISBN 5-94157-232-8.