МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра технологий обработки и защиты информации

Реферат

Динамический анализ кода

Направление 10.03.01 Информационная безопасность
Профиль Безопасность компьютерных систем

Обучающийся _____ А. А. Володина, 4 курс

Руководитель _____ М. А. Дрюченко, к. т. н., доцент

Воронеж 2023

Содержание

Введение

По мере развития проекта стоимость устранения дефектов программного обеспечения может экспоненциально возрастать. Чем раньше будет обнаружена программная ошибка, тем быстрее и дешевле ее можно будет исправить. Именно поэтому инструменты, которые обеспечивают поиск ошибок на ранних этапах жизненного цикла программного обеспечения, представляют довольно высокую ценность. В средах высокопроизводительных вычислений запускаются сложные приложения, созданные с использованием разных языков программирования, платформ и технологий, с тысячами потоков и процессов одновременно. Простого изучения кода на наличие различных проблем недостаточно, чтобы выявить и изолировать различные ошибки и проблемы с производительностью, которые могут появиться во время выполнения.

Инструменты динамического анализа кода могут помочь разработчикам благодаря простой отладке запущенных потоков и процессов. Тестирование с помощью динамического анализа покажет, хорошо ли работает приложение, и наоборот, выявит ошибки, указывающие на то, что приложение не работает должным образом.

Целью данной работы является изучение понятия динамического анализа кода, его видов и принципов работы, а также выделение преимуществ и недостатков данного метода.

1. Определение динамического анализа кода

Динамический анализ кода (Dynamic program analysis, DPA) – это анализ программного обеспечения, производящийся при помощи выполнения программ на реальном или виртуальном процессоре. Проще говоря, это способ анализа программы непосредственно при её выполнении.

2. Процесс динамического анализа кода

Процесс динамического анализа можно разделить на несколько этапов:

- Подготовка исходных данных.
- Проведение тестового запуска программы.
- Сбор необходимых параметров.
- Анализ полученных данных.

Динамический анализ выполняется с помощью набора данных, которые подаются на вход исследуемой программе. Поэтому эффективность анализа напрямую зависит от качества и количества входных данных для тестирования. Именно от них зависит полнота покрытия кода, которая будет получена по результатам тестирования. Для генерации входных тестовых данных нередко используются различные сторонние утилиты. Некоторые из них используют следующий подход: они помечают входные данные и отслеживают их перемещение по мере выполнения программы. При следующем тестовом запуске утилита сформирует новый набор входных параметров, и так до тех пор, пока не получит нужный результат.

В то время как при статическом анализе исходный код интерпретируется как текст и все выводы основаны на данных синтаксического анализатора, без выполнения команд, динамический анализ предоставляет данные о коде в другом режиме. Код анализируется во время выполнения, демонстрируя покрытие кода тестами, достаточность и качество модульных тестов, утечки памяти и другие потенциальные уязвимости. С другой стороны, при тестовом запуске исполнение программы может выполняться как на реальном, так и на виртуальном процессоре, но в обязательном порядке должен быть получен исполняемый файл, то есть нельзя таким способом проанализировать код, содержащий ошибки компиляции или сборки.

Используя динамическое тестирование, можно получить следующие метрики и предупреждения:

- Используемые ресурсы: время выполнения программы в целом или ее отдельных модулей, количество внешних запросов

(например, к базе данных), количество используемой оперативной памяти и других ресурсов.
- Степень покрытия кода тестами и другие метрики программы.
- Программные ошибки: деление на ноль, разыменование нулевого указателя, утечки памяти, "состояние гонки".
- Детектировать некоторые уязвимости.

Процесс анализа состоит из нескольких шагов:

- Наблюдение.

Прежде всего, происходит захват ошибок среды выполнения, обнаружение мест утечки памяти и отображение всех результатов в среде IDE.

- Корректировка.

Далее разработчик имеет возможность провести трассировку каждой ошибки назад до нарушающей работу строки исходного текста. При хорошей интеграции в среду IDE каждая ошибка будет отображаться на экране. Разработчику нужно просто щелкнуть мышью на строке ошибки, и откроется фрагмент исходного кода со строкой, нарушающей работу. Во многих случаях разработчик может быстро устранить проблему, используя доступную трассировку стека и дополнительные инструменты работы с исходным кодом в среде IDE (средства просмотра вызовов функций, средства трассировки вызовов и т.д.).

- Профилирование.

Устранив обнаруженные ошибки и утечки памяти, разработчик может проанализировать использование ресурсов во времени, включая пиковые ситуации, среднюю загрузку и избыточное расходование ресурсов. В идеальном случае инструмент анализа выдаст визуальное представление долговременного использования ресурсов, позволяя немедленно идентифицировать всплески в распределении памяти и иные аномалии.

- Оптимизация.

Используя информацию этапа профилирования, разработчик может провести "тонкий" анализ использования ресурсов программой. Среди прочего, подобная оптимизация может минимизировать случаи пикового использования ресурсов и их избыточное расходование, включая работу с ОЗУ и использование времени ЦПУ.

3. Виды динамического анализа кода

3.1 По данным об исходном коде

Динамический анализ кода может производиться по разным принципам в зависимости от наличия и использования данных об исходном коде проверяемо программы:

- Принцип белого ящика.

Имеются данные о программном коде.

- Принцип черного ящика.

Не имеется данных о коде программы.

- Принцип серого ящика.

Известна структура программы, но в самом тестировании эти данные не используются.

3.2 По способу взаимодействия с программой

Программы для динамического анализа различаются по способу взаимодействия с проверяемой программой:

- Размещение вставок в исходный код на этапе предпроцессорной обработки.

В исходный текст приложения до начала компиляции вставляется специальный фрагмент кода для обнаружения ошибок. При таком подходе не требуется детального знания среды исполнения, в результате чего такой метод пользуется популярностью среди инструментов тестирования и анализа встраиваемых систем.

- Размещение вставок в объектный код.

Для такого инструмента динамического анализа необходимо обладать достаточными знаниями относительно среды исполнения, чтобы иметь возможность вставлять код непосредственно в исполняемые файлы и библиотеки. При этом подходе не нужно иметь доступа к исходному тексту программы или делать перекомпоновку приложения.

- Вставка кода во время компиляции.

Разработчик использует специальные ключи (опции) компилятора для внедрения в исходный код. Используется способность компилятора обнаруживать ошибки.

- Специализированные библиотеки этапа исполнения.

Для обнаружения ошибок в передаваемых параметрах разработчик использует отладочные версии системных библиотек. Дурной славой пользуются функции типа strcpy() из-за возможности появления нулевых или ошибочных указателей во время исполнения. При использовании отладочных версий библиотек такие "плохие" параметры обнаруживаются. Данная технология не требует перекомпоновки приложения и в меньшей степени влияет на производительность, чем полноценное использование вставок в исходный/объектный код.

4. Преимущества динамического анализа кода

К основным преимуществам динамического анализа кода относят:

- Возможность проводить анализ программы без необходимости доступа к её исходному коду. Здесь стоит сделать оговорку, так как программы для динамического анализа различают по способу взаимодействия с проверяемой программой и для некоторых случаев доступ к коду проверяемой программы все же будет необходим.
- Возможность обнаружения сложных ошибок, связанных с работой с памятью: выход за границу массива, обнаружение утечек памяти.
- Возможность проводить анализ многопоточного кода непосредственно в момент выполнения программы, тем самым обнаруживать потенциальные проблемы, связанные с доступом к разделяемым ресурсами, возможные deadlock ситуации.
- В большинстве реализаций появление ложных срабатываний исключено, так как обнаружение ошибки происходит в момент ее возникновения в программе; таким образом, обнаруженная ошибка является не предсказанием, сделанным на основе анализа модели программы, а констатацией факта ее возникновения.
- Захватываются ошибки в контексте работающей системы, как в реальных условиях, так и в режиме имитационного моделирования.
- Динамический анализ не только позволяет обнаружить ошибки, но и помогает обратить внимание разработчика на подробности расходования памяти, циклов ЦПУ, дискового пространства и других ресурсов.

5. Недостатки динамического анализа кода

Недостатки, которые присущи динамическому анализу кода:

- Нельзя гарантировать полного покрытия кода, т.е., скорее всего, процент кода программы, который был проанализирован в процессе

динамического тестирования, не будет равен ста процентам. На практике, даже хорошие тесты покрывают не более 80% программного кода

- Динамический анализ обнаруживает ошибки только в области, определяемой конкретными входными данными; ошибки, находящиеся в других частях программы, не будут обнаружены;
- Почти не выявляются ошибки логического типа. Например, с точки зрения динамического анализатора, всегда истинное условие не является ошибкой, так как такая некорректная проверка просто исчезает ещё на этапе компиляции программы.
- Тяжело локализовать место с ошибкой в исходном коде.
- Более высокая сложность использования по сравнению со статическим анализом, так как для достижения большей эффективности динамического анализа тестируемой программе требуется подача достаточного количества входных данных, чтобы получить более полное покрытие кода.
- Процесс динамического анализа требует огромного количества оперативной памяти и дискового пространства для сбора результатов покрытия кода тестами.
- Также при динамическом тестировании требуется позаботиться о минимальном воздействии инструментирования на исполнение тестируемой программы (временные характеристики, потребляемые ресурсы или программные ошибки).
- При тестировании на реальном процессоре исполнение некорректного кода может привести к непредсказуемым последствиям.
- В конкретный момент времени может быть проверен только один вариант выполнения программы, что требует большого количества тестовых запусков для большей полноты тестирования.

Вывод

В ходе данной работы был изучен динамический анализ кода. Так же были рассмотрены основные этапы данного анализа, различные виды его работы, выделены преимущества и недостатки.

Из всего этого можно сделать вывод, что нет одной технологии, которая бы позволяла выявлять ошибки всех типов, один вид анализа не способен полностью заменить другой. Будь то динамический или статический анализ, все это просто разные методологии, которые имеют свои преимущества и недостатки, но дополняют друг друга.

Динамическое тестирование наиболее важно в тех областях, где главным критерием является надежность программы, время отклика или потребляемые ресурсы. Это может быть, например, система реального времени, управляющая ответственным участком производства, или сервер базы данных. В таких областях любая допущенная ошибка может оказаться критической.

В конечном счёте смысл использования динамического анализа кода, как и всех подобных техник сводится к поднятию качества программного продукта и сокращению затрат при его разработке.

Список использованных источников

1. Динамический анализ кода [Электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%BA%D0%BE%D0%B4%D0%B0.
2. What's the Use of Dynamic Analysis When You Have Static Analysis? [Электронный ресурс]. – URL: https://pvs-studio.com/en/blog/posts/0643/.
3. Dynamic code analysis [Электронный ресурс]. – URL: https://pvs-studio.com/en/blog/terms/0070/.
4. Static and Dynamic Code Analysis [Электронный ресурс]. – URL: https://pvs-studio.com/en/blog/posts/0248/.
5. Myths about static analysis. The third myth - dynamic analysis is better than static analysis [Электронный ресурс]. – URL: https://pvs-studio.com/en/blog/posts/0117/.
6. Зачем нужен динамический анализ кода, на примере проекта PVS-Studio [Электронный ресурс]. – URL: https://habr.com/ru/company/pvs-studio/blog/580196/.
7. Анализ кода: проблемы, решения, перспективы [Электронный ресурс]. – URL: https://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%BA%D0%BE%D0%B4%D0%B0:_%D0%BF%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D1%8B,_%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D1%8F,_%D0%BF%D0%B5%D1%80%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D1%8B.
8. Статический и динамический анализ кода: 5 полезных советов [Электронный ресурс]. – URL: https://auriga.ru/blog/2019/static-dynamic-analysis-medical-device-software/.
9. Что такое динамический анализ кода? - определение из техопедии [Электронный ресурс]. – URL: https://ru.theastrologypage.com/dynamic-code-analysis.
10. What Is Dynamic Analysis? [Электронный ресурс]. – URL: https://totalview.io/blog/what-dynamic-analysis.