Санкт-Петербургский государственный университет

Кафедра системного программирования

Устранение ложных срабатываний статических анализаторов кода символьным исполнением для платформы .NET

Седлярский Михаил Андреевич, 21.M07-мм группа

Научный руководитель: к.ф.-м.н. Д.А. Мордвинов, доцент кафедры системного программирования

Санкт-Петербург
2023

Введение

- Современная разработка нуждается в автоматическом анализе кода
- Методы статического анализа предрасположены к ложноположительным срабатываниям
- Символьное исполнение лишено такого недостатка, но плохо масштабируется

Существующие подходы

- Легковесные анализаторы, микрограмматики (Google Sanitizers)
- Межпроцедурный анализ, сепарационная логика (Infer)
- Большие данные, обработка крупных семантических графов (Graspan)

Существующие подходы

В данной работе предлагается объединить методы статического анализа и символьного исполнения. Можно подкреплять и валидировать результаты статического анализатора таргетированным исполнением в символьной виртуальной машине (прим. TASMAN)

Постановка задачи

Целью разработка и реализация приложения для платформы .NET, подтверждающее срабатывания статических анализаторов кода символьным исполнением

Задачи:

- провести сравнительный анализ статических анализаторов кода для платформы .NET;
- запустить статические анализаторы на настоящих проектах. На основании полученных результатов выбрать анализатор кода, с которым будут проводиться дальнейшие работы;
- разработать и реализовать алгоритм преобразования результатов статического анализатора кода в цели для движка символьного исполнения.

Постановка задачи

Задачи (продолжение):

- разработать и реализовать способ передачи целей в движок символьного исполнения;
- разработать и реализовать алгоритм ранжирования результатов статического анализа на основе результатов символьного исполнения
- провести эксперименты и оценить качество полученного инструмента.

Сравнение анализаторов

Таблица: Сводная информация о статических анализаторах .NET

Название	Поддерживаемые платформы			Поддержка SARIF	Open source
	win	linux	macos		
Roslyn analysers	+	+	+	+	+
ReSharper	+	+	+	+	- (есть бесплатные лицензии)
PVS-Studio	+	+	+	+ (через plog-converter)	- (есть бесплатные лицензии)
InferSharp	+ (через wsl)	+	± (работает через docker, но теоретически можно собрать нативно)	+	+
Sonar Scanner for .NET + sonar-dotnet	+	+	+	+	+
CHECKMARX SAST	web решение				- (можно получить демо версию)

Сравнение анализаторов (Infer#)

Проект	NULLPTR_ DEREFERENCE	PULSE_ RESOURCE_LEAK	THREAD_ SAFETY_VIOLATION	STACK_VARIABLE _ADDRESS_ESCAPE
efcore	107	2	8	0
litedb	1	7	6	0
moq4	0	1	0	0
NLog	30	75	44	0
nunit	26	192	7	0
xunit	0	0	0	0
btcpayserver	4	6	4	0
AutoMapper	0	0	0	0
spbu-homeworks-1	0	1	4	0
parallel-programming-1-sync	0	0	0	0
parallel-programming-1-thread-pool	0	0	0	0
parallel-programming-3-thread-pool	0	0	0	0
BenchmarkDotNet	2	11	0	0
ILSpy (ILSpy.XPlat.slnf)	48	4	0	2
OpenRA	34	49	1	0
Newtonsoft.Json	14	271	5	0
RestSharp	0	53	0	0

Таблица: Результаты запуска Infer#

Сравнение анализаторов (sonar)

Проект	roslyn	sonar-dotnet	прочее
efcore	264	2893	487
litedb	0	470	9
moq4	214	441	5
NLog	17	266	12
nunit	218	856	4
xunit	292	1092	0
btcpayserver	267	1062	1
AutoMapper	88	167	0
spbu-homeworks-1	33	105	0
parallel-programming-1-sync	0	2	0
parallel-programming-1-thread-pool	11	3	6
parallel-programming-3-thread-pool	8	10	0
BenchmarkDotNet	18	19	0
ILSpy (ILSpy.XPlat.slnf)	873	1829	0
OpenRA	0	2990	0
Newtonsoft.Json	1478	740	0
RestSharp	28	47	9

Таблица: Результаты запуска sonar-scanner-msbuild (roslyn + sonar-dotnet)

Сравнение анализаторов (PVS-Studio)

Проект	V3022 (Expression <exp> is always <value>)	V3080 (Possible null dereference)	V3146 (Possible null dereference. A method can return default null value)	V3106 (Possibly index is out of bound)	Всего
efcore	анализ решения съедает все ресурсы на сервере				
litedb	15	19	1	0	110
moq4	4	2	0	0	61
NLog	34	9	0	1	163
nunit	28	8	0	0	203
xunit	19	11	0	0	127
btcpayserver	37	99	13	0	374
AutoMapper	2	1	0	0	38
spbu-homeworks-1	0	0	0	2	5
parallel-programming-1-sync	0	0	0	0	0
parallel-programming-1-thread-pool	0	0	0	0	0
parallel-programming-3-thread-pool	0	0	0	0	2
BenchmarkDotNet	14	0	0	0	82
ILSpy (ILSpy.XPlat.slnf)	98	128	0	17	792
OpenRA	13	39	0	2	342
Newtonsoft.Json	30	10	0	34	265
RestSharp	0	1	0	0	16

Таблица: Результаты запуска PVS-Studio

Сравнение анализаторов

Был выбран PVS-studio т.к. выдаёт интересные с точки символьного исполнения результаты, обладает относительным разнообразием правил

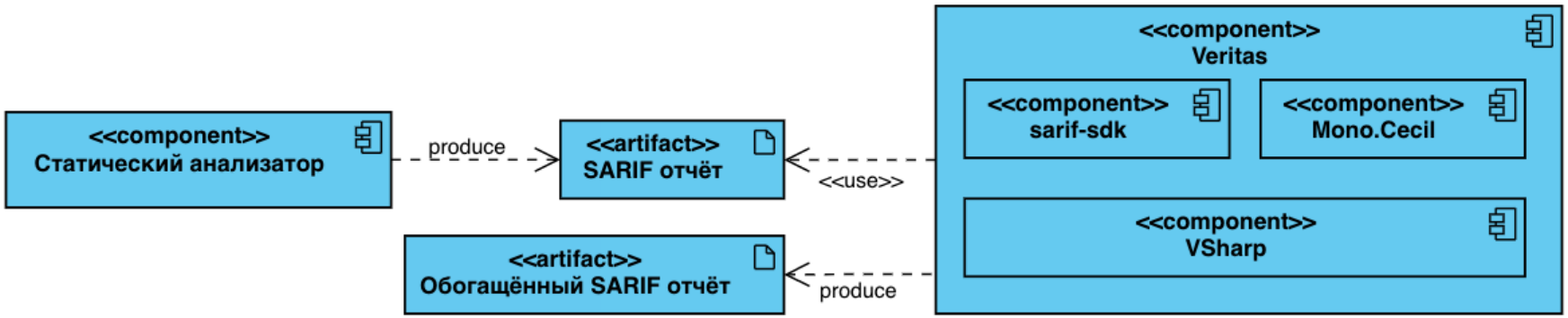
Побочные результаты:

- были найдены ошибки в Infer# и PVS-Studio; Собранная диагностика помогла авторам проектов оперативно исправить неисправности

Общая структура решения

Veritas – обёртка вокруг V#, которая преобразует SARIF отчёты анализаторов в цели символьного исполнения, а затем ранжирует и обогащает изначальный отчёт основываясь на результатах символьного анализа

Рис.: Диаграмма развёртывания



Метод решения

- Фильтрация результатов статического анализатора (Possible null dereference, Possibly index is out of bound);
- Построение целей для символьного исполнения;
- Передача целей в символьный движок V# и интерпретация результатов;
- Срабатывания без цели дополнительно проверяются на корректность локализации;
- Ранжирование результатов и обогащение;

Экспериментальное исследование

Постановка эксперимента

- Проверим насколько полно Veritas может покрыть целями (targets) ошибки из отчётов PVS-Studio, подтверждать срабатывания и оценим долю ошибок, указывающих на некорректную локацию в коде;
- Генератор целей Veritas будет запускаться на 14 проектах;
- Затем подсчитывается доля срабатываний, для которых получилось сгенерировать цели, доля подтверждённых срабатываний и доля срабатываний с некорректной локализацией;
- Рассматриваются срабатывания только типов: V3080, V3146, V3106;

Экспериментальное исследование

Проект	Кол-во поддерживаемых срабатываний	Доля срабатываний с целями, %	Доля срабатываний с неправильной локализацией, %	Доля подтверждённых срабатываний с целями, %	Кол-во найденных исключений
litedb	20	60,00%	40,00%	33,33%	52
NLog	10	70,00%	30,00%	0,00%	4
btcpayserver	114	58,77%	27,19%	5,97%	28
moq4	2	0,00%	0,00%	-	0
nunit	8	12,50%	75,00%	0,00%	0
xunit	11	81,82%	0,00%	0,00%	1
AutoMapper	7	42,86%	57,14%	0,00%	5
spbu-homeworks-1	1	0,00%	100,00%	-	0
ILSpy (ILSpy.XPlat.slnf)	146	47,26%	41,10%	2,90%	123
OpenRA	97	95,88%	0,00%	34,41%	289
Newtonsoft.Json	46	97,83%	0,00%	0,00%	10
parallel-programming-1-sync	0				
parallel-programming-1-thread-pool	0				
parallel-programming-3-thread-pool	0				

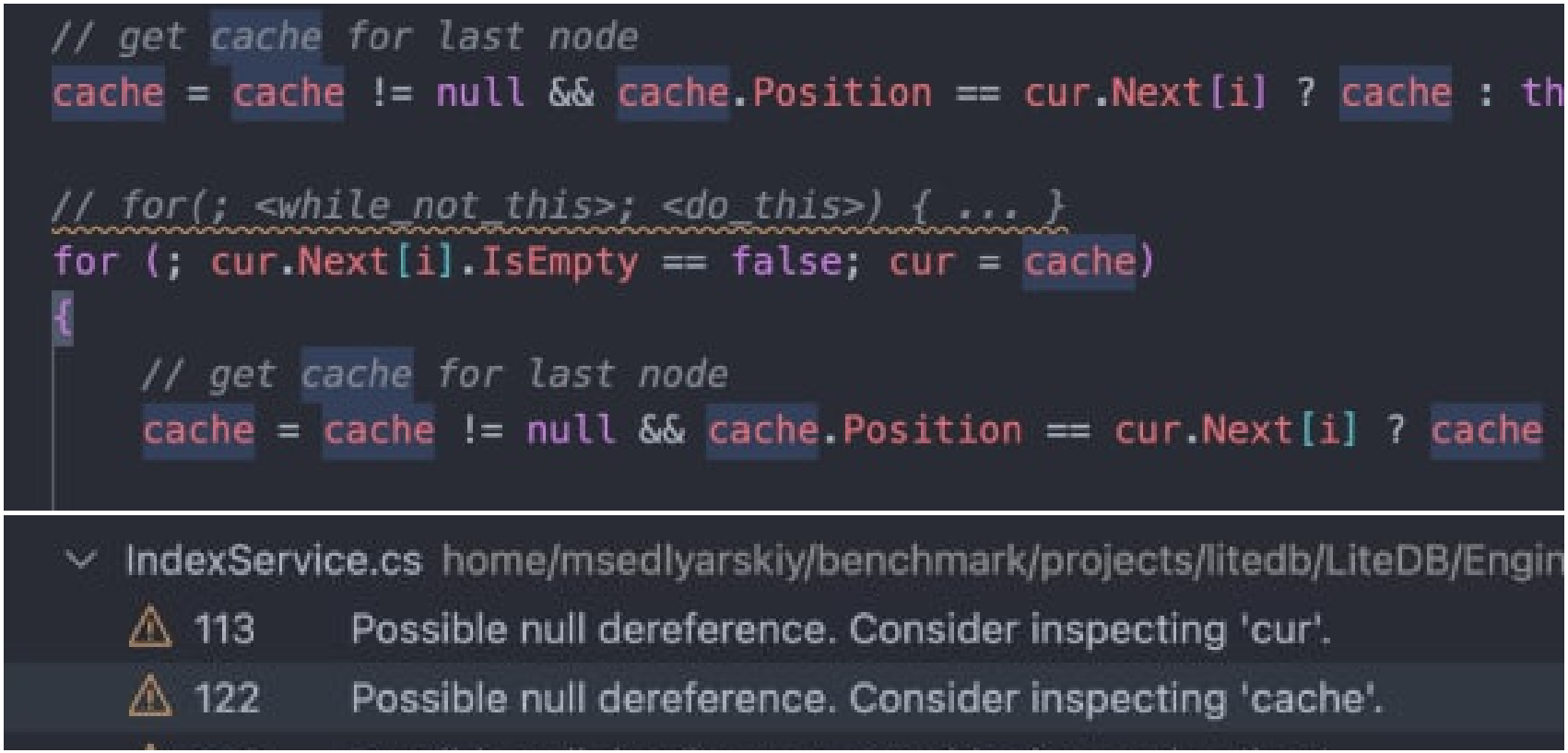
Таблица: Результаты запуска Veritas

Экспериментальное исследование

- Чем можно объяснить, что в среднем 48,46% результатов PVS-Studio не удаётся отобразить на IL инструкцию?
- В среднем, 33,67% результатов PVS-Studio, оставшихся без целей, имеют некорректную привязку к местоположению в коде – локализацию;
- Подтверждается до трети срабатываний с целями – с ростом стабильности V# покрытие будет увеличиваться;
- Символьное исполнение находит ряд новых ошибок;

Экспериментальное исследование

Рис.: Пример некорректной локализации ошибок анализатором PVS-Studio



Результаты

- проведён сравнительный анализ статических анализаторов кода для платформы .NET;
- собран бенчмарк из 17 проектов, на котором были запущены статические анализаторы на настоящих проектах;
- разработан и реализован алгоритм преобразования результатов статического анализатора кода в цели и передача их в движок символьного исполнения V#;
- проведён эксперимент, оценивающий качество полученного инструмента;
- разработан алгоритм загрузки сборок и зависимостей для платформы .net core;
- были выявлены ошибки в анализаторах Infer# и PVS-Studio; оказано содействие авторам проектов в устранении неполадок;