Динамическая символьная интерпретация приложений архитектуры RISCV64

Логунова Влада Игоревна
vlada@ispras.ru
Институт системного программирования им. В.П. Иванникова РАН, Москва

Ежедневное применение программных продуктов успело стать неотъемлемой частью жизни современного человека. Персональные устройства, электроприборы, сервера компаний и организаций, транспорт и умные дома вплетают в повседневность пользователя качественно новую инфраструктуру. Безопасность и надёжность при создании такой инфраструктуры определяется стандартами и отлаженностью соответствующих процессов. Концепция жизненного цикла безопасной разработки ПО [1, 2] существует для того, чтобы минимизировать последствия появления ошибок и уязвимостей в программных продуктах. Данный подход предполагает наличие разноплановых видов тестирования, включая статический и динамический анализ. Динамический анализ, в отличие от статического, использует запуск тестируемого приложения и предоставляет возможность наблюдать результаты проявления ошибок непосредственно в среде исполнения, что сокращает количество ложных срабатываний.

Как правило, для динамического анализа применяются автоматизированные инструменты фаззинг-тестирования. Наиболее распространенный вид фаззинга основан на множественных запусках программы с различными входными данными в надежде увеличить метрику достигнутого покрытия кода. Генерация новых входных данных осуществляется с помощью случайных мутаций данных из предыдущих запусков. Подобный подход называется фаззингом с обратной связью по покрытию или фаззингом "методом серого ящика" [3, 4]. Гибридный фаззинг дополнительно использует входные данные от динамического символьного интерпретатора [5] ("метод белого ящика"), извлекающего более подробную информацию о запуске. В первую очередь, это логические условия ветвления, зависящие от изначальных помеченных данных. Новые входные данные вычисляются аналитически с целью изменить направление ветвления при будущем запуске. Несмотря на то, что символьная интерпретация требует больше времени на один запуск, своевременное получение фаззером вычисленных данных способно повышать общую эффективность анализа [6].

Среди динамических символьных интерпретаторов преимущественно представлены инструменты [7, 8, 9] для архитектуры x86. Однако многообразие программных решений не ограничивается единственной платформой, так что одним из важных аспектов применения технологий динамического анализа является переносимость. В качестве перспективного направления для системного создания безопасного ПО можно рассмотреть открытую процессорную архитектуру RISCV. С точки зрения наличия архитектурно-зависимых компонентов динамический символьный интерпретатор в процессе инструментации может отслеживать преобразования входных данных непосредственно на уровне машинного кода либо на уровне промежуточного представления. В первом случае требуется символьная семантика набора инструкций для каждой анализируемой платформы, во втором кросс-платформенность обеспечивается за счет дополнительного этапа трансляции.

В данной работе представлены результаты применения техник символьной интерпретации для приложений открытой процессорной архитектуры RISCV64 с помощью инструмента Sydr [10]. При построении формул преобразований символьных данных в нем используется открытый символьный фреймворк Triton [11], в который была добавлена символьная семантика соответствующего набора целочисленных инструкций, включающего псевдоинструкции и сокращенные инструкции. Предложенный подход символьной интерпретации не требует наличия исходного кода или дополнительного этапа трансляции в промежуточное представление, обладая при этом обширным набором техник и оптимизаций, включая предикаты безопасности и метод точного определения границ при анализе косвенных переходов. Инструмент был апробирован на наборе синтетических тестов и

реальных приложений, а также проведено экспериментальное сравнение с открытым символьным интерпретатором SymQEMU [12], работающим с применением трансляции бинарного кода в промежуточное представление. Кроме того, добавлена возможность проведения гибридного фаззинга приложений архитектуры RISCV64 с применением фреймворка Sydr-Fuzz [13].

Список литературы:

[1] M. Howard, S. Lipner. The security development lifecycle. Microsoft Press Redmond, 2006.

[2] ГОСТ Р 56939-2016: Защита информации. Разработка безопасного программного обеспечения. Общие требования. — Национальный стандарт РФ, 2016.

[3] K. Serebryany. Continuous fuzzing with libFuzzer and AddressSanitizer. 2016 IEEE Cybersecurity Development (SecDev), page 157. IEEE, 2016.

[4] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse. AFL++: combining incremental steps of fuzzing research. 14th USENIX Workshop on Offensive Technologies (WOOT 20), 2020.

[5] P. Godefroid, M. Y. Levin, D. Molnar. Automated whitebox fuzz testing. Proceedings of the 2008 Network and Distributed System Security Symposium, pages 151–166, 2008.

[6] Fuzzbench (Google). DSE+Fuzzing Experiment Report. 2021. https://www.fuzzbench.com/reports/experimental/2021-07-03-symbolic/index.html.

[7] I. Yun et al. QSYM: a practical concolic execution engine tailored for hybrid fuzzing. 27th USENIX Security Symposium, pages 745–761, 2018.

[8] L. Borzacchiello, E. Coppa, and C. Demetrescu. FUZZOLIC: mixing fuzzing and concolic execution. Computers & Security, 2021.

[9] S. Poeplau and A. Francillon. Symbolic execution with SymCC: don’t interpret, compile! 29th USENIX Security Symposium (SDL) (USENIX Security 20), pages 181–198, 2020.

[10] A. Vishnyakov, A. Fedotov, D. Kuts, A. Novikov, D. Parygina, E. Kobrin, V. Logunova, P. Belecky, and S. Kurmangaleev. Sydr: cutting edge dynamic symbolic execution. 2020 Ivannikov ISPRAS Open Conference (SDL) ISPRAS), pages 46–54. IEEE, 2020.

[11] F. Saudel and J. Salwan. Triton: a dynamic symbolic execution framework. Symposium sur la sécurité des technologies de l’information et des communications, pages 31–54, 2015.

[12] S. Poeplau and A. Francillon. SymQEMU: compilation-based symbolic execution for binaries. Proceedings of the 2021 Network and Distributed System Security Symposium, 2021.

[13] A. Vishnyakov, D. Kuts, V. Logunova, D. Parygina , E. Kobrin, G. Savidov, A. Fedotov. Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle. 2022 Ivannikov ISPRAS Open Conference, 2022.