DOI: 10.55648/1998-6920-2024-18-2-69-78 УДК 004.56

Разработка метода оценки покрытия кода при фаззинг-тестировании программного обеспечения методом черного ящика с использованием аппаратной виртуализации для оценки тестового покрытия

Н. Н. Самарин

ФГУП «Научно-исследовательский институт «Квант»» (г. Москва)

Аннотация: В настоящей статье представлен разработанный метод оценки покрытия при проведении фаззинг-тестирования программного обеспечения, использующий технологии аппаратной виртуализации. Тестируемое программное обеспечение рассматривается как черный ящик. Особенностью предлагаемого метода является возможность выполнения контроля состояния виртуальной машины, в которой осуществляется фаззинг-тестирование, в том числе осуществление мониторинга состояния процессора и входных данных в режиме реального времени. Проведенные эксперименты показали, что разработанный метод позволяет получить оценку тестового покрытия кода с высокой точностью, сопоставимой с методом оценки на основе статической инструментации, который, однако, применим только при проведении фаззинг-тестирования методом белого ящика.

Ключевые слова: информационная безопасность, тестирование, аппаратная виртуализация, оценка покрытия, фаззинг черного ящика.

Для цитирования: Самарин Н. Н. Разработка метода оценки покрытия кода при фаззинг-тестировании программного обеспечения методом черного ящика с использованием аппаратной виртуализации для оценки тестового покрытия // Вестник СибГУТИ. 2024. Т. 18, № 2. С. 69–78. https://doi.org/10.55648/1998-6920-2024-18-2-69-78.

Контент доступен под лицензией Creative Commons Attribution 4.0 License

© Самарин Н. Н., 2024

Статья поступила в редакцию 10.11.2023;
принята к публикации 13.02.2024.

1. Введение

Фаззинг-тестирование – одно из наиболее активно развивающихся направлений поиска архитектурных и логических ошибок, которые являются причиной возникновения уязвимостей в программном обеспечении (ПО) [1]. Основной задачей фаззинг-тестирования является автоматизация поиска уязвимостей с минимизацией вовлечения в процесс человека [1, 2]. При традиционном фаззинг-тестировании вовлечение эксперта необходимо на этапе запуска фаззера – для подготовки входных данных и оценки результатов после завершения тестирования.

Одной из проблем фаззинг-тестирования является поиск способов оценки покрытия исполняемого кода программы, когда она представляет собой черный ящик [1]. В таком случае эксперт не имеет возможности проведения инструментации тестируемой программы на уровне исходных текстов ввиду их отсутствия. Наиболее популярным методом инструмента-

ции в этом случае является бинарная инструментация, однако она ограничена возможностями операционной системы (ОС) [3].

Существуют методы инструментации на базе эмуляторов, к примеру, эмулятора QEMU, позволяющие отслеживать состояния тестируемой программы на низком уровне [3, 4]. Однако эмуляторы накладывают некоторые ограничения – при таком подходе не учитывается состояние физического процессора и целевой (для исполнения тестируемой программы) ОС. Более того, оценка покрытия с использованием эмулятора также ограничена возможностями самого эмулятора и требует больших вычислительных ресурсов для сокращения времени фаззинг-тестирования [3].

В качестве решения обозначенной проблемы предлагается использовать разработанный в рамках данной работы метод покрытия на основе аппаратной виртуализации. Суть метода заключается в том, что электронно-вычислительная машина (ЭВМ), на которой производится фаззинг-тестирование, находится под управлением прозрачного гипервизора одной виртуальной машины. Применение аппаратной виртуализации в контексте оценки покрытия позволяет не только полностью контролировать состояние программы в оперативной памяти, но также контролировать состояние процессора и операционной системы, тем самым фиксируя трассы, исполняемые тестируемой программой.

Аппаратную виртуализацию поддерживают лидирующие компании по производству процессоров – Intel и AMD [5]. В общих чертах реализация поддержки аппаратной виртуализации у данных производителей схожа, однако детали реализации отличаются.

В данной статье проведен анализ технологии аппаратной виртуализации, реализованной как в процессорах Intel, так и в процессорах AMD, а также приведено описание разработанного метода. Проведена практическая апробация разработанного метода с целью оценки покрытия при фаззинг-тестировании на хостовой ОС семейства Linux, эмуляторе QEMU и виртуализированной ОС семейства Linux.

2. Особенности аппаратной виртуализации

Аппаратная виртуализация подразумевает установку и функционирование гипервизора или монитора виртуальных машин на аппаратной составляющей ЭВМ [5]. Аппаратная виртуализация выступает в качестве промежуточного узла между ПО, функционирующим на более высоком уровне абстракции (к примеру, ОС), и аппаратным обеспечением. К наиболее распространенным средствам аппаратной виртуализации относятся VMware vSphere и Microsoft Hyper-V [5, 6, 7].

Отличительной особенностью аппаратной виртуализации от программной является тип установки гипервизора. В случае с аппаратной виртуализацией гипервизор устанавливается на аппаратное обеспечение без какой-либо прослойки – фактически становится своего рода ОС, главной задачей которой является реализация возможности запуска других ОС [6]. При этом гипервизор получает полный контроль над ОС, установленными внутри виртуальных машин, ввиду работы на более низком уровне абстракции.

Зачастую средства аппаратной виртуализации поддерживают несколько виртуальных машин, создаваемых внутри своего пространства. Это достигается благодаря разделению аппаратных ресурсов и выделению для каждой виртуальной машины собственного экземпляра виртуального устройства – будь то центральный процессор или контроллер USB [8].

Выделяется три типа аппаратной виртуализации: полная виртуализация, паравиртуализация и представление с аппаратной поддержкой [9]. Полная виртуализация подразумевает полную имитацию оборудования, в результате чего создается легкопереносимая среда. При использовании паравиртуализации разработчиком создается специальная сборка ОС, в которой изменен набор используемых компонентов в зависимости от предоставляемых ей ресурсов. Представление с аппаратной поддержкой реализует для виртуальной машины полностью виртуализированную среду без доступа к физическому аппаратному обеспечению.

К главному преимуществу аппаратной виртуализации относят исключительную простоту настройки и гибкость масштабирования при увеличение аппаратных ресурсов [7, 10]. Кроме того, важным аспектом является возможность создания 64-разрядной виртуальной машины даже на базе 32-разрядного процессора. При этом программным компонентам, исполняющимся внутри виртуальной машины, не удастся отличить виртуальное оборудование от реального [10].

Наиболее распространенными технологиями аппаратной виртуализации являются Intel VT-x и AMD-V. Далее будет кратко рассмотрена каждая технология.

3. Обзор Intel VT-x и AMD-V

Intel VT-x является расширением для виртуализации архитектуры процессора IA-32 [11]. Оно включает в себя набор инструкций обеспечения аппаратной виртуализации и два режима работы процессора. Основными преимуществами Intel VT-x являются:

– возможность полной изоляции виртуальных машин между собой и аппаратных компонентов ЭВМ;

– упрощенный интерфейс для управления аппаратным обеспечением, с помощью которого можно динамически выделять необходимые ресурсы для виртуальных машин в режиме реального времени;

– надежный механизм обеспечения безопасности данных за счет разграничения ресурсов на низком уровне абстракции.

Добавление дополнительных режимов работы к архитектуре IA-32 обусловлено необходимостью эффективного перехвата и эмуляции ряда служебных инструкций [11]. Первый режим работы – «root» – используется в качестве монитора виртуальных машин, а второй – «non-root» – для монитора гостевых окружений. Вход в режим «root» обеспечивается посредством выполнения инструкции VMXON, а входы в режим «non-root» – посредством выполнения инструкций VMLAUNCH/VMRESUME.

Важным процессом при разработке и применении технологии аппаратной виртуализации является сохранение состояния процессора при переключении между хостовой и гостевой операционными системами, а также при вызове функционирования в режиме монитора виртуальных машин. Для хранения состояния применяется структура VMCS (Virtual Machine Control Structure), которая создается индивидуально для каждой гостевой ОС [7, 11].

Кроме этого, в реализации технологии аппаратной виртуализации Intel применяется механизм, который позволяет гипервизору управлять порядком разметки виртуальных адресов и их проецирования на физические адреса памяти. Таким механизмом является EPT (Extended Page Table) [8, 11]. Он представляет собой таблицу трансляции адресов второго уровня и обеспечивает значительный прирост производительности работы блока управления памятью [11].

Реализация технологии аппаратной виртуализации от производителя AMD во многих архитектурных аспектах совпадает с реализацией от Intel. Так, процессоры AMD поддерживают режим работы процессора GuestMode, аналогичный режиму работы «non-root» в процессорах Intel, структуру VMCB – аналог VMCS – инструкцию для перехода в виртуальное окружение VMRUN и аналог EPT – Real Mode w/ Paging [12]. Однако в сравнении с Intel выделяется и ряд отличий. Например, для поддержки механизма Real Mode w/ Paging используется аппаратный контроллер памяти с поддержкой виртуализации. Это необходимо в первую очередь для обеспечения безопасности контроллера DMA (Direct Memory Access) [12]. Кроме того, используется защищенный монитор виртуальных машин – SVM (Secure Virtual Monitor), который обеспечивает безопасный запуск и контроль гостевой ОС на базе доверенного модуля платформы – TPM (Trusted Platform Module) [12].

Алгоритмы работы аппаратной виртуализации от Intel и AMD также аналогичны и предоставляют не только изоляцию тестируемой программы от аппаратных компонентов, но

и позволяют отслеживать состояние ее исполнения для оценки покрытия при фаззинг-тестировании. Поддержка аппаратной виртуализации двумя наиболее распространенными производителями процессоров позволяет обеспечить предлагаемый метод переносимостью без необходимости его доработки под конкретное аппаратное обеспечение.

4. Существующие методы оценки покрытия

Для оценки покрытия традиционно используются два метода – инструментация и отслеживание входных корпусов [13]. При этом инструментация делится на статическую (на уровне исходных текстов или на уровне исполняемого кода) и динамическую [13, 14]. Статическая инструментация на уровне исходных текстов включает в себя процесс компиляции тестируемого ПО с целью размещения внутри функциональных объектов специальных маркеров, при достижении которых средство фаззинг-тестирования получает сведения о прохождении трассы [15]. В результате сбора и накопления таких сведений рассчитывается общее покрытие тестируемого ПО.

Статическая инструментация на уровне исполняемого кода также заключается в добавлении специальных маркеров, однако может быть выполнена в отношении различных уровней детализации ПО – начиная от уровня инструкция и заканчивая уровнем образа исполняемого файла [16]. Так, например, средство для профилирования Valgrind осуществляет инструментацию на уровне функциональных объектов по следующему принципу:

1. В исполняемом коде выявляются все вызовы функциональных объектов.
2. Valgrind создает теневой стек с адресами возврата выявленных функций.
3. В ходе профилирования сравнивается значение адреса возврата функции с адресом, приведенным на теневом стеке.
4. Если адрес возврата на теневом и оригинальном стеках не совпадают, запоминается адрес инструкции, которая изменила стек.
5. В результате профилирования полученные адреса агрегируются, и на их основе рассчитывается итоговое покрытие.

Существенным недостатком такого подхода является тот факт, что сведения о покрытии могут быть получены только в результате нарушения работы тестируемой программы, что сказывается на точности оценки покрытия при низком количестве нарушений работ.

Динамическая инструментация работает схожим образом и применяется при невозможности использования статической инструментации [14, 16]. В этом случае маркеры устанавливаются в оперативной памяти, в которую загружена тестируемая программа. Динамическая инструментация имеет качественно более низкие оценки покрытия в связи с тем, что представление функциональных объектов в памяти может быть изменено компилятором на безусловные переходы вместо традиционного вызова [15]. Кроме того, в данном методе остается существенный недостаток, связанный с возможностью фиксирования показателей покрытия только при нарушении работы программы, что также накладывает ограничения при оценке итогового покрытия в контексте ее точности [17].

Реализация конкретного метода динамической инструментации может отличаться в зависимости от исходного алгоритма. Одним из наиболее распространенных алгоритмов является промежуточное представление исполняемого кода в памяти, которое модифицируется средством инструментации [18]. Это, в свою очередь, позволяет обеспечить целостность памяти, в которую размещен исполняемый код, и исключить возможные нарушения работы, связанные именно с процессом инструментации, такие как нарушение целостности данных программы, которые контролируются самой программой.

Отслеживание входных корпусов реализовано в виде режима фаззинг-тестирования на базе QEMU, а также в таких средствах инструментации, как Gcov и Lcov [3, 19]. Данные средства минимизируют объем входного корпуса с целью сбора как можно большего покрытия в кратчайшие сроки. При этом в памяти отслеживается состояние входного корпуса, и в

случае его уникальной модификации в процессе работы тестируемой программы считается, что достигнута новая трасса и, как следствие, получен прирост к покрытию.

Отслеживание входных корпусов основано на принципе создания большого количества входных данных фиксированного небольшого объема. На каждой итерации для функциональных объектов, в которые передаются входные данные, отслеживается их состояние, и в случае изменения данных фиксируется адрес, по которому вызывается модифицирующий код. По результатам тестирования множество адресов, по которым произведена модификация входных данных, анализируется, затем рассчитывается оценка покрытия. Важно отметить, что в случае, если входные данные в тестируемой программе не модифицируются, оценка покрытия будет нулевая, что, в сущности, не будет соответствовать действительности.

Таким образом, наиболее надежным способом инструментации является статическая инструментация на уровне исходных текстов, однако она неприменима при проведении фаззинг-тестирования черного ящика [19]. Предлагаемый метод, описанный далее, позволяет добиться точности оценки покрытия, близкой к точности оценки при статической инструментации, за счет возможности отслеживания состояния и содержимого аппаратных ресурсов ОС, на которой проводится фаззинг-тестирование.

5. Описание предлагаемого метода на основе аппаратной виртуализации для оценки покрытия

Одной из ключевых особенностей предлагаемого метода является поддержка большинства известных средств для проведения фаззинг-тестирования. Это обеспечивается за счет функционирования метода на уровне абстракции ниже ОС, что позволяет реализовать совместимость и мониторинг тестируемого ПО вне зависимости от среды исполнения.

Виртуальная машина, находящаяся под управлением гипервизора, предоставляет необходимые программные интерфейсы для доступа к состоянию виртуального процессора, оперативной памяти и видеопамяти [9]. В ходе мониторинга состояния виртуальной машины, на которой производится фаззинг-тестирование, гипервизор фиксирует уникальные трассы, исполняемые процессором, и отслеживает входные данные, находящиеся в оперативной памяти или видеопамяти. При выявлении уникальных трасс фиксируется прирост покрытия. Дополнительным источником сведений о приросте покрытия выступают данные в памяти, которые также отслеживаются в части их уникальной модификации.

Разработанный метод состоит из следующих шагов:

1. В виртуальной машине под управлением гипервизора запускается фаззинг-тестирование с заранее подготовленными входными корпусами для их дальнейшей мутации.
2. Осуществляется запуск тестируемой программы и передача ей входных данных.
3. Для каждого функционального объекта в рамках одной итерации гипервизор фиксирует трассу.
   3.1. Извлекается состояние процессора из структуры VMCS.
   3.2. Функциональные объекты определяются по инструкциям call и по подготовке регистров к вызову функции вместе с инструкцией безусловного перехода.
   3.3. Фиксируется состояние входных данных, переданных средством фаззинг-тестирования ПО в памяти на предмет их модификации.
4. По окончании одной итерации код гипервизора, функционирующий внутри ОС, создает файл-отчет, содержащий представление пройденных трасс и путей модификации входных данных.
5. Входные данные проходят процесс мутации штатным или доработанным модулем средства фаззинг-тестирования, затем происходит повторная итерация в соответствии с шагом 2. Если цели фаззинг-тестирования по покрытию или времени достигнуты, алгоритм завершается.

6. По окончании фаззинг-тестирования код гипервизора, функционирующий внутри ОС, агрегирует файлы отчетов покрытия, на основе совокупности которых рассчитывается итоговое покрытие для тестируемого ПО.

Состояние процессора во время фаззинг-тестирования извлекается из структуры VMCS, а оперативная память и видеопамять отслеживаются в режиме реального времени с целью мониторинга состояния переданных входных данных. За счет возможности отслеживания состояния процессора и памяти исключается возможность ложного определения или пропуска новой трассы, что, в свою очередь, положительно сказывается на точности оценки покрытия. Для подтверждения работы предложенного метода проведен эксперимент, описанный далее.

6. Проведение экспериментов для оценки покрытия с использованием предложенного метода

В ходе проведения экспериментов точность определения степени покрытия оценивалась относительно статической инструментации с использованием средства фаззинг-тестирования AFL (S1). В качестве тестируемого программного обеспечения выбраны утилиты cURL – используется для передачи данных по ссылке типа URL, iptables – утилита для настройки и ограничения сетевых подключений и vlc – медиаплеер с возможностью запуска проигрывания из командной строки. Кроме статической инструментации AFL, для оценки покрытия в эксперименте использовалась оценка на базе эмулятора QEMU (S2) и инструментация с использованием Gcov (S3). Фаззинг-тестирование проводилось на протяжении 24 часов. Для чистоты эксперимента модулю мутации AFL передана инициирующая входная последовательность, которая обеспечит мутацию одинаковыми методами и алгоритмами для каждого экземпляра средства фаззинг-тестирования.

Результаты эксперимента приведены в табл. 1. Предложенный метод условно отмечен в таблице как S4.

Таблица 1. Результаты эксперимента по оценке покрытия с использованием предложенного метода

Метод оценки покрытия / **Программное обеспечение**	**S1**	**S2**	**S3**	**S4**
cURL	21.8 %	19.9 %	17.6 %	20.2 %
iptables	16.3 %	15.4 %	14.1 %	15.9 %
vlc	7.8 %	2.5 %	6.6 %	8.3 %

В ходе оценки результатов выявлено, что покрытие при фаззинг-тестировании vlc методом S2 значительно ниже покрытия, определенного другими методами. QEMU в режиме оценки покрытия не имеет графического интерфейса, и при попытке открытия vlc для воспроизведения работа плеера завершалась с ошибкой, свидетельствующей о том, что в ходе работы не удалось создать окно.

Предложенный метод показал наиболее близкую к референсному значению, представленному методом S1, оценку покрытия.

7. Заключение

Технология аппаратной виртуализации предоставляет широкие возможности по контролю состояния виртуальной машины. При этом исполняемый код, функционирующий внутри виртуальной машины, исполняется, как если бы он исполнялся на физическом оборудовании. Это позволяет исключить потенциально нестабильное поведение тестируемого программного обеспечения, что в конечном счете обеспечивает чистоту результатов работы средства фаззинг-тестирования и оценки покрытия при проведении тестирования методом черного ящика.

Разработанный метод в ходе эксперимента показал удовлетворительные результаты, приближенные к результатам, полученным при статической инструментации исходных текстов, что свидетельствует о его применимости для фаззинг-тестирования методом черного ящика.

Полученные результаты позволяют продолжить исследования в области методов на основе аппаратной виртуализации для фаззинг-тестирования. За счет возможности взаимодействия с виртуальной машиной и отслеживания ее состояния становится возможным создание универсального монитора процесса фаззинг-тестирования, который не только фиксирует состояние данных и процессора, но и позволяет восстанавливать спецификацию программы.

Дальнейшие исследования должны быть направлены на анализ методов фаззинг-тестирования программного обеспечения, которое требует взаимодействия с различными аппаратными компонентами.

Литература

1. *Ерышов В. Г.* Фаззинг-тестирование. Классификация современных средств фаззинга // Сборник избранных статей по материалам научных конференций ГНИИ «Нацразвитие»: Международные научные конференции, Санкт-Петербург, 26–31 августа 2021 года. С. 287–289. DOI 10.37539/AUG298.2021.94.77.007.
2. *Zhang C., Chen J.* Fuzzing Methods Recommendation Based on Feature Vectors // Proc. 36th IEEE/ACM International Conference on Automated Software Engineering, 2021. P. 1079–1081. DOI 10.1109/ASE51524.2021.9678630.
3. *Eisele M., Maugeri M., Shriwas R. et al.* Embedded fuzzing: a review of challenges, tools, and solutions // Cybersecurity. 2022. V. 5, № 1. P. 1–18. DOI 10.1186/s42400-022-00123-y.
4. *Kim S., Cho Ja., Lee Ch., Shon T.* Smart seed selection-based effective black box fuzzing for IIoT protocol // The Journal of Supercomputing. 2020. V. 76, № 12. P. 10140–10154. DOI 10.1007/s11227-020-03245-7.
5. *Silakov D. V.* The use of hardware virtualization in the context of information security // Programming and Computer Software. 2012. V. 38, № 5. P. 276–280. DOI 10.1134/S0361768812050064.
6. *Epishkina A. V., Kanner A. M., Kanner T. M.* Comprehensive Testing of Software and Hardware Data Security Tools Using Virtualization // Mechanisms and Machine Science (book series). 2020. V. 80. P. 79–87. DOI 10.1007/978-3-030-33491-8_9.
7. *Куликов С. С., Клименков Е. И.* Миграция и виртуализация как технологии обеспечения совместимости аппаратных и программных средств в образовательном процессе // Информатизация образования. 2010. № 1 (58). С. 82–94.
8. *Борисенко Б. Б., Килюшева Е. Д.* Обнаружение гипервизора, использующего технологию аппаратной виртуализации // Проблемы информационной безопасности. Компьютерные системы. 2014. № 4. С. 76–84.

9. *Егоров В. Ю., Карпов И. В., Матвеев Е. А.* Методика управления оперативной памятью в технологиях аппаратной виртуализации VT и SVM // Наукоемкие технологии. 2010. Т. 11, № 4. С. 35–45.
10. *Du J., Zwaenepoel W., Sehrawat N.* Performance profiling in a virtualized environment // Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing, 2010, Boston, MA.
11. *Юлюгин Е. А.* Корректное и быстрое исполнение отдельных инструкций архитектуры Intel® 64 в виртуальном окружении // Информационные технологии. 2019. Т. 25, № 3. С. 157–164. DOI 10.17587/it.25.157-164.
12. Intel software development manual [Электронный ресурс]. URL: `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html` (дата обращения: 01.12.2023).
13. AMD documentation hub [Электронный ресурс]. URL: `https://www.amd.com/en/search/documentation/hub.html` (дата обращения: 01.12.2023).
14. *Радаев В. А.* Методика тестирования приложений на основе анализа покрытия исходного кода и фаззинга // Материалы Межвузовской научно-технической конференции студентов, аспирантов и молодых специалистов им. Е. В. Арменского, Москва, 25 февраля – 04 марта 2020 года. С. 165.
15. *Воронина Е. Н.* Сравнительный анализ подходов к поиску уязвимостей программного обеспечения методом фаззинг-тестирования //Сборник трудов Десятой международной научно-технической конференции «Безопасные информационные технологии», Москва, 03–04 декабря 2019 года. С. 75–80.
16. *Вишняков А. В.* Поиск ошибок в бинарном коде методами динамической символьной интерпретации: специальность 23.50.00: диссертация на соискание ученой степени кандидата физико-математических наук, 2022. 131 с.
17. *Созин М. В.* Фаззинг на основании состояния исполнения программы // Материалы VIII Всероссийской молодежной школы-семинара по проблемам информационной безопасности «Перспектива», Таганрог, 10–13 октября 2019 года. С. 12–17.
18. *Валеев Д. Р., Котенко И. В.* Анализ подходов к автоматической обработке результатов фаззинг-тестирования // Сборник научных статей XII Международной научно-технической и научно-методической конференции «Актуальные проблемы инфотелекоммуникаций в науке и образовании» (АПИНО). Санкт-Петербург, 28 февраля – 01 марта 2023 года. Т. 1. С. 219–224.
19. *Попов В. И., Тюрин М. Е., Семибратов А. С.* Анализ современных инструментальных средств, реализующих фаззинг-тестирование // Сборник трудов II Всероссийской научно-технической конференции «Состояние и перспективы развития современной науки по направлению «ИТ-технологии»», Анапа, 23–24 марта 2023 года. Т. 2. С. 83–94.

Самарин Николай Николаевич

к.т.н., ФГУП «Научно-исследовательский институт «Квант»» (125438, Москва, 4-й Лихачёвский переулок, 15), e-mail: `samarin_nik@mail.ru`, ORCID: 0009-0007-4911-8471, AuthorID: 1100538, SPIN-код: 2176-0939.

Автор прочитал и одобрил окончательный вариант рукописи.
Автор заявляет об отсутствии конфликта интересов.

Development of the Method for Assessing Code Coverage During Black-Box Fuzz-Testing of Software Using Hardware Virtualization to Evaluate Test Coverage

Nikolay N. Samarin

Scientific Research Institute "Kvant" (Moscow)

Abstract: This article presents a developed method for assessing code coverage during fuzz-testing of software using hardware virtualization. The tested software is considered as a black box. The proposed method's feature is the ability to monitor the state of the virtual machine in which the fuzz testing is carried out, including monitoring the processor's state and input data in real-time. The experiments conducted showed that the developed method allows us to obtain an accurate assessment of code test coverage comparable to the static instrumentation-based method, which is only applicable when conducting white-box fuzz testing.

Keywords: information security, testing, hardware virtualization, coverage assessment, black-box fuzzing.

For citation: Samarin N. N. Development of the method for assessing code coverage during black-box fuzz-testing of software using hardware virtualization to evaluate test coverage (in Russian). *Vestnik SibGUTI*, 2024, vol. 18, no. 2, pp. 69-78. https://doi.org/10.55648/1998-6920-2024-18-2-69-78.

Content is available under the license Creative Commons Attribution 4.0 License

© Samarin N. N., 2024

The article was submitted: 10.11.2023;
accepted for publication 13.02.2024.

References

1. Yeryshov V. G. Fazzing testirovaniye. Klassifikatsiya sovremennykh sredstv fazzinga. *Sbornik izbrannykh statey po materialam nauchnykh konferentsiy GNII "Natsrazvitiye»: Mezhdunarodnyye nauchnyye konferentsii*, Sankt-Peterburg, 26-31 August, 2021, pp 287-289. DOI 10.37539/AUG298.2021.94.77.007.
2. Zhang C., Chen J. Fuzzing Methods Recommendation Based on Feature Vectors. *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering*, 15-19 November, 2021 pp. 1079-1081. DOI 10.1109/ASE51524.2021.9678630. EDN BXHZDT.
3. Eisele M., Maugeri M., Shriwas R., et al. Embedded fuzzing: a review of challenges, tools, and solutions. *Cybersecurity*, 2022. vol. 5, no. 1, pp 1-18. DOI 10.1186/s42400-022-00123-y. EDN JCXYPZ.
4. Kim S., Cho Ja., Lee Ch., Shon T. Smart seed selection-based effective black box fuzzing for IIoT protocol. *The Journal of Supercomputing*, 2020, vol. 76, no. 12, pp. 10140-10154. DOI 10.1007/s11227-020-03245-7. EDN KYWZAP.
5. Silakov D. V. The use of hardware virtualization in the context of information security. *Programming and Computer Software*, 2012, vol. 38, no. 5, pp. 276-280. DOI 10.1134/S0361768812050064. EDN RGNETF.
6. Epishkina A. V., Kanner A. M., Kanner T. M. Comprehensive Testing of Software and Hardware Data Security Tools Using Virtualization. *Mechanisms and Machine Science (book series)*, 2020, vol. 80, pp. 79-87. DOI 10.1007/978-3-030-33491-8_9. EDN EIFAXP.
7. Kulikov S. S., Klimenkov Ye. I. Migratsiya i virtualizatsiya kak tekhnologii obespecheniya sovmestimosti apparatnykh i programmnykh sredstv v obrazovatel'nom protsesse [Migration and virtualization as technologies for ensuring compatibility of hardware and software in the educational process]. *Informatizatsiya obrazovaniya*, 2010, no. 1(58), pp. 82-94. EDN XMRDYT.
8. Borisenko B. B., Kilyusheva Ye. D. Obnaruzheniye gipervizora, ispol'zuyushchego tekhnologiyu apparatnoy virtualizatsii [Detecting a hypervisor that uses hardware virtualization technology *Problemy informatsionnoy bezopasnosti. Komp'yuternyye sistemy*, 2014, no. 4, pp. 76-84. EDN TJZKKV.
9. Yegorov V. YU., Karpov I. V., Matveyev Ye. A. Metodika upravleniya operativnoy pamyat'yu v tekhnologiyakh apparatnoy virtualizatsii VT i SVM [Methodology for managing RAM in VT and SVM

hardware virtualization technologies]. *Naukoyemkiye tekhnologii*, 2010, vol. 11, no. 4, pp. 035-045. EDN QCTKPH.
10. Du J., Zwaenepoel W., Sehrawat N. Performance profiling in a virtualized environment []. *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2010: 2*, Boston, MA, 22 June, 2010. EDN NZEKCW.
11. Yulyugin Ye. A. Korrektnoye i bystroye ispolneniye otdel'nykh instruktsiy arkhitektury Intel® 64 v virtual'nom okruzhenii / Ye. A. Yulyugin [Correct and fast execution of individual Intel® 64 architecture instructions in a virtual environment]. *Informatsionnyye tekhnologii*, 2019, vol. 25, no. 3, pp. 157-164. DOI 10.17587/it.25.157-164. EDN VWTPFI.
12. *Intel software development manual*, available at: `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html` (accessed 01.12.2023).
13. *AMD documentation hub*, available at: `https://www.amd.com/en/search/documentation/hub.html` (accessed 01.12.2023).
14. Radayev V. A. Metodika testirovaniya prilozheniy na osnove analiza pokrytiya iskhodnogo koda i fazzinga [Methodology for testing applications based on source code coverage analysis and fuzzing]. *Mezhvuzovskaya nauchno-tekhnicheskaya konferentsiya studentov, aspirantov i molodykh spetsialistov im. Ye.V. Armenskogo*, Moscow, 25 Febrary, 2020, p. 165. EDN ZQSZQP.
15. Voronina Ye. N. Sravnitel'nyy analiz podkhodov k poisku uyazvimostey programmnogo obespecheniya metodom fazzing-testirovaniya [Comparative analysis of approaches to searching for software vulnerabilities using fuzzing testing]. *Bezopasnyye informatsionnyye tekhnologii: Sbornik trudov Desyatoy mezhdunarodnoy nauchno-tekhnicheskoy konferentsii*, Moscow, 03–04 December, 2019. pp. 75-80. EDN DXEHYH.
16. Vishnyakov A. V. Poisk oshibok v binarnom kode metodami dinamicheskoy simvol'noy interpretatsii: spetsial'nost' [Comparative analysis of approaches to searching for software vulnerabilities using fuzzing testing]. Abstract of Ph. D. thesis., 2022, p.131. EDN MRBRIF.
17. Sozin M. V. Fazzing na osnovanii sostoyaniya ispolneniya programmy [Fuzzing based on program execution state]. *Perspektiva-2019: Materialy VIII Vserossiyskoy molodezhnoy shkoly-seminara po problemam informatsionnoy bezopasnosti*, Taganrog, 10-13 October, 2019, pp. 12-17. EDN HVJPAW.
18. Valeyev D. R., Kotenko I. V. Analiz podkhodov k avtomaticheskoy obrabotke rezul'tatov fazzing-testirovaniya [Analysis of approaches to automatic processing of fuzzing testing results]. *Aktual'nyye problemy infotelekommunikatsiy v nauke i obrazovanii (APINO 2023): Sbornik nauchnykh statey. XII Mezhdunarodnaya nauchno-tekhnicheskaya i nauchno-metodicheskaya konferentsiya*, Sankt-Peterburg, 28 Fevrary, 2023, vol. 1. pp. 219-224. EDN HOECIC.
19. Popov V. I., Tyurin M. Ye., Semibratov A. S. Analiz sovremennykh instrumental'nykh sredstv, realizuyushchikh fazzing-testirovaniye [Analysis of modern tools that implement fuzzing testing]. *Sostoyaniye i perspektivy razvitiya sovremennoy nauki po napravleniyu IT-tekhnologii: Sbornik trudov II Vserossiyskoy nauchno-tekhnicheskoy konferentsii*, Anapa, 23-24 March, 2023, vol. 2. pp. 83-94. EDN DZVUMH.

Nikolay N. Samarin

Cand. of Sci. (Engineering), Scientific Research Institute "Kvant" (125438, Moscow, 4th Likhachevsky per. 15), e-mail: `samarin_nik@mail.ru`, ORCID: 0009-0007-4911-8471, AuthorID: 1100538, SPIN code: 2176-0939.

Research interests: information security, information protection, software analysis for undocumented capabilities, fuzzing testing, security assessments of automated systems.