УДК 004.056.5

Е.А. Деркач, А.А. Шелупанов

Методологические подходы к обеспечению безопасности программных продуктов

Поддержание безопасности программных средств является одним из ключевых аспектов при их разработке. Безопасности уделяется внимание на всех этапах жизненного цикла производства программного обеспечения (ПО), начиная от анализа требований, проектирования и разработки и заканчивая тестированием, выпуском и последующей поддержкой. В рамках данной работы рассматривается отечественный и зарубежный опыт в обеспечении безопасности будущего ПО как на отдельных этапах жизненного цикла его разработки, так и в комплексном формате. Каждый из методов имеет свои преимущества и недостатки. Изучение опыта исследователей позволит применить полученные знания при разработке собственного методологического подхода, содержащего в себе минимальный комплекс мер, необходимых для обеспечения достаточного уровня безопасности разрабатываемого программного обеспечения.

Ключевые слова: создание безопасного программного обеспечения, меры обеспечения безопасности, методы тестирования безопасности.
DOI: 10.21293/1818-0442-2024-27-1-37-43

Современные темпы развития информационных технологий приводят к повсеместному внедрению различного программного обеспечения (ПО). Такое ПО может обладать уязвимостями разной степени критичности, которые без контроля качества и предварительного тестирования попадут к конечному пользователю и приведут к негативным последствиям. По сегодняшним данным в сфере информационной безопасности наблюдается существенный рост числа атак на основе эксплуатации уязвимостей [1–2], которые могут быть случайно допущены ещё на этапе разработки ПО, что в конечном счёте приведёт к утечке персональных данных и получению конфиденциальной информации злоумышленниками.

Процесс создания безопасного ПО направлен на обеспечение защиты конечных программных продуктов и может включать в себя большой перечень мер, направленных на недопущение какого-либо вмешательства во внутреннюю структуру ПО. Выделяется несколько основных этапов разработки ПО: анализ требований; проектирование архитектуры; разработка и тестирование. На каждом этапе находят своё применение различные меры по обеспечению безопасности, однако с учётом огромных темпов и скорости развития ИТ-индустрии появляется необходимость в оптимизации доступных ресурсов, используемых при разработке программных средств.

Выбор наиболее эффективных и точных мер обеспечения безопасности беспокоит как отечественных исследователей в области информационных технологий, так и их зарубежных коллег уже достаточно давно. Одни исследователи акцентируют своё внимание на отдельных методах обеспечения безопасности [3], другие занимаются разработкой целого комплекса мер [4]. Некоторыми учёными даже предпринимаются попытки применения машинного обучения [5] при обеспечении безопасности разрабатываемого ПО.

Отечественный опыт в тестировании

Коллективы из разных городов России занимаются изучением эффективных способов обеспечения безопасности ПО. Например, исследователи из Москвы и Московской области [6] в качестве основного вида тестирования предлагают проводить тестирование безопасности программного кода или Security Code Review, который осуществляется на этапе разработки ПО. Ключевым моментом данного подхода является тот факт, что основной задачей стоит выявление на раннем этапе ряда уязвимостей самого кода, которые могут быть потенциально использованы злоумышленниками. Среди таких уязвимостей могут быть:

– отсутствие инициализации данных;

– утечка, нехватка и использование освобождённой памяти;

– выход вычислений за пределы диапазона при преобразовании переменных числового типа;

– ошибки определения времени и десинхронизация;

– переполнение, чтение и запись вне буфера;

– формирование отрицательного значения длины строк или числа элементов в массиве.

Учёные предлагают в качестве основных методов тестирования безопасности кода применять:

– ручную инспекцию кода;

– статический (шаблонный) анализ кода и

– динамический анализ кода при его исполнении (рис. 1).

Ручная инспекция кода является наиболее эффективной с точки зрения полноты и точности проверок, однако требует высокого уровня квалификации эксперта и обладает некоторой степенью субъективизма, что может исказить конечные результаты проверки.

Статический анализ кода хоть и является автоматизированным средством и позволяет заниматься поиском ряда уязвимостей (например, XSS-запросов, инъекций кода, ошибок входных данных и др.),

однако может давать ложные срабатывания и сильно зависит от шаблонов запросов.

Последний метод – динамический анализ – предназначен для отслеживания действий программы во время её работы, что позволяет обнаружить проблемы при установке, изменении прав доступа, работе с памятью, пересылке паролей и пр.

Рис. 1. Виды методов тестирования безопасности программного кода с краткой характеристикой [6]

Отдельно учёные выделяют фаззинг-тестирование как один из перспективных методов испытания ПО благодаря лёгкости автоматизации и непрерывности проведения, но считают, что он больше подходит для тестирования надёжности и качества ПО, чем безопасности.

Другой коллектив учёных из Москвы [7] занимался анализом уязвимостей программного обеспечения, а именно тестированием ПО в условиях имитации компьютерных атак.

В работе выделяют два основных метода для определения уязвимостей: сканирование и зондирование (рис. 2).

Первый метод является пассивным и достаточно простым в реализации, в ходе применения которого производится косвенное подтверждение наличия уязвимостей. Сканирование направлено на:

– сбор информации о сети;

– составление перечня ресурсов сети (ОС, служб и узлов) без влияния на производительность;

– поиск неидентифицированных устройств в сети.

Второй метод – зондирование – является активным, где происходит имитация компьютерной атаки, использующей проверяемую уязвимость. Он более точный, но при этом имеет большую продолжительность относительно сканирования.

Сканирование и зондирование подразделяют на категории. Для пассивного анализа (сканирования) они приводят следующие алгоритмы:

– алгоритм проверки пакетов;

– алгоритм статистического анализа исходного кода программ. В случаях активного анализа (зондирования) приводят следующие позиции:

– алгоритм подготовки некорректных данных на вход программ;

– алгоритм имитации компьютерных атак.

Рис. 2. Методы определения уязвимостей с краткой характеристикой [7]

Сканирование может проводиться по двум алгоритмам. Первый является проверкой заголовков пакетов, позволяющий проводить идентификацию открытых портов устройств, собирать и анализировать полученную информацию, сравнивая с базой данных потенциальных уязвимостей и формируя вывод о наличии уязвимостей. А второй – алгоритм статистического анализа исходного кода программ – позволяет обнаружить следующие уязвимости:

– ошибки синхронизации;

– испорченный ввод;

– ошибки форматных строк;

– переполнение буфера.

Методы зондирования осуществляются также по двум алгоритмам:

– алгоритму подготовки некорректных данных на вход программ, заключающемуся в направлении большого массива некорректных данных с целью определения поведения программы (своего рода фаззинг-тестирование, тестирование на отказ);

– алгоритму имитации компьютерных атак, благодаря которому имитируются компьютерные атаки, однако не всегда это уместно, поскольку имеется потенциальный риск нанести вред тестируемой инфраструктуре.

Учёные из Санкт-Петербурга [8] в своих исследованиях определили, что в качестве мер по обеспечению безопасности на примере средств защиты информации (СЗИ) необходимо пользоваться только прямым моделированием наиболее распространённых атак, т.е. зондированием потенциальных уязви-

мостей. В частности, они рассматривают применение следующих мер (рис. 3):

– Server-Side Includes (SSI) Injection атак, основанных на модификации серверных команд в коде HTML с последующим их запуском на стороне сервера;

– Cross-Site Scripting (XSS) атак, схожих с предыдущим видом, но рассчитанных на исполнение на стороне клиента;

– Cross Site Request Forgery (XSRF или CSRF) атак, перенаправляющих клиента на фишинговые сайты, используя уязвимости протокола HTTP;

– инъекций кода, направленных на исполнение части вредоносного кода, с целью получения доступа до важной системной информации;

– DDoS-атак, предназначенных для перегрузки системы значительным количеством запросов;

– Authorization Bypass – получение несанкционированного доступа к сведениям других пользователей.

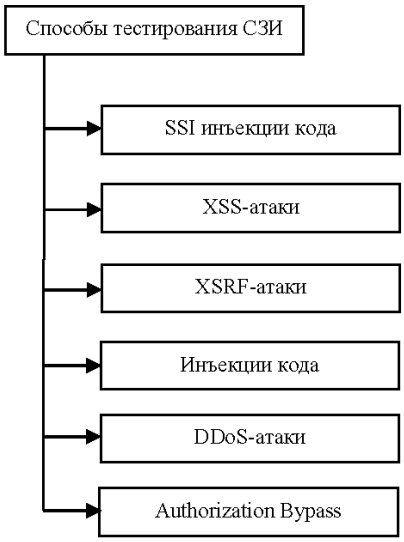

Рис. 3. Способы тестирования средств защиты информации [8]

Кроме того, исследователи предлагают выделить в качестве возможных методов тестирования СЗИ:

– системное;

– регрессионное и

– нагрузочное.

Для выполнения всех тестовых сценариев допускается не только автоматизация процесса тестирования, но и привлечение экспертов для ручных проверок.

Группа исследователей из Москвы [9] рассматривала методы тестирования на примере средств защиты информации. В частности, изучались подходящие методы тестирования с учётом особенностей программно-аппаратных СЗИ, функционирующих до загрузки в ОС, в самой ОС, а также расположенных на флеш-памяти на примере ПСКЗИ ШИПКА. Главной идеей работы являлось определение тестовых методов, подходящих для такого рода задач.

В качестве методов тестирования флеш-памяти рассматривались скорость чтения и записи, а также проводились нагрузочные испытания, где в качестве анализируемых результатов рассматривались:

– скорость чтения и записи (пакет мелких файлов до 500 Кб);

– скорость чтения и записи (одиночный файл на несколько Гб);

– время доступа при чтении и записи при работе с блоками (4, 64 и 1024 Кб);

– работа с отдельными логическими томами;

– работа со всей доступной памятью.

При тестировании СЗИ, функционирующих до авторизации в ОС и в самой ОС, исследователи предлагают проводить тестирование функциональное, нефункциональное и тестирование, связанное с изменениями. Авторы считают, что ряд проверок возможно автоматизировать, однако далеко не всё, поскольку имеется аппаратная часть, которую приходится «пробрасывать» вручную при эмуляции на виртуальных машинах. При автоматизации используются специализированные программные комплексы для эмуляции поведения пользователя, заключающиеся в подготовке скриптов. Исследователи исключают автоматизацию тестирования ПО до авторизации в ОС, связывая с рядом трудностей, по всей видимости, по причине отсутствия тестового ПО, способного запускаться также до авторизации в ОС.

Учёные из Воронежа [10] определяли в своей работе методы тестирования программных средств с учётом двух параметров: скорости и приоритетности, поскольку стремительная скорость разработки ПО множеством конкурирующих компаний приводит к тому, что выпускаемые приложения необходимо быстро тестировать, при этом выбирая определённый функционал в качестве максимально приоритетного. В связи с этим они выделяют три ключевых вида тестирования:

– функциональное;

– тестирование производительности и

– безопасности.

Функциональное тестирование необходимо для валидации и верификации заложенных требований на первом этапе жизненного цикла (ЖЦ) разработки – анализе требований. Тестирование производительности определяет то, насколько быстро, стабильно и надёжно работает приложение. Тестирование безопасности показывает, каков уровень стойкости ПО в вопросах противостояния потенциальным угрозам. В качестве методов тестирования безопасности исследователи рассматривают фаззинг-тестирование.

Тестирование ПО разделяют на несколько этапов:

– этап анализа функциональных и нефункциональных требований, на котором определяются тест-планы, включающие в себя основные тестовые сценарии;

– этап подготовки тестовых сценариев, на котором уже разрабатываются подробные тестовые случаи, которые необходимо проверить и провести подготовку тестового окружения;

– этап тестирования, во время которого собирается результирующая информация с описанием всех найденных и заведённых дефектов.

Коллективом исследователей из Северодвинска [11] рассматривалась модернизированная методика «белого ящика». В работе приводится полноценная поэтапная методика проведения тестирования, состоящая из нескольких этапов (рис. 4).

1. Подготовительный этап (планирование), состоящий из:

– проведения анализа и оценки рисков и угроз;

– разработки стратегии тестирования;

– разработки детального плана тестирования;

– разработки сценариев тестов и определения области покрытия каждого теста;

– подготовки тестовой среды.

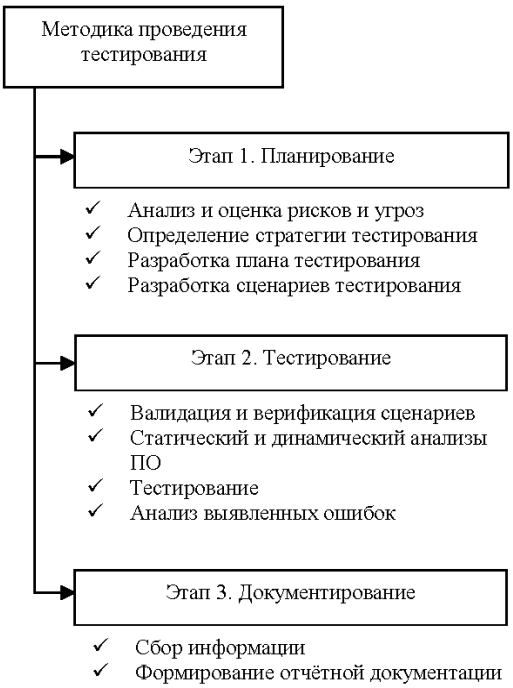

Рис. 4. Методика проведения тестирования программного обеспечения [11]

2. Проведение тестирования, заключающееся в:

– проведении валидации и верификации тестовых сценариев;

– проведении статического анализа ПО и его компонентов;

– проведении динамического анализа ПО и его компонентов;

– проведении тестовых сценариев;

– проведении анализа выявленных ошибок, дефектов и уязвимостей ПО.

3. Этап формирования отчетной документации о результатах проведения исследований.

Ключевой особенностью исследования является применение профилировщика для оценки скорости проведения обработки данных при тестировании. Повышение эффективности процесса возможно за счёт минимизации влияния тестовой среды и совершенствования программных средств для анализа данных.

Иностранный опыт в тестировании

Иностранные исследователи, как и отечественные, рассматривают потенциал применения методов тестирования безопасности как на ранних этапах ЖЦ, так и во время полноценного тестирования и рекомендуют применять комплексный подход.

Коллективом учёных из Польши и Нидерландов рассматривается потенциал модульных тестов во время проверки скомпилированного кода [12]. Они позиционируют такой подход как один из популярных, широко распространённых и перспективных в поиске и устранении уязвимостей в коде. Ключевым вопросом в работе стоят три главные проблемы:

– создание тестовых сценариев (тест-кейсов);

– покрытие тест-кейсами программного кода и

– обслуживание и поддержание тест-кейсов в актуальном состоянии.

В работе предлагается инструмент, позволяющий автоматически генерировать тестовые сценарии и выбирать наиболее подходящие в соответствии с запросом. Прототип такого инструмента синхронизируется с модулем Parasoft C/C++test, который используется для поиска дефектов и мониторинга соответствия со стандартами безопасности MISRA, AUTOSAR или CERT.

Генерацией тест-кейсов интересовались и авторы работы из Индии [13] с целью применения такого подхода для тестирования критического пути, ориентированного на проверку и изучение функционала, который используется большинством пользователей, т.е. в условиях поведения реального пользователя. В таких условиях генерация количества тестовых данных может оказаться эффективным способом определения проблем и дефектов.

В работе представлено усовершенствование метода тестирования критического пути, основанного на автоматическом создании большого количества тестовых данных и их дальнейшем сравнении. Автоматическая генерация тестовых сценариев позволяет охватить не только типичные и стандартные случаи, но и рассмотреть гибридные и нестандартные ситуации, что и позволяет получить 100% покрытие.

Группа учёных из Италии [14] предлагает использовать комплексный подход к обеспечению безопасности ПО и применении тестирования на нескольких этапах ЖЦ разработки: во время написания кода, его компиляции и сборки и на этапе полноценного тестирования.

На этапе написания кода обязательно осуществляется проверка кода на наличие потенциальных угроз безопасности по соответствующим базам данных и каталогам и проводится сравнение с эталонной моделью безопасности.

На этапе сборки создаётся репозиторий, доступный нескольким разработчикам, задействованным в проекте, и запускается серия сквозных модульных тестов, которые позволяют выявить потенциальные проблемы и оперативно их устранить.

В качестве методов тестирования безопасности уже на этапе готовых и функционирующих сборок учёные предлагают использовать тестирование на проникновение (пентест) в комплексе с технологией

DAST (Dynamic Application Security Testing). Такой подход обусловлен тем, что оба метода по отдельности имеют существенные недостатки, а в режиме симбиоза дают отличный результат. С одной стороны, метод DAST не способен всесторонне проверить приложение, однако лёгок для применения самими разработчиками. Такой метод зачастую выбирается компаниями с малым бюджетом, где весьма проблематично организовать отдельную группу сотрудников для тестирования. А с другой, пентест достаточно дорогостоящий и трудоёмкий, и он обязует привлекать к тестированию экспертов по безопасности, но позволяет найти серьёзные уязвимости и не допустить существенных финансовых потерь.

Среди всех методов тестирования безопасности, применяемых на этапе тестирования готового программного продукта, наиболее часто используется фаззинг – один из передовых методов тестирования безопасности, обладающий широким спектром возможностей [15–17], что позволяет современным исследователям использовать его даже для тестирования нейросетей [18]. Преимуществом такого метода является то, что он позволяет детектировать ошибки в ПО с использованием подачи на вход разнородных неожиданных данных. Применяются разные варианты данного метода (рис. 5), классифицирующиеся по способу подготовки данных для тестирования:

– с формированием специальных уже готовых файлов с заданными граничными значениями;

– с автоматическим применением случайных данных;

– с самостоятельным ручным вводом произвольных данных;

– метод грубой силы – с формированием и передачей тестировщиком специальных искажённых данных в каждом пакете;

– автоматизированный вариант метода грубой силы с необходимым исследованием для правильной подготовки спецификации протокола и файла.

Рис. 5. Виды фаззинг-тестирования [18]

Для проведения фаззинг-тестирования используют специализированные приложения и утилиты, занимающиеся доставкой произвольных данных – фаззеры, различающиеся по своим характеристикам:

– локальные фаззеры, включающие в себя фаззеры командной строки, переменной среды, а также фаззеры формата файла;

– фаззеры удалённого доступа, включающие в себя фаззеры сетевых протоколов (простые и сложные), а также фаззеры веб-приложений и веб-браузеров;

– фаззеры оперативной памяти;

– интегрированные среды фаззеров, представляющие собой уже готовые библиотеки фаззеров для определённых задач [19, 20].

Заключение

На сегодняшний день в производстве программных средств применяется множество мер по обеспечению их безопасности, направленных на поиск уязвимостей и угроз. Каждый из них имеет свои преимущества и недостатки, однако современные исследования учёных из Российской Федерации и других стран показывают, что работы в направлении устранения недостатков и повышения эффективности известных мер по обеспечению безопасности активно ведутся. При этом наблюдаются как модификации самостоятельных методов, так и применение нескольких методов тестирования в симбиозе.

На основе опыта исследователей можно судить о том, что наибольшей эффективности удаётся добиться применением не какого-то одного средства, а только путём применения комплекса мер, охватив каждый этап жизненного цикла разработки безопасного ПО. Такой подход на первый взгляд может показаться избыточным и трудозатратным, однако он позволяет обнаружить ряд дефектов, которые были упущены на предыдущих этапах разработки.

Применение комплексного подхода также позволит сократить количество поздно обнаруженных дефектов LDB (Late Detected Bugs), которые могут оказаться как малозначимыми, так и критическими и принести вред не только конечному потребителю, но и привести к репутационным и экономическим потерям компании.

Литература

1. Барабанов А.В. Семь безопасных информационных технологий / А.В. Барабанов, А.В. Дорофеев, А.С. Марков, В.Л. Цирлов. – М.: ДМК Пресс, 2017. – 224 с.

2. Daud M.I. Secure Software Development Model: A Guide for Secure Software Life Cycle // Proceedings of the International MultiConference of Engineers and Computer Scientists. – 2010. – Vol. 1. – P. 1–5.

3. Trautsch F. Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects / F. Trautsch, S. Herbold, J. Grabowski // The Journal of Systems and Software. – 2020. – Vol. 159. – P. 1–15.

4. Kshirasagar N. Software testing and quality assurance. Theory and practice / N. Kshirasagar, T. Priyadarshi. – A John Wiley & Sons, Inc., Publication, 2008. – P. 616.

5. Herbold S. Smoke testing for machine learning: simple tests to discover severe bugs / S. Herbold, T. Haar // Empirical Software Engineering. – 2022. – Vol. 27. – P. 1–30.

6. Марков А.С. Тестирование и испытания программного обеспечения по требованиям безопасности информации / А.С. Марков, В.Л. Цирлов, И.А. Олексенко, В.Г. Маслов // Известия Института инженерной физики. – 2009. – № 2. – С. 2–6.

7. Мукминов В.А. О новых методах и алгоритмах тестирования программного обеспечения / В.А. Мукминов, Ю.В. Войнов, А.В. Баландин // Двойные технологии. – 2011. – № 2. – С. 22–25.

8. Ляпустин А.Е. Способы испытания средств защиты информации / А.Е. Ляпустин, М.Е. Ляпустин // Евразийский научный журнал. – 2015. – № 6. – С. 193–196.

9. Борисова Т.М. Тестирование средств защиты информации / Т.М. Борисова, А.В. Кузнецов, А.И. Обломова // Информационное противодействие угрозам терроризма. – 2013. – № 21. – С. 59–67.

10. Бугаева А.А. Процесс тестирования, методы и типы тестирования программного обеспечения / А.А. Бугаева, В.В. Денисенко // Синергия наук. – 2022. – № 72. – С. 92–102.

11. Бедердинова О.И. Совершенствование метода тестирования программного обеспечения «Белый ящик» / О.И. Бедердинова, Л.А. Иванова // Вестник Северного (Арктического) фед. ун-та. – 2014. – № 2. – С. 113–123.

12. Zielinski M. Using Advanced Code Analysis for Boosting Unit Test Creation / M. Zielinski, R. Groenboom // IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). – 2021. – Vol. 1. – P. 1–5.

13. Mishra D.B. Test Case Generation and Optimization for Critical Path Testing Using Genetic Algorithm / D.B. Mishra, R. Mishra, K.N. Das, A.A. Acharya // Soft Computing for Problem Solving. – 2022. – Vol. 1. – P. 67–80.

14. Casola V. Secure software development and testing: A model-based methodology / V. Casola, A. Benedictis, C. Mazzocca, V. Orbinato // Computers & Security. – 2024. – Vol. 137. – P. 1–16.

15. Luo D. Grammar-based fuzz testing for microprocessor RTL design / D. Luo, T. Li, L. Chen, H. Zou, M. Shi // Integration. – 2022. – P. 64–73.

16. Park L.H. Mixed and constrained input mutation for effective fuzzing of deep learning systems / L.H. Park, J. Kim, J. Park, T. Kwon // Information Sciences. – 2022. – Vol. 614. – P. 497–517.

17. Zhao X. AMSFuzz: An adaptive mutation schedule for fuzzing / X. Zhao, H. Qu, J. Xu, Sh. Li, G.-G. Wang // Expert Systems with Applications. – 2022. – Vol. 208. – P. 118162.

18. Tao Ch. DLRegion: Coverage-guided fuzz testing of deep neural networks with region-based neuron selection strategies / Ch. Tao, Y. Tao, H. Guo, Zh. Huang, X. Sun // Information and Software Technology. – 2023. – Vol. 162. – P. 107266.

19. Chen Ch. A systematic review of fuzzing techniques / Ch. Chen, B. Cui, J. Ma, R. Wu, J. Guo // Computers & Security. – 2018. – Vol. 75. – P. 118–137.

20. Godefroid P. Fuzzing: Hack, Art and Science // Communications of the ACM. – 2020. – Vol. 63. – P. 70–76.

Деркач Евгений Александрович
Аспирант каф. комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)
Томского университета систем управления и радиоэлектроники (ТУСУР)
Ленина пр-т, 40, г. Томск, Россия, 634050
Тел.: +7-913-887-87-87
Эл. почта: ead@fb.tusur.ru

Шелупанов Александр Александрович
Д-р техн. наук, проф., президент ТУСУРа
Ленина пр-т, 40, г. Томск, Россия, 634050
ORCID: 0000-0003-2393-6701
Тел.: +7 (382-2) 90-71-55
Эл. почта: saa@fb.tusur.ru

Derkach E.A., Shelupanov A.A.

Domestic and foreign experience in security testing

Testing is considered as a separate and independent life cycle stage of any secure software development. However, the testing is present at another stages, such as development stage or the technical support stage after software release. This paper describes the domestic and foreign experience in security testing which is considered both at specific life cycle stage of secure software development and as a whole. Each method has its own benefits and drawbacks. The experience of researchers will be useful in own methodology for software security testing.

Keywords: testing, security testing, testing methods, fuzzing testing.
DOI: 10.21293/1818-0442-2024-27-1-37-43

References

1. Babarinov A.V., Dorofeev A.V., Markov A.S., Tzirlov V.L. *Sem' bezopasnyh informacionnyh tekhnologij* [Seven secure information technologies]. Moscow, DMK Press, 2017, 224 p. (in Russ.).

2. Daud M.I. Secure Software Development Model: A Guide for Secure Software Life Cycle. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2010, vol. 1, pp. 1–5.

3. Trautsch F., Herbold S., Grabowski J. Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects *Journal of Systems and Software*, 2020, vol. 159, pp. 1–15.

4. Kshirasagar N., Priyadarshi T. Software testing and quality assurance. Theory and practice. A John Wiley & Sons, Inc. Publication, 2008, 616 p. (in Eng.).

5. Herbold S., Haar T. Smoke testing for machine learning: simple tests to discover severe bugs *Empirical Software Engineering*, 2022, vol. 27, pp. 1–30.

6. Markov A.S., Tzirlov V.L., Oleksenko I.A., Maslov V.G. *Testirovanie i ispytaniya programmnogo obespecheniya po trebovaniyam bezopasnosti informacii* [Software testing according to information security requirements] *Izvestiya Instituta inzhenernoy phiziki*, 2009, no. 2, pp. 2–6 (in Russ.).

7. Mukminov V.A., Voinov Y.V., Balandin A.V. *O novyh metodah i algoritmah testirovaniya programmnogo obespecheniya* [About new methods and algorithms for software testing] *Dvoinye Tekhnologii*, 2011, no. 2, pp. 22–25 (in Russ.).

8. Lyapustin A.E., Lyapustin M.E. *Sposoby ispytaniya sredstv zashchity informacii* [Methods for testing information security tools] *Evrazijskij nauchnyj zhurnal*, 2015, no. 6, pp. 193–196 (in Russ.).

9. Borisova T.M., Kuznetsov A.V., Oblomova A.I. *Testirovanie sredstv zashchity informacii* [Testing information security tools] *Informacionnoe protivodejstvie ugrozam terrorizma*, 2013, no. 21, pp. 59–67 (in Russ.).

10. Bugaeva A.A., Denisenko V.V. *Process testirovaniya, metody i tipy testirovaniya programmnogo obespecheniya* [Testing process, methods and types of software testing] *Sinergiya Nauk*, 2022, no. 72, pp. 92–102 (in Russ.).

11. Bederdinova O.I., Ivanova L.A. *Sovershenstvovanie metoda testirovaniya programmnogo obespecheniya «Belyj yashchik»* [Improving the White Box Software Testing Method] *Vestnik Severnogo (Arkticheskogo) federalnogo universiteta*, 2014. no. 2, pp. 113–123 (in Russ.).

12. Zielinski M., Groenboom R. Using Advanced Code Analysis for Boosting Unit Test Creation *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2021, vol. 1, pp. 1–5.

13. Mishra D.B., Mishra R., Das K.N., Acharya A.A. Test Case Generation and Optimization for Critical Path Testing Using Genetic Algorithm *Soft Computing for Problem Solving*, 2022, vol. 1, pp. 67–80.

14. Casola V., Benedictis A., Mazzocca C., Orbinato V. Secure software development and testing: A model-based methodology, *Computers & Security*, 2024, vol. 137, p. 1–16.

15. Luo D., Li T., Chen L., Zou H., Shi M. Grammar-based fuzz testing for microprocessor RTL design *Integration*, 2022, pp. 64–73.

16. Park L.H., Kim J., Park J., Kwon T. Mixed and constrained input mutation for effective fuzzing of deep learning systems *Information Sciences*, 2022, vol. 614, pp. 497–517.

17. Zhao X., Qu H., Xu J., Li Sh., Wang G.-G. AMSFuzz: An adaptive mutation schedule for fuzzing *Expert Systems with Applications*, 2022, vol. 208, p. 118162.

18. Tao Ch., Tao Y., Guo H., Huang Zh., Sun X. DLRegion: Coverage-guided fuzz testing of deep neural networks with region-based neuron selection strategies // *Information and Software Technology*, 2023, vol. 162, p. 107266.

19. Chen Ch., Cui B., Ma J., Wu R., Guo J. A systematic review of fuzzing techniques, *Computers & Security*, 2018, vol. 75, pp. 118–137.

20. Godefroid P. Fuzzing: Hack, Art and Science *Communications of the ACM*, 2020, vol. 63, pp. 70–76.

Evgeny A. Derkach
Postgraduate student, Department of Complex Information Security of Computer Systems, Tomsk State University of Control Systems and Radioelectronics (TUSUR)
40, Lenin pr., Tomsk, Russia, 634050
Phone: +7-913-887-87-87
Email: ead@fb.tusur.ru

Alexander A. Shelupanov
Doctor of Science in Engineering, Professor, President of TUSUR
40, Lenin pr., Tomsk, Russia, 634050
ORCID: 0000-0003-2393-6701
Phone: +7 (382-2) 90-71-55
Email: saa@fb.tusur.ru