Тестирования безопасности приложений

В.А. Фатхи, Н. В. Дьяченко

Донской государственный технический университет Ростов-на-Дону

Аннотация: Рассматривается тестирование безопасных приложений, с использованием различных методик и предложением графорасчетной модели тестограмм. С использованием данного метода можно по результатам моделирования найти значимые места для работы всей системы в целом. Также использование тестограмм во время эксплуатации может указать на узел или узлы с ошибочными значениями, вследствие чего можно проанализировать влияние одного значения на другие.

Ключевые слова: тестирование безопасности приложений, тестограммы, тестовые деревья.

Введение

На сегодняшний день можно уверенно говорить о том, что веб-технологии прочно вошли в жизнь любого бизнеса, являясь наиболее удобным способом предоставления информации: коммерческие и государственные структуры весьма активно предоставляют свои услуги, используя публичные и частные сети для всех видов пользователей. Преимущества такого подхода очевидны - улучшаются наиболее критичные показатели бизнес-процессов: производительность, оперативность, доступность, стоимость и т.п.

Основная часть исследования

Тестирование безопасности приложений (AST) - это процесс повышения устойчивости приложений к угрозам безопасности путем выявления слабых мест и уязвимостей в исходном коде.

АСТ начинался как ручной процесс. Сегодня, в связи с растущей модульностью корпоративного программного обеспечения, огромным количеством компонентов с открытым исходным кодом и большим количеством известных уязвимостей и векторов угроз, АСТ должна быть автоматизирована. Большинство организаций используют комбинацию нескольких средств защиты приложений [1, 2].

Инструменты SAST используют метод тестирования белого ящика, при котором тестировщики проверяют внутреннюю работу приложения. SAST проверяет статический исходный код и сообщает о слабых сторонах безопасности.

Инструменты статического тестирования могут быть применены к некомпилированному коду для поиска таких проблем, как синтаксические ошибки, математические ошибки, проблемы проверки ввода, недопустимые или небезопасные ссылки. Они также могут работать на скомпилированном коде с использованием анализаторов двоичного и байтового кода.

Динамическое тестирование безопасности приложений (DAST). Инструменты DAST используют подход тестирования черного ящика. Они выполняют код и проверяют его во время выполнения, обнаруживая проблемы, которые могут представлять собой уязвимости в системе безопасности. Это может включать проблемы со строками запросов, запросами и ответами, использованием скриптов, утечкой памяти, обработкой файлов cookie и сеансов, аутентификацией, выполнением сторонних компонентов, инъекцией данных и инъекцией DOM.

Инструменты DAST можно использовать для проведения крупномасштабных сканирований, имитирующих большое количество неожиданных или вредоносных тестовых случаев, и отчетов об ответе приложения.

Инструменты IAST — это эволюция инструментов SAST и DAST, сочетающих два подхода для обнаружения более широкого спектра слабых мест в системе безопасности. Как и инструменты DAST, инструменты IAST работают динамически и проверяют программное обеспечение во время выполнения. Однако они запускаются из сервера приложений, что позволяет им проверять скомпилированный исходный код, как это делают инструменты IAST.

Инструменты IAST могут предоставить ценную информацию о первопричине уязвимостей и конкретных строках кода, которые затронуты, что значительно упрощает исправление. Они могут анализировать исходный код, поток данных, конфигурацию и сторонние библиотеки и подходят для тестирования API.

Тестирование безопасности мобильных приложений.

Инструменты MAST объединяют статический анализ, динамический анализ и исследование криминалистических данных, генерируемых мобильными приложениями. Они могут тестировать уязвимости безопасности, такие как SAST, DAST и IAST, а также решать специфические для мобильных устройств проблемы, такие как джейлбрейк, вредоносные сети Wi-Fi и утечка данных с мобильных устройств.

Анализ состава программного обеспечения (SCA)

Инструменты SCA помогают организациям проводить инвентаризацию сторонних коммерческих компонентов и компонентов с открытым исходным кодом, используемых в их программном обеспечении. Корпоративные приложения могут использовать тысячи сторонних компонентов, которые могут содержать уязвимости безопасности. SCA помогает понять, какие компоненты и версии фактически используются, выявить наиболее серьезные уязвимости безопасности, влияющие на эти компоненты, и понять самый простой способ их устранения.

Самозащита приложения времени выполнения (RASP)

Инструменты рашпиля эволюционировали от SAST, DAST и IAST. Они способны анализировать трафик приложений и поведение пользователей во время выполнения, обнаруживать и предотвращать киберугрозы.

Как и предыдущее поколение инструментов, RASP обладает видимостью исходного кода приложения и может анализировать слабые и уязвимые места. Он делает еще один шаг вперед, определяя, что были

использованы слабые места в системе безопасности, и обеспечивая активную защиту, завершая сеанс или выдавая предупреждение.

Инструменты RASP интегрируются с приложениями и анализируют трафик во время выполнения, а также могут не только обнаруживать и предупреждать об уязвимостях, но и фактически предотвращать атаки. Наличие такого типа углубленной проверки и защиты во время выполнения делает SAST, DAST и IAST гораздо менее важными, позволяя обнаруживать и предотвращать проблемы безопасности без дорогостоящих работ по разработке.

Новые организационные практики, такие как DevSecOps, подчеркивают необходимость интеграции безопасности на каждом этапе жизненного цикла разработки программного обеспечения. Инструменты AST могут:

- помочь разработчикам понять проблемы безопасности и внедрить рекомендации по обеспечению безопасности на этапе разработки.

- помочь тестировщикам выявить проблемы безопасности на ранней стадии, прежде чем программное обеспечение будет отправлено в производство.

Продвинутые инструменты, такие как RASP, могут выявлять и блокировать уязвимости в исходном коде в процессе производства.

Тестируйте внутренние интерфейсы, а не только API и UIs.

Естественно, что тестирование безопасности приложений фокусируется на внешних угрозах, таких как вводимые пользователем данные через веб-формы или публичные запросы API. Однако еще чаще злоумышленники используют слабую аутентификацию или уязвимости во внутренних системах, оказавшись уже внутри периметра безопасности. ACT следует использовать для проверки надежности входных сигналов, соединений и интеграций между внутренними системами.

Каждый день обнаруживаются новые уязвимости, и корпоративные приложения используют тысячи компонентов, любой из которых может выйти из строя или потребовать обновления системы безопасности. Крайне важно как можно чаще тестировать критические системы, расставлять приоритеты по вопросам, связанным с критическими для бизнеса системами и высокоэффективными угрозами, и выделять ресурсы для быстрого их устранения.

Безопасность кода третьей стороны

Организации должны применять практику AST к любому стороннему коду, который они используют в своих приложениях. Никогда не верьте, что компонент от третьей стороны, будь то коммерческий или с открытым исходным кодом, является безопасным. Сканируйте сторонний код так же, как вы сканируете свой собственный. Если вы обнаружите серьезные проблемы, применяйте исправления, консультируйтесь с поставщиками, создавайте свои собственные исправления или рассматривайте возможность переключения компонентов.

Кроме того, стоит обеспечивать многоуровневую защиту для обеспечения легкодоступности и безопасности веб-сайтов и приложений. Эти решения для обеспечения безопасности приложений включают в себя:

- защиту от DDoS-атак-поддержание безотказной работы во всех ситуациях. Предотвратите любой тип DDoS-атаки любого размера, чтобы предотвратить доступ к вашему сайту и сетевой инфраструктуре.

CDN - повышение производительности веб-сайта и сокращение затраты на пропускную способность с помощью CDN, разработанного для разработчиков. Кэширование статических ресурсов на границе при одновременном ускорении API и динамических веб-сайтов.

Cloud WAF - разрешение легального трафика и предотвращение плохого трафика. Защитите свои приложения на краю с помощью облачного WAF корпоративного класса.

Gateway WAF- сохраняет приложения и API внутри вашей сети.

Аналитика атак-смягчение и реагирование на реальные угрозы безопасности эффективно и точно с помощью действенной разведки на всех уровнях вашей защиты [3-5].

Защита от захвата учетных записей —использует процесс обнаружения намерений для идентификации и защиты от попыток захвата учетных записей пользователей в вредоносных целях.

Безопасность API-защищает API, гарантируя, что только желаемый трафик может получить доступ к вашей конечной точке API, а также обнаруживая и блокируя эксплойты уязвимостей.

Расширенная защита ботов-анализирует ваш трафик ботов, чтобы точно определить аномалии, выявляет плохое поведение ботов и проверяет его с помощью механизмов вызова, которые не влияют на трафик пользователей.

Пусть задана некоторая гипотетическая структура, обладающая двумя управляющими входными, тремя исполнительскими выходами и определённой структурой, отображающей движение и логическое преобразование командной информации. Векторная диаграмма исследуемой структуры приведена на рисунке 1.

Входы x_1, x_2 – и управляющие. Выходы y_1, y_2, y_3 – исполнительские. Узел 5 дизъюнктивного типа «УДТ», а узел 6 – конъюнктивного типа (УКТ). Остальные операторы графа одновходовые, а поэтому выполняют лишь функции преобразования входной информации в её выходную форму.

Будем предполагать, что в структуре может отсутствовать нарушитель (назовем это нулевым состоянием или присутствием, но не более, чем один (таких состояний будет 8)).

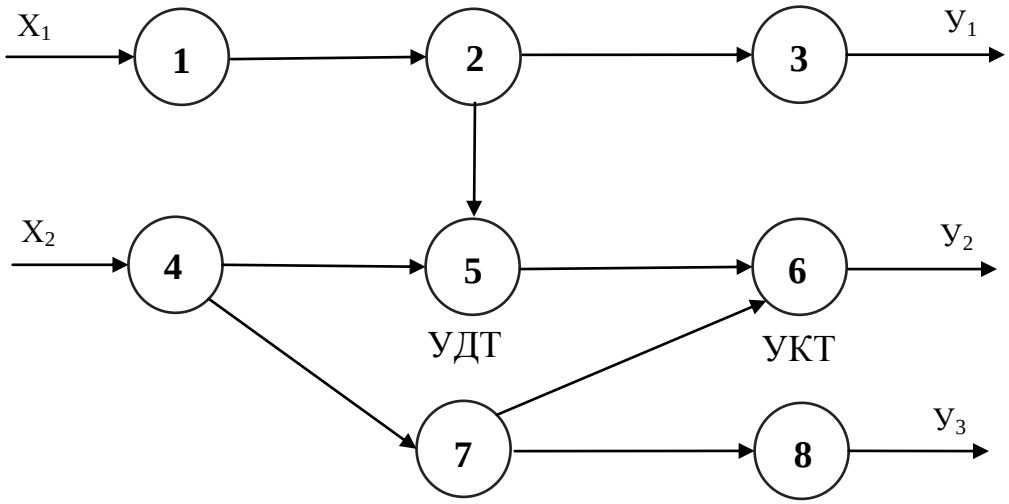

Рис 1 - Векторная диаграмма исследуемой структуры

Синдромы состояний системы можно отобразить наличием (обозначим это 1) или отсутствием (обозначим это 0) командной информации на исполнительских выводах при различных комбинациях управляющих команд на выходах (1 – наличие команды, 0 – отсутствие таковой). Таблица синдромов приобретает вид:

Таблица № 1

Синдромы состояний системы

Состояния входов x_1, x_2	Нарушитель (номер в структуре)								
	0	1	2	3	4	5	6	7	8
00	000	100	100	100	000	000	010	001	001
01	011	111	111	111	000	001	001	000	010
10	100	000	000	000	111	100	110	101	101
11	111	011	011	011	100	101	101	100	110

По таблицам синдромов могут быть построены тестовые последовательности для определения места нарушителя (Рисунок 2). Длина тестовой последовательности может быть разной, а в некоторых случаях она может состоять даже из одного шага. Так, нарушитель с номером 6 обнаруживается сразу же при отсутствии входных воздействий и появлении на выходе y_3 несанкционированного сигнала.

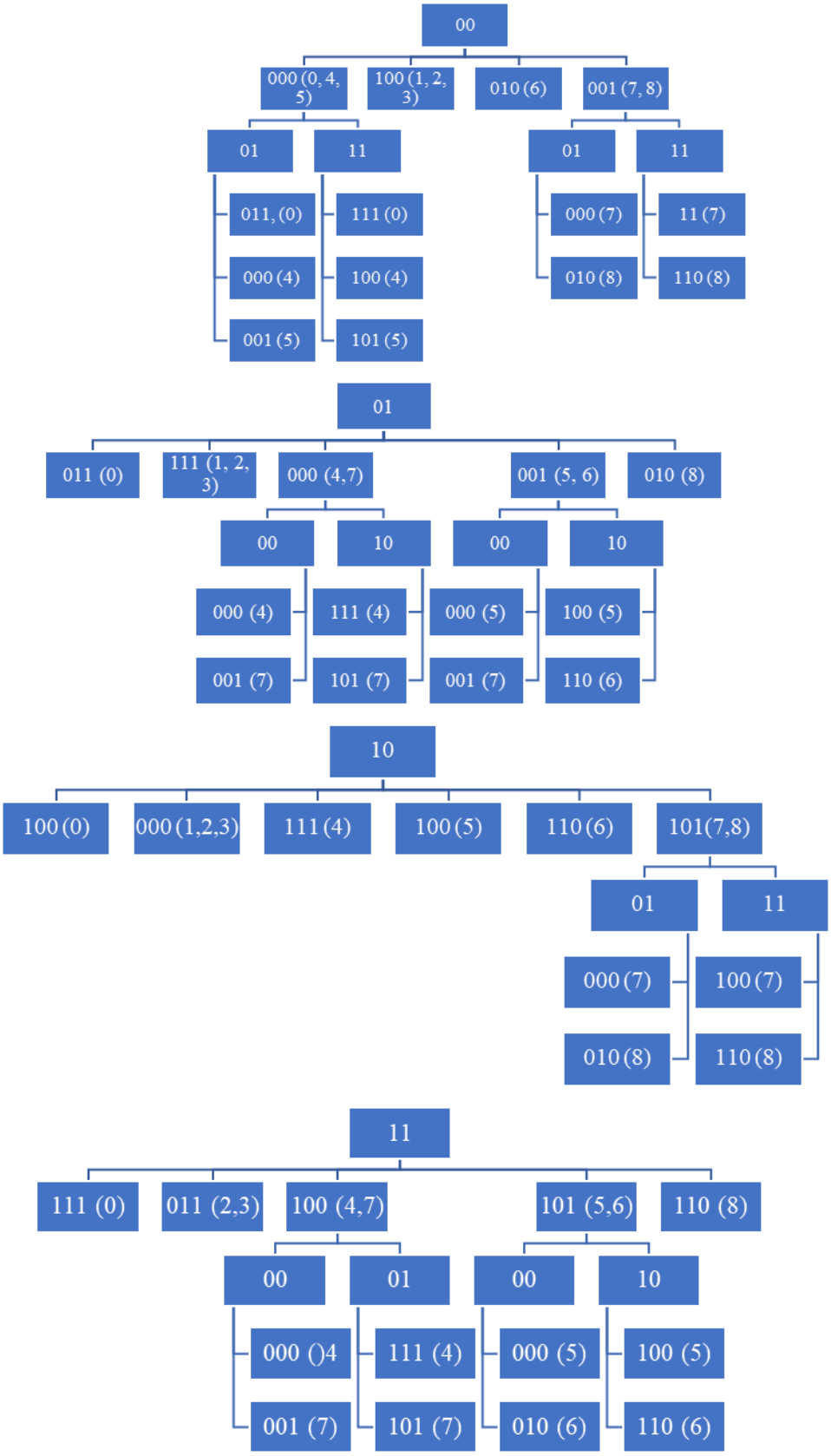

Рис 2 – Тестовые деревья

Считается, что возможна минимизация тестовых проверок, в которых по определенным правилам изменяются состояния входов, а анализируются состояния выходов. Описанная модель является детерминированной, но возможен и вероятностный подход [6-8].

По результатам анализа синдромов можно предположить возможность решения нескольких задач. Первая связана с минимизацией количества тестовых проверок для выявления места нарушения, то есть в процессе тестирования имеется возможность подавать на выходы любые возможные комбинации и в любой последовательности для рассматриваемого примера комбинация входов «01» выявляет четыре возможных места нарушения за один шаг, с другой комбинацией «00» за один шаг выявляется результат «0» (нарушений нет) и в восьмом разряде. Другая задача может состоять в том, чтобы за единственную тестовую комбинацию входов получить информацию о наличии или нарушении в одном из максимального множества узлов, такой комбинацией является «10», с помощью которой получается информация о возможных нарушениях в пяти узлах. Еще одной задачей испытаний может быть такая, которая выявляет нарушения без специальных входных воздействий на систему; в рассмотренном примере такая задача позволяет проверить лишь шестой узел «00». Если стоит задача минимальных изменений входных воздействий для контроля конкретного узла, то такая задача решается и ее итог будет зависеть от того, какой контрольный узел подлежит проверке. В рассмотренном примере для проверки узлов шесть и восемь достаточно изменения тестовых значений лишь на одном входе из двух, а для проверки четвертого и седьмого также достаточно изменения тестовой информации на одном, но уже другом входе [9-11]. Если необходимо убедиться в том, что решение задачи достоверно для какого-либо узла, проверки можно проводить так, чтобы исключить знак, или в конечном результате в одном или нескольких испытаниях. В рассмотренном

примере комбинация «01» с последующей входной комбинацией «00» выделяется в качестве возможного места нарушений четвертого и седьмого узла, а также соответствует состоянию без нарушений. При проведении тестовых комбинаций «00» и «11», из множества ноль, четыре и семь выделяется семь, если на входе будет «000», для выделения нулевого или четвертого можно воспользоваться тестовой комбинацией «11» при получении на входе «011», чье состояние соответствует исправному нулевому, «100» - нарушение в четвертом узле. Такая задача не является однозначной. Возможны и другие комбинаторные задачи, позволяющие повышать разрешающую способность теста или достоверность окончательного решения.

В случае, когда нарушения носят случайный характер, задача приобретает вероятностный смысл и ее решения возможны лишь с применением соответствующих стохастических подходов [5- 6].

Литература

1 Евсин В.А., Тихонов Н.А., Воробьев С.П. Разработка модуля оптимального размещения информационных ресурсов на узлах вычислительной сети: описание реализуемых методов и структур данных // Инженерный вестник Дона, 2019, №1. URL:ivdon.ru/ru/magazine/archive/n1y2019/5493

2 Берёза Н. В. Современные тенденции развития мирового и российского рынка информационных услуг // Инженерный вестник Дона, 2012, №2. URL: ivdon.ru/magazine/archive/n2y2012/758

3 Зотов А.И., Ганжур М.А., Авакьянц А.В., Характеристика управленческой структуры и системы прохождения команд, Проблемы современного педагогического образования. 2018. № 58-3. С. 111-116

4 Змитрович А.И. Интеллектуальные информационные системы. Мн.: НТООО «ТетраСистемс», 1997. С. 368.

5 Фатхи В.А., Фатхи Дм.В., Фатхи Д.В. Функция достижимости и сетевая производная нечеткой сети Петри на основе μ-значной логики // Математические методы в технике и технологиях – ММТТ-19: сб.трудов XIX Междунар. науч. конф.: в 10-и т. Т.6/ под общ. ред. В.С. Балакирева. – Воронеж, Воронеж. гос. технол. акад., 2006. С. 33-37.

6 Ganzhur M.A., Ganzhur A.P., Smirnova O.V. Modeling of critical systems implementing negative events using dual Petri nets. MATEC Web of Conferences Volume 226 (2018), XIV International Scientific-Technical Conference “Dynamic of Technical Systems” (DTS-2018). URL:doi.org/10.1051/matecconf/201822604001

7 Marković N., Živanić J., Lazarević Z., Iričanin B. The Mathematical Model for Analysis and Evaluation of the Transient Process of the three–phase Asynchronous Machine Performance. Serbian journal of electrical engineering (DTS-2018). URL: journal.ftn.kg.ac.rs/Vol_15-3/05-Markovic-Zivanic-LazarevicIricanin.pd

8 Гинис Л.А. Развитие инструментария когнитивного моделирования для исследования сложных систем // Инженерный вестник Дона, 2013, №3. URL: ivdon.ru/ru/magazine/archive/n3y2013/1806

9 Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И.В. Романовского. — СПб. Невский Диалект; БХВ-Петербург, 2003. — 654 с.

10 Поликарпова Н.И., Шалыто А.А. Автоматное программирование. - СПб. СПбГПУ, 2008. - 227с.

11 Андреев А.А. Методика выбора базовой архитектуры реконфигурируемой вычислительной системы на основе методов теоретико-игровой оптимизации // Инженерный вестник Дона, 2013, №1. URL: ivdon.ru/magazine/archive/n1y2013/1569

References

1 Evsin V.A., Tixonov N.A., Vorob`ev S.P., Inzhenernyj vestnik Dona, 2019, №1. URL:ivdon.ru/ru/magazine/archive/n1y2019/5493

2 Beryoza N. V., Inzhenernyj vestnik Dona, 2012, №2. URL: ivdon.ru/magazine/archive/n2y2012/758

3 Zotov A.I., Ganzhur M.A., Avak`yancz A.V., Problemy` sovremennogo pedagogicheskogo obrazovaniya. 2018. № 58-3, p. 111-116

4 Zmitrovich A.I. Intellektual'nye informacionnye sistemy [Intelligent information systems]. M: NTOOO «TetraSistems», 1997, p. 368.

5 Fatxi V.A., Fatxi Dm.V., Fatxi D.V., MMTT-19: sb. trudov XIX Mezhdunar. nauch. konf.: v 10-i t. T.6 pod obshh. red. V.S.Balakireva. Voronezh, Voronezh. gos. texnol. akad., 2006, p. 244.

6 Ganzhur M.A., Ganzhur A.P., Smirnova O.V., MATEC Web of Conferences Volume 226 (2018), XIV International Scientific-Technical Conference “Dynamic of Technical Systems” (DTS-2018). URL:doi.org/10.1051/matecconf/201822604001

7 Marković N., Živanić J., Lazarević Z., Iričanin B., Serbian journal of electrical engineering (DTS-2018). URL: journal.ftn.kg.ac.rs/Vol_15-3/05 Markovic Zivanic LazarevicIricanin.pd

8 Ginis L.A., Inzhenernyj vestnik Dona, 2013, №3. URL: ivdon.ru/ru/magazine/archive/n3y2013/1806

9 Gasfild D. Stroki, derev'ya i posledovatel'nosti v algoritmah: Informatika i vychislitel'naya biologiya [Strings, Trees, and Sequences in Algorithms: Computer Science and Computational Biology]. SPb.: Nevskij Dialekt; BXV-Peterburg, 2003, p. 654.

10 Polikarpova N.I., Shaly`to A.A. Avtomatnoe programmirovanie [Automatic programming]. SPb.: SPbGPU, 2008, p. 227.

11 Andreev A.A. Inzhenernyj vestnik Dona, 2013, №1 URL: ivdon.ru/magazine/archive/n1y2013/1569