УДК 004.056.53

*В. Нитья, С. Сентилкумар*

# ОБНАРУЖЕНИЕ И ПРЕДОТВРАЩЕНИЕ АТАК С ПРОВЕРКОЙ ВВОДА В ВЕБ-ПРИЛОЖЕНИЯХ С ИСПОЛЬЗОВАНИЕМ ДЕТЕРМИНИРОВАННЫХ АВТОМАТОВ С МАГАЗИННОЙ ПАМЯТЬЮ

**Ключевые слова:** атака с проверкой кода, статический и динамический анализ, обнаружение, предотвращение, детерминированные автоматы с магазинной памятью.

## Введение

С расширением рамок использования Интернета был разработан новый класс функционирующих в сети систем информационных технологий — веб-приложения, которые обеспечивают связь, доступ к информации и онлайн-услуги, что позволяет компаниям реализовать свои возможности. Следовательно, многие из них начали использовать веб-приложения, подключив свои системы к сети. С появлением веб-приложений появились новые возможности ведения бизнеса — бронирование как железнодорожных, так и авиабилетов, приобретение любого продукта, кроме того, телефонные счета можно оплатить всего одним щелчком мыши, без посещения соответствующего офиса, что экономит время и усилия. Каждая организация старается представить свой бизнес на мировом уровене с помощью веб-приложений.

Со времен отображения простой страницы и появления современных многофункциональных сайтов веб-приложения претерпели изменения — они обрабатывают двусторонний поток данных между клиентом и сервером так эффективно, что заменяют многие настольные приложения. Это важная часть коммерческого, развлекательного и социального взаимодействия. Ранее разработки веб-приложений были реализованы на сервере.

Клиентская часть состояла только из веб-страницы HTML, которая отображалась на веб-браузере. Если приложению требовались некоторые данные от пользователя, обращались к HTML-форме, и данные сразу отправлялись на сервер без какой-либо обработки или проверки.

Сервер веб-приложений получает входные данные от других уровней: пользовательский ввод от веб-браузера и наборы результатов с сервера базы данных. Неспособность надлежащим образом обезвредить входные данные может поставить под угрозу безопасность веб-приложения. В то же время зачастую уязвимость обнаруживается из-за ошибок программирования и отсутствия информации об известных эксплойтах и сценариях атак. Наличие чего-то похожего на безобидную модификацию объектной модели документа (Document Object Model — DOM) может быть достаточно для установления вредоносных программ, что может привести к прямому доступу к системе жертвы. Пока злоумышленники используют все возможные недостатки (дыры) в безопасности, эксперты по безопасности и поставщики услуг постепенно осознают, что крайне важна осведомленность о таких атаках. Например, проект Open Web Application Security Project (OWASP) предоставляет список из десяти наиболее распространенных угроз безопасности в веб-приложениях. Он используется во многих исследованиях по клас-

сификации уязвимостей веб-приложений, и исследователи пришли к консенсусу относительно его обоснованности и эффективности. Поскольку веб-приложение широко используется во всем мире, ошибки в нем могут легко использоваться злоумышленниками. Основная причина взлома веб-приложения — неправильная проверка в приложении при таких ведущих атаках, как SQL-инъекция и межсайтовый скриптинг.

Предлагаемый в работе метод основан на контекстно-свободных языках (context-free languages — CFL) и детерминированных автоматах с магазинной памятью (Deterministic push down automata — DPDA). Класс CFL играет важную роль в нескольких областях информатики. Помимо его определения с использованием контекстно-свободных грамматик, он имеет другие характеристики: наиболее известна та, которая связана с использованием DPDA. Автоматы с магазинной памятью (Pushdown automata — PDA) — это конечные автоматы, дополненные магазинным стеком. PDA может только читать вершину стека и не знать, насколько он большой. Кроме того, он может добавлять или удалять элементы с вершины стека. Автоматы с магазинной памятью естественно моделируют управление потоком последовательных вычислений в типичных языках программирования со вложенными и потенциально рекурсивными вызовами программных модулей, таких как процедуры и обращение к программе методов. Следовательно, разнообразие вопросов анализа программы, оптимизации компилятора и проверки модели можно сформулировать как решение проблем для PDA. Детерминированный автомат с магазинной памятью — это вариант автомата с магазинной памятью. Класс DPDA допускает детерминированный CFL как подходящее подмножество CFL.

В данной работе в разд. 1 даны общие сведения о проверке входных данных и описана связанная с этим работа, сосредоточенная на анализе атак веб-приложений, обнаружений и предотвращений, используемых несколькими известными сканерами уязвимостей, инструментами, и представлены некоторые недостатки этих методов. В разд. 2 предлагаемая модель рассмотрена на базе атак с проверкой кода (Input validation attacks — IVA). В подразделе 2.1 поверхностно представлена атака и чит-код веб-страницы. Детали статического анализа, основанного на детерминированных автоматах с магазинной памятью, изложены в подразделе 2.2. Детали динамического анализа на основе детерминированных автоматов с магазинной памятью описаны в подразделе 2.3. В 2.4 представлен валидационный процесс, выполняемый между статической и динамической недетерминированными структурами автоматов. Этап классификации, цель которого — классификация вредоносных веб-страниц, представлен в 2.5. В разд. 3 предложены эксперименты, выполненные для проверки рассматриваемого подхода и оценки эффективности предложенного метода.

## 1. Предпосылки и связанные с ними работы

Межсайтовый скриптинг (Cross Site Scripting — XSS) и атака SQL-инъекций (SQL Injection Attack — SQLIA ) — два основных вида атак с проверкой ввода (IVA), используемые злоумышленниками для взлома веб-приложения. Основное различие между этими атаками заключается в том, что XSS-атаки используются для перенаправления пользователей на веб-сайты, где злоумышленники могут украсть данные из SQLIA, используемые для кражи информации. Напротив, атака межсайтового скриптинга (XSS) с помощью вредоносного кода перенаправляет пользователей на вредоносные веб-сайты, воровство куки-файлов и порчи веб-сайтов.

Йонг Джун Парк (YongJoon Park), Дже Чул Парк (JaeChul Park ) предложили систему обнаружения вторжений веб-приложений (Web Application Intrusion Detection System — WAIDS), которая эффективно обнаруживает атаки с проверкой кода (2008) [21]. Уильям Дж. Дж. Халфонд (William G.J. Halfond) и Алессандро Орсо (Alessandro Orso) (2008) [5] представили веб-приложение предотвращения SQL-инъекций (SQL Injection Preventer) для исключения внедрения SQL во время выполнения программы. Биштеталь (Bishtetal) предложил метод динамической оценки кандидатов (Dynamic Candidate Evaluations method) для автоматического предотвращения атак SQL-инъекций (CANDID) (2010) [3]. Иньонг Ли (Inyong Lee), Сунки Чжон (Soonki Jeong), Сангсо Йео (Sangsoo Yeo), Джонсаб Мун (Jongsub Moon,) (2011) представили простой метод обнаружения SQL-инъекций путем удаления значений атрибутов SQL-запросов во время выполнения программы и проверки на SQL-запросы во время компиляции [7].

Такеши Мацуда (Takeshi Matsuda) разработал новый алгоритм обнаружения атак межсайтового скриптинга путем извлечения особенностей атаки межсайтовых скриптов с учетом внешнего вида и частоты символов (2012) [15].

A. Razzaq, K. Latif, H.F. Ahmad, A. Hur, Z. Anwar, P.C. Bloods предоставили онтологию на базе подхода, определяющего семантические правила для атак веб-приложений. Этот подход обнаруживает и классифицирует атаки веб-приложений (2013) [1]. Лвин Кхин Шар (Lwin Khin Shar), Хи Бенг Куан (Hee Beng Kuan) разработали метод, основанный на статических атрибутах кода для прогнозирования атак SQLIA и XSS. Успех предложенной системы основан на исторических данных, которые отражают статические атрибуты (2013) [16]. Андерс Моллер (Anders Moller) и Матиас Шварц (Mathias Schwarz) разработали метод мониторинга манипуляций со стороны клиента и предоставления инструментов предупреждения, которые помогают разработчику приложения определить уязвимости перед развертыванием веб-приложения (2014).

Исату Гидара (Isatou Hydara), Абу Бакар Мд Султан (Abu Bakar Md Sultan), Хазура Зулзалил (Hazura Zulzalil), Новия Адмодисастро (Novia Admodisastro) предложили метод на основе OWASP ESAPI Security Guidelines (Руководства по безопасности OWASP ESAPI) для устранения уязвимостей межсайтового скриптинга в веб-приложении (2015) [17]. Сангвук Чо (Sangwook Cho), Гесик Ким (Gyoosik Kim), Сон Чжэ (Seong-je Cho), Жонгму Чой (Jongmoo Choi), Минкю Парк (Minkyu Park), Сангчул Хан (Sangchul Han) предложили метод с использованием статического инструментария байт-кода и ввода во время выполнения проверки для контроля входных значений приложений на основе Java (2016) [18].

Иберия Медейрос (Ibéria Medeiros), Нуно Невес (Nuno Neves), Мигель Коррейа (Miguel Correia) предложили WAP-инструмент (2016) на основе анализа статического кода источника и интеллектуального анализа данных для выявления уязвимостей веб-приложений [6]. Сунг Су Ким (Sung Soo Kim) и другие авторы исследовали и усовершенствовали инструменты обнаружения уязвимостей (Vulnerability Detection Tools — VDT) на основе открытого API, используемого Open VAS, для запрета проблем, связанных с безопасностью (2016) [13].

Васегипана Мехрноуш (Vaseghipanah Mehrnoush), Модири Насер (Modiri Nasser) и Джаббедар Сэм (Jabbehdar Sam) предложил метод обнаружения атак с проверкой кода, используя искусственную нейронную сеть и многослойный персептрон (Multi-Layer Perceptron — MLP network) (2017) [19]. К. Бакаре (K. Bakare), Айени (Ayeni), Б. Джунайду (B. Junaidu), Сахалу (Sahalu), Р. Колаволе (R. Kolawole), Адеянджу (Adeyanju) предложили интеллектуальный инструмент нечеткого вывода для обнаружения XSS-атак в веб-приложении, в предложенных рамках используется нечеткая логика для классификации слабых мест XSS (2018) [20].

**1.1. Обзор ограничений подходов.** Имеющиеся методы эффективно обрабатывали несколько атак веб-приложений. Но одни обеспечивают решение только для SQL-инъекций атак, другие — только для межсайтовый скриптинг атаки. Существующий подход в основном сосредоточен на предсказании атаки, а не на обеспечении полного и простого экономически эффективного решения для обнаружения и устранения атак с проверкой кода, особенно в промышленности и академической науке. Для этого были реализованы автоматические инструменты и системы безопасности, но ни одно из этих средств не является полным или достаточно точным, чтобы гарантировать абсолютный уровень безопасности веб-приложения. Важен механизм, который легко развертывается и обеспечивает хорошую производительность для обнаружения и предотвращения атаки с проверкой кода IVA. Таким образом, предложен новый подход, использующий детерминированные магазинные автоматы, основанные на статическом и динамическом анализе.

## 2. Обзор предлагаемой конструкции

Представим среду обнаружения и предотвращения IVA для исправления недостатков проверки входных данных в приложении с использованием детерминированных автоматов с магазинной памятью. Чтобы разработать решение для уменьшения недостатков, которые анализируются в существующих методах, предложили новую конструкцию, способную обнаруживать и предотвращать атаки с проверкой кода, используя статический и динамический анализ на основе детерминированных автоматов с магазинной памятью. Сначала проиллюстрируем работу конструкции обнаружения и предотвращения IVA, показанную на рис. 1, с помощью последовательности операций и действий, связанных со статическим и динамическим анализом, — на рис. 2. Затем опишем, как конструкция обнаруживает и предотвращает атаки с проверкой кода IVA: с использованием SQL-инъекций (SQLIA), с использованием межсайтовых сценариев XSS, и снижает затраты и время тестирования, а также эффективно направляет разработчика/тестера на ту часть исходного кода, на которой следует сосредоточить усилия при выявлении и предотвращении атаки с проверкой кода IVA в веб-приложении. Данная конструкция иллюстрируется с помощью мотивирующего примера выборки веб-приложения. Воздействие конструкции в создании детерминированных автоматов с магазинной памятью и распознавании ввода с помощью контекстно-свободной грамматики демонстрируются на примере. Атаки с проверкой кода происходят из-за неправильной проверки со стороны клиента и сервера в веб-приложении. Эта проблема решается в рамках статического и динамического анализа (рис. 1).

Важность статического и динамического анализа можно продемонстрировать с помощью следующего примера. Рассмотрим веб-приложение с ошибкой или введем неверные утверждения в логику веб-страницы. Первоначально исходный код веб-приложения, который может быть представлен на любом языке, извлекается путем анализа веб-приложения, как показано на шаге 1 рис. 2. Анализ строк выполняется для определения всех возможных значений для каждой точки доступа в исходном коде, статическая строка анализа языков полезна для проверки ошибок в динамически SQL-запросах и вредоносного ввода, как показано на шаге 2 рис. 2. Статический анализ включает анализ исходного кода, возможный ввод в приложение. Грамматика генерирует конечный набор правил для структурирования в строку. Контекстно-свободная грамматика

(Context free grammar — CFG) полезна для распознавания образов для ценного сценария. CFG генерируется на основе процесса анализа строк и правил; их синтаксис и описание идентифицированы, как показано на шаге 3 рис. 2.

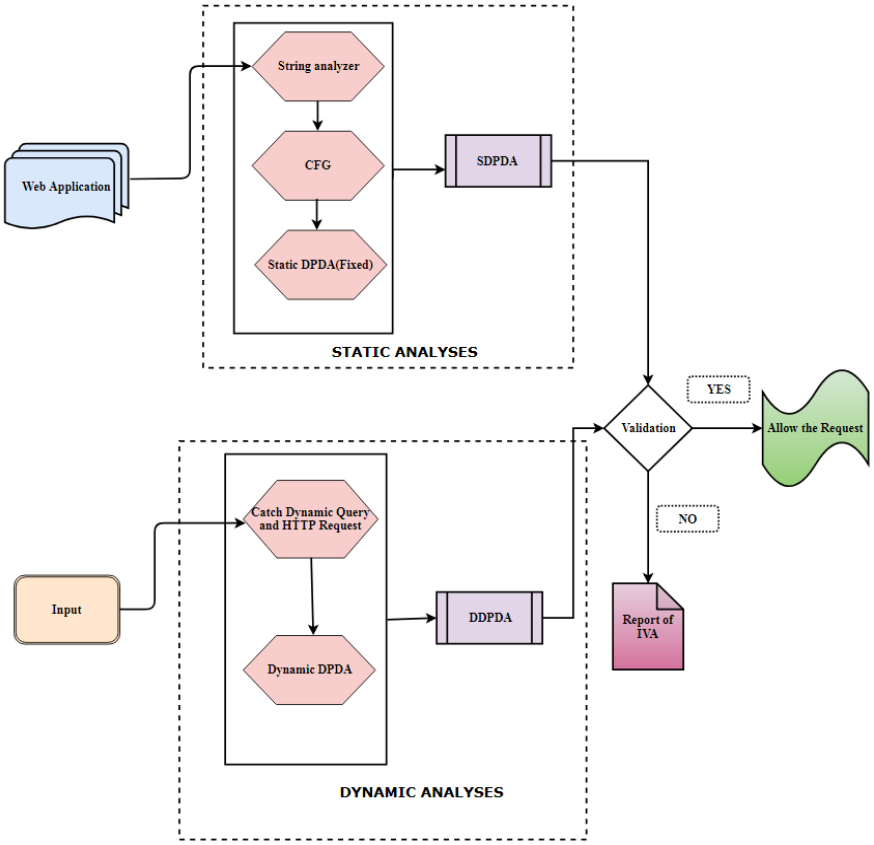


Рис. 1

Соответствующий детерминированный PDA (DPDA) генерируется на основе контекстно-свободной грамматики статически, как показано на шаге 4 рис. 2. DPDA — автомат с магазинной памятью, его действие — это полностью определенная ситуация, а не выбор между несколькими альтернативными действиями. DPDA не могут использовать язык с неоднозначностью грамматики. Детерминированные автоматы с магазинной памятью между каждой операцией должны найти уязвимости XSS и SQL-инъекции. Теперь при вводе данных на веб-странице во время выполнения извлекается DPDA и выполняется проверка между статическим и динамическим PDA для идентификации любого вредоносного ввода, добавляется при вводе входных данных в веб-форму, как показано на шагах 5, 6 (см. рис. 2). Во время процесса проверки, если динамический запрос содержит вредоносный код, он не будет соответствовать статическому PDA-автомату, и запрос будет считаться вредоносным, и выполнение запроса не разрешается, в противном случае, если запрос соответствует статическому PDA-автомату, выполнение запроса разрешается, как показано на шагах 7a, 7b рис. 2. Наконец выполняется шаг классификации, направленный на классификацию вредоносных запросов веб-страниц, таких как SQLIA, XSS или другие атаки и генерирование отчета об атаке с проверкой ввода (IVA) на шаге 8 рис. 2 для веб-приложения.

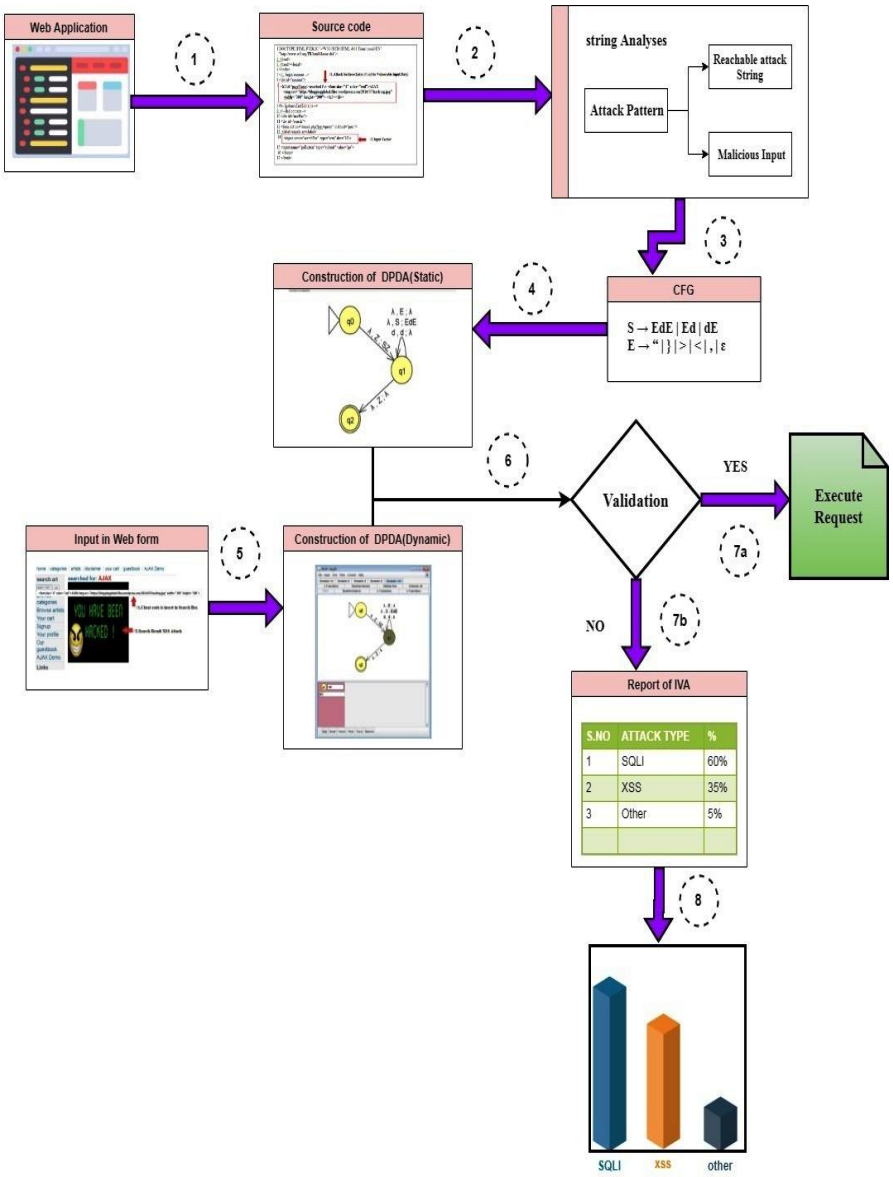


Рис. 2

Методика состоит из следующих этапов.

**2.1. Нахождение поверхности атаки (Attack Surface — AS) и чит-код (Cheat Code — CC) поверхности атаки (Attack Surface — AS) веб-приложения.** Определение поверхности атаки веб-приложения подразумевает все точки входа (Entry Points — EP), через которые осуществляется внешний ввод данных в приложение.

Упрощенное определение поверхности атаки относится к объему кода, функциональности и интерфейсам системы, доступной злоумышленникам. Идея этого понятия заключается в том, что злоумышленники могут использовать уязвимости только в тех частях системы, которые система выставляет противникам.

Поверхность атаки описывает все различные точки, где злоумышленник может попасть в систему, и где он мог бы получить данные.

Поверхностью атаки приложения являются:

1) сумма всех путей для данных в приложение и из него (точки входа и выхода);

2) код, который защищает эти пути, включает в себя соединение с ресурсом и аутентификацию, авторизацию, регистрацию активности, проверку данных и кодирование;

3) все ценные данные, используемые в приложении, в том числе секреты и ключи, интеллектуальная собственность, важные бизнес-данные, личные данные.

Проанализировав точки входа и выхода, создадим список чит-кодов атаки с проверкой ввода после сбора кодов от хакерских приемов и разных сайтов. Здесь список чит-кодов атаки с проверкой ввода поможет протестировать уязвимости SQLI и XSS, полезные для обхода фильтров.

Таблица 1

| Номер | Тип IVA | Поверхность атаки (AS) | Чит-код (CC) |
|---|---|---|---|
| 1 | SQLIA | Пользовательский интерфейс (UI) формы и поля<br>Базы данных<br>Точка входа в систему/аутентификация<br>Интерфейсы администратора<br>Запросы и функции поиска<br>Формы ввода данных (CRUD) | or 1=1—<br>admin' or '1'='1<br>1234 ' AND 1=0 UNION ALL SELECT 'admin',<br>') or '1'='1—<br>'; drop table xyz –<br>';SHUTDOWN; -- |
| 2 | XSS | URL s<br>Праметры<br>HTTP заголовки и куки<br>API s<br>Файлы<br>Форма обратной связи<br>Поле комментариев | “><script>alert(23);</script><br>“};alert(23);a={“a”:<br><img src="javascript:alert('XSS');"><br><link rel="import" href="test.svg" /><br><input onfocus=write(1) autofocus><br><iframe src=http://hacers.org/scriptlet.htm ><br><BODY ONLOAD=alert(‘XSS’)><br>";!--"<XSS>=&{()} |

Здесь AS и CC — веб-приложения, опишем чит-код и поверхность атаки на следующем примере веб-страницы.


Рис. 3

В приведенном примере, где чит-код вставлен в поле поиска веб-формы на рис. 3, просмотрев исходный код, можно найти точку входа для атаки с проверкой ввода на рис. 4.

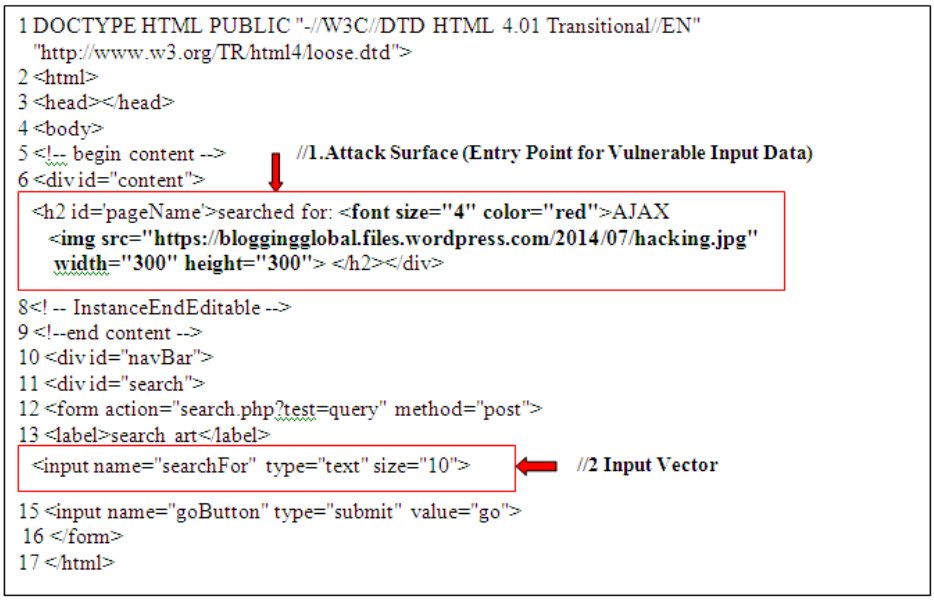


Рис. 4

**2.2. Статический анализ на основе DPDA.** Статический анализ выполняется в неисполняемой среде. Как правило, статический анализ будет проверять исходный код на предмет всех возможных вариантов поведения во время выполнения и наличия уязвимого кода. Статический подход использует CFG для представления точек входа и выхода приложения. CFG — наиболее подходящие и наиболее широко используемые методы описания синтаксиса языков программирования. Их можно использовать для генерации синтаксических анализаторов (parsers), которые преобразуют кусок исходного кода в древовидное представление синтаксической структуры кода. Эти деревья парсера можно использовать для дальнейшей обработки или анализа исходного текста. В этом смысле грамматики составляют основу многих инженерных и обратных инженерных приложений, таких как компиляторы, интерпретаторы и инструменты для анализа и преобразования программного обеспечения.

Детерминированный автомат с магазинной памятью построен как синтаксический анализатор предсказания/совпадения. Каждый шаг либо предсказывает, какую продукцию использовать, либо сравнивает некоторые символы ввода, и детерминированный автомат с магазинной памятью — это PDA с дополнительным свойством, которое состоит в том, что для каждого состояния автомата с PDA и для любой комбинации текущего входного символа и текущего символа стека, определен не более чем один переход.

Например, из рис. 4 фрагмент кода CFG генерируется для строки поиска (точка входа для ввода) строка 14, 7. DPDA представляет статический запрос и грамматическую модель и все возможные значения ввода, которые могут быть введены.

Преимущество использования DPDA для статического анализа состоит в следующем, если можно найти DPDA для CFG, то можно эффективно распознавать строки на этом языке. Ниже на рис. 5 описан статический анализ процесса для веб-программы и определен DPDA из CFG.

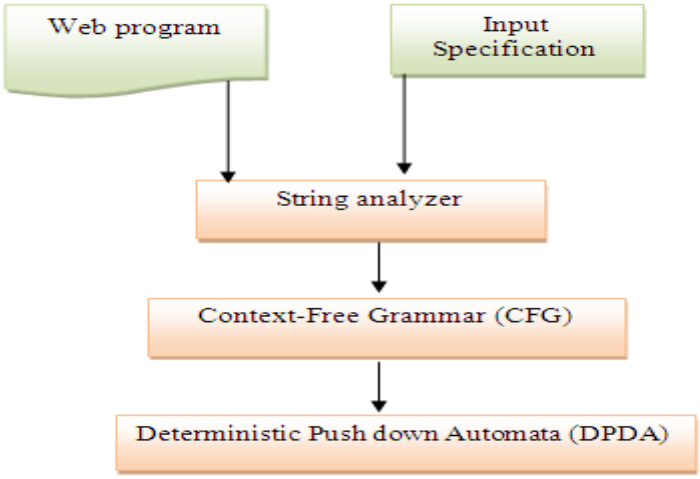


Рис. 5

На рис. 3 CFG генерируется для строки поиска (точка входа для ввода). Рассмотрим допустимый ввод в виде d, который является терминалом, и хакер может добавить чит-код наряду с действительным кодом «d». Итак, рассмотрим случай, когда E — не терминал, т.е. он может или не может быть действительным или неверным кодом, зависит от типа пользователя.

Таблица 2

Начальный символ: **S**
**S** → **E**d**E** | **E**d | d**E**
**E** → “} | > | < |, | ε

| Правило | Приложение | Результат |
|---|---|---|
| Start → **S** | Start | **S** |
| **S** → **E**d**E** | **S** | **E**d**E** |
| **E** → ε | **E**d**E** | d**E** |
| **E** → ε | d**E** | d |

С помощью инструмента JFLAP конвертируем CFG в статический DPDA из CFG. JFLAP является инструментом для автоматизации, но для этого требуются опытные ученики, которые знают основы автоматов, чтобы полнее использовать его возможности.

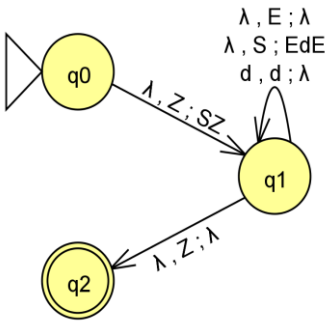


Рис. 6

Из приведенной выше диаграммы (рис. 6) можно описать статический DPDA, где в JFLAP ε представлен λ. Статический DPDA фиксирован, неизменен и используется для проверки входных данных.

**2.3. Динамический анализ на основе DPDA.** Динамический анализ использует противоположный подход и выполняется во время работы програм-

мы (по некоторым входам). Динамический анализ контролирует функциональное поведение, время отклика и общую производительность системы. В данной статье используется динамический анализ загрязнений для отслеживания потока каждого входного символа, информация используется для восстановления входной грамматики. Динамический анализ основан на трансформации поведенческих аспектов программного детерминированного автомата с магазинной памятью, мониторинге запросов во время выполнения и сопоставление их со статическим DPDA. Динамический анализ принимает входные данные или запрос, сформированные во время выполнения, и динамический запрос или грамматическую модель для него. Для строк запроса токенизация в виде SQL-токенов является динамической моделью запроса. Динамический анализ — это захват модели, а мониторинг времени выполнения автоматизирован, но преобразование поведенческих аспектов программы в DPDA в основном ручные или автоматизированные с помощью инструмента JFLAP.

Образец формы поиска вредоносного ввода вставлен как <font size="4" color="red">AJAX<img src="https://bloggingglobal.files.wordpress.com/2014/07/hacking.jpg" width="300" height="300"> (см. рис. 3). Рассмотрим d как действительный ввод **d=AJAX**, где добавлен скрипт к нормальному вводу d, который вызывает атаку с проверкой ввода из-за неправильной проверки, рассматриваемый как **E**, который нетерминальный. Анализируя табл. 1 образца чит-кода веб-приложения, находим что вредоносный код может начинаться со специальных символов “ ,> , }, ; ,<, > etc….

Вход

```
< font size="4" color="red">   AJAX
  <img src="https://bloggingglobal.files.wordpress.com/2014/07/hacking.jpg"
    width="300" height="300">
```

Ввод, вставленный, как показано на рис. 3, представлен **< d <** (< — начальный символ, добавленный до и после действительный ввода **d**).

Таблица 3

| Правило | Приложение | Результат |
|---|---|---|
| Start → **S** | Start | **S** |
| **S** → **E**d**E** | **S** | **E**d**E** |
| **E** → < | **E**d**E** | <d**E** |
| **E** → < | <d**E** | <d< |

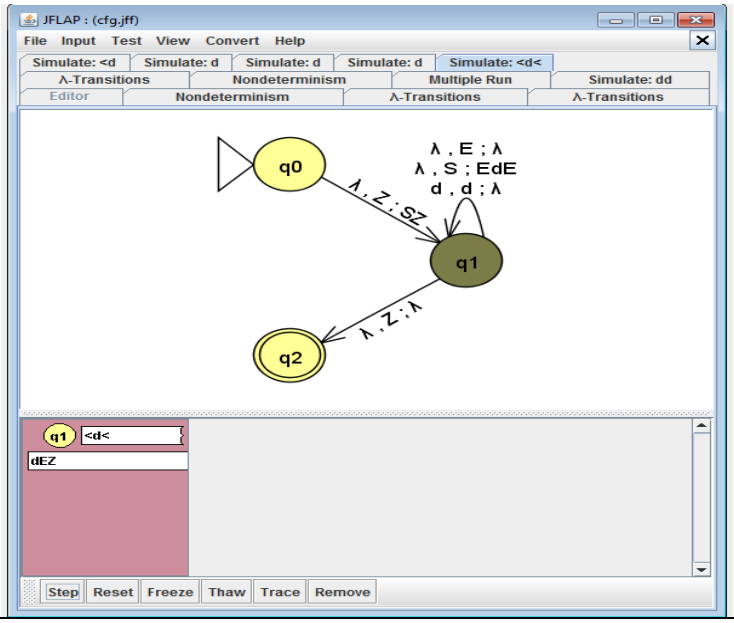


Рис. 7


ISSN 0572-2691

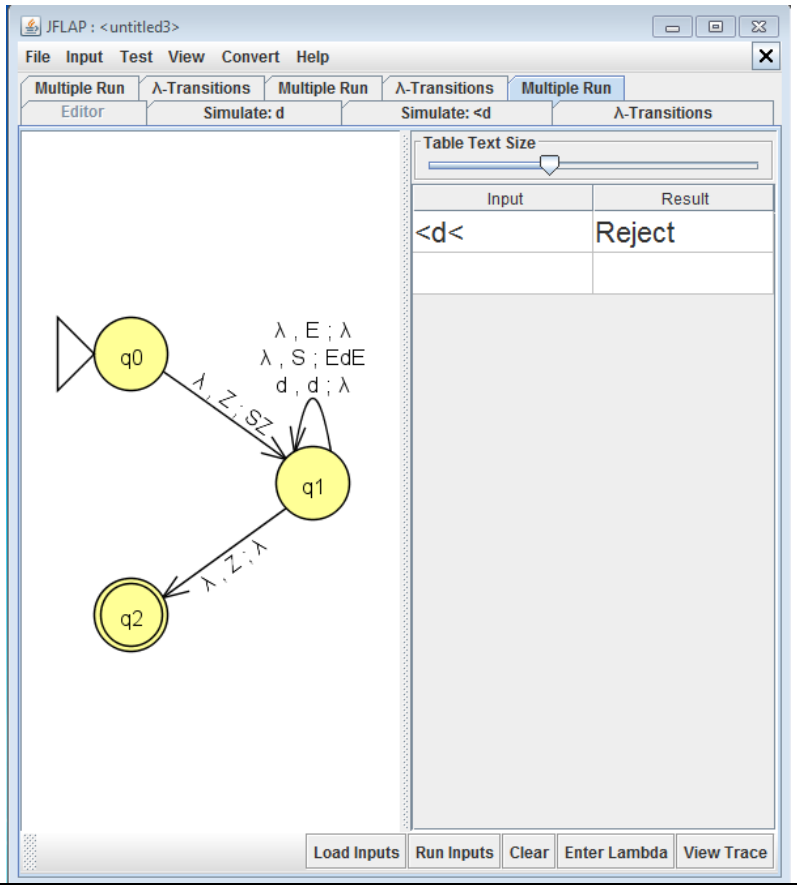


Рис. 8

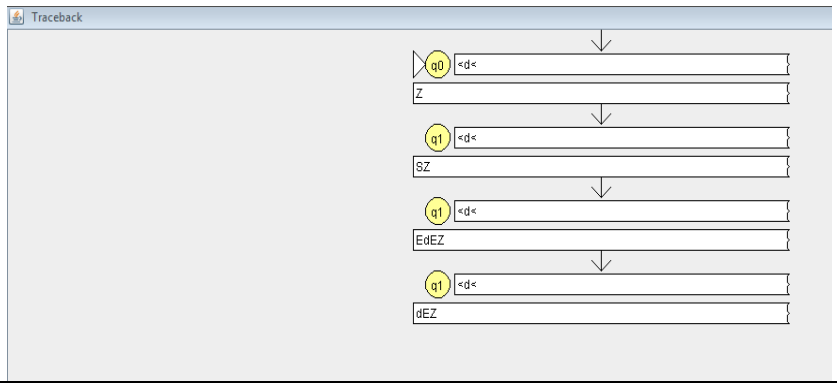


Рис. 9

Из рис. 7–9 следует, что **<font size="4" color="red">AJAX<img src="https://bloggingglobal.files.wordpress.com/2014/07/hacking.jpg"width = "300" height="300">** (поисковое слово используется на веб-странице) является недопустимым вводом, поскольку вводится с вредоносным чит-кодом вместе с действительным вводом AJAX, который представлен как **<d<**, поскольку ввод вставлен с вредоносным кодом, не принятым конечным состоянием q2 в DPDA, этот тип вредоносного запроса отклоняется DPDA, как показано на рис. 8.

Рассмотрим еще один пример, как действительный ввод анализируется с помощью предлагаемой конструкции.


Рис. 10

Вход вставлен, как показано на рис. 10, с входом AJAX, представляет d, не добавлен с вредоносным кодом в форму поиска на веб-странице.

Таблица 4

| Правило | Приложение | Результат |
|---|---|---|
| Start → **S** | Start | **S** |
| **S** → **E**d**E** | **S** | **E**d**E** |
| **E** → ε | **E**d**E** | d**E** |
| **E** → ε | d**E** | d |

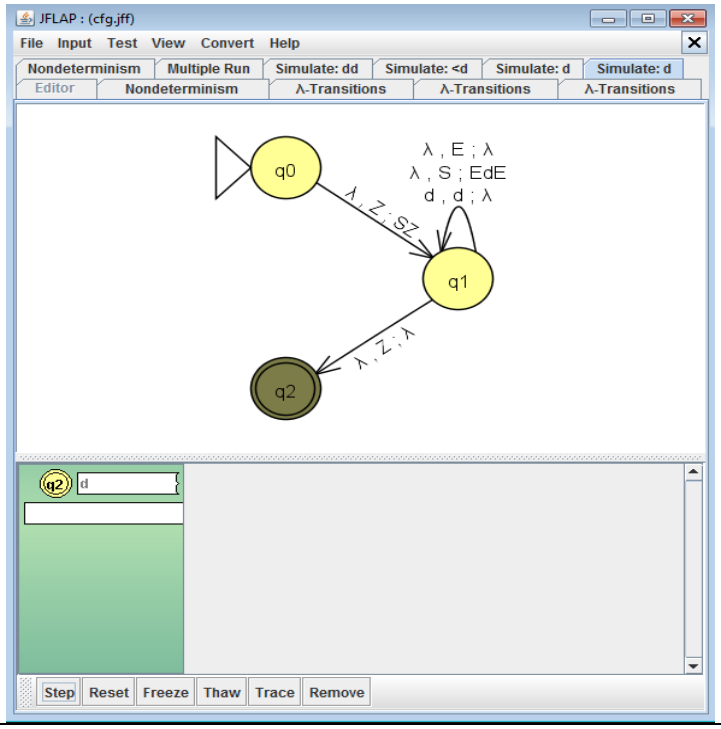


Рис. 11

Из рис. 11–13 видно, что AJAX (поисковое слово, используемое на веб-странице) является допустимым вводом, который не вставляется с каким-либо вредоносным, так как ввод свободен от вредоносного кода, принятого конечным состоянием q2 в DPDA.


ISSN 0572-2691

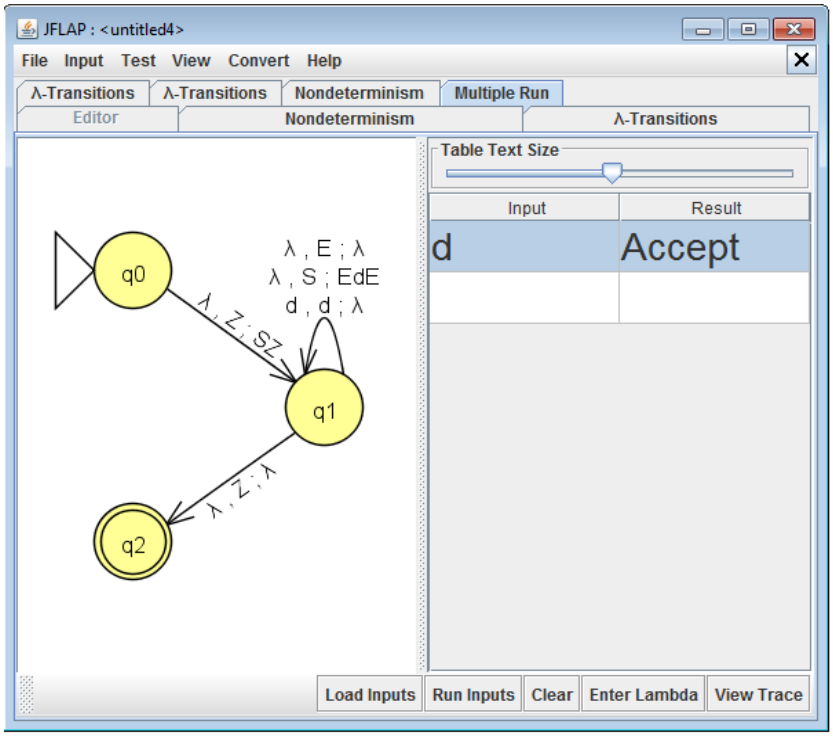


Рис. 12

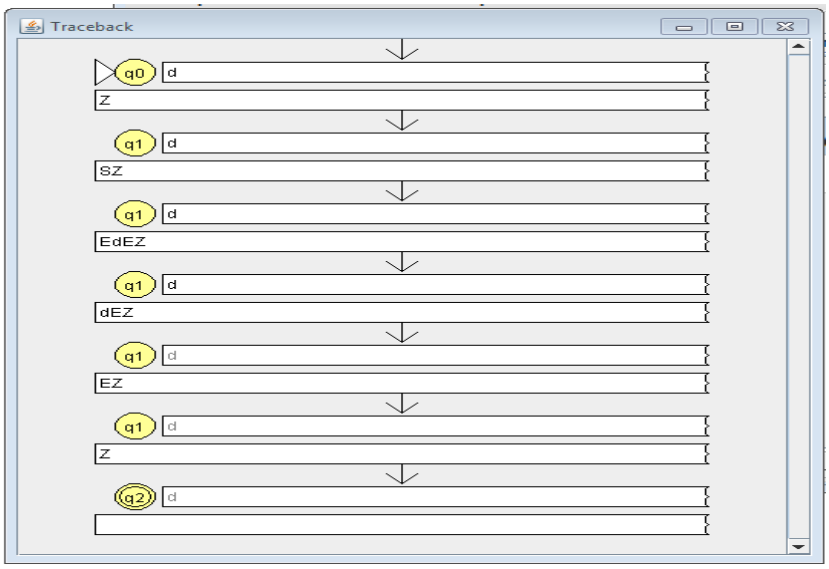


Рис. 13

**2.4. Валидация.** Предлагается метод проверки для обнаружения атак с проверкой ввода на основе статического и динамического анализа. Этот метод генерирует структуру динамических детерминированных автоматов (DDPDA) для выполнения ввода во время выполнения и сравнивает их с грамматикой статического детерминированного автомата (SDPDA), структура которого проанализирована заранее. Затем программисту предоставляется сводка результатов статического и динамического сопоставления. Символы, используемые в предлагаемом алгоритме, приведены в табл. 5.

Таблица 5

| Номер | Символы | Описание |
|---|---|---|
| 1 | $I_u$ | Пользовательские данные |
| 2 | $f_s$ | Функция для генерации статической DPDA модели |
| 3 | $f_d$ | Функция для генерации динамической модели DPDA |
| 4 | $C_S$ | Фрагмент статического кода |
| 5 | $C_D$ | Фрагмент динамического кода |
| 6 | $S_m$ | Статическая модель DPDA (фиксированная) |
| 7 | $D_m$ | Динамическая модель DPDA |

Предложенный метод обнаружения использует функцию $f_s$, которая генерирует статическую DPDA и динамическую модель DPDA. Функция показана в формулах (1) и (2), фиксированный DPDA и DPDA во время выполнения в веб-приложении и сгенерированы:

$$S_m = f_s\left(C_S\right) \tag{1}$$

$$D_m = f_d\left(C_D\right) \tag{2}$$

$$S_m \oplus D_m \begin{cases} = \text{нормальный} \\ \neq \text{ненормальный.} \end{cases} \tag{3}$$

Формула (3) применяется независимо от того, является пользовательский ввод $\left(I_u\right)$ нормальным или ненормальным. Здесь символ $\oplus$ представляет исключающий оператор ИЛИ. Таким образом, (1), (2) логически исключают один другого логическим оператором OR. Если эту формулу применить к приведенному выше примеру, получим

$$S_m \oplus D_{m=}0 = \text{нормальный (принимается),} \tag{3′}$$

$$S_m \oplus D_{m=}0 \neq \text{ненормальный (отклоняется).} \tag{3″}$$

Если найден результат (3′), то процесс проверки считают нормальным и выполнение запроса пользователя разрешается. Если обнаружен результат (3″), то процесс проверки считается ненормальным, запрещающим выполнение вредоносного запроса, содержащим уязвимый код (чит-код).

$N$: Общее число элементов поверхности атаки (точки входа и выхода) в веб-приложении

$I_u$: Пользовательские данные

$f_s$: Функция для генерации статической модели DPDA

$f_d$: Функция для генерации динамической модели DPDA

$C_S$: Фрагмент статического кода

$C_D$: Фрагмент динамического кода

$Sm_i$: Статическая модель DPDA (фиксированная)

$Dm_i$: Динамическая модель DPDA, сгенерированная из $Sm_i$

$Sm_i = \{Sm_1 . . . Sm_n\}$,

$Dm_i = \{Dm_1, . . . , Dm_n\}$,

// Static analyss

1. For $i$=1 to $N$
2. Get $C_S$
3. $Sm_i = f_s\left(C_{Si}\right)$
4. End {For}
5. // Dynamic analysis (running time)
6. While (Normal & $\forall k \in N$)


ISSN 0572-2691

7. Get $C_{Dk}$ from the web with $I\{t, f\}$
8. $Dm_k = f_d\ (C_{Dk})$
9. If $(Sm_k \bigoplus Dm_k) = 0$ then
10. Result = Normal (Accept)
11. Else
12. Result = Abnormal (Reject)
13. End {If}
14. End {While}

**Алгоритм 1.** Алгоритм обнаружения IVAs (рис. 14).

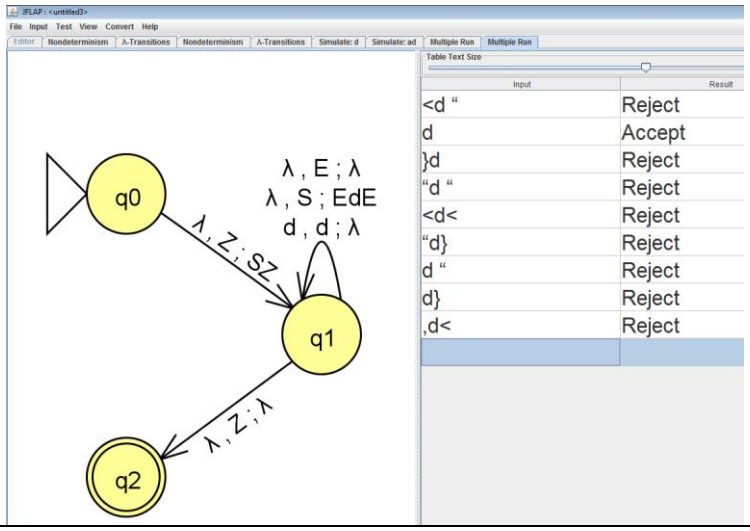


Рис. 14

Из результатов сопоставления статического (SDPDA) и динамического (DDPDA) типов принимает только действительный ввод d, где другой возможный вредоносный код вместе с действительным вводом DPDA отклоняет. Вход, который DPDA отклоняет, обнаруживается как ненормальный пример (см. рис. 3) и запрос не выполняется.

Рассмотрим отчет об атаке с проверкой ввода (IVA). Если обнаружен вредоносный запрос из-за уязвимого ввода, то он считается атакой с проверкой ввода. Основанная история набора данных IVA классифицируется как атака SQL-инъекций (SQLIA), межсайтовый скриптинг атака и любые другие виды атак. Этот последний этап, на котором обнаружена атака с проверкой ввода, заблокирован для обработки вредоноснго запроса и выводится сообщение о типе попытки ПВА атаковать веб-приложение.

## 3. Результаты и обсуждения

Предлагаемый метод предоставляет надежную информацию, чтобы направлять тестировщика/разработчиков, или предоставляет инструменты/методы обнаружения IVA в сценарии, описанном выше. Предлагаемая методика — неотъемлемая часть общего метода. Экспериментальная оценка метода продолжается. Тем не менее предварительные результаты показывают, что данная методика эффективна и осуществима в любом веб-приложении, использующем JSP, PHP, Java и т.д. Предлагаемая система моделируется путем реализации онлайн-приложений, таких как онлайн-банкинг, книжный магазин, образовательный сайт, которые, как известно, уязвимы к атакам с проверкой ввода. Приложение развернуто на сервере Glass Fish с MySQL 5.5 как база данных в NetBeans8.1.

Таблица 6

| Предотвращение и метод обнаружения | Атаки SQL | | | | | Межсайтовый скриптинг атаки | | |
|---|---|---|---|---|---|---|---|---|
| | Тавтологии | Комбинированный запрос | Логически неверно | Союзный запрос | Хранимая процедура | Отраженный МС | Постоянный МС | МС на основе ОМД |
| Метод удаление атрибута | ▲ | ▲ | ▲ | ▲ | ▲ | ▼ | ▼ | ▼ |
| Входные шаблоны санации | △ | △ | △ | △ | △ | △ | ▼ | ▼ |
| Семантическая безопасность | ▲ | ▲ | ▲ | ▲ | △ | △ | △ | △ |
| PHP структура надзора | ▼ | ▼ | ▼ | ▼ | ▼ | ▲ | △ | △ |
| Классификация и анализ | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | ▼ | ▼ |
| Санации на основе размера запроса | ▲ | ▲ | ▲ | ▲ | ▲ | ▼ | ▼ | ▼ |
| Конструкция предсказания уязвимости | △ | △ | △ | △ | △ | △ | △ | △ |
| Статический анализ и сбор данных | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | △ |
| Инструменты обнаружения уязвимости | △ | △ | △ | △ | △ | △ | △ | △ |
| Контекстные отпечатки script | ▼ | ▼ | ▼ | ▼ | ▼ | ▲ | ▲ | ▲ |
| Предложенный метод | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ | ▲ |

Символы: ▲ — возможно, △ — частично возможно, ▼ — невозможно.

В табл. 6 приведен сравнительный анализ предотвращения атаки с проверкой ввода и методов обнаружения и типов атак. Символ ▲ используется для метода, который может успешно остановить все атаки этого типа. Символ ▼ используется для техники, которая не может прекратить атаки такого типа. Символ △ относится к методу, который останавливает только тип атаки частично из-за ограничений основного подхода. Хотя многие подходы определены как методы обнаружения или предотвращения, лишь немногие из них были внедрены практически. Следовательно, это сравнение основано не на эмпирическом опыте, а на аналитической оценке. Из приведенных выше результатов, улучшая метод статического и динамического анализа, можно обнаружить все типы атак с проверкой ввода.

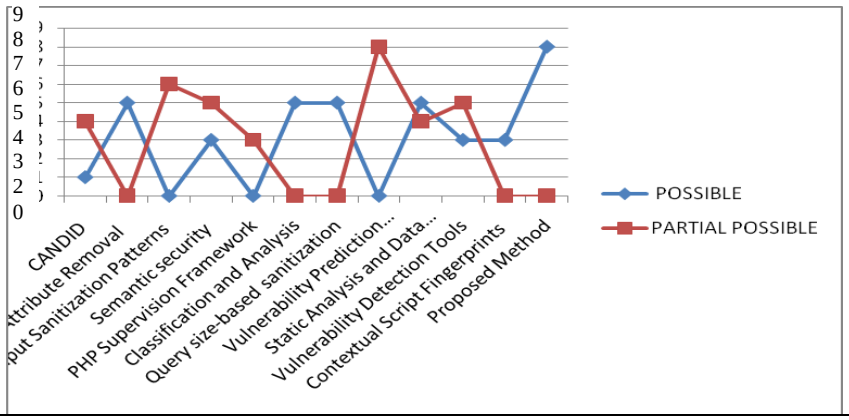


Рис. 15


ISSN 0572-2691

На рис. 15 описан график количества атак с проверкой ввода и различными подходами предложенного метода. Существующие методы только обнаруживают и предотвращают несколько типов атак. Предлагаемый метод, основанный на DPDA, может предотвратить и обнаружить все различные типы атак с проверкой ввода.

**Заключение**

В данной статье показано, как поверхность атаки идентичности, которая является горячей точкой для атаки с проверкой ввода, может указать разработчикам приложений и тестировщикам, какая часть исходного кода должна быть проанализирована в ходе тестирования веб-приложения для выявления и предотвращения IVAs. Отсутствие этой достоверной информация приводит к потере времени и ресурсов для проверки уязвимостей в приложении, однако не является проблемой для небольших приложений, и наоборот, их увеличение порождает проблему. Существующие подходы к обеспечению безопасности традиционных приложений не всегда достаточны при рассмотрении веб-парадигмы и часто ltkf.n конечных пользователей ответственны за защиту ключевых аспектов обслуживания. Эта ситуация должна быть предотвращена, так как при не очень хорошем управлении она может допустить ненадлежащее использование веб-приложения и привести к нарушению требований его безопасности.

Предлагается статический и динамический анализ на основе детерминированного автомата с магазинной памятью DPDA для обнаружения и предотвращения IVAs в веб-приложении. DPDA основаны на CFG, статический и динамический анализ взят из приложения и DPDA создаются и сравниваются для идентификации любого нового внедрения уязвимого кода. Экспериментальная оценка предлагаемого метода утверждает, что статический и динамический анализ может эффективно обнаруживать и удалять атаки с проверкой ввода. Предварительные результаты показывают, что предложенный подход эффективен и выполним для прикладной среды. В следующей публикации будет рассмотрена классификация атак с проверкой ввода и степени их серьезности в веб-приложении.

*В. Нитья, С. Сентилкумар*

## ВИЯВЛЕННЯ ТА ЗАПОБІГАННЯ АТАК З ПЕРЕВІРКОЮ ВВОДУ У ВЕБ-ДОДАТКИ ПРИ ВИКОРИСТАННІ ДЕТЕРМІНОВАНИХ АВТОМАТІВ ІЗ МАГАЗИННОЮ ПАМ'ЯТТЮ

Належна перевірка вводу та дезинфекції є критичною проблемою при разробці веб-додатків. Помилки та недоліки в операціях перевірки спричиняють зловмисну поведінку у веб-додатку, злочинцям неважко буде їх використати. Оскільки зловмисники стрімко набувають навиків та здібностей, вони зосереджені на вивченні вразливостей у веб-додатках та намагаються наразити на небезпеку конфіденційність, цілісність та доступність інформаційної системи. Атаки з перевіркою вводу (Input Validation Attacks — IVA) — це коли хакер відправляє шкідливі дані (чіт-коди), щоб заплутати веб-додатки та отримати доступ або знищити кінець додатка без відома користувача. Перевірка вводу — це перша лінія захисту для таких атак. Приклади атак з перевіркою вводу включають в себе міжсайтовий скриптинг (Cross Site Scripting — XSS), атаку SQL-ін'єкцій (SQL Injection Attack — SQLIA), переповнення буфера та зворотний шлях у каталогах. Використовуючи атаки з перевіркою коду, хакери можуть викрасти конфіденційні дані, що знижують ринкову вартість організації. У цьому проекті досліджено проблему виявлення та усунення похибок перевірки як клієнтського, так і

серверного коду з використанням даного підходу. Запропоновано нову ідею, що сприяє виявленню та усуванню атаки з перевіркою коду з використанням статичного та динамічного аналізу.

**Ключові слова:** атака з перевіркою коду, статичний і динамічний аналіз, виявлення, запобігання, детерміновані автомати з магазинною пам'яттю.

*V. Nithya, S. Senthilkumar*

# DETECTION AND AVOIDANCE OF INPUT VALIDATION ATTACKS IN WEB APPLICATION USING DETERMINISTIC PUSH DOWN AUTOMATA

The proper validation of input and sanitization is critical issue in developing web applications. Errors and flaws in validation operations resulting in malicious behavior in web application can be easily exploited by attackers. Since attackers are rapidly developing their skills and abilities they focus on exploring vulnerabilities in the web applications and try to compromise confidentiality, integrity and availability of information system. Input Validation Attacks (IVAs) are the attacks where a hacker sends malicious inputs (cheat codes) to confuse a web application in order to have access or destroy back end of application without knowledge of users. Input validation serves as the first line of defense for such attacks. Examples of input validation attacks include Cross Site Scripting (XSS), SQL Injection Attack (SQLIA), buffer overflow and directory traversal. Using Input validation attacks hackers can steal the sensitive data which decrease organization market value. In this project, we investigate the problem of detection and removal of validation bugs both at the client-side and the server-side code by using our approach. In this paper we proposed new idea that makes it possible to able detect and prevent input validation attack using Static and Dynamic Analysis.

**Keywords:** Input validations attack, static, dynamic, detection, prevention, deterministic push down automata.

1. Razzaq Abdul, Latif Khalid, Ahmad H. Farooq, Hur Ali, Zahid Anwar, Bloodsworth Peter Charles, Semantic Security against Web Application Attacks, *Information Sciences*. 2013. P. 19–38.
2. Moller A., Schwarz M. Automated detection of client-state manipulation vulnerabilities. *ACM Trans. Softw. Eng. Methodol.* 2014. **23**, N 4.
3. Bisht P., Madhusudan P., Venkatakrishnan V.N. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. *ACM Transactions on Information and System Security*. 2010. **13**, N 2. DOI: 10.1145/1698750. 1698754
4. Diomidis Spinellis, and Angelos D. Keromytis, How to train your browser: Preventing XSS attacks using contextual script fingerprints. *ACM Trans. Priv. Secur.* 2016. **19**, N 1. Article 2.
5. Halfond W.G.J., Orso A. Member and Panagiotis Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation. *IEEE Transactions on Software Engineering*. 2008. **34**, N 1. P. 65–81.
6. Medeiros I., Neves N., Correia M. Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. *IEEE Transactions on Reliability*. 2016. **65**, N 1. P. 54–69.
7. Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon, A novel method for SQL injection attack detection based on removing SQL query attribute values. *Mathematical and Computer Modelling*. 2012. **55**. P. 58–68.
8. Jang Y.S., Choi J.Y. Detecting SQL injection attacks using query result size. *Computers & Security*. 2014. **44**. P. 104–118.
9. Lwin Khin Shar, Lionel C. Briand, Hee Beng Kuan Tan. Web application vulnerability prediction using hybrid program analysis and machine learning. *IEEE Trans. Dependable Sec. Comput.* 2015. **12**, N 6. P. 688–707.
10. NithyaV., Lakshmana Pandian S. and Regan R. The SQL injection attack and prevention by classification and analysis. *Asian Journal of Information Detection Technology*. 2013. **12**. P. 131–139.

11. Nithya V., Regan R. & Vijayaraghavan J. A survey on SQL injection attacks, their detection and prevention techniques. *International Journal of Engineering and Computer Science*. 2013. **2**. N 4. P. 886–90.
12. Rim Akrout, Eric Alata, Mohamed Kaaniche and Vincent Nicomette. An automated black box approach for web vulnerability identification and attack scenario generation. *Journal of the Brazilian Computer Society.* 2014.
13. Sung Soo Kim et. al, Vulnerability detection mechanism based on open API for multi-user's convenience. IEEE. 2016.
14. Takeshi Matsuda, Daiki Koizumi and Michio Sonoda. Cross site scripting attacks detection algorithm based on the appearance position of characters. *The 5th International. Conference on Communications, Computers and Applications.* Istanbul. Turkey. 2012. P. 65–70.
15. Prokhorenko V., Kim-Kwang Raymond Choo, Helen Ashman. Intent-Based Extensible Real-Time PHP Supervision Framework, Institute of Electrical and Electronics Engineers. *IEEE Transactions on Information Forensics and Security*. 2012. **10**, N 11. P. 2215–2226.
16. Lwin Khin Shar, Hee Beng Kuan Tan. Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. *Information and Software Technology*. 2013. **55**. P. 1767–1780.
17. Isatou Hydara , Abu Bakar Md Sultan , Hazura Zulzalil. Novia Admodisastro Removing "Cross-Site Scripting Vulnerabilities from Web Applications using the OWASP ESAP I Security Guidelines, Hydara, 2015. **8**, N 30.
18. Sangwook Cho, Gyoosik Kim, Seong-je Cho, Jongmoo Choi, Minkyu Park, Sangchul Han. Runtime Input Validation for Java Web Applications using Static Bytecode Instrumentation Proceeding, RACS '16. *Proceedings of the International Conference on Research in Adaptive and Convergent Systems.* 2016. P. 148–152.
19. Mehrnoush Vaseghipanah, Nasser Modiri and Sam Jabbehdari, Detecting Input Validation Attacks of Web Apps and Developing Metrics for Their Ranks, *IJCSNS International Journal of Computer Science and Network Security*. 2017. **17**, N 6.
20. Bakare K. Ayeni , Junaidu B. Sahalu, and Kolawole R. Adeyanju. Detecting Cross-Site Scripting in Web Applications Using Fuzzy Inference System. *Journal of Computer Networks and Communications.* 2018. Article ID 8159548.
21. Park Y.J., Park J. Web Application Intrusion Detection System for Input Validation Attack. IEEE 2008. located on : http://portal.acm.org/citation.cfm?id= 1472086
22. Nithya V., S. Lakshmana Pandian, C. Malarvizhi. A survey on detection and prevention of cross-site scripting attack. *International Journal of Security and Its Applications*. 2017. **3**, N 3. P. l39–152.

*Получено 29.05.2019*