**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и вычислительной техники

Кафедра ИиСП

**Реферат по дисциплине «Контроль качества»**

**на тему «Безопасность web-приложений»**

Выполнил:

студент группы: ПС-31

Смирнов Фёдор

Проверил:

Рядова Ольга Владимировна

г. Йошкар-Ола

2020

**СОДЕРЖАНИЕ:**

1. Безопасность web-приложений 3
2. Основные угрозы.

2.1. Классы атак 5

2.2. Основные уязвимости 8

1. Способы выявления 11

3.1. Метод получения идентифицирующей информации 12

3.2. Метод тестирования на проникновение 14

3.3. Метод статического анализа 17

3.4. Метод динамического анализа 20

1. Сравнение инструментов тестирования безопасности 22
2. Литература 30
3. БЕЗОПАСТНОСТЬ WEB-ПРИЛОЖЕНИЙ

Развитие глобальной сети интернет и увеличение доступности ресурсов в ней, привело к расширению спектра задач, решаемых с использованием WEB-технологий. В связи с этим получил распространение особый вид приложений — WEB-приложения, от которых напрямую зависит функционирование бизнес-процессов многих организаций.

WEB-приложение — это клиент-серверное приложение, где в качестве клиента выступает браузер, который отображает пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него. А серверная часть представляет собой WEB-сервер, обрабатывающий запросы клиентов. Взаимодействие между клиентом и сервером, как правило, осуществляется посредством протокола HTTP. Архитектура WEB-приложений имеет три уровня, которые показаны на рисунке 1.



Рисунок 1. Архитектура WEB-приложения

Основными особенностями, повлиявшими на распространение WEB-приложения являются:

1. Доступность — данный вид приложений не привязан к определенному терминалу или локальной сети, доступ может осуществляться из любой точки Земного шара, где присутствует соединение с Интернет.
2. Кроссплатформенность — клиенту достаточно иметь браузер, соответствующий стандартам, а операционная система и тип устройства не имеют никакого значения.
3. Не требует установки и настройки отдельного приложения на клиентском устройстве.
4. Автоматическое обновление — клиент всегда работает с самой актуальной версией приложения, т. к. все обновления происходят на стороне сервера.

Но в тоже время в процессе эволюции WEB-приложений разработчики сталкивались с рядом проблем, среди которых особняком стоит проблема безопасности WEB-приложений. Проблемы безопасности вытекают из основных особенностей WEB-приложений, например, из-за невозможности изолировать WEB-приложение от попыток несанкционированного доступа извне в виду их доступности.

Актуальность проблем безопасности WEB-приложений подкрепляется тем, что в них используется конфиденциальная информация, а также осуществляются бизнес-процессы компании, например:

* хранение и обработка персональные данных сотрудников и клиентов;
* сведения о финансовых операциях компании;
* сведения, составляющие коммерческую тайну;
* сведения о взаимодействии, а также взаимодействие с клиентами компании;
* взаимодействие между сотрудниками и отделами;

1. ОСНОВНЫЕ УГРОЗЫ
   1. Классы атак

### Аутентификация (Authentication)

* Подбор (Brute Force)
* автоматизированный процесс проб и ошибок, использующийся для того, чтобы угадать имя пользователя, пароль, номер кредитной карточки, ключ шифрования и т.д.
* Недостаточная аутентификация (Insufficient Authentication)
* эта уязвимость возникает, когда Web-сервер позволяет атакующему получать доступ к важной информации или функциям сервера без должной аутентификации
* Небезопасное восстановление паролей (Weak Password Recovery Validation)
* эта уязвимость возникает, когда Web-сервер позволяет атакующему несанкционированно получать, модифицировать или восстанавливать пароли других пользователей

### Авторизация (Authorization)

* Предсказуемое значение идентификатора сессии (Credential/Session Prediction)  
  предсказуемое значение идентификатора сессии позволяет перехватывать сессии других пользователей
* Недостаточная авторизация (Insufficient Authorization)  
  недостаточная авторизация возникает, когда Web-сервер позволяет атакующему получать доступ к важной информации или функциям, доступ к которым должен быть ограничен
* Отсутствие таймаута сессии (Insufficient Session Expiration)  
  в случае если для идентификатора сессии или учетных данных не предусмотрен таймаут или его значение слишком велико, злоумышленник может воспользоваться старыми данными для авторизации
* Фиксация сессии (Session Fixation)  
  используя данный класс атак, злоумышленник присваивает идентификатору сессии пользователя заданное значение

### Атаки на клиентов (Client-side Attacks)

* Подмена содержимого (Content Spoofing)  
  используя эту технику, злоумышленник заставляет пользователя поверить, что страницы сгенерированны Web-сервером, а не переданы из внешнего источника
* Межсайтовое выполнение сценариев (Cross-site Scripting, XSS)  
  наличие уязвимости Cross-site Scripting позволяет атакующему передать серверу исполняемый код, который будет перенаправлен браузеру пользователя
* Расщепление HTTP-запроса (HTTP Response Splitting)  
  при использовании данной уязвимости злоумышленник посылает серверу специальным образом сформированный запрос, ответ на который интерпретируется целью атаки как два разных ответа

### Выполнение кода (Command Execution)

* Переполнение буфера (Buffer Overflow)  
  эксплуатация переполнения буфера позволяет злоумышленнику изменить путь исполнения программы путем перезаписи данных в памяти системы
* Атака на функции форматирования строк (Format String Attack)  
  при использовании этих атак путь исполнения программы модифицируется методои перезаписи областей памяти с помощью функций форматирования символьных переменных
* Внедрение операторов LDAP (LDAP Injection)  
  атаки этого типа направлены на Web-серверы, создающие запросы к службе LDAP на основе данных, вводимых пользователем
* Выполнение команд ОС (OS Commanding)  
  атаки этого класса направлены на выполнение команд операционной системы на Web-сервере путем манипуляции входными данными
* Внедрение операторов SQL (SQL Injection)  
  эти атаки направлены на Web-серверы, создающие SQL запросы к серверам СУБД на основе данных, вводимых пользователем
* Внедрение серверных сценариев (SSI Injection)  
  атаки данного класса позволяют злоумышленнику передать исполняемый код, который в дальнейшем будет выполнен на Web-сервере
* Внедрение операторов XPath (XPath Injection)  
  эти атаки направлены на Web-серверы, создающие запросы на языке XPath на основе данных, вводимых пользователем

### Разглашение информации (Information Disclosure)

* Индексирование директорий (Directory Indexing)  
  атаки данного класса позволяют атакующему получить информацию о наличии файлов в Web каталоге, которые недоступны при обычной навигации по Web сайту
* Идентификация приложений (Web Server/Application Fingerprinting)  
  определение версий приложений используется злоумышленником для получения информации об используемых сервером и клиентом операционных системах, Web-северах и браузерах
* Утечка информации (Information Leakage)  
  эти уязвимости возникают в ситуациях, когда сервер публикует важную информацию, например комментарии разработчиков или сообщения об ошибках, которая может быть использована для компрометации системы
* Обратный путь в директориях (Path Traversal)  
  данная техника атак направлена на получение доступа к файлам, директориям и командам, находящимся вне основной директории Web-сервера.
* Предсказуемое расположение ресурсов (Predictable Resource Location)  
  позволяет злоумышленнику получить доступ к скрытым данным или функциональным возможностя

### Логические атаки (Logical Attacks)

* Злоупотребление функциональными возможностями (Abuse of Functionality)  
  данные атаки направлены на использование функций Web-приложения с целью обхода механизмов разграничение доступа
* Отказ в обслуживании (Denial of Service)  
  данный класс атак направлен на нарушение доступности Web-сервера
* Недостаточное противодействие автоматизации (Insufficient Anti-automation)  
  эти уязвимости возникаеют, в случае, если сервер позволяет автоматически выполнять операции, которые должны проводиться вручную
* Недостаточная проверка процесса (Insufficient Process Validation)  
  уязвимости этого класса возникают, когда сервер недостаточно проверяет последовательность выполнения операций приложения
  1. Список уязвимостей OWASP (Open Web Application Security Project)
* Инъекции (Injection) — внедрение в запросы к базе данных кода, дополняющего данный запрос и дающего злоумышленнику неавторизованный доступ к базе данных.
* Уязвимости аутентификации (Broken Authentification) – распространенная уязвимость, связанная с недостаточно проработанной системой валидации пользователей в приложении, приводит к получению неавторизованного доступа.
* Незащищенность важных данных (Sensitive Data Exposure) — многие приложения не используют механизмов для защиты передаваемых данных, таких как, например, HTTPS.
* Внедрение внешних сущностей в XML (XML External Entities) — вид инъекции, основанный на внедрении в XML-запрос к серверу  атрибутов и сущностей, позволяющих получить неавторизованный доступ к данным.
* Небезопасный контроль доступа (Broken Access Control) — уязвимость в методах авторизации, позволяющие злоумышленнику получить повышенные привилегии.
* Небезопасная конфигурация (Security Misconfiguration) — WEB-приложение — это сложная система, состоящая из многих компонентов, таких как WEB-сервер, СУБД и др. Неверная конфигурация одного из компонентов может привести к серьезным проблемам с безопасностью всего приложений.
* XSS(Cross-Site Scripting) – внедрение (инъекция) вредоносного кода в HTTP-ответ, получаемый клиентом и выполняющийся на стороне клиента.
* Небезопасная десериализация (Insecure Deserialization) — десериализация преобразует последовательность бит в структурированные данные, зачастую на данном этапе не уделяется достаточно внимания безопасности, например, отсутствует валидация типов данных, что приводит к их подмене.
* Использование компонентов с известными уязвимостями (Using Components with Known Vulnerabilities) — зачастую при разработки WEB-приложений используются библиотеки, фреймворки и компоненты сторонних разработчиков, которые могут содержать различные недостатки (уязвимости), в связи с этим важно использовать самые актуальные версии, в которых исправляются известные уязвимости.
* Недостаточное журналирование и мониторинг (Insufficient Logging&Monitoring) — для своевременного обнаружения несанкционированного доступа, утечки информации и т. д., необходимо использовать средства автоматизированного мониторинга трафика, а также журналирования, которые помогут понять сущность атаки и разработать в кратчайшие сроки устранить уязвимости, а также вернуть работоспособность и состояние WEB-приложений.

1. **СПОСОБЫ ВЫЯВЛЕНИЯ УГРОЗ**

Методы обнаружения уязвимостей web-приложений На сегодняшний день согласно исследованиям OWASP наиболее эффективным способом обнаружения уязвимостей   
web-приложений является экспертный анализ исходных кодов web-приложения   
(code review). Этот способ весьма трудоемок, требует высокой квалификации эксперта и не защищен от ошибок эксперта. Поэтому активно развиваются методы автоматического обнаружения уязвимостей web-приложений.

Методы автоматического обнаружения уязвимостей web-приложений можно разделить на две основные группы:

1. методы, анализирующие работу развернутого на стенде web-приложения без обращения к исходным кодам web-приложения;
2. методы, анализирующие исходные коды web-приложения и конфигурационные настройки.

Первая группа методов рассматривает web-приложение с точки зрения внешнего пользователя, то есть потенциального злоумышленника. В эту группу входят следующие методы:

* Метод получения идентифицирующей информации о web-приложении и выявления его уязвимостей с помощью бюллетеней безопасности (security advisory).
* Метод тестирования на проникновение.

Вторая группа состоит из следующих методов:

* Метод статического анализа исходных кодов web-приложения.
* Метод динамического анализа исходных кодов web-приложения.
  1. Метод получения идентифицирующей информации о web-приложении.

Метод получения идентифицирующей информации о web-приложении основан на посылке от имени обычного пользователя web-приложению набора HTTP-запросов, ответы на которые позволят сделать вывод о том, на каком web-сервере работает приложение, с помощью какой технологии оно разработано, какие версии программного обеспечения использует, какие стандартные компоненты оно использует и т.д. Для определения типа и версии web-сервера используется техника fingerprint, которая основана на том, что каждый web-сервер по-своему обрабатывает HTTP-протокол, в результате чего можно с высокой степенью вероятности определить тип и даже версию web-сервера путем посылки серверу набора корректных и некорректных запросов и анализа соответствующих ответов. Для определения остальных параметров используется анализ HTTP-ответов и HTML-страниц средствами поиска регулярных выражений. Шаблоны для поиска задаются экспертом. Например, использование расширения .jsp в URI может свидетельствовать о том, что web-приложение было разработано с использованием технологии JSP.

Возможность получения идентифицирующей информации пользователем приложения является потенциальной уязвимостью ввиду следующих причин: в сети Интернет накоплены огромные объёмы данных по уязвимостям программных продуктов с указанием версий программ, методов реализации атак, бюллетени безопасности Bugtraq и Security Focus ежедневно публикуют отчёты о новых найденных уязвимостях. Эта информация предназначена для администраторов и специалистов в области информационной безопасности, в тоже время она доступна через сеть Интернет всем желающим. Администраторы информационных систем часто не устанавливают обновления безопасности вовремя. В результате становится возможным взломать информационную систему путем использования всем известной уязвимости, которую администратор не закрыл установкой соответствующего обновления. В последние годы на хорошо известные и незакрытые уязвимости операционных систем было проведено большое количество атак, вызвавших широко известные эпидемии Интернет-червей.

Метод получения идентифицирующей информации о web-приложении и выявления его уязвимостей с помощью бюллетеней безопасности получил широкое практическое применение при проведении атак ввиду своей простоты и доступности. При этом сам метод не позволяет найти новые уязвимости web-приложения, а его полезность для обнаружения уязвимостей новых приложений состоит в том, что он позволяет указать на саму возможность утечки информации о web-приложении.

Данный метод требует от эксперта настройки шаблонов, по которым производится поиск идентифицирующей информации. Эти шаблоны в большинстве случаев специфичны для конкретной технологии разработки web-приложений.

* 1. Метод тестирования на проникновение.

Метод тестирования на проникновение (penetration testing) рассматривает web-приложение с точки зрения внешнего пользователя, то есть потенциального злоумышленника. При этом считается, что злоумышленник обладает такими же возможностями, как и обычный пользователь, т.е. не имеет доступа к исходным кодам web-приложения, конфигурационным настройкам и т.п. Метод предусматривает тестирование работающего на стенде web-приложения путем посылки запросов которые эмулируют пользовательскую активность, включающую в том числе и некорректные запросы, соответствующие действиям злоумышленника.

При поиске уязвимостей в web-приложении методом тестирования на проникновение возникают три основные задачи :

* Получение и анализ структуры web-приложения.
* Построение набора тестовых HTTP-запросов на основе построенной структуры web-приложения.
* Прогон тестового набора с анализом ответов web-приложения для выявления уязвимостей.

Задача получения и анализа структуры web-приложения состоит в том, чтобы построить полный список URI web-приложения, методов доступа к ним и списков их параметров, выделить URI, защищённые аутентификацией. Данная информация необходима для построения набора тестовых запросов к web-приложению. Для автоматического получения структуры web-приложения используются сетевые роботы (web crawlers). В случае статических HTML-страниц работа робота сводится к обходу всех доступных ссылок webприложения, а в случае наличия форм и скриптов – к заполнению форм и извлечению гиперссылок из скриптов. Получение полной структуры web-приложения возможно не во всех случаях – основные проблемы возникают при заполнении web-форм и интерпретации скриптов.

Web-формы делятся на две категории: формы, содержащие только элементы с ограниченными областями значений (select, option и т.д.) и формы, содержащие элементы с неограниченными областями значений (например, textarea). Обход страниц, связанных с посылкой форм первого типа осуществляется последовательным перебором всех комбинаций значений полей формы. Для переходов, связанных с заполнением текстовых полей, возможны два способа обхода: ручной или автоматизированный. При ручном способе управление процессом передаётся человеку, а при автоматизированном способе применяются эвристические методы заполнения текстовых полей, основанные на словарях, например, VeriWeb, HiWE. Для форм, содержащих элементы с неограниченной областью значений, автоматизированный обход не гарантирует нахождения всех URI приложения.

Для автоматического обхода страниц, содержащих ссылки, формируемые средствами скриптовых языков, необходима возможность не только интерпретации скриптового языка, но и эмуляции действий пользователей для формирования тех или иных событий, например, onClick, onMove. Существующие средства интерпретации скриптовых языков не гарантируют выявления всех гиперссылок.

Задача построения тестового набора запросов к web-приложению состоит в том, чтобы по исходным данным (список URI приложения, методы доступа, принимаемые параметры) подобрать запросы так, чтобы было обнаружено как можно больше уязвимостей. Существующие способы построения таких запросов рассмотрены ниже.

Построение запросов по базе ресурсов подразумевает, что существует база ресурсов, которые потенциально могут встретиться в структуре web-приложения. Наличие в web-приложении того или иного ресурса из базы свидетельствует об уязвимости, связанной с возможным доступом к этому ресурсу. Например, в web-приложении присутствуют имена известных уязвимых CGI-сценариев и имена конфигурационных файлов web-серверов. Поиск ресурсов происходит по всей структуре web-приложения. В каждом каталоге (из структуры web-приложения) по очереди запрашиваются все имена, представленные в базе ресурсов. Способ построение запросов по базе ресурсов полностью автоматический и опирается на накопленную информацию о характерных уязвимостях web-приложений. Таким образом, данный способ имеет ограничения, аналогичные рассмотренным выше в методе получения идентифицирующей информации, но в тоже время позволяет выявить новые уязвимости web-приложения, основанные на типовых ошибках разработчиков и администратора.

Генерация запросов по шаблону с типизированными параметрами состоит в том, что для каждого URI задаётся шаблон, параметры которого типизированы, после чего происходит автоматическая генерация запросов по заданному шаблону со случайным выбором значений конкретных параметров. Значения параметров могут задаваться регулярными выражениями. Данный способ используется для обнаружения ошибок проверки корректности введенных пользователем данных. Критерий наличия уязвимости вводит пользователь – если ответ web-сервера соответствует некоторому заданному регулярному выражению, то делается вывод об уязвимости приложения. Описанным способом также реализуются переборы паролей. Данный способ требует привлечения эксперта и тонкой настройки, так как необходимо для каждого ресурса webприложения составить наборы значений параметров. При этом данный способ не зависит от технологии, на которой разработано webприложение, так как работает только в терминах протокола HTTP.

Анализ настроек каталогов web-приложения заключается в том, что по структуре web-приложения проверяются типовые уязвимости, связанные с неправильным конфигурированием webприложения и web-сервера. Сюда относятся проверка возможности автоматического построения индекса каталога, выполнения HTTPметодов PUT и DELETE, возможность обращения к ресурсам из областей аутентификации напрямую, возможность получения исходных кодов web-приложения.

Задача прогона тестового набора и анализа ответов сервера состоит с том, чтобы сделать правильный вывод о том, демонстрирует ли данный HTTP-запрос наличие уязвимости в web-приложении или нет. Данная задача тесно связана с задачей построения тестового набора, а основная проблема состоит в определении критериев наличия уязвимости. В настоящее время для решения этой задачи применяется метод, в котором распознавание осуществляется регулярными выражениями, задаваемыми экспертом. Таким образом, данный метод также требует тонкой настройки экспертом, при этом метод работает в терминах протокола HTTP и не зависит от технологии, на которой разработано web-приложение.

Метод тестирования на проникновение существенно меньше зависит от технологии разработки web-приложения, чем другие методы. Данный метод позволяет накапливать знания эксперта в виде набора правил построения запросов и набора шаблонов для анализа HTTP ответов. Этот метод получил широкое распространение для выявления уязвимостей разработанных web-приложений, когда необходимо оценить наличие хотя бы типовых ошибок при разработке и настройке web-приложения. Достоинством данного метода является то, что метод позволяет оценивать развернутое и настроенное web-приложение, выявляя не только ошибки кодирования, но и ошибки конфигурирования web-сервера и web-приложения.

* 1. Метод статического анализа.

Метод статического анализа исходных кодов web-приложения не предусматривает реального выполнения web-приложения. Вместо этого производится построение графов управления и зависимостей по данным и обнаружение уязвимостей посредством анализа этих графов. Для обнаружения уязвимостей используется два основных подхода: анализ типов безопасности и анализ потоков данных. В каждом из подходов уязвимость определяется, как нарушение в программе свойства noninterference.

При анализе типов безопасности вводится набор типов безопасности (t1, t2, …, tn), над которыми вводится отношение частичного порядка ≤. Каждой переменной программы ставится в соответствие её тип безопасности. Существуют два способа определения типов безопасности всех переменных – ручной и автоматический. Ручной способ предполагает, что при каждом объявлении переменной программист должен явно задать её тип безопасности. Автоматический способ предполагает, что даны: разметка типами безопасности переменных, содержащих входные данные, разметка функций, осуществляющих пользовательский ввод правила вывода типов безопасности ещё неразмеченных переменных. Таким образом, при анализе программы последовательно от начала до конца все неразмеченные переменные получают свой тип безопасности автоматически. Тип безопасности переменной постоянен на всё время жизни переменной.

Для выявления уязвимостей необходимо специфицировать, требуемые уровни безопасности параметров функций. В реализациях метода данная спецификация часто находится в отдельном конфигурационном файле и не зависит от конкретной программы (аннотируются стандартные библиотечные функции). Следующий пример иллюстрирует данный подход:

Типы безопасности: {untainted, tainted}

Начальная разметка: int main (int argc, tainted char \*argv[])

Сама программа:

void LogMessage(char \*fmt, …)

{

…

fprintf(fd, fmt, arg);

}

int main (int argc, tainted char \*argv[])

{

char \*arg2 = argv[2]; …

char \*Mess = new [strlen(arg2) + strlen(“arg2 = ”) + 1];

strcat(strcpy(Mess, “arg2 =”), arg2);

LogMessage(Mess);

}

Программа имеет уязвимость форматной строки. Методом анализа уровней безопасности данная уязвимость будет обнаружена, если функция fprintf будет аннотирована, как принимающая второй аргумент только уровня безопасности untainted.

В работе было доказано, что в языках со строгой типизацией данный подход обнаруживает все уязвимости, связанные с некорректными потоками информации.

Анализ типов безопасности имеет существенный недостаток – постоянную привязку типа безопасности к переменной. В результате тип безопасности определяется без учета того, из какого источника данные попали в переменную, что приводит к большому числу ошибок первого рода. Для того чтобы избежать большого числа ошибок первого рода, было предложено привязывать тип безопасности не к переменной, а к её значению. В результате каждая переменная теоретически может получать любой тип безопасности на протяжении своей жизни. Этот подход получил название анализа потоков данных.

В рамках анализа потоков данных каждой конструкции языка программирования сопоставляются формальные правила вывода результирующего типа безопасности переменных, участвующих в этой конструкции. А для библиотечных функций создается разметка, описывающая типы безопасности параметров и результата функции. На основе разметки и формальных правил вывода анализатор определяет в каждом участке кода тип безопасности конкретных данных. Сопоставляя аннотации библиотечных функций и типы безопасности их параметров, можно выявить несоответствия типа безопасности требованиям разметки, что указывает на наличие уязвимости в web-приложении.

В рамках данного подхода для части конструкций программы программист сам должен указывать тип безопасности переменных, например, при необходимости повысить тип безопасности после проверки корректности введенных данных.

Обнаружение уязвимостей web-приложений методом статического анализа предусматривает решение следующих основных задач:

* Определение конструкций языка и библиотечных функций, которые возвращают данные, потенциально контролируемые злоумышленником. Например, это параметры HTTP запросов GET и POST, пользовательские cookie и т.д. Вся информация, полученная из таких конструкций, помечается меткой tainted.
* Разработка аксиом распространения метки tainted между переменными приложения. Данные аксиомы должны поддерживать не только тривиальные присваивания, но и более сложные конструкции языка, такие, например, как присваивания через массивы, работу через ссылки и т.д. Данная система аксиом должна содержать правила, по которым информация, помеченная как tainted, может вновь стать безопасной, untainted. Полнота и точность метода обнаружения уязвимостей преимущественно зависит от качества модели распространения метки tainted по приложению.
* Определение конструкций языка, которые не должны принимать данные с типом безопасности tainted. В этот список должны входить функции для работы с системным окружением, с СУБД, с почтовыми сервисами и т.д.

Метод статического анализа исходных кодов web-приложения позволяет обнаруживать уязвимости, связанные с неустойчивостью web-приложения к некорректным входным данным. Другие классы уязвимостей данный метод обнаруживать не позволяет. Метод специфичен для каждой технологии создания web-приложений и требует от программиста аннотирования функций проверки корректности входных данных.

* 1. Метод динамического анализа

Метод динамического анализа исходных кодов web-приложения по своей сути аналогичен методу статического анализа исходных кодов за исключением того, что анализ производится в процессе выполнения web-приложения без построения графов программ. Вместо этого в процессе выполнения программы (т.е. неявно сформирован один из путей в графе управления) для каждой переменной производится вычисление типа безопасности, а для каждой размеченной функции сравниваются типы безопасности параметров со спецификацией, указанной в разметке.

Метод динамического анализа позволяет осуществлять обнаружение уязвимостей для широко применяемых при создании web-приложений динамических языков, как, например, Perl, PHP, Python, Ruby. В настоящее время для интерпретируемых языков webприложений метод динамического анализа часто интегрируется в интерпретатор языка, например, Perl Tainted Mode для языка Perl, PHPrevent для языка PHP и Ruby’s Tainted Mode для языка Ruby.

Основной проблемой метода динамического анализа является проблема снятия метки tainted с данных. Существует несколько способов решения данной проблемы: дополнительная разметка исходного кода, применение стандартных библиотеки «очищающих» функций из состава технологии, на которой разработано web- приложение, применение различного рода эвристик. Так, в технологии Perl любое регулярное выражение над tainted данными снимает с них метку tainted. Таким образом, Perl полностью доверяет программисту в проведённых очистках входной информации. В технологии Ruby программист должен явно указать после своих проверок изменение типа безопасности на untainted.

Метод динамического анализа, в отличие от статического анализа, позволяет осуществлять обнаружение уязвимостей в генерируемом на лету коде. В тоже время, метод динамического анализа анализирует лишь один из возможных графов выполнения программы и не позволяет сделать вывод об отсутствии уязвимостей во всей программе. Метод динамического анализа, аналогично статическому анализу специфичен для каждой технологии создания web-приложений.

1. **СРАВНЕНИЕ ИНСТРУМЕНТОВ ТЕСТИРОВАНИЯ БЕЗОПАСТНОСТИ.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | основные возможности | дополнительные | результаты php.testsparker.com | результаты premium.bgabank.com |
| OWASP ZAP | Man-in-the-middle Proxy | Dynamic SSL certificates | H: Advanced SQL Injection — AND boolean- | H: Server Side Include |
| free | Traditional and AJAX spiders | Smartcard and Client Digital Certificates support | based blind — WHERE or HAVING clause | H: Reflected Cross Site Scripting |
|  | Automated scanner | Web sockets support | M: X-Frame-Options Header Not Set | M: X-Frame-Options Header Not Set |
|  | Passive scanner | Support for a wide range of scripting languages | L: X-Content-Type-Options Header Missing | M: Application Error Disclosure |
|  | Forced browsing | Plug-n-Hack support | L: Web browser xss protection | M: Directory Browsing |
|  | Fuzzer | Authentication and session support | is not enabled | M: Secure Pages Include Mixed Content |
|  |  | Powerful REST based API |  | (Including Scripts) |
|  |  | Automatic updating option |  | L: X-Content-Type-Options Header Missing |
|  |  | Integrated and growing marketplace of add-ons |  | L: Web browser xss protection is not enabled |
|  |  |  |  | L: Cross-Domain JavaScript Source File Inclusion |
|  |  |  |  | L: Incomplete or No Cache-control and |
|  |  |  |  | Pragma HTTP Header Set |
|  |  |  |  | L: Cookie No HttpOnly Flag |
|  |  |  |  | L: Cookie Without Secure Flag |
|  |  |  |  | L: Content-Type Header Missing |
|  |  |  |  | L: Private IP Disclosure |
|  |  |  |  | I: Image Exposes Location or Privacy Data |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | основные возможности | дополнительные | результаты php.testsparker.com | результаты premium.bgabank.com |
| W9scan | Fingerprint detection | Сollect message | svn | possible condactions vectors xxe injection |
| free | Can identify common website CMS fingerprints ( 300+) | Emails (based on reptiles) | possible dowload payload's ways | xxs injection |
|  | Recognizable common website frame | Private IP (based on reptiles) | version of using services | services's version determined |
|  | Identify common port service fingerprints | E-mail (based on reptiles) | possible condactions vectors xxe injection | dirrectories's version determined |
|  | Detect website scripting language | Detecting Warnings, Fatal Error,… | xxs injection | subdomein's verion determined |
|  | Detect operating system type | PHP version identification | configuration server's files |  |
|  | Detection Website Firewall (WAF) | IIS information disclosure | subdomein search |  |
|  |  | IP address attribution |  |  |
|  | Attack parameter | Integrated Wappalyzer recognition script |  |  |
|  | SQL injection (based on crawlers) | robots.txt Analysis |  |  |
|  | XSS injection (based on reptiles) | Detecting unsafe headers in headers |  |  |
|  | A large number of Fuzz parameter scans | Detecting unsafe factors in cookies |  |  |
|  | CVE vulnerability |  |  |  |
|  | struts Vulnerability collection (including automatic detection) | Violent cracking |  |  |
|  | Shellshock cgi test | Backup files and directories (based on crawlers) |  |  |
|  | heartbeat Bleeding heart | Backup files and directories (based on domain name) |  |  |
|  | IIS parsing vulnerability | Common directory |  |  |
|  | IIS Put vulnerability | Common file |  |  |
|  |  | Subdomain violence analysis |  |  |
|  |  | fckeditorPath enumeration |  |  |
|  |  | Common mdbdatabase enumeration |  |  |
|  |  | git svn Leak identification |  |  |
|  |  | TOMCAT web.xml Give way |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | основные возможности | дополнительные | результаты php.testsparker.com | результаты premium.bgabank.com |
| Wapiti | file disclosure (Local and remote include/require, fopen, readfile...) | поддержка прокси (HTTP, HTTPs и SOCKS5 | Blind SQL Injection | Cross Site Scripting |
| free | Database Injection (PHP/JSP/ASP SQL Injections and XPath Injections) | поддержка различных методов аутентификации | Cross Site Scripting |  |
|  | XSS (Cross Site Scripting) injection (reflected and permanent) | (Basic, Digest, Kerberos, NTLM) | Commands execution |  |
|  | Command Execution detection (eval(), system(), passtru()...) | поддержка SSL-сертификатов |  |  |
|  | CRLF Injection (HTTP Response Splitting, session fixation...) | возможность добавления различных |  |  |
|  | XXE (XML External Entity) injection | HTTP-заголовков или настроек user-agent |  |  |
|  | SSRF (Server Side Request Forgery) |  |  |  |
|  | Use of know potentially dangerous files |  |  |  |
|  | Weak .htaccess configurations that can be bypassed |  |  |  |
|  | Presence of backup files giving sensitive information |  |  |  |
|  | Shellshock |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | основные возможности | дополнительные | результаты php.testsparker.com | результаты premium.bgabank.com |
| Arachni | active testing | passive testing | Cross-Site Scripting (XSS) in script context | H: Cross-Site Request Forgery |
| free | CSRF detection | HTTP PUT | Blind SQL Injection (differential analysis) | M: Mixed Resource |
|  | Code injection | Insufficient Transport Layer Protection for password forms | Code injection | M: HTTP TRACE |
|  | Blind code injection using timing attacks | WebDAV detection ( webdav). | Code injection (timing attack) | M: Common directory |
|  | LDAP injection | HTTP TRACE detection | Operating system command injection (timing attack) | M: Missing 'Strict-Transport-Security' header |
|  | Path traversal | Credit Card number disclosure | Operating system command injection | L: Private IP address disclosure |
|  | File inclusion | CVS/SVN user disclosure | H: File Inclusion |  |
|  | Response splitting | Private IP address disclosure | H: Cross-Site Scripting (XSS) in HTML tag |  |
|  | OS command injection | Common backdoors | H: Cross-Site Scripting (XSS) |  |
|  | Blind OS command injection using timing attacks | .htaccess LIMIT misconfiguration | H: Path Traversal |  |
|  | Remote file inclusion | Interesting responses | M: Backup file |  |
|  | Unvalidated redirects | HTML object grepper | M: Common directory |  |
|  | Unvalidated DOM redirects | E-mail address disclosure | M: HTTP TRACE |  |
|  | XPath injection | US Social Security Number disclosure | L: Missing 'X-Frame-Options' header |  |
|  | XSS | Forceful directory listing | L: Password field with auto-complete |  |
|  | Path XSS | Mixed Resource/Scripting | L: Insecure client-access policy |  |
|  | XSS in event attributes of HTML elements | Insecure cookies | L: Insecure cross-domain policy (allow-access-from) |  |
|  | XSS in HTML tags | HttpOnly cookies | L: Common sensitive file |  |
|  | XSS in script context | Auto-complete for password form fields. |  |  |
|  | DOM XSS | Origin Spoof Access Restriction Bypass |  |  |
|  | DOM XSS script context | Form-based upload |  |  |
|  | Source code disclosure | localstart.asp |  |  |
|  | XML External Entity | Cookie set for parent domain |  |  |
|  |  | Missing Strict-Transport-Security headers for HTTPS sites |  |  |
|  | plagins | Missing X-Frame-Options headers |  |  |
|  | Passive Proxy, Dictionary attacker for HTTP Auth, | Insecure CORS policy |  |  |
|  | Cookie collector, WAF Detector | Insecure cross-domain policy |  |  |
|  |  | Insecure cross-domain policy |  |  |
|  |  | Insecure client-access policy |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | основные возможности | дополнительные | результаты php.testsparker.com | результаты premium.bgabank.com |
| Paros | A1: Injection — SQLinjection, SQLinjection Fingerprint (места, где потенциально может быть SQLinj) | Поиск включенного автозаполнения для форм паролей. При этом если у поля input есть атрибут type=«password»,  получается ложное срабатывание. | H: SQL injection | M: Directory browsing |
| free | A6: Security Misconfiguration — Directory browsing, ISS default file, Tomcat source file disclosure, IBM WebSphere default files и некоторые другие стандартные или устаревшие файлы (﻿Obsolete file), содержащие исходный код и прочее. | CRLF injection | M: XSS | M: Использование автозаполнения в формах с важной информацией (пароли и прочее). |
|  | A7: XSS | Secure page browser cache (кэширование страниц в браузере c важной информацией) | M: ﻿Устаревшие файлы с исходным кодом |  |
|  |  | Возможность сканирования защищенной области пользователя (личный кабинет) | M: Использование автозаполнения в формах с важной информацией (пароли и прочее). |  |
|  |  | Возможность сканирования веб-приложений в локальной сети | L: Раскрытие внутренних IP |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | основные возможности | дополнительные | результаты php.testsparker.com | результаты premium.bgabank.com |
| Tenable.io | almost cover OWASP TOP 10 | сканирование сети на предмет известных уязвимостей | H: Уязвимости компонентов | H: Уязвимости компонентов |
| 250000 руб/год |  | сканирование сети на поиск хостов | — вышедшая из поддержки версия PHP | вышедшая из поддержки версия PHP |
|  |  |  | — вышедшая из поддержки версия Apache | уязвимости Apache |
|  |  |  | H: Code injection | уязвимости Bootstrap |
|  |  |  | H: SQLinj | уязвимости JQuery |
|  |  |  | H: XSS |  |
|  |  |  | H: LFI | M: Web server phpinfo() |
|  |  |  | H: Path Traversal | M: Совместное использование HTTP и HTTPS |
|  |  |  | M: Раскрытие ценных данных — полный путь, бекапы | M: Отсутствие перенаправления с HTTP на HTTPS |
|  |  |  | M: Раскрытие внутренних IP | M: Directory browsing |
|  |  |  | M: Cookie без флага HTTPOnly | M: Найдены backup файлы |
|  |  |  | M: Отправка пароля по HTTP | M: Использование небезопасной версии протокола SSL |
|  |  |  | L: Использование автозаполнения в формах с важной информацией | M: Истечение срока сертификата SSL/TLS |
|  |  |  | L: Ответ сервера на TRACE запросы | L: Раскрытие внутренних IP |
|  |  |  | L: Не установлены header`ы Cache-Control, X-Content-Type-Options, X-Frame-Options, X-XSS-Protection | L: Cookie без флага HTTPOnly |
|  |  |  |  | L: Ответ сервера на TRACE запросы |
|  |  |  |  | L: Не установлены header`ы Strict-Transport-Security, Cache-Control, X-Content-Type-Options, X-Frame-Options, X-XSS-Protection |

|  |  |  |  |
| --- | --- | --- | --- |
|  | основные возможности | результаты php.testsparker.com | результаты premium.bgabank.com |
| Burp Suite Pro | Proxy | H: OS command injection | H: Cross-site scripting (reflected) |
| 27500 руб/год | Spider | H: File path traversal | M: SSL cookie without secure flag set |
|  | Scanner | H: Out-of-band resource load (HTTP) | M: SSL certificate (not trusted or expired) |
| enterprise | Intruder | H: Server-side template injection | L: Cookie without HttpOnly flag set |
| 275000 руб/год | Repeater | H: Cross-site scripting (reflected) | L: Password field with autocomplete enabled |
|  | Sequencer | H: Flash cross-domain policy | L: Strict transport security not enforced |
|  | Decoder | H: Silverlight cross-domain policy |  |
|  | Comparer | H: Cleartext submission of password |  |
|  | Extender | H: External service interaction (DNS) |  |
|  |  | H: External service interaction (HTTP) |  |
|  |  | M: SSL certificate (not trusted or expired) |  |
|  |  | L: Password field with autocomplete enabled |  |
|  |  | L: Form action hijacking (reflected) |  |
|  |  | L: Unencrypted communications |  |
|  |  | L: Strict transport security not enforced |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | основные возможности | результаты php.testsparker.com | результаты premium.bgabank.com |
| Acunetix | SQL injection, | H: Apache 2.2.14 mod\_isapi Dangling Pointer | L: Clickjacking: X-Frame-Options header missing |
| за 1 сайт | Cross site scripting, | H: Blind SQL Injection |  |
| 300000 руб | CRLF injection | H: Cross site scripting |  |
|  |  | H: Cross site scripting (verified) |  |
| pro |  | H: Directory traversal |  |
| 500000 руб |  | H: File inclusion |  |
|  |  | H: PHP code injection |  |
|  |  | H: Server-side template injection |  |
|  |  | H: SVN repository found |  |
|  |  | H: User controllable script source |  |
|  |  | M: Access database found |  |
|  |  | M: Apache 2.x version older than 2.2.9 |  |
|  |  | M: Apache httpd remote denial of service |  |
|  |  | M: Apache httpOnly cookie disclosure |  |
|  |  | M: Application error message |  |
|  |  | M: Backup files |  |
|  |  | M: Directory listing |  |
|  |  | M: HTML form without CSRF protection |  |
|  |  | M: Insecure clientaccesspolicy.xml file |  |
|  |  | M: Partial user controllable script source |  |
|  |  | M: PHP hangs on parsing particular strings as floating point number |  |
|  |  | M: PHP preg\_replace used on user input |  |
|  |  | M: Source code disclosure |  |
|  |  | M: User credentials are sent in clear text |  |
|  |  | L: Apache 2.x version older than 2.2.10 |  |
|  |  | L: Apache mod\_negotiation filename bruteforcing |  |
|  |  | L: Clickjacking: X-Frame-Options header missing |  |
|  |  | L: Login page password-guessing attack |  |
|  |  | L: Possible relative path overwrite |  |
|  |  | L: Possible sensitive directories |  |
|  |  | L: Possible sensitive files |  |
|  |  | L: TRACE method is enabled |  |

1. **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**
2. <https://ru.wikipedia.org/wiki/Безопасность_приложений>
3. https://lvk.cs.msu.su/~ddk/pubs/methods\_for\_webapp\_vuln\_scanning.pdf
4. <https://habr.com/ru/company/owasp/blog/335820/>
5. <https://sibac.info/conf/technology/v/95451>
6. <http://www.infosecurity.ru/iprotect/websec/classification/>
7. <https://www.software-testing.ru/library/testing/testing-automation/2889-automation-testing-tools>
8. <https://itfb.com.ua/instrumenty-testirovaniya-bezopasnosti/>
9. <https://habr.com/ru/company/tomhunter/blog/456892/>