УДК 004.056.53

Р.Х. Хамдамов, К.Ф. Керимов

МЕТОДЫ БЛОКИРОВАНИЯ УЯЗВИМОСТЕЙ ВИДА XSS НА ОСНОВЕ СЕРВИС-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЫ

Ключевые слова: уязвимость, XSS, выявление угроз, адаптивный анализ.

Введение

Веб-приложения разрабатываются на нескольких языках и развертываются в различных операционных системах. Это связано с различными функциями, которые веб-приложение предоставляет своим пользователям. Приложения электронной коммерции должны учитывать различные интерфейсы, необходимые для взаимодействия, безопасности и доступности веб-приложения. Следовательно, приложения разрабатываются с использованием различных языков, таких как PHP, ASP, JSP, .NET, Python и т.д., на основе требований веб-приложения. Приложения постоянно проверяются на наличие уязвимостей, и если они оказываются уязвимыми, могут подвергнуться атакам. Недавние исследования показывают [1], что атаки на веб-приложения увеличиваются, так как запускаются на порт 80, который остается всегда открытым. SSL и брандмауэры неэффективны против атак на уровне приложений, поскольку не могут предотвратить атаки на порт 80. Эти атаки могут вывести из строя сервер веб-приложений, а также получить доступ к внутренним базам данных, содержащим конфиденциальную информацию: номера кредитных карточек клиентов, информацию об учетной записи и личную информацию.

Уязвимость веб-приложения при атаках XSS

Пользователи взаимодействуют с веб-сайтами, переходя по ссылкам или заполняя и отправляя HTML-формы в браузере. В результате на сервер отправляется список пар «имя–значение» в виде http-запроса. Запрос может содержать другую информацию, такую как список файлов cookie, URL-адрес источника ссылки и т.д. Уязвимость вида XSS использует гиперссылки или клиентские скрипты, такие как JavaScript, VBScript, ActiveX, XHTML, Flash и другие веб-приложения [2, 3]. Злоумышленник, использующий атаку вида XSS (Cross-Site scripting — Межсайтовый скриптинг), обычно прибегает к механизму внедрения вредоносного кода в сеанс пользователя или целевой веб-сервер, чтобы перенаправить пользователя на вредоносную гиперссылку или вызвать сценарий, получающий данные у всех пользователей, которые заходят на уязвимый веб-ресурс. Эта XSS-атака потенциально приводит к перехвату информации учетной записи пользователя, краже cookie-файлов или информации о сеансе, модификации пользовательского контента, порче веб-сайта, изменению привилегий пользователей и т.д. [4].

Приведем возможности, с помощью которых можно внедрить XSS:

- любые способы отправки данных на сервер методом GET и POST.
- все процедуры, возвращающие данные в браузер, такие как сообщения об ошибках, информация для пользователей и предупреждения.

Современное состояние решения проблемы

Ранее исследователи предлагали довольно много серверных решений [5], механизмов безопасности на основе информационных потоков [6], клиентских решений [7], сканеров и инструментов для устранения уязвимостей вида XSS [8].

Решения на стороне клиента, предложенные исследователями [9], полностью опираются на конфигурацию пользователя, а количество исследований доказало, что решение на стороне клиента не является надежным [10]. Данные исследований показывают, что число атак с нулевой датой увеличивается [11], и, следовательно, решение на стороне клиента не может предотвратить новые угрозы с немедленным эффектом.

Как видно из вышесказанного, ни одно решение не предназначено для использования механизмов безопасности, необходимых веб-приложениям, которые разрабатываются на различных языках и с различными целями. Например, одно приложение может разрешить теги во входных данных, в то время как другие приложения могут не разрешать их. Исследования показывают, что около 80 % веб-приложений [12] уязвимы для XSS-атак из-за вышеуказанных фактов. В данной статье авторы предлагают сервис-ориентированную архитектуру для предотвращения таких уязвимостей на разных языках. Решение разработано с использованием языка PHP, расширяемого языка разметки XML и определений схем XML (XSD) и протестировано в веб-приложении, разработанном в JSP/Servlets, развернутом на сервере JBOSS.

Предлагаемая процедура решения

Решение направлено на предоставление независимых сервисов с определенными интерфейсами, которые могут быть вызваны для выполнения их задач стандартным способом, без предварительного знания сервисом вызывающего приложения и без знания приложением того, как сервис фактически выполняет свои задачи. Решение основано на подходе сервис-ориентированной архитектуры (SOA, Service-oriented architecture). В литературе SOA определяется как слабо связанные программные сервисы для поддержки требований бизнес-процессов и пользователей программного обеспечения. Эти сервисы взаимодействуют на основе формального определения, независимого от базовой платформы и языка программирования. Процедура решения использует XML и XSD для взаимодействия между сервисами.

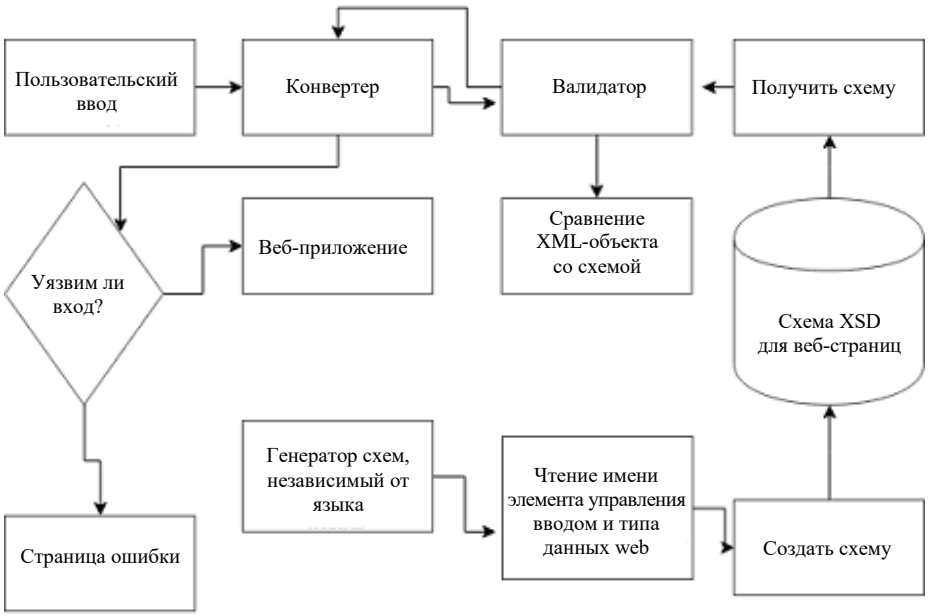
Разработан метод блокирования уязвимостей вида XSS, в основе которого лежит возможность защиты приложений от XSS-атак с помощью XML и XSD. Это включает в себя создание XML-документа на основе всех элементов управления формы, представленных пользователем. Этот XML-документ будет проверен на соответствие схеме на стороне сервера. Любой вредоносный сценарий приведет к созданию недопустимого или неправильно сформированного XML-файла и, таким образом, не позволит отправить вредоносные сценарии.

Техническое проектирование предлагаемого подхода

На рисунке показана схема работы метода блокирования уязвимостей вида XSS на основе сервис-ориентированной архитектуры.

Рассмотрим подробно каждый элемент схемы.

Конвертер. Это — компонент интерфейса между приложением и пользователем. Запросы http настроены на отправку запросов преобразователю, который преобразует объект запроса в XML-пару «значение–имя». Затем этот объект XML передается компоненту средства проверки. Результатом проверки является состояние уязвимости входа, что приводит к следующему действию конвертера. Если входные данные допустимы, конвертер передает их веб-приложению. В противном случае он выдает исключение и выполняет запрограммированное действие, если запрос признан недействительным.



Валидатор. Схема для элементов управления вводом генерируется генератором схемы и хранится в репозитории. Валидатор получает объект XML-запроса от преобразователя и соответствующую схему для запроса. Валидатор проверяет входные данные, указанные как пара «значение–имя» в объекте XML, и путем сопоставления ограничений схемы проверяет их на уязвимость. Результат валидатора отправляется на вход компонента конвертера, что приводит к дальнейшим действиям для входа.

Приложение-генератор схем. Документ схемы XML создается с помощью языка PHP. Это основная часть предлагаемой архитектуры, и приложение-генератор схем генерирует схему для каждой страницы на основе входных данных, предоставленных разработчиком; сгенерированная схема хранится в файловой системе или базе данных. Когда валидатор получает объект XML, который необходимо проверить на уязвимость вида XSS, он извлекает соответствующую схему веб-страницы из репозитория и проверяет входные данные на основе правил, указанных в схеме.

Форма входных данных получает все имя входного элемента управления и тип данных имени входного элемента управления веб-страницы от разработчика для создания документа схемы для этой веб-страницы.

Существует восемь параметров, принятых с помощью формы для генерации схемы документа. Описание входных параметров представлено в табл. 1.

Таблица 1

Имя параметра	Описание
Имя ввода	Имя элемента экземпляра документа (например, имя пользователя)
Пример входного значения	Пример значения для указанного выше входа (например, текст-образец)
Тип данных	Тип данных входного параметра (например, строка). В настоящее время поддерживаются типы данных «string», «integer», «decimal»
Минимальные, максимальные значения	Диапазон значений входного параметра (например, – 100, 100). Для типа входных данных «string» минимальные и максимальные значения переводятся как минимальная и максимальная длины строки. Заголовок столбца изменяется соответствующим образом после выбора типа данных

Продолжение таблицы 1

Входной формат	Любые конкретные ограничения формата для входного значения (например, SSN, номер кредитной карточки etc). Гибкость встроена, чтобы принимать регулярные выражения
Особые символы	Если функции приложения требуют специальных символов, которые должны рассматриваться как допустимые входные данные
Разметка разрешена	Это логическое значение, которое устанавливается как *true*. Если входные значения должны содержать какой-либо размеченный текст, это можно сделать, установив флажок. Значение по умолчанию — *false*
Обязательное значение	Это логическое значение, которое устанавливается как *true*. Обязательный ввод должен произойти во входном сообщении для успешного подтверждения сообщения

Генератор схемы генерирует правило для поля User name, чтобы принять теги, проверяется атрибут Mark-up allowed для поля User name. Но для поля Password атрибут Mark-up allowed не отмечен, и, следовательно, правило, сгенерированное генератором схемы, должно запретить теги, введенные в поле Password.

Подход генератора схем основан на регулярных выражениях, и, следовательно, при генерации схемы ограничения генерируются автоматически и включаются в схему, которая используется валидатором для проверки входных данных на наличие вредоносных шаблонов. Разработано семь методов, включенных в генератор схем.

Опишем функциональные возможности методов.

• RElement () — создает узел элемента для запроса. Это сложный элемент, так как содержит другие элементы и атрибуты.

• MElement () — обрабатывает имена и значения элементов данных. Создает правила, используя следующие функции, основанные на типе данных указанного ввода.

• SaveData () — вызывается для сохранения схемы в базе данных или по указанному разработчиком пути.

• CreateTWM () — генерирует минимальный и максимальный фасеты для параметра, заданного разработчиком, через форму входных данных. Здесь в форме ввода данных для поля User name допустимо количество символов от 10 до 60.

• TypeFN () — генерирует фасеты регулярного выражения для integer и decimal.

• BaseTypeFS () — создает шаблоны регулярных выражений в строке ввода.

Содержимое строк ограничено, поэтому не может содержать теги <script> </script>, а также другие функции скрипта, используемые, в основном, для внедрения уязвимости XSS.

Ниже приведены ограничения, если разработчик не выбрал отметку:

Pattern = @ «^ (^ (<\ s * (\ S +) (\ S [^>] *)?> [/ S / S] * <\ c * \ / \ 1 \ s *>)) $».

Значения шаблона приведены в табл. 2.

Таблица 2

Значения шаблона	Функция, адресуемая значением шаблона
@»^	С начала строки ввода
(^(	Отрицание соответствия — отвечает всем, кроме тегов
<	Начало определения тега (& lt;)
\s*	Соответствие нулю или нескольким символам пробела

Когда разметка разрешена, создается другой шаблон регулярного выражения, чтобы предотвратить основные теги, такие как <Script> и др., которые помогают выполнять функции сценария.

Когда входные данные предоставлены, как указано в табл. 1, создается документ экземпляра XML, который сохраняется и отображается для проверки, и его схема проверки. Созданная схема используется для проверки содержимого входных данных, введенных пользователем на веб-странице.

Взаимодействие компонентов

Ниже приведены действия, выполняемые до и после получения HTTP-запроса на стороне сервера.

- Схема для каждой веб-страницы, на которой присутствует элемент управления вводом, создается и хранится разработчиком в автономном режиме в структуре папок или базе данных.
- При получении запроса http-запрос передается преобразователю.
- Конвертер преобразует входные данные в объект XML и отправляет его в валидатор.
- Средство проверки извлекает соответствующую схему для запроса и сопоставляет объект XML с документом схемы. Если входные данные сопоставляются со схемой, то преобразователю возвращается состояние «да», в противном случае — «нет».
- Если статус «да» получен от валидатора, то запрос перенаправляется в веб-приложение. В противном случае — на страницу ошибки.

Заключение

Большое количество веб-приложений уязвимы для XSS-атак. Преимущества данного подхода заключаются в следующем. Основная часть системы защиты от уязвимостей вида XSS не зависит от платформы и языка. Методы, разработанные авторами, позволяют создать различные системы защиты, основанные на технологиях XML и XSD. Предлагаемый подход строится на модульном принципе, а также на основе шаблонов регулярных выражений.

Р.Х. Хамдамов, К.Ф. Керімов

МЕТОДИ БЛОКУВАННЯ ВРАЗЛИВОСТЕЙ ВИДУ XSS НА ОСНОВІ СЕРВІС-ОРІЄНТОВАНОЇ АРХІТЕКТУРИ

Веб-додатки розробляються на декількох мовах і розгортаються на різних операційних системах. Це пов'язано з різними функціями, які веб-додаток надає своїм користувачам. Додатки електронної комерції повинні враховувати різні інтерфейси, необхідні для взаємодії, безпеки та доступності веб-додатків. Отже, додатки розробляються з використанням різних мов, таких як PHP, ASP, JSP, .NET, Python і т.д., на основі вимог веб-додатків. Вони постійно перевіряються на наявність вразливостей, як тільки стають вразливими, можуть піддатися атакам. Дані досліджень показують, що близько 70 % веб-додатків уразливі для атак виду XSS. Це пов'язано з тим, що користувачам дозволено вводити дані в текстових полях в формах веб-додатків, що збільшує загрозу для веб-додатків та дозволяє хакерам впроваджувати шкідливий вміст в них. В статті представлено нове рішення для блокування атак XSS, яке не залежить від мов, на яких розробляються веб-додатки, і усуває вразливості XSS, що виникають з інших інтерфейсів. Рішення направлено на надання незалежних сервісів з певними інтерфейсами, які можуть бути викликані для виконання їх завдань стандартним способом, без попереднього знання сервісом викликаного додатку і без знання додатком того, як сервіс фактично виконує свої завдання. Рішення базується на підході сервіс-орієнтованої архітектури SOA. Розроблено метод блокування вразливостей виду XSS, в основі якого лежить можливість захисту додатків від XSS-атак за допомогою XML і XSD. Це включає в себе створення XML-документа на основі всіх елементів керування форми, представлених користувачем.

Ключові слова: вразливість, XSS, виявлення загроз, адаптивний аналіз.

R.Kh. Khamdamov, K.F. Kerimov

METHODS OF BLOCKING THE VULNERABILITIES OF THE XSS TYPE BASED ON SERVICE-ORIENTED ARCHITECTURE

Web applications are developed in several languages and deployed in various operating systems. This is due to the various functions that the web application provides to its users. E-commerce applications must take into account the various interfaces required for interoperability, security, and availability of a web application. Consequently, applications are developed using various languages, such as PHP, ASP, JSP, .NET, Python, etc. based on web application requirements. Applications are constantly checked for vulnerabilities, and when they are vulnerable, they can be attacked. Research data shows that about 70 % of web applications are vulnerable to attacks from the XSS form. This is due to the fact that users are allowed to enter data in text fields in web application forms. This increases the threat to the web application, allowing hackers to embed malicious content into the web application. This article presents a new solution for blocking Cross-Site Scripting (XSS) attacks, which does not depend on the languages in which web applications are developed, and eliminates XSS vulnerabilities arising from other interfaces. The solution aims to provide independent services with specific interfaces that can be invoked to perform their tasks in a standard way, without prior knowledge of the calling application by the service and without the application knowing how the service actually performs its tasks. The solution is based on a service-oriented architecture (SOA) approach. A method has been developed for blocking vulnerabilities of the XSS type based on the ability to protect applications from XSS attacks using XML and XSD. This includes creating an XML document based on all form controls submitted by the user.

Keywords: visibility, XSS, threat identification, adaptive analysis

1. Opanasenko V.N., Kryvyi S.L. Synthesis of adaptive logical networks on the basis of Zhegalkin polynomials. *Cybernetics and Systems Analysis*. 2015. **51**, 6. P. 969–977. DOI: 10.1007/s10559-015-9790-1.
2. Керимов К.Ф. Модель выявления угроз информационной безопасности в электронных ресурсах. *Перспективы развития техники и технологии и достижения горно-металлургической отрасли за годы независимости Республики Узбекистан*: Тез. докл. Респ. науч. конф. 12-14 мая 2011. Навои. С. 339–340.
3. Козлов Д.Д., Петухов А.А. Методы обнаружения уязвимостей в web-приложениях. *Программные системы и инструменты*. 2006. № 7. С. 156–166.
4. Кондрашова Н.В. Согласование внешнего критерия и способа разбиения выборки при решении задачи структурно-параметрической идентификации методом группового учета аргументов. *Международный научно-технический журнал «Проблемы управления и информатики»*. 2015. № 5. С. 20–33.
5. Низамутдинов М.К. Тактика защиты и нападения на ИТ-приложения. Санкт-Петербург : БХВ-Петербург, 2005. С. 10–30.
6. Пазизин С.В. Основы защиты информации в компьютерных системах. М. : ТВП-ОпиПМ, 2003. 73 с.
7. Петренко С. А., Петренко А. А. Аудит безопасности Intranet. М. : ДМК Пресс, 2002. 416 с.
8. Ржавский К.В. Информационная безопасность: практическая защита информационных технологий и телекоммуникационных систем: Учебное пособие. Волгоград : ВолГУ, 2002. 122 с.
9. Рябко Д.М. Подход к тестированию уязвимости web-приложений от атак типа SQL-инъекций. УкрПРОГ, Киев, Украина, 2006. 585 с.
10. Керимов К.Ф., Салахутдинов В.Х. Методика оценки риска информационной безопасности электронных ресурсов компьютерной сети при угрозах несанкционированного доступа. *Проблемы информатики и энергетики*.. 2018. № 5. 86 с.
11. Хорев П.Б. Методы и средства защиты информации в компьютерных системах. М. : Гелиос, 2006. 62 с.
12. Керимов К.Ф., Мухсинов Ш.Ш., Исматуллаев С.О. Брандмауэр баз данных, основанный на обнаружении аномалий. *Проблемы информатики и энергетики*. 2015. № 3. 63 с.

Получено 01.04.2019
После доработки 25.05.2019