Тема 3. XSS

Итак, мы разобрались с первым бастионом защиты пользователя от злодеев и жуликов — самим браузером. Дальше посмотрим, какие ещё опасности могут нас поджидать.

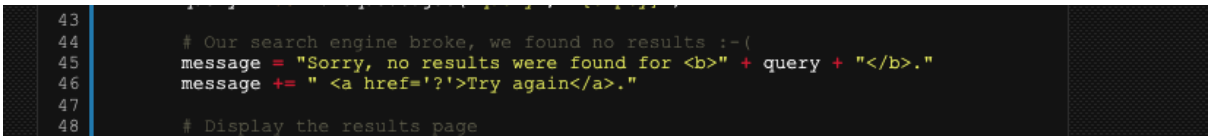
А поджидать нас будет самая распространённая проблема на стороне клиента — Cross-site-scripting. Сайт OAWSP из года в год включает инъекции (в том числе и SQL) в список десяти самых частых уязвимостей.

Как только браузер научился работать с куками и сессиями, сразу же появились желающие этими кукис злонамеренно воспользоваться. А JS-код отлично умеет работать с этой сущностью браузера. Основная задача этой атаки — заставить выполниться вредоносный код на стороне атакуемого, причём в контексте изначально домена. Например, пользователь заходит на сайт своего банка, вбивает пароль, но в ту же секунду начинает происходить что-то странное — и вжух, денег опять как не бывало.

Выделяют три вида XSS-атак: Reflected (отражённые атаки), Stored (хранимые атаки) и атаки на DOM. Известная компания Google даже сделала игру, где задача пользователя — провести одну из таких атак. Давайте и мы в неё немного поиграем.

Первый тип атаки, как и первый уровень игры, — это Reflected XSS. Это так называемые отражённые, или моментальные атаки. Эти атаки невозможно провести без активного участия жертвы, то есть злоумышленнику помимо навыков программирования ещё нужно быть обаятельным малым и иметь навыки социальной инженерии. Посмотрим на примере игры.

На первом уровне нас приветствует вымышленный поисковик, и наша задача — запустить JS-код, а именно вывести в браузере alert-команду — всплывающее окно с текстом. Тут мы можем посмотреть на пример кода бэкенда и внимательно его прочитать. Отражённый тип атаки предполагает, что жертва сама приводит вредоносный код в действие. Если мы посмотрим на 45-ю строчку кода сервера, то увидим там интересное выражение, которое говорит: извините, ничего не нашлось по такому-то запросу.



Для начала попробуем отправить туда какой-нибудь валидный запрос.

Sorry, no results were found for **смысл**. Try again.

Работает. Теперь включаем режим жулика и пробуем отправить туда что-то менее безобидное. Например, тег script — обычный HTML-тег, который содержит внутри себя JS-код. Этот код выполнится в момент загрузки страницы.

```
<script>alert("Hacked!!")</script>
```

И вот жулики победили. Понятно, что способов провести такую атаку миллион, но тут главное уловить суть — жертва сама инициализирует атаку на свой браузер посредством запроса на сервер. Запрос уже содержит вредоносный код, и именно этот код возвращается пользователю.

Идём на следующий уровень и к следующему типу XSS-атак. Хранимые процедуры XSS — это когда злоумышленники делают так, что их вредоносный код начинает где-то храниться, чаще всего это база данных. Например, как в игре, это могут быть комментарии какого-нибудь блога. То есть хакер пишет некий комментарий, но непростой, а с JS-инъекцией внутри. И все последующие пользователи сайта будут загружать себе эту инъекцию.

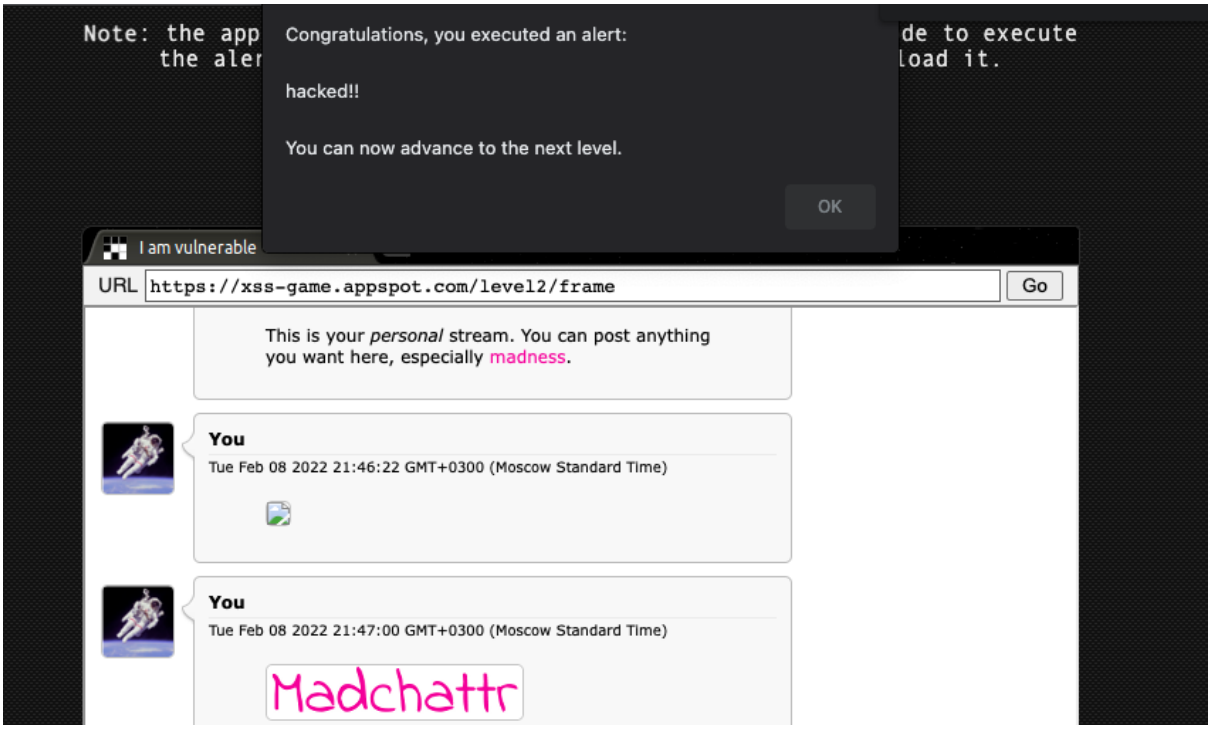
Попробуем пройти следующий уровень.

Тут у нас блог и уютное предложение что-нибудь запостить. Давайте, прямо как по заветам Григория Остера, запостим тут вредоносный XSS:

```
<img src="/static/logos/level2.png" onload="alert('hacked!!');">
```

Запостим тег картинки с лого сайта и заполним атрибут onload нашим кодом. Этот тег выполнится в момент успешной загрузки картинки.

И действительно, мы смогли сохранить в базе JS-код, и теперь все посетители этого вымышленного сайта будут видеть, что мы тут побывали. Заметим, что браузер не будет исполнять тег script после загрузки страницы, поэтому эта атака тут не поможет.



Следующий тип атак — манипуляция с DOM (Document Object Model). Это то, как JS работает с HTML-тегами и страницей браузера вообще. Код выполняется на странице жертвы, и сама HTML-страница не меняется, как в предыдущем типе, но сам код веб-страницы начинает исполняться несколько неожиданным образом.

Попробуем взломать следующий уровень. Тут несколько фреймов, и наша задача — вывести alert. Для прохождения этого уровня уже потребуются базовые знания JS, так что перейдём сразу к самому интересному.

Обратим внимание, что на 37-й строчке кода выполняется следующая процедура:

```
chooseTab(unescape(self.location.hash.substr(1)) || "1");
```

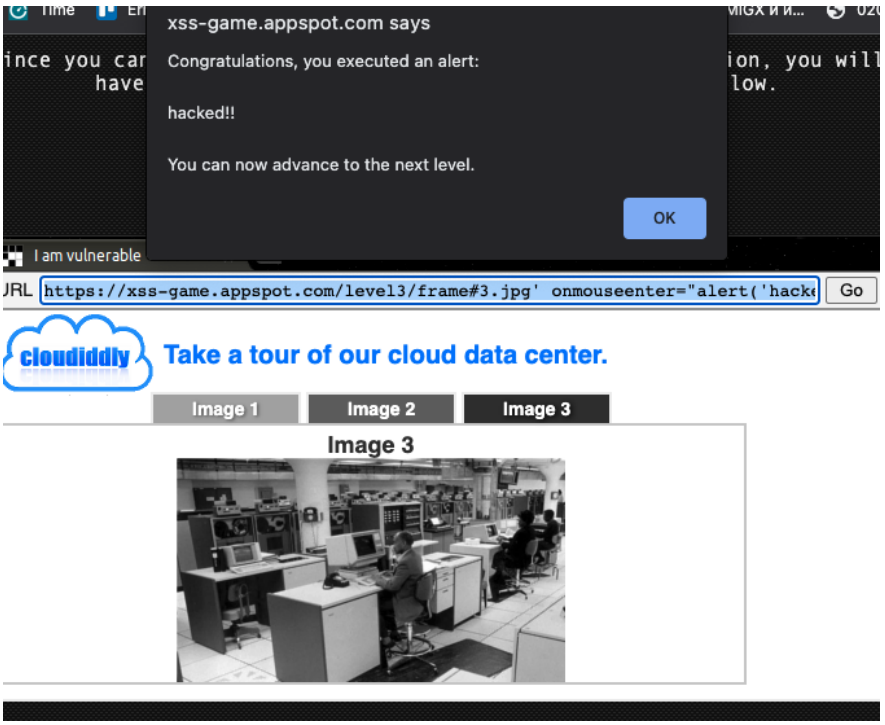
Тут мы, по сути, парсим браузерную строку и достаём то, что находится за знаком диеза. Далее это используется для генерации ссылки на картинку. Отличное место, чтобы вставить инъекцию!

Например, мы можем сделать вот такую хитрую атаку:

```
https://xss-game.appspot.com/level3/frame#3.jpg'
onmouseenter="alert('hacked!!')"
```

Так мы воспользуемся встроенным в JS ивентом, который сработает при наведении мышки на картинку, — в итоге должна выскочить всплывашка.

Попробуем. Как и ожидалось, работает.



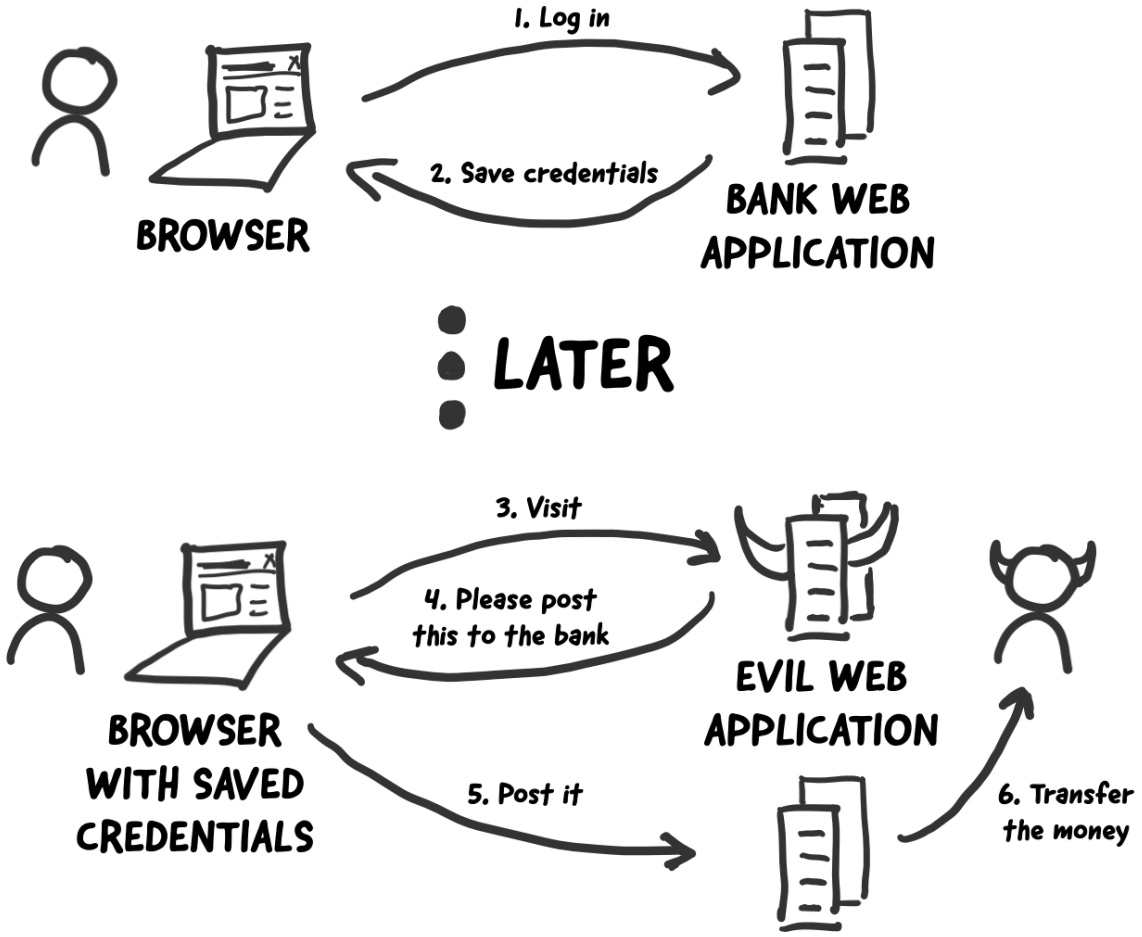
Как мы можем от этого себя защитить? От большинства атак, где мы напрямую работаем с HTML, уже есть какая-никакая защита. Что в темплейтах Flask, что в джанго. При рендеринге нам достаточно не выключать стандартное поведение и знать парочку команд при работе с jinja — подробнее смотрите по ссылке.

Плюс в защите от XSS может помочь настройка content security policy. Это похожая по смыслу и по реализации на CORS защита: мы позволяем запускать JS/CSS-скрипт, только полученный из определённых источников, в том числе источников в документе. Например, можно запретить исполнение inline-скриптов, которые мы писали ранее, а разрешить только те, что подключаются в виде файлов в момент рендера страницы, причём только из указанных нами доменов. Подробнее про этот механизм смотрите по ссылке.

Ну и конечно, лучшая защита от атак, где злоумышленники выходят с вами на прямое взаимодействие, — ваш ум и осторожность.

Ещё один тип атаки, схожий по сути, но достаточно сильно отличающийся по реализации — CSRF (Cross Site Request Forgery). На русский язык это можно перевести как межсайтовая подделка запроса.

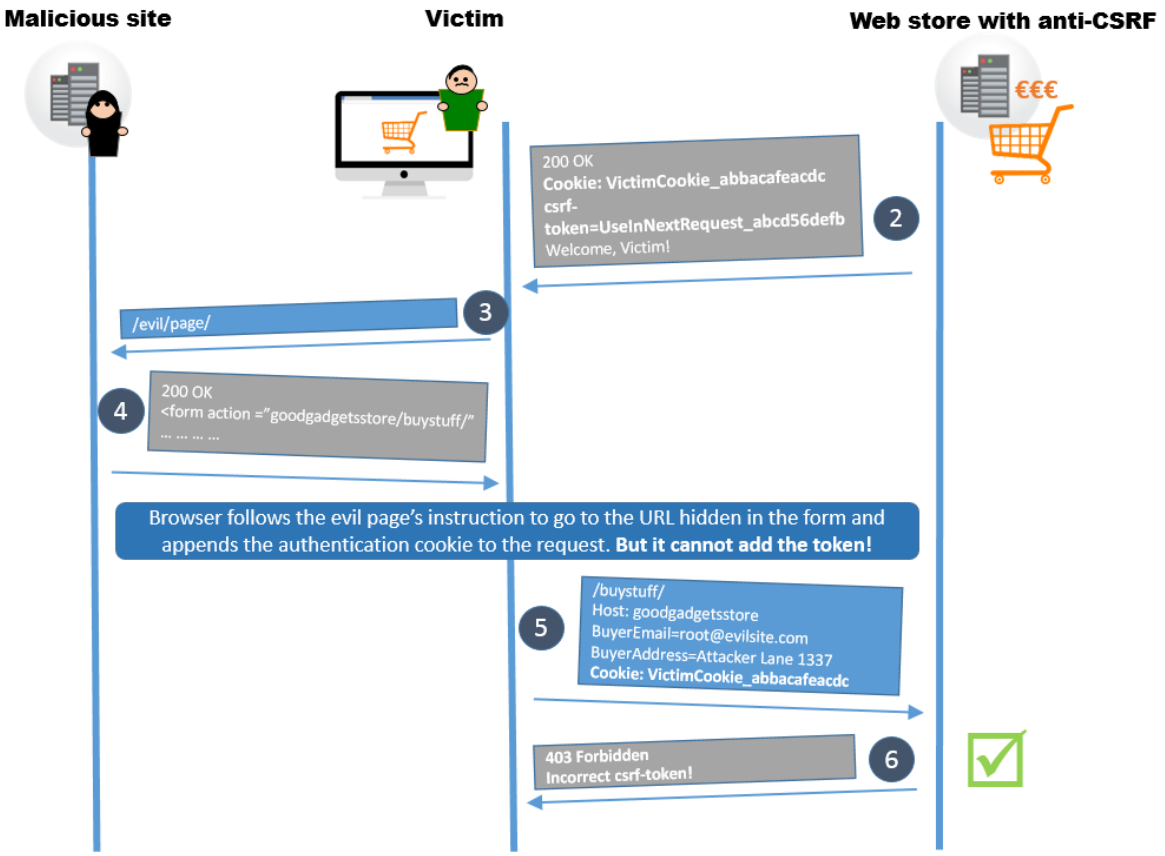
Как это работает? Злоумышленник так или иначе завлекает нас на свой сайт. Например, всё та же жёлтая реклама от бабушки. Мы перешли на сайт, при этом мы всё ещё авторизованы в нашем банковском аккаунте на уровне браузера. Сайт может выглядеть вполне безобидно, мы с удовольствием читаем его контент, например решили заполнить HTML-форму на этом сайте.



Но оказывается, эта форма ведёт не на бэкенд открытого сайта, а, например, на сайт вашего банка (к счастью, мы не можем отправить деньги кому-то обычной HTML-формой, но для примера сойдёт) — и, по старой традиции, прощайте ваши денежки. Всё дело в том, что браузер не очень умеет думать о контексте отправки запроса. Например, его логика работает так: «Мы отправляем запрос на сайт банка, а у меня как раз хранятся к нему кукис и токен сессии. Думаю, это отличная идея добавить их к запросу». Да, тут другой origin, но, как мы знаем из темы про CORS, он всё равно его отправит, хоть и не отрендерит ответ, но это тут и не нужно.

От этого есть несколько типов защиты. Самая известная и базовая, и мы даже с ней уже встречались, когда работали с wtforms, — CSRF token.

Для каждой сессии пользователя бэкенд создаёт токен, причём такой, что его сложно подделать. Этот токен напрямую помещается в HTML-код страницы как невидимый атрибут формы. Теперь бэкенд ожидает получить этот токен или в теле запроса, или в заголовке и таким образом верифицировать его. Атакующий не знает этого токена, ведь он не рендерил HTML-страницу с ним, и поэтому рассмотренная выше атака не будет работать.



В прошлый раз мы эту защиту выключили, потому что ручная отправка токена довольно сложное занятие, когда нам нужно тестировать сервис руками. По ссылке вы можете найти документацию wtforms насчёт работы с CSRF token. Тот же самый механизм реализован и в джанго.

Большинство прочих видов защиты предполагают более сложную работу с явными системами аутентификации. Например, session token / JWT в header или session token в body param. Разбор этих подходов несколько выходит за рамки курса, так что смотрите по ссылке.

Правда, если сайт уязвим к XSS-атакам, то все эти защиты против CSRF можно будет обойти. Ведь и правда, если мы можем выполнить произвольный скрипт на стороне клиента, то достать секретное значение для отправки формы будет совсем просто.

Итак, теперь вы точно знаете, что подвохов и опасностей в интернете полно. Но знать их в лицо уже большое дело! И со многими из них мы можем справиться. Впереди чуть-чуть практики, и далее поговорим ещё об одном типе атак.