! Scalability

- HTML
- HTTPS
- Database
  - APIs and environment
- JavaScript
  - CSRF

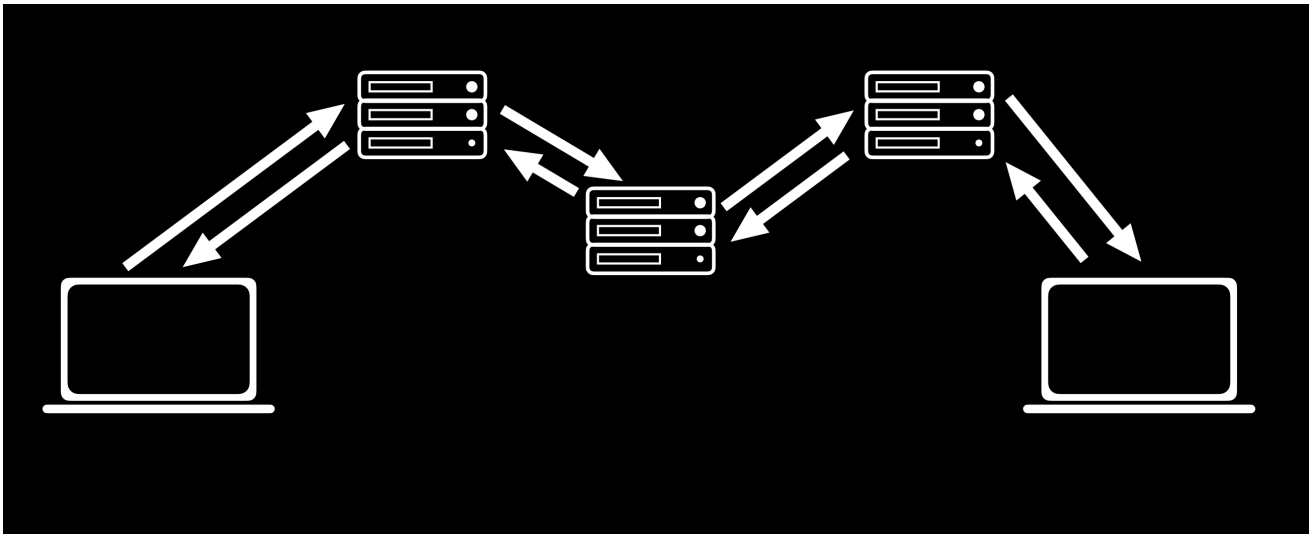
Вступление

Теперь мы изучим как улучшить безопасность нашего сайта.

> HTML
>
> # HTML
>
> При работе с html есть множество способов взлома аккаунта. Самый простой метод - это *фишинг атака*. Она хоть и напрямую не связана с разработкой, но очень важно понимать это при работе в интернете.
> Например такой html код является фишингом:
>
> ```
> !DOCTYPE html>
> <html lang="en">
>     <head>
>         <title>Link</title>
>     </head>
>     <body>
>         <a href="https://cs50.harvard.edu/">https://www.google.com/</a>
>     </body>
> </html>
> ```

> HTTPS
>
> # HTTPS
>
> Ранее мы говорили о протоколе http, но сейчас почти все сервисы находятся на протоколе https.

HTTPS - это *зашифрованный http*. Как мы знаем, чтобы данные дошли до нужного места, они должны перейти еще по множеству серверов, как показано ниже. В таком случае данные могут перехватить и узнать их.



Все данные передаваемый с компьютера по https будут шифроваться и при перехвате пакетов данных их значение злоумышленику не узнать.

Для шифрования данных используются методы **секретных и публичных** ключах.

Криптография секретных ключей

В таком случае получатель и отправитель имеют секретный ключ, которые знают только они. В таком случае этим ключом можно зашифровать сообщение и разошифровать его. Это очень безопасно, правда на практике это тяжело применимо.

Криптография публичных ключей

Одним из величайших открытий в криптографии - это метод шифрования публичным ключом. Благодаря нему интернет функционирует так, как мы это видим сейчас. В данном методе, у нас 2 ключа: *публичный*, который открыт для всех и *приватный*, который доступен только 1 пользователю. После создания пары ключей математическим методов, данные шифруются публичным ключом, а приватным они расшифруются. *HTTPS шифрует данные публичным ключом, а приватный ключ есть только у нас*.

Database

Базы данных и хэш

Так-же использую наше приложения мы должны убедиться, что наша база даных безопасна. В базе данных нельзя просто хранить скрытые данные, как показано в изображение ниже:

id	username	password
1	harry	hello
2	ron	password
3	hermione	12345
4	ginny	abcdef
5	luna	qwerty

Чтобы обезопасить нашу базу данных, надо использовать **хэш функции**. В таком случае наша база данных не будет иметь строку с паролем, а лишь хэш данного пароля.

id	username	password
1	harry	48c8e8c3f9e80b68ac67304c7c510e9fcb
2	ron	6024aba15e3f9be95e3c9e6d3bf261d78e
3	hermione	90112701066c0a536f2f6b2761e5edb09e
4	ginny	b053b7574c8a25751e2a896377e5d477c5
5	luna	a4048eaaee50680532845b2025996b44a9

Так-же нужно понимать, что хэш функции работают только в *одну сторону*. Именно поэтому компания не может подсказать вам ваш старый пароль и просит создать новый. Так-же хэш функции для определенного значения возвращают один и тот же хэш. Поэтому при вводе пароля, приложение с начало хэширует вводные данные, а потом сравнивает их с данными из базы.

APIs

API

Мы часто используем js в spa (single page application) в связке с сервисным api. Для обеспечения большей безопасности API, есть несколько способов:

- **API ключ** - запросы API доступны только тем пользователем, у кого есть api ключ.
- **Лимит использования** - лимит использования api добавит отказоустойчивость для api и убережет от DDOS атак.
- **Аунтефикация маршрутка** - передавать особые данные, только для особых пользователей.

Environment Variables

Просто хранить API ключи или пароли в нашем коде довольно небезопасно, ведь данные могут случайно утечь в наш публичный репозиторий. Для безопасности наших ключей, лучше использовать **переменные окружения**, в таком случае наши данные будут храниться в локальном окружении и не смогут утечь.

JavaScript

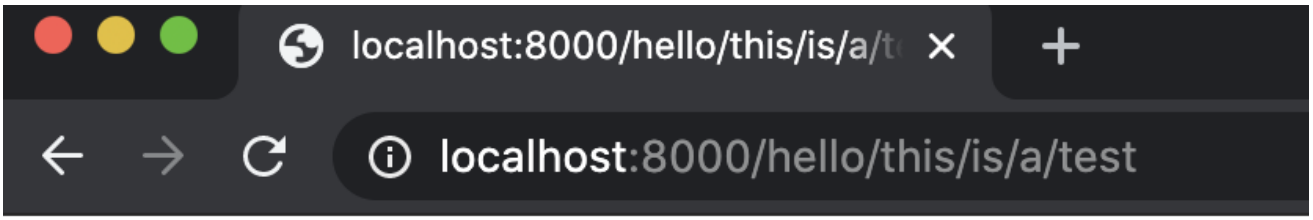
JavaScript

Так-же существует множество способов атаковать сервер с помощью js. Один из таких примеров - это **Cross-Site Scripting**, когда пользователь может вводить произвольный код для сайта. Для пример у нас есть такое приложение на django:

```
urlpatterns = [
    path("<path:path>", views.index, name="index")
]
```

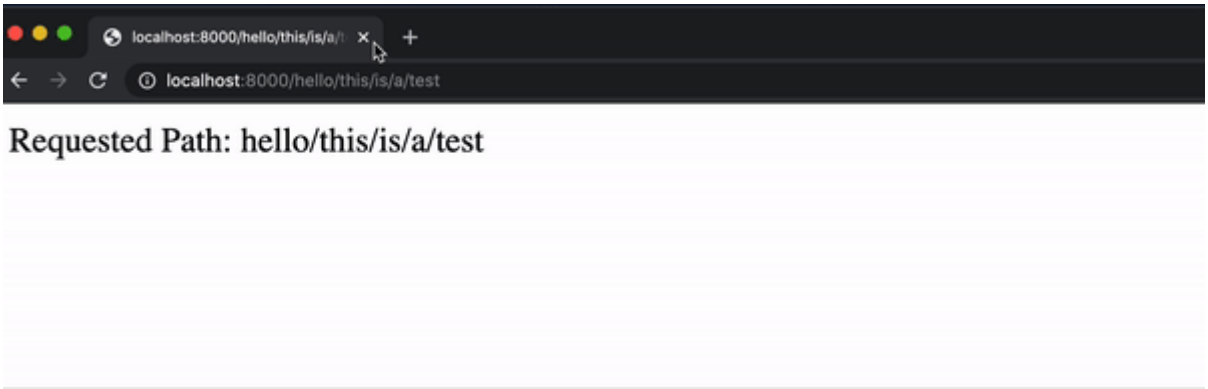
```
def index(request, path):
    return HttpResponse(f"Requested Path: {path}")
```

Вебсайт будет показывать url, который мы вводили на сайте:



Requested Path: hello/this/is/a/test

Но так-же пользователь может вести код в строку:



И `alert` - это еще самое безобидное, что может сделать пользователь.

CSRF

Cross-Site Request Forgery

Подделка межсайтовых запросов

Ранее мы использовали csrf защиту в django, но что будет если не обезопасить наш сайт? Для пример возьмем банк, в котором перейдя на ссылку можно перевести деньги.

Человек может легко создать ссылку, которая сделает такой перевод:

```
<a href="http://yourbank.com/transfer?to=brian&amt=2800">
    Click Here!
</a>
```

Эта атака может быть даже более изощренной, чем ссылка. Если URL-адрес помещен в изображение, то доступ к нему будет получен при попытке браузера загрузить

изображение:

```
<img src="http://yourbank.com/transfer?to=brian&amt=2800">
```

Поэтому всякий раз, когда вы создаете приложение, которое может принять некоторое изменение состояния, это должно быть сделано с помощью POST-запроса. Даже если банк требует POST-запрос, скрытые поля формы все равно могут обмануть пользователя и заставить его случайно отправить запрос. Следующая форма даже не ждет, пока пользователь нажмет кнопку мыши; она автоматически отправляет запрос!

```
<body onload="document.forms[0].submit()">
    <form action="https://yourbank.com/transfer"
    method="post">
        <input type="hidden" name="to" value="brian">
        <input type="hidden" name="amt" value="2800">
        <input type="submit" value="Click Here!">
    </form>
</body>
```

Чтобы предотвратить такие атаки мы должны создавать **CSRF токены** при загрузке страницы и принимать запросы от форм только с проверенными токенами.