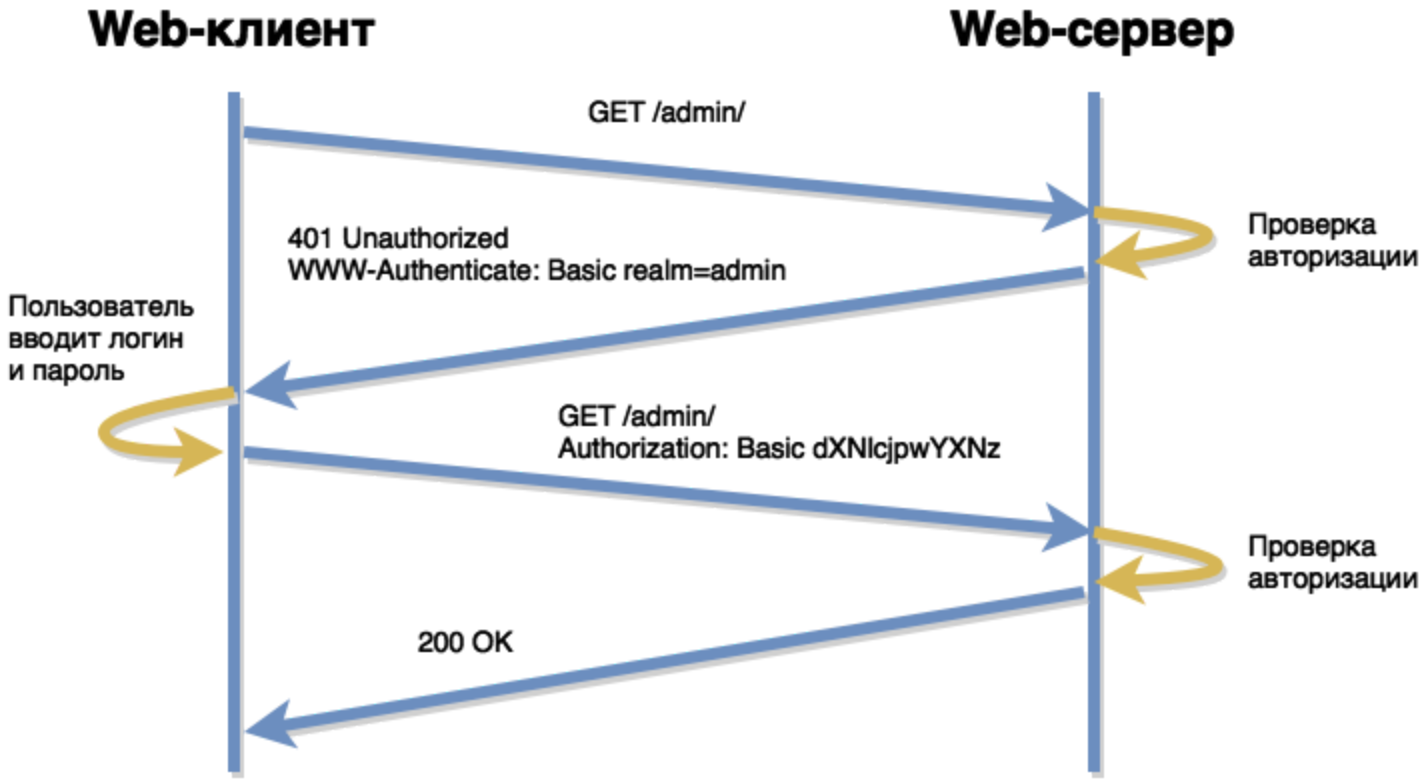
Авторизация в Web-приложениях

Авторизация в Web-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.

Basic HTTP Authorization

Basic HTTP Authorization



Заголовки и коды ответа

- `401 Unauthorized` - для доступа к ресурсу нужна атворизация
- `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

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.

Аттрибуты Cookie

- `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.

Правила выбора Cookies

Путсть URL=`http://my.app.site.com/blog/post/12`

Браузер выберет все cookies, у которых:

- Не истек срок `Expires`
- `Domain` совпадает с `my.app.site.com` или является .суффиксом, например `Domain=.site.com`
- `Path` является префиксом `/blog/post/12`, например `Path=/blog/`
- Не стоит флага `Secure`

Работа с cookie в Django

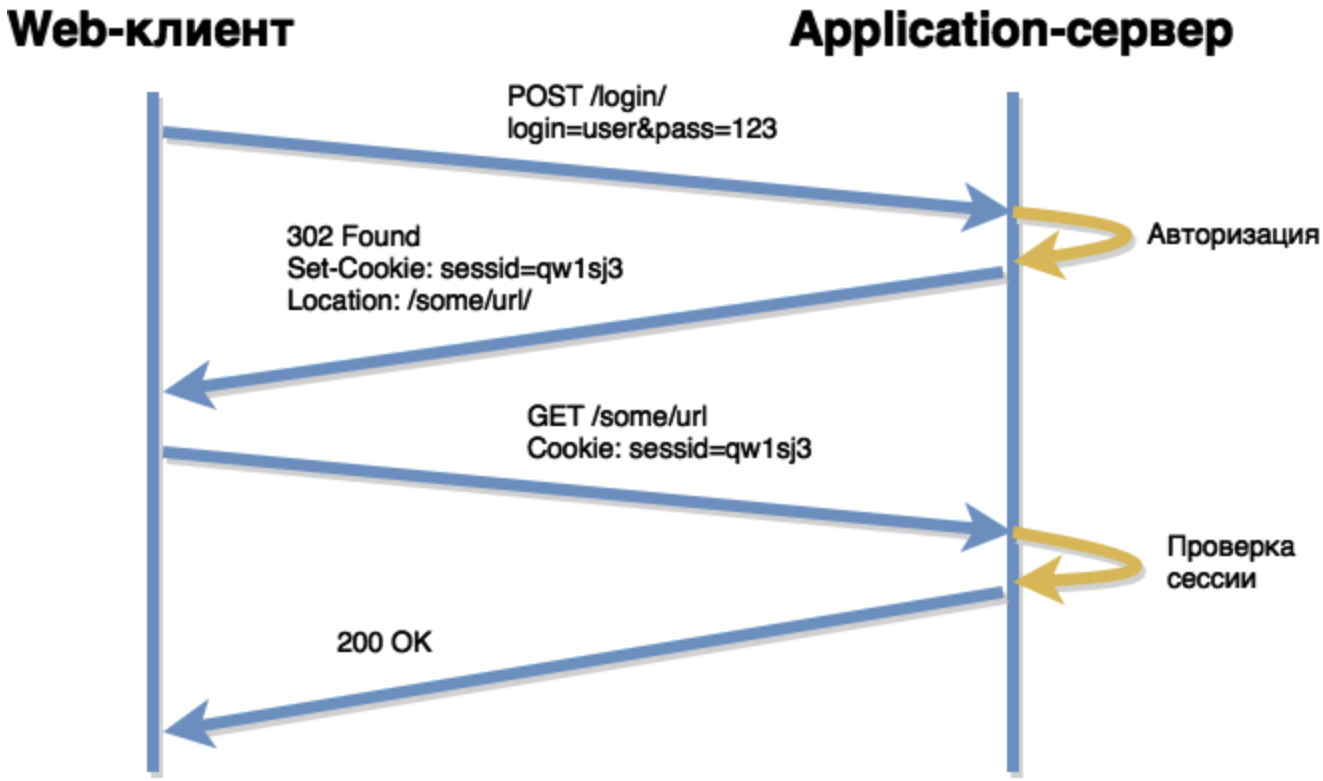
```
# установка
resp.set_cookie('sessid','asde132dk13d1')
resp.set_cookie('sessid','asde132dk13d1',
    domain='.site.com', path='/blog/',
    expires=(datettime.now() + timedelta(days=30)))

# удаление
resp.delete_cookie('another')

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

Cookie-based авторизация

Cookie-based авторизация



Необходимые модели

```
class User(models.Model):
    login    = models.CharField(unique=True)
    password = models.CharField()
    name     = models.CharField()

class Session(models.Model):
    key  = models.CharField(unique=True)
    user = models.ForeignKey(User)
    expires = models.DateTimeField()
```

Вход на сайт

URL = `/login/`

- Клиент отправляет login / password на сервер
- Сервер проверяет login / password и создает **сессию**
- Сервер устанавливает cookie, содержащий **ключ сессии**
- Сервер делает перенаправление на целевую страницу

```
def login(request):
    error = ''
    if request.method == 'POST':
        login = request.POST.get('login')
        password = request.POST.get('password')
        url = request.POST.get('continue', '/')
        sessid = do_login(login, password)
        if sessid:
            response = HttpResponseRedirect(url)
            response.set_cookie('sessid', sessid,
               domain='.site.com', httponly=True,
               expires = datetime.now()+timedelta(days=5)
            )
            return response
        else:
            error = u'Неверный логин / пароль'
    return render(request, 'login.html', {'error': error })
```

```
def do_login(login, password):
    try:
        user = User.objects.get(login=login)
    except User.DoesNotExist:
        return None
    hashed_pass = salt_and_hash(password)
    if user.password != hashed_pass
        return None
    session = Session()
    session.key = generate_long_random_key()
    session.user = user
    session.expires = datetime.now() + timedelta(days=5)
    session.save()
    return session.key
```

Проверка сессии

При запросе по любому URL:

- Клиент передает в заголовке `Cookie` свой `sessid`
- Сервер загружает сессию из базы данных по `sessid`
- Сервер загружает объект пользователя по `id` из сессии

Как правило, для проверки сессии используются `middleware`.

Middleware в Django

Middleware - это Python класс, в котором есть один из указанных ниже методов. Список всех активных middleware указан в настройке `MIDDLEWARE_CLASSES`.

- `process_request(request)`
- `process_view(request, view, args, kwargs)`
- `process_response(request, response)`
- `process_exception(request, exception)`

```
# project/project/middleware.py
class CheckSessionMiidleware(class):
    def process_request(request):
        try:
            sessid = request.COOKIE.get('sessid')
            session = Session.objects.get(
                key=sessid,
                expires__gt=datetime.now(),
            )
            request.session = session
            request.user = session.user
        except Session.DoesNotExist:
            request.session = None
            request.user = None
```

Выход из приложения

Для выхода из приложения достаточно удалить объект сессии:

```
def logout(request):
    sessid = request.COOKIE.get('sessid')
    if sessid is not None:
        Session.objects.delete(key=sessid)
    url = request.GET.get('continue', '/')
    return HttpResponseRedirect(url)
```

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
```

Безопасность

Безопасность паролей

Главная задача - максимально затруднить доступ злоумышленика к исходному паролю пользователя. Меры безопасности:

- Отправка формы входа (login / password) по **https**
- Пароли хранятся в виде хэшей с добавлением соли
- Защита от перебора в форме логина, например **captcha**

Безопасность сессий

Основное направление атаки - кража cookie, хранящей ключ сессии, т.е. кража авторизации. Меры безопасности:

- Ключ сессии невозможно подобрать перебором
- `HttpOnly` флаг для сессионой cookie
- Привязка сессии к IP адресу
- Ограничение сессий по времени
- Запрос пароля при критических действиях: смене пароля и т.д.