ПОДРОБНЫЙ РАЗБОР • 20 СЛАЙДОВ

CSRF:
Одна незаметная форма, и браузер сам оплатит счёт

Разбираем **Cross-Site Request Forgery (CSRF)**:
как это работает, в чём риск и как защититься.

bank.example.com

Вы вошли в систему

Баланс
€ 12,480

session=abc123

Банк доверяет этой сессии

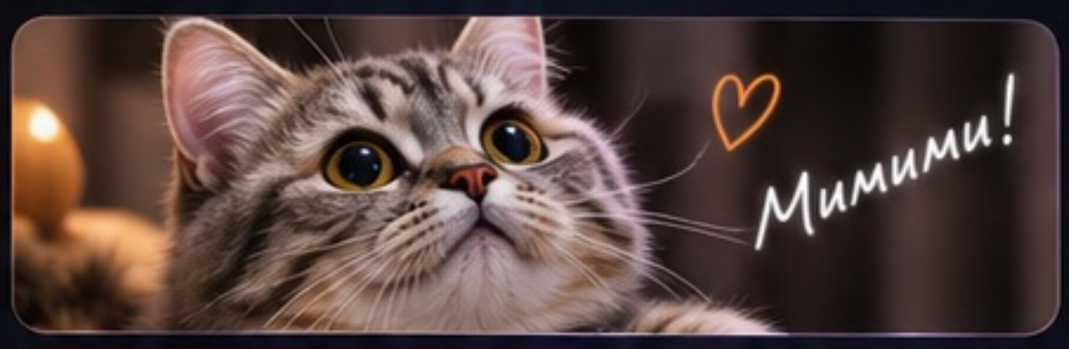
Недавние операции

- Зарплата + € 3,200
- Продукты - € 84
- Аренда - € 1,100

– € 5,000
Перевод на неизвестный счёт

cute-cats.example

Самые милые котики 2026 🐱



Скрытая форма (hidden form)
Автоотправка на bank.example

Выполняется при загрузке:

```
POST bank.example/transfer
to=attacker   amount=5000
+ your_session_cookie
```

Вы видите котиков.
А ваш браузер отправляет перевод.

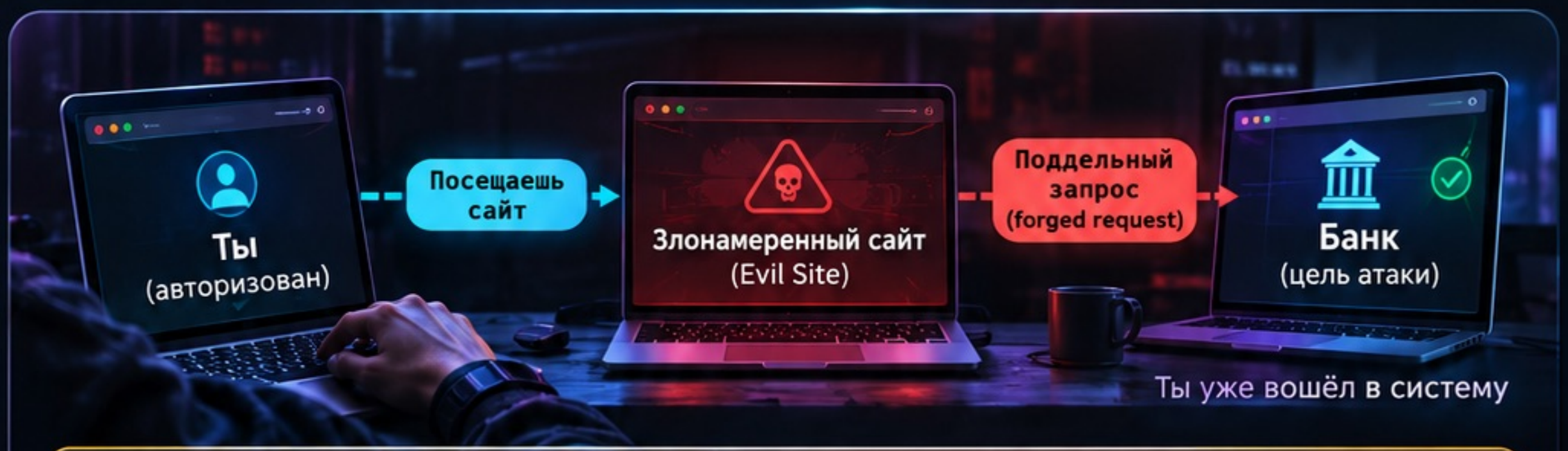
← Подделанный запрос + ваш cookie

Как работает CSRF | Сценарий атаки | Реальные последствия | Как защититься

@lionsoulbatyshchikov

Что такое CSRF?

CSRF (*Cross-Site Request Forgery*) — это атака, при которой ты, уже будучи авторизован в целевом сервисе, незаметно отправляешь запрос, который не планировал. Сайт видит валидную сессию и выполняет действие от твоего имени.



Запрос автоматически содержит твой `session cookie` — именно это и позволяет выполнить действие от твоего имени.

C **Cross-Site**

Запрос инициируется с другого сайта (не целевого).

S **Request**

Обычный HTTP request, который отправляет твой браузер.

R **Forgery**

Запрос подделан злоумышленником, а не тобой.

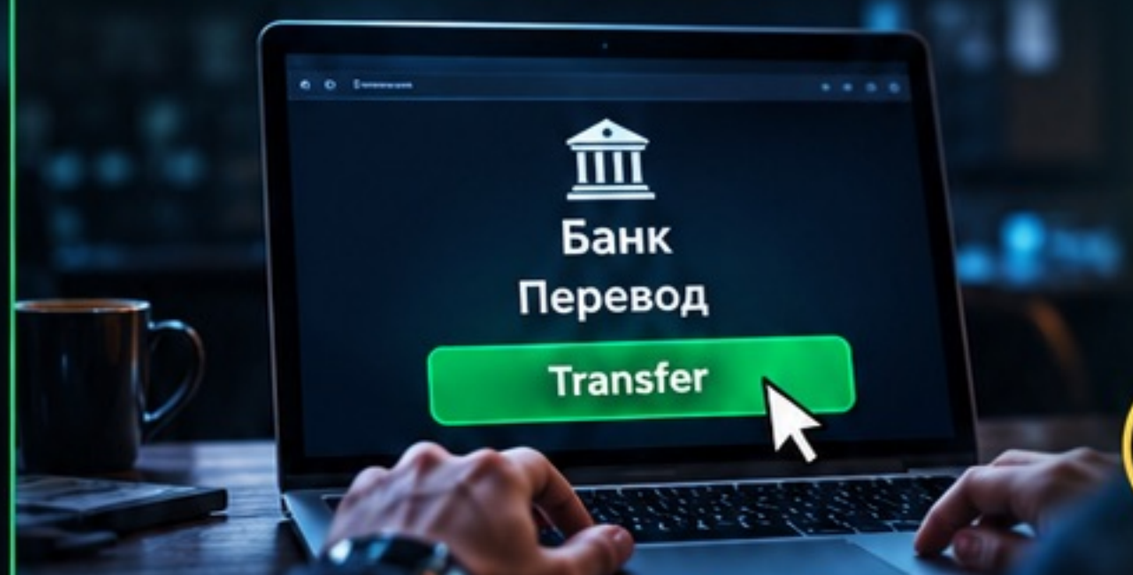
КЛЮЧЕВАЯ ИДЕЯ

Также известна как **XSRF** или **«session riding»** атака. Сайт не может надёжно отличить твой реальный клик от подделанного запроса злоумышленника.

Как cookie попадает в запросы?

При входе на сайт сервер выдаёт браузеру **session cookie**. После этого браузер автоматически добавляет её ко всем запросам к этому сайту, независимо от того, с какой страницы был инициирован запрос.

Легитимный запрос (с сайта банка)

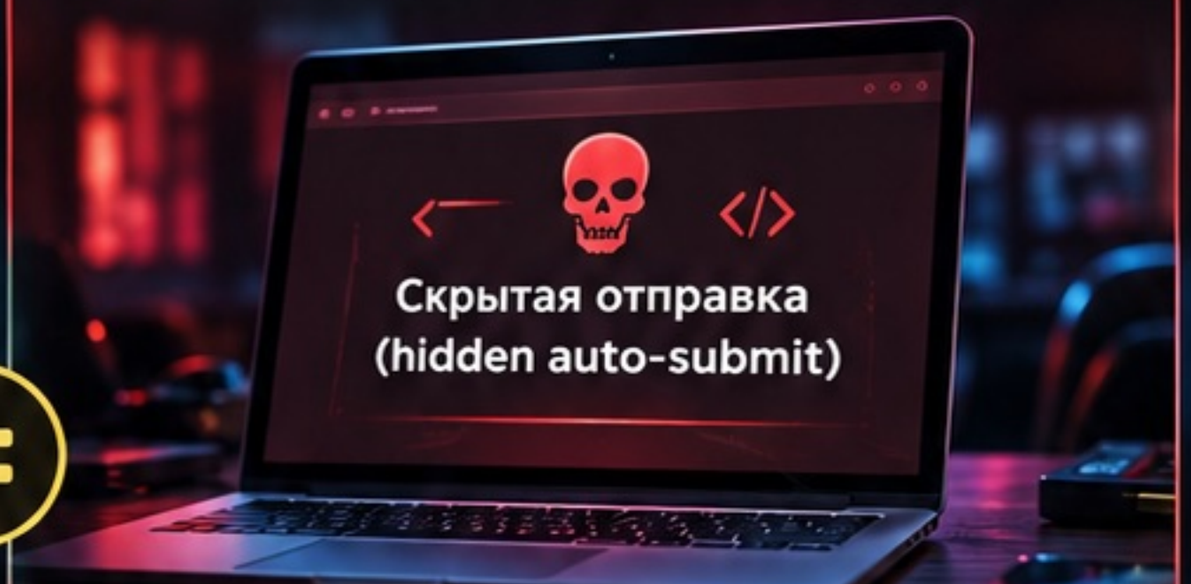


session=abc123

Cookie прикреплён

=

Поддельный запрос (с вредоносного сайта)



session=abc123

Тот же cookie прикреплён

КЛЮЧЕВАЯ ИДЕЯ

Браузер привязывает **cookie** к целевому сайту, а не к тому, кто отправил запрос. Для сервера оба запроса выглядят одинаково, поэтому он доверяет им обоим. Именно это доверие и делает **CSRF** возможным.

РЕЦЕПТ АТАКИ

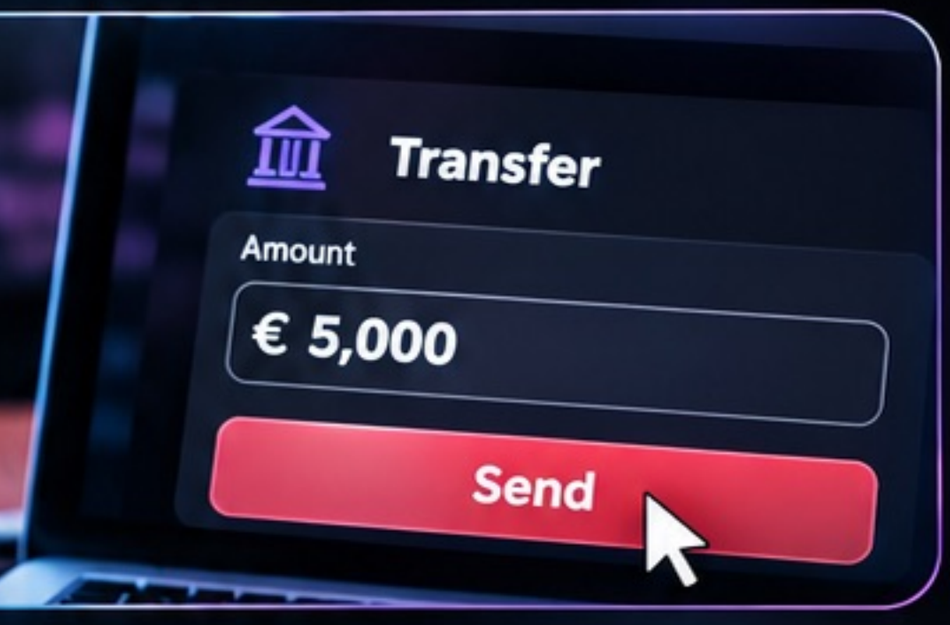
Три условия для CSRF

Атака возможна только тогда, когда одновременно выполняются все три условия. Убери любое — и классический CSRF не сработает.

1 Действие, меняющее состояние

Запрос выполняет значимое действие: перевод средств, смена e-mail, удаление данных и т. д. Страницы только для чтения (read-only) не уязвимы.

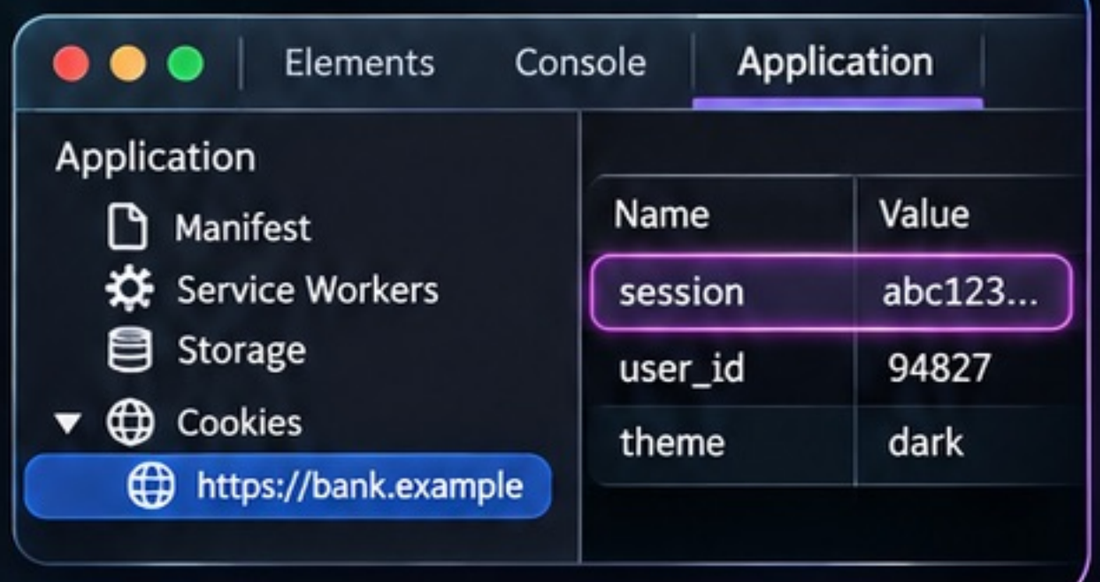
`transfer` `change email` `delete`



2 Сессия на основе cookie

Сайт аутентифицирует пользователя через cookie, и браузер автоматически отправляет его в каждом запросе. Токены, которые передаются вручную (например, в JS), не подставляются автоматически.

`session rides along for free`



3 Предсказуемый запрос

Атакующий может заранее собрать корректный запрос: известны URL, HTTP method и параметры. Никаких секретных или уникальных данных не требуется.

`no per-request secret to supply`

```
POST /transfer HTTP/1.1
Host: bank.example
Content-Type: application/x-www-form-urlencoded

amount=5000&to=attacker
```

Действие (state change) + **Сессия** (cookie) + **Предсказуемый запрос** = **Уязвимость (Exploitable)**

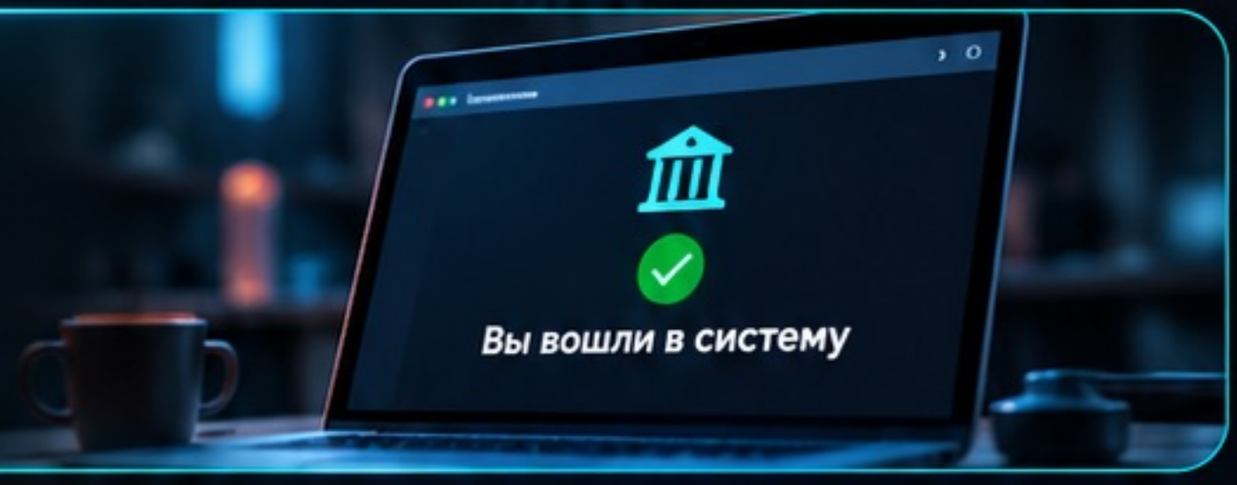
@lionsoulbatyshchikov

Атака шаг за шагом

Никакого вредоносного ПО и кражи паролей. Злоумышленнику достаточно, чтобы ты был авторизован и открыл страницу под его контролем.

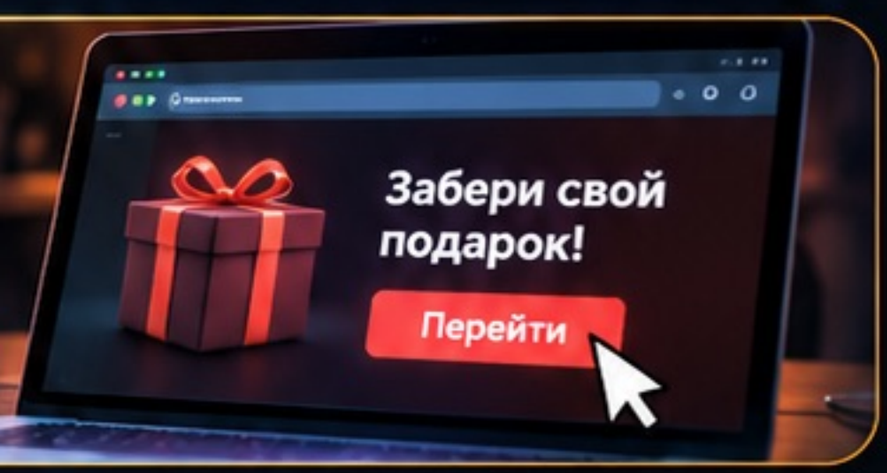
1 Ты входишь в банк

Проходишь авторизацию, и в браузере сохраняется `session cookie`, который остаётся валидным.



2 Ты открываешь страницу злоумышленника

Переходишь по ссылке из письма, сообщения, рекламы или просто с виду безобидного сайта.



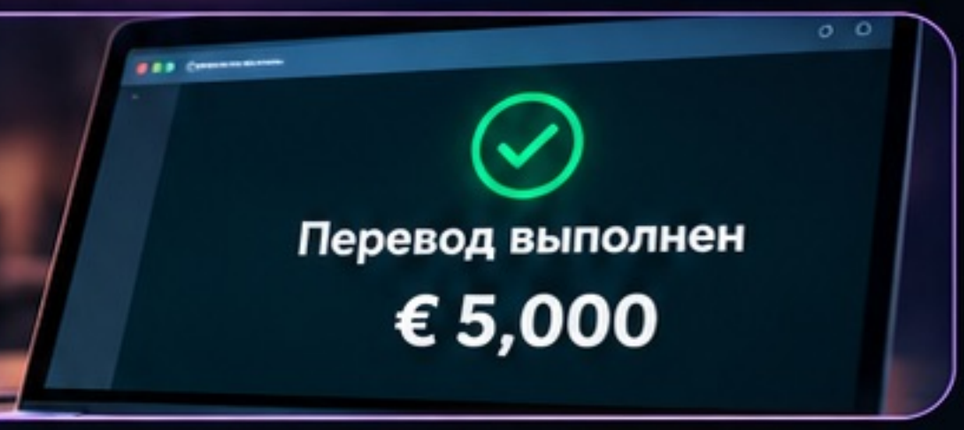
3 Страница незаметно отправляет запрос

На странице автоматически выполняется скрытый запрос (например, через form) — например, на перевод средств.

```
<form action="/transfer" method="POST">
  <input type="hidden" name="amount" value="5000">
  <input type="hidden" name="to" value="attacker">
</form>
```

4 Банк доверяет cookie

Сервер видит валидную сессию, считает, что запрос отправил ты, и выполняет действие.



КЛЮЧЕВАЯ ИДЕЯ

Первые два шага выглядят для тебя абсолютно обычно: нет предупреждений, всплывающих окон и дополнительных действий. Весь вред наносится **в фоне**, без твоего участия.

СРАВНЕНИЕ ЗАПРОСОВ

Легитимный vs поддельный запрос

Почему сервер обманывается? Поддельный запрос — это побайтовая копия настоящего. Отличается только заголовок, который злоумышленник контролировать не может.

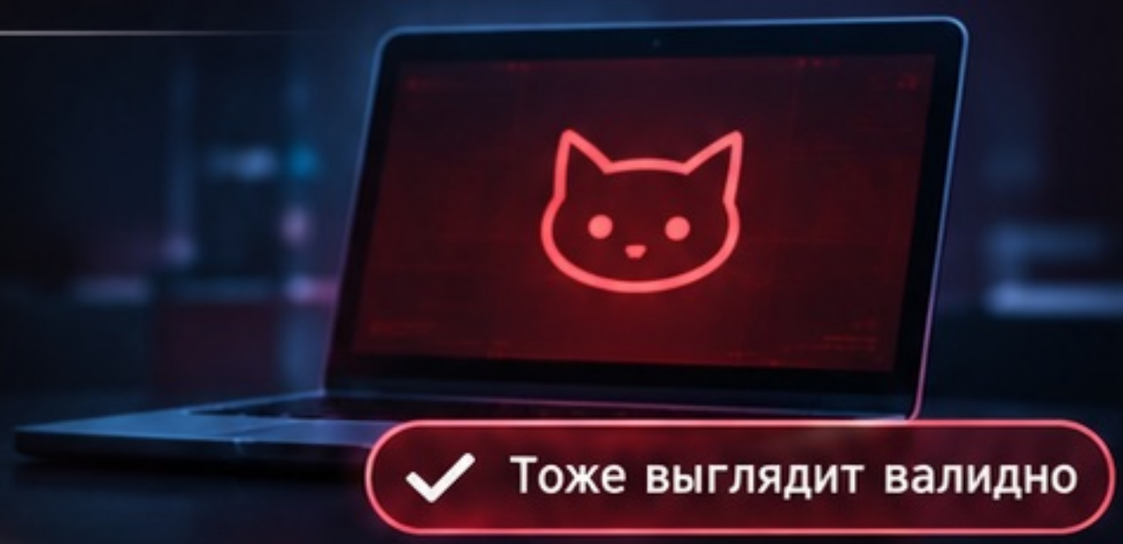
Реальный запрос • из штатной формы банка

```
1  POST /transfer
2  Host: bank.example
3  Cookie: session=abc123
4  Origin: bank.example
5  to=mum&amount=100
```



Поддельный запрос • с вредоносной страницы

```
1  POST /transfer
2  Host: bank.example
3  Cookie: session=abc123
4  Origin: cute-cats.example
5  to=attacker&amount=5000
```



В ЧЁМ РАЗНИЦА?

Одинаковые cookie и тело запроса. Различается только заголовок **Origin**. Именно его злоумышленник не может подделать или убрать — на этом и строится большинство защит от **CSRF**.

Векторы доставки CSRF

Злоумышленнику не нужно ничего взламывать — достаточно, чтобы твой авторизованный браузер загрузил его payload. Любая страница, на которой можно разместить ссылку или контент, подходит.



Image tag

Незаметный <img>, который при загрузке страницы выполняет GET-запрос на нужный URL.

```
<img src="https://
bank.example/transfer
?amount=5000">
```

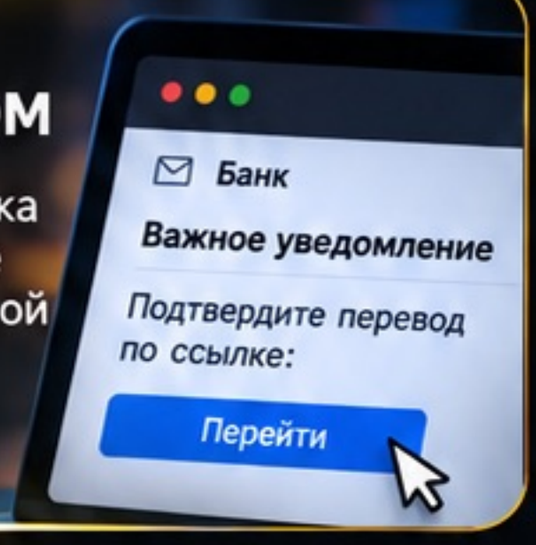
Автоотправка формы

Скрытая форма, которая автоматически отправляется сразу после открытия страницы.

```
<form action="/transfer"
 method="POST">
 <input type="hidden"
 name="amount" value="5000">
</form>
<script>document.forms[0]
.submit()</script>
```

Ссылка в письме или DM

Обычная на вид ссылка в email, мессенджере или рекламе, по которой ты переходишь уже в авторизованной сессии.



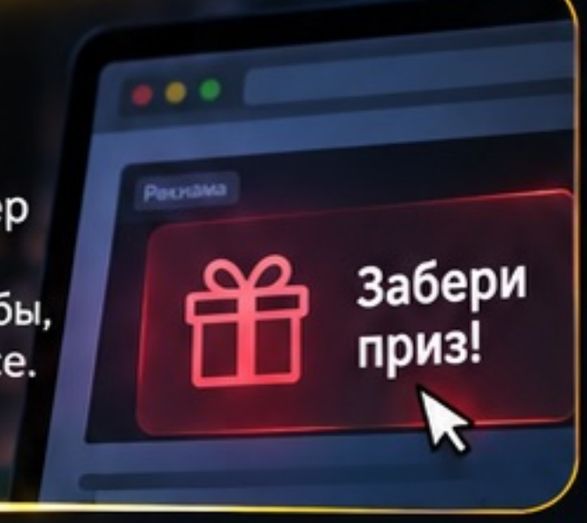
Форум или комментарий

Пост на сайте, где разрешён пользовательский контент (HTML или Markdown).

```
B  I  </>
[Перевод средств!]
(https://bank.example
/transfer)
```

Реклама или iframe

Вредоносный баннер или встраиваемый iframe на, казалось бы, доверенном ресурсе.



Сокращённый URL

Короткая ссылка, за которой скрывается реальный адрес, выполняющий целевое действие.

https://t.ly/a7F9

https://bank.example/
transfer?amount=5000

КЛЮЧЕВАЯ ИДЕЯ

Самые простые в реализации — атаки на основе **GET:** достаточно изображения, загруженного на любой странице. Поэтому изменять состояние через обычный **GET** — опасно само по себе.

ПРОСТОЙ ПРИМЕР

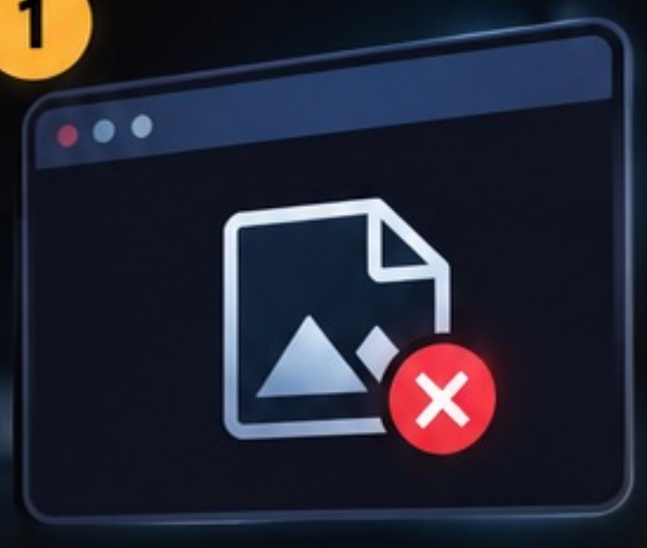
GET CSRF: одна картинка

Если изменение состояния возможно через GET-запрос, для атаки достаточно всего одного тега <img>. Браузер сам загрузит указанный URL сразу при открытии страницы.

ВЕСЬ PAYLOAD

```
<img src="bank.example/transfer?to=attacker&amount=5000">
```

1



Сломанная картинка

<img> не отображает изображение, и это никого не настораживает.

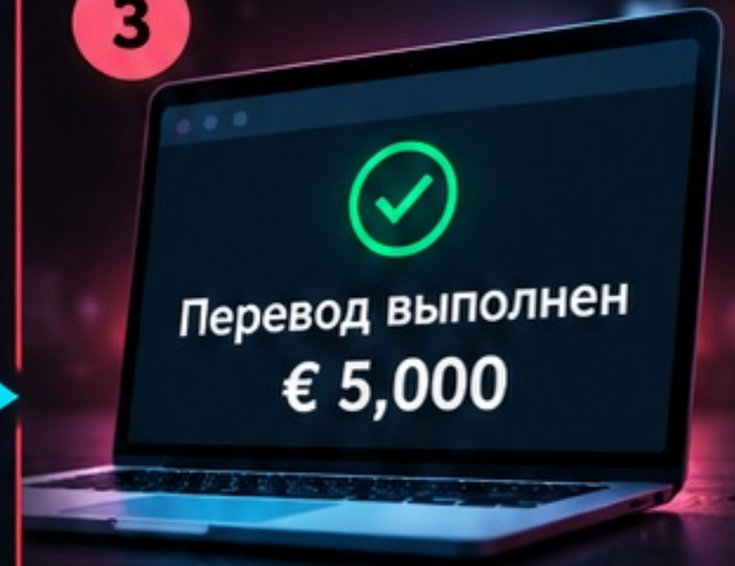
2



Выполняется GET

Браузер отправляет запрос по этому URL, автоматически прикрепляя cookie с вашей сессией.

3



Деньги ушли

Сервер видит валидный GET-запрос и выполняет перевод.

КЛЮЧЕВАЯ ИДЕЯ

По спецификации HTTP, метод **GET** должен быть “безопасным” (safe): он предназначен только для чтения и не должен менять состояние. Если банк позволяет перевод средств через **GET** — это уязвимость, даже без участия **CSRF**.

РАЗБИРАЕМ УЯЗВИМОСТЬ

09 / 20

POST CSRF: скрытая форма



Большинство атак с использованием CSRF работают через POST. Никаких проблем для злоумышленника: он отправляет скрытую форму с нужной нагрузкой. Вы видите кошек; браузер отправляет перевод.

Что вы видите

cute-cats.example



Прокрути ещё котиков!

Всего: 10 лучших пород

- ничего не кликать
- ничего не подозревать
- предупреждений нет

Скрытая форма

```
<form action="bank.example/transfer"
      method="POST" hidden>
    <input type="hidden"
        name="amount" value="5000">
  </form>

 <script>
    form.submit();
</script>


// ни кнопки, ни клика,
// отправляет моментально
// при загрузке страницы
```

Ваш браузер отправляет запрос

POST bank.example/transfer

Host: bank.example

Cookie: session=abc123 (отправляется автоматически)

to=attacker&amount=5000

Перевод выполнен

Главный вывод

POST-запросы в CSRF не требуют клика, поэтому скрытая форма может незаметно отправить запрос и выполнить действие.

ЗАЧЕМ ЭТО ЗЛОУМЫШЛЕННИКУ

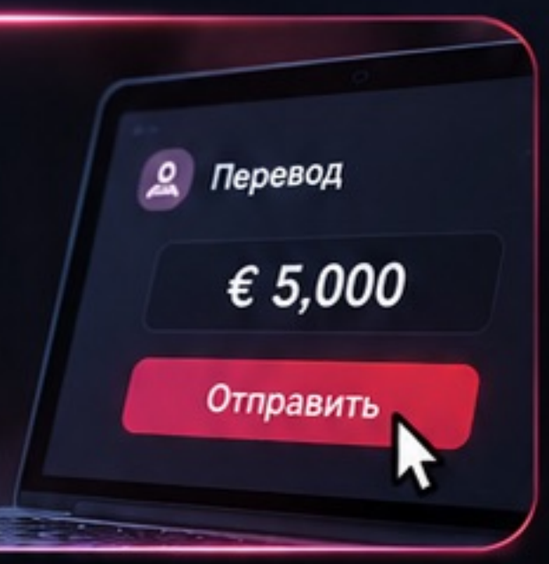
Что получает атакующий с помощью CSRF

CSRF позволяет использовать вашу активную сессию в своих целях. Злоумышленник может инициировать любое допустимое действие, пока вы авторизованы.



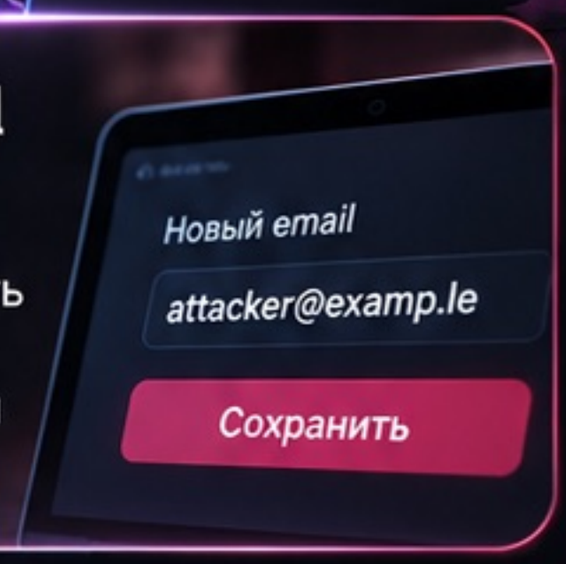
Перевод средств

Списать деньги или совершить платёж с вашего аккаунта.



Смена email

Изменить почту, а затем запустить восстановление пароля на новый адрес.



Захват аккаунта

Изменить данные для восстановления и фактически заблокировать вас.



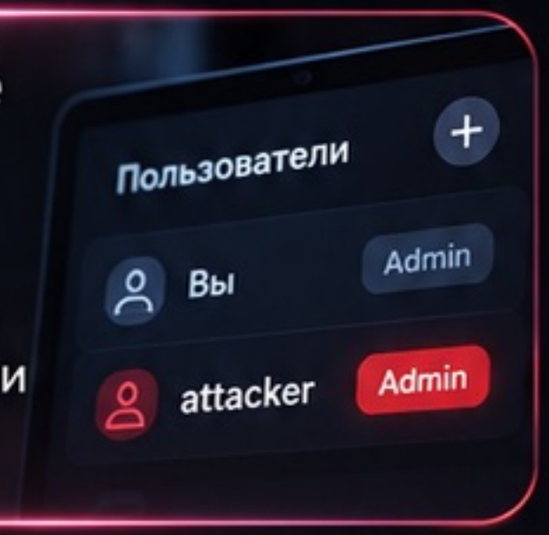
Смена адреса

Подменить адрес доставки или выставления счета, перенаправив заказы на себя.



Добавление админа

Создать нового администратора в приложении или на сервисе.



Изменение настроек

Поменять конфигурацию устройства, роутера или IoT (например, DNS, доступ и т. п.).



КЛЮЧЕВОЙ ВЫВОД

У классического CSRF есть важное ограничение: злоумышленник отправляет запрос «вслепую» и не может прочитать ответ из-за **Same-Origin Policy**. Поэтому **CSRF** — это про выполнение действий, а не кражу данных.

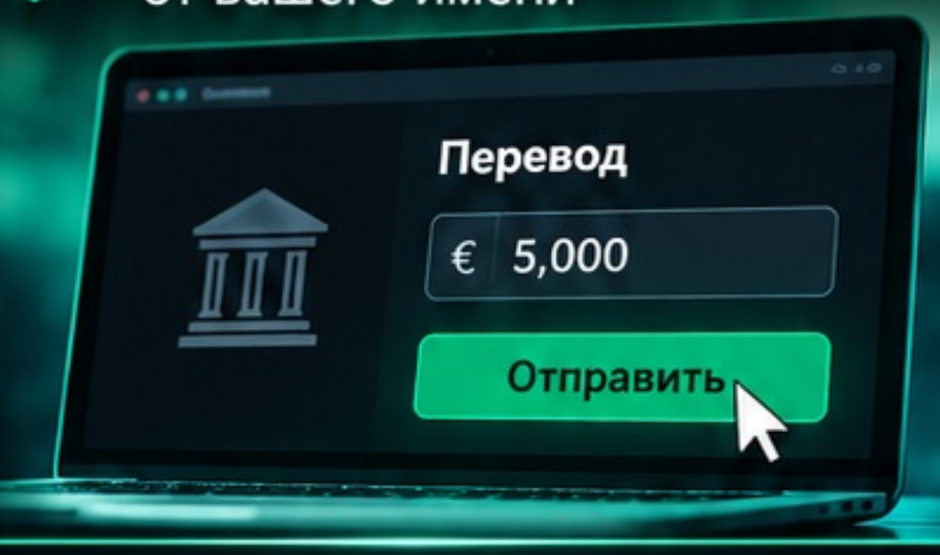
@lionsoulbatyshchikov

CSRF vs XSS

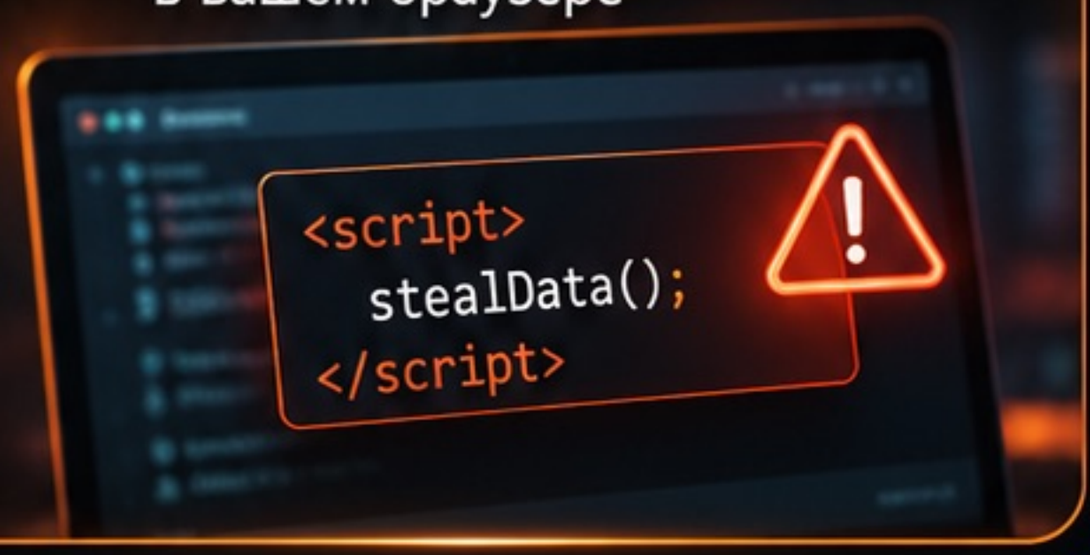
Атаки часто упоминаются вместе, но они используют разный тип доверия и решают разные задачи. Понимание различий помогает правильно выбирать защиту.

Критерий	CSRF	XSS
Суть атаки	Эксплуатирует доверие сайта к вашему браузеру (валидная сессия).	Эксплуатирует ваше доверие к сайту.
Выполнение кода	Код атакующего не выполняется.	Выполняется **JavaScript** атакующего в контексте сайта.
Что нужно	Быть авторизованным на целевом сайте.	Уязвимость для инъекции (внедрение кода в страницу).
Доступ к ответу	Нет — запрос уходит «вслепую» (**blind**).	**Да** — полный доступ к контенту страницы и ответам.
Основная защита	**anti-CSRF tokens** + **SameSite** (и проверка Origin/Referer).	**Output encoding** + **CSP** (и валидация пользовательского ввода).

CSRF: выполняет действие от вашего имени



XSS: запускает свой код в вашем браузере



КЛЮЧЕВОЙ ВЫВОД

XSS обходит любую защиту от **CSRF**. Если злоумышленник может выполнить **JavaScript** на вашем сайте, он прочитает **anti-CSRF token** и отправит корректный запрос.
Сначала устраняем **XSS** — иначе **CSRF**-защита бессмысленна.

РАСПРОСТРАНЁННОЕ ЗАБЛУЖДЕНИЕ

Почему Same-Origin Policy вас не защищает

Многие считают, что Same-Origin Policy (SOP) блокирует кросс-сайтовые запросы. На самом деле SOP запрещает читать ответ, но сам запрос всё равно уходит — вместе с вашими cookies и контекстом сессии.





ЧТО ТАКИЕ ЗАПРОСЫ НЕ ТРЕБУЮТ РАЗРЕШЕНИЯ

<img> — загрузка ресурса
без preflight, с отправкой cookies

<form> — отправка формы
без preflight, с отправкой cookies

<iframe> — встраивание страницы
без preflight, с отправкой cookies

<a> — переход по ссылке
без preflight, с отправкой cookies

КЛЮЧЕВОЙ ВЫВОД

Same-Origin Policy не останавливает **CSRF**. Он лишь не даёт прочитать ответ. Для защиты от таких запросов нужны отдельные механизмы: **anti-CSRF tokens**, **SameSite** и проверка **Origin/Referer**.

ЗАЩИТА 1 ИЗ 3

Anti-CSRF token



Базовая и самая надёжная защита: добавить в запрос секретный токен, неизвестный злоумышленнику. Сервер генерирует случайный токен и проверяет его при каждом **state-changing** запросе.

1

Сервер генерирует токен

Создаёт случайный токен, привязанный к вашей сессии, и встраивает его в форму.

```
Set-Cookie: session=abc123
csrf_token = f9b8e2a4d1c7...
```

2

Токен передаётся скрыто

Форма отправляется с токеном в hidden field (или заголовке).

```
<form method="POST">
  <input type="hidden"
         name="csrf_token"
         value="f9b8e2a4d1c7...">
```

3

Сервер сравнивает значения

Проверяет, что полученный токен совпадает с сохранённым в сессии.



4

Поддельный запрос отклоняется

У злоумышленника нет доступа к токену, поэтому его запрос не проходит проверку.



КЛЮЧЕВОЙ ВЫВОД

Защита работает благодаря **Same-Origin Policy**: атакующий может отправить запрос на ваш сервер, но не может прочитать страницы вашего сайта и узнать токен. Нет токена — нет действия.

Четыре способа внедрить защиту



Одна и та же идея — в разных реализациях. Выбор зависит от того, храните ли вы состояние на сервере (**session**) и какой клиент используете: обычные формы или **API** / **SPA**.

Synchronizer token

Токен хранится в серверной сессии (**session**) и сверяется при отправке формы. Классический и наиболее надёжный подход.



`требует session storage`

```
// сервер
session["csrf_token"] = crypto.random()
// проверка
session["csrf_token"] === form.token
```

Double-submit cookie

Один и тот же токен отправляется в двух местах: в **cookie** и в поле формы. Сервер проверяет, что они совпадают. Подходит для **stateless**-сервисов.



`проще, но слабее при XSS`

```
Set-Cookie: csrf_token=f9b8e2a4d1c7...
<form>
  <input type="hidden" name="csrf_token"
         value="f9b8e2a4d1c7...">
</form>
```

Signed double-submit

Токен из double-submit подписывается (или привязывается к session), чтобы его нельзя было подделать.

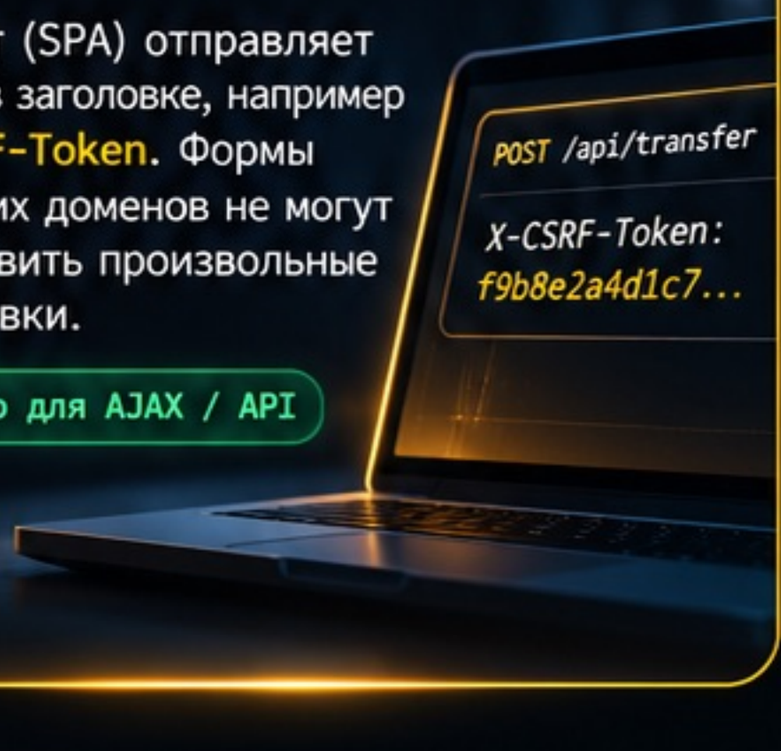


`stateless и значительно надёжнее`

```
csrf_token = sign(random(), session_id)
// сервер проверяет подпись и совпадение
```

Custom request header

Клиент (SPA) отправляет токен в заголовке, например **X-CSRF-Token**. Формы с других доменов не могут установить произвольные заголовки.



`идеально для AJAX / API`

КЛЮЧЕВОЙ ВЫВОД

Большинство фреймиворков уже поддерживают эти подходы из коробки. Включите встроенную защиту и не реализуйте логику токенов вручную.

ЗАЩИТА 2 ИЗ 3

SameSite cookies

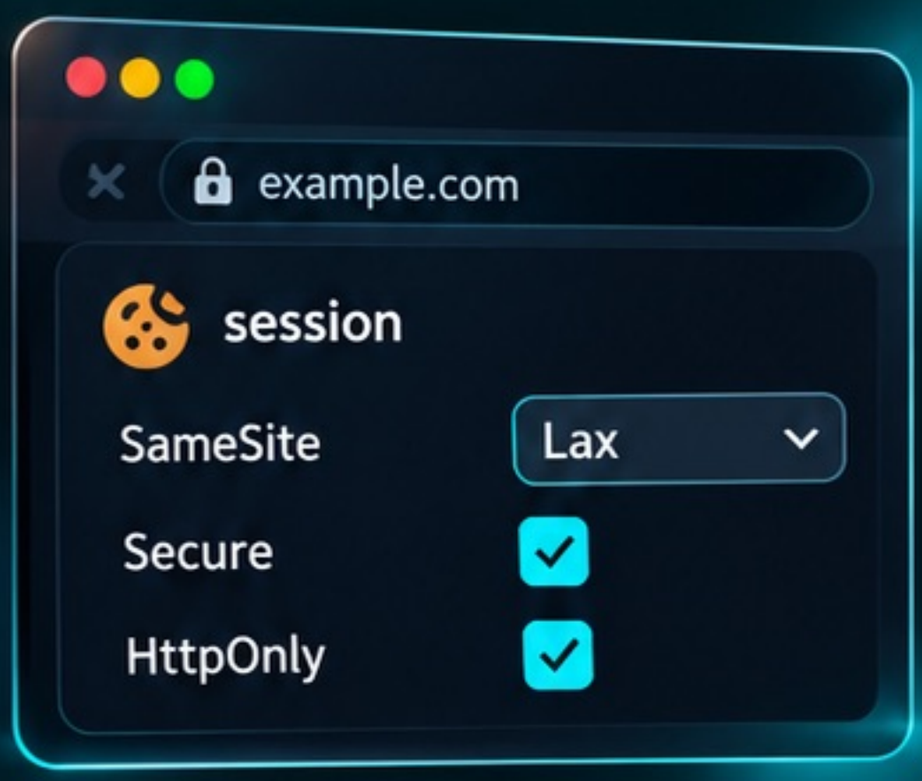
Это флаг cookie, который говорит браузеру, когда отправлять её в кросс-сайтовых запросах. Правильная настройка не даст поддельному запросу уйти вместе с вашей сессией.



Значение	Кросс-сайт POST	Top-level GET	Что это означает
Strict	Блокируется	Блокируется	Максимальная защита, но ломает переходы по внешним ссылкам (например, вход по ссылке).
Lax	Блокируется	Отправляется	Актуальный дефолт для современных браузеров: блокирует кросс-сайт POST, но допускает безопасные переходы (top-level GET).
None	Отправляется	Отправляется	Cookie всегда отправляется. Требует флаг Secure (только по HTTPS).

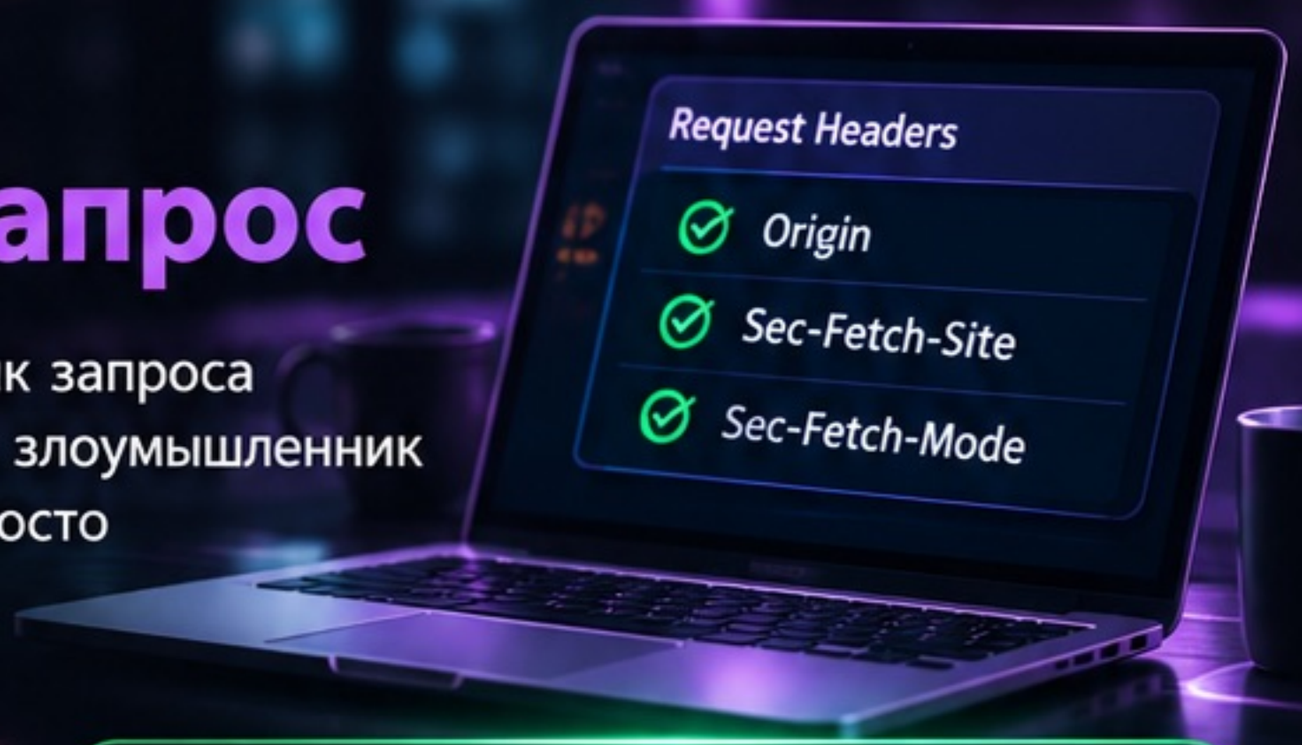
КЛЮЧЕВОЙ ВЫВОД

Сейчас **Lax** — дефолт в большинстве браузеров, и он уже нейтрализует классическую кросс-сайт **POST**-атаку. Но не стоит полагаться только на него: не везде поведение одинаковое, а действия через **GET** всё ещё возможны.



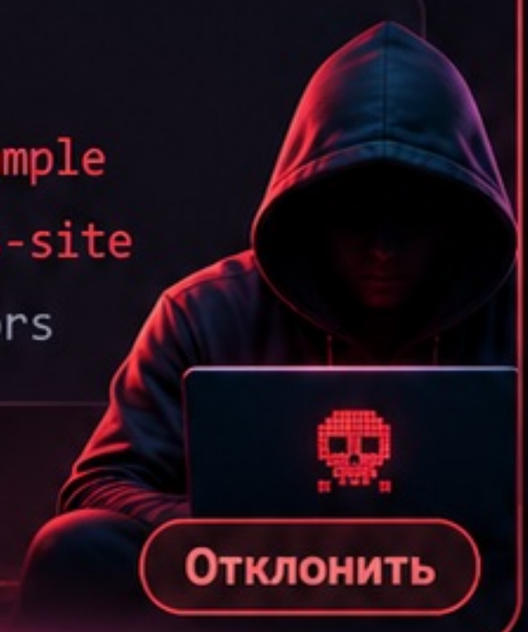
Проверяйте, откуда пришёл запрос

Браузер сам сообщает серверу источник запроса в специальных заголовках. Эти значения злоумышленник не может подделать, а сервер может просто отклонять кросс-сайт запросы.



✕ Поддельный запрос

```
POST /transfer
Origin: cute-cats.example
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
```



Отклонить

✓ Легитимный запрос

```
POST /transfer
Origin: bank.example
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
```



Разрешить

</> Пример проверки на сервере

```
if (Sec-Fetch-Site === "cross-site") {
    reject(request); // запрос не с нашего сайта
}
```

КЛЮЧЕВАЯ МЫСЛЬ

Злоумышленник не может задать заголовки **Origin**, **Referer** или **Sec-Fetch-*** — ими управляет браузер. **Fetch Metadata** — современный и удобный слой защиты, но рассматривайте его как часть **defense in depth**, а не единственную меру.

Что на самом деле не защищает от CSRF

Многие «фиксы» кажутся логичными, но оставляют уязвимость открытой. Разбираем популярные заблуждения и что реально работает.



✕ «Используем только POST»

Формы тоже могут отправлять кросс-сайт запросы. Один POST сам по себе ничего не защищает.



✕ «Проверяем Referer»

Заголовок может отсутствовать или быть подменён, а злоумышленник может отправить пустой Referer.



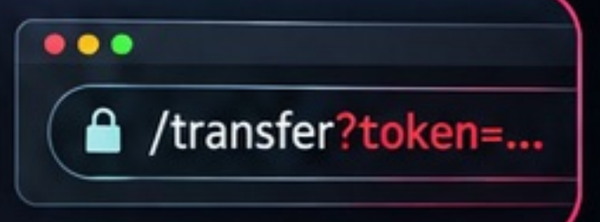
✕ «Cookie защищён»

Cookie отправляется браузером автоматически. Злоумышленнику её не нужно читать или знать.



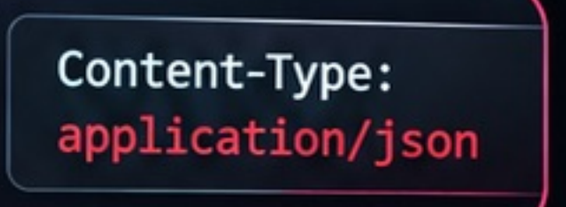
✕ «Передаём токен в URL»

Токен попадёт в историю браузера, логи и заголовок Referer — это утечка и плохая практика.



✕ «Требуем JSON content-type»

Слегка усложняет атаку, но легко обходится и не является полноценной защитой.



ЧТО РЕАЛЬНО РАБОТАЕТ

Комбинация мер: непредсказуемый **anti-CSRF token**, флаг **SameSite** для cookies и проверка **Origin** или **Sec-Fetch-***. Многослойная защита, а не одна отдельная настройка.

Как проверить защиту от CSRF

Проверить просто — не нужен готовый эксплойт. Найдите действие, которое меняет состояние, удалите защиту (например, токен) и посмотрите, примет ли сервер запрос.



Найдите чувствительное действие

Например, перевод, изменение настроек или удаление.

Уберите токен

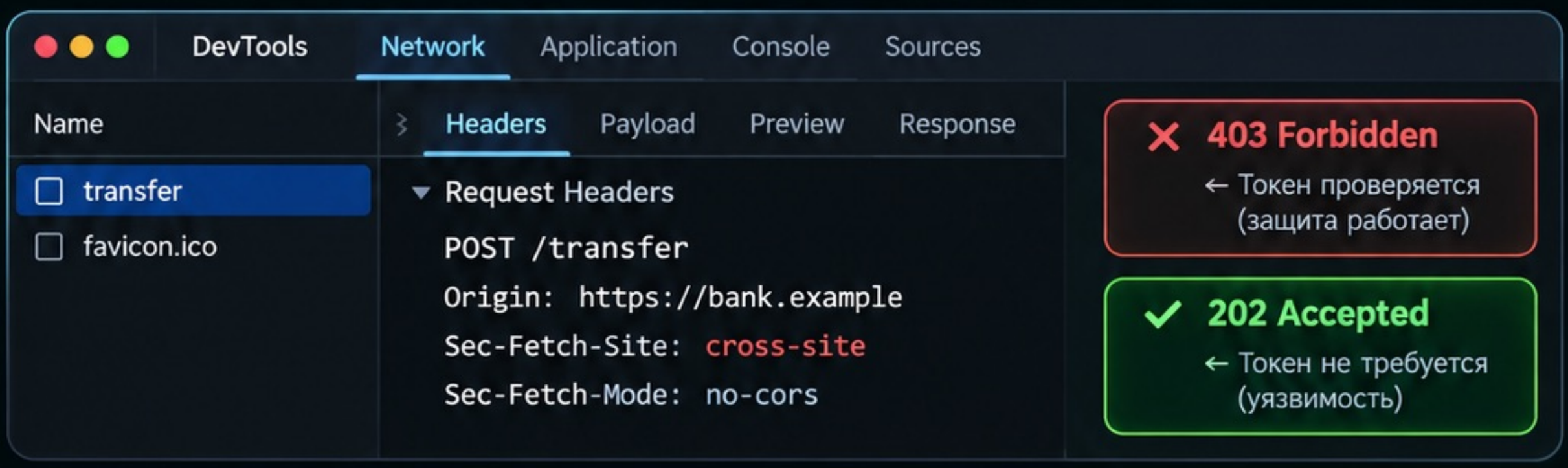
Удалите или подмените поле с **anti-CSRF token** (в форме или в запросе).

Отправьте запрос заново

Повторите то же действие, но уже без токена.

Проверьте SameSite

Посмотрите флаги **cookie** в **DevTools** (Application → Cookies).



ПОПРОБУЙТЕ САМИ

1. Откройте DevTools, найдите запрос, который меняет состояние, и отправьте его без CSRF-поля.
2. Ответ **202** (или аналогичный успех) — значит, есть уязвимость. Ответ **403** — проверка токена работает.
3. Тестируйте только свои приложения или те, на которые у вас есть явное разрешение.

КОРОТКО И ПО ДЕЛУ

CSRF: шпаргалка

Главное о CSRF — что нужно для успешной атаки, какие слои защиты работают и что стоит делать на практике.



Что нужно злоумышленнику

1. **Действие, меняющее состояние**
   например, перевод или изменение настроек
2. **Сессионная аутентификация на базе cookies**
   пользователь уже авторизован в приложении
3. **Предсказуемый запрос**
   сервер принимает его без дополнительной проверки

Слои защиты (используйте в совокупности)

Anti-CSRF token	непредсказуемый токен, проверяется на сервере	1
SameSite cookies	флаг SameSite=Lax или Strict для сессионных cookies	2
Origin / Sec-Fetch check	проверка Origin и/или Sec-Fetch-* для отклонения кросс-сайт запросов	3
Re-auth на критичные действия	повторная аутентификация (пароль, 2FA и т. п.) для операций с высоким риском	4

Делайте

- Выполняйте изменение состояния через POST
- Включайте встроенную CSRF-защиту фреймворка
- Задавайте SameSite для сессионных cookies



Не делайте

- Менять состояние через GET
- Опирайтесь только на Referer
- Передавать токен в URL



@lionsoulbatyshchikov

CSRF: ИТОГИ

Было полезно?

Сохраните, чтобы вернуться при ревью безопасности, и поделитесь с коллегой, который всё ещё меняет состояние по GET.



Сохранить | Поделиться | Подписаться

ТЕПЕРЬ У ВАС ЕСТЬ БАЗА

1 Token

Непредсказуемый anti-CSRF token

2 SameSite

Флаг для cookie (Lax или Strict)

3 Origin check

Проверка Origin или Sec-Fetch-*

ПОГРУЗИТЬСЯ ГЛУБЖЕ

- Типы XSS и их отличия
- Базовые принципы Web Security
- Другие веб-атаки

@lionsoulbatyshchikov