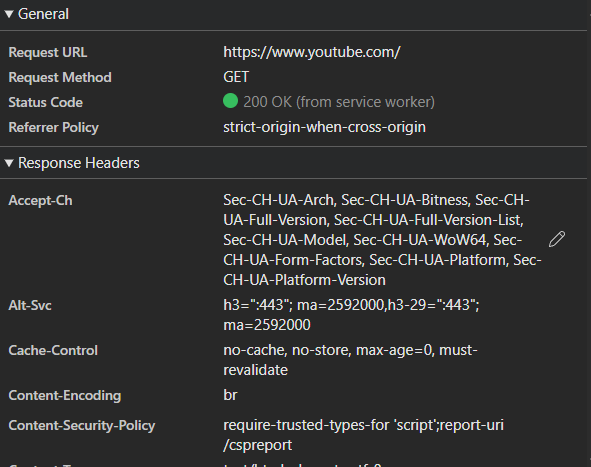
1 Откройте любую страницу, например https://example.com, и перейдите во вкладку Network в инструментах разработчика браузера. Найдите первый GETзапрос и зафиксируйте его метод, статус ответа, заголовки запроса и ответа, размер тела и используемую версию протокола. Также найдите запрос с редиректом (код 3xx) и опишите, что при этом произошло: куда произошёл переход, каким методом, и как клиент обрабатывает такое поведение.

*Анализ HTTP-запросов YouTube в DevTools*

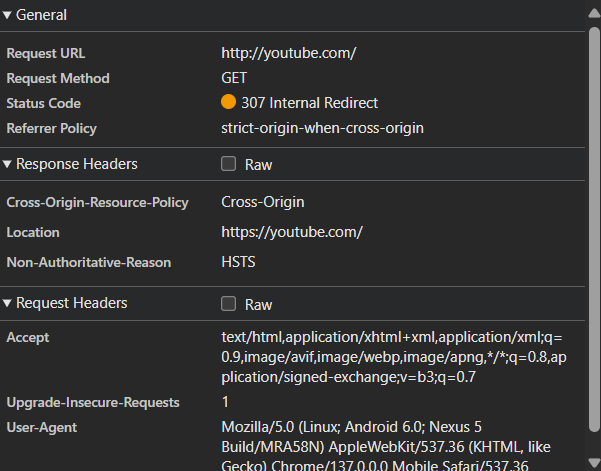
1. Основной GETзапрос (https://www.youtube.com/)



Запрос к основной странице YouTube содержит следующие характеристики:

* URL запроса: https://www.youtube.com/
* Метод: GET (стандартный метод получения данных)
* Статус ответа: 200 OK
* Используемый протокол: HTTP/2 или HTTP/3 (подтверждается заголовком altsvc: h3)
* Заголовки запроса включают информацию о браузере (UserAgent), данные о клиенте (SecCHUA\*) и запрос на переход на защищенное соединение (UpgradeInsecureRequests)
* Заголовки ответа содержат:
* contenttype: text/html (тип возвращаемого контента)
* cachecontrol: nostore (запрет на кэширование)
* stricttransportsecurity: maxage=31536000 (HSTS принудительное использование HTTPS)
* Размер тела ответа оптимизирован за счет сжатия Brotli (contentencoding: br)
* Политика безопасности включает защиту от XSS и Clickjacking (contentsecuritypolicy, xframeoptions: SAMEORIGIN)
* HSTS автоматически перенаправляет все запросы на HTTPS
* Кэширование отключено для обеспечения актуальности контента
* Поддержка современных протоколов HTTP/3 через altsvc: h3=":443"

2. Запрос с редиректом (307 Internal Redirect)



* Исходный URL: http://youtube.com/
* Метод: GET
* Статус ответа: 307 Internal Redirect (временное перенаправление)
* Направление перенаправления: https://youtube.com/
* Причина: nonauthoritativereason: HSTS (браузер знает о необходимости HTTPS)
* Обработка: автоматическое создание нового GETзапроса на HTTPS

Процесс редиректа:

1. Браузер отправляет GET http://youtube.com/

2. Срабатывает механизм HSTS

3. Браузер автоматически перенаправляет на https://youtube.com/

4. Завершается успешным ответом 200 OK

Заголовки запроса:

* UpgradeInsecureRequests: 1 запрос на переход с HTTP на HTTPS
* UserAgent идентификация браузера (в данном случае Nexus 5 на Android)
* SecCHUA\* подробные данные о версии браузера и ОС (Client Hints)

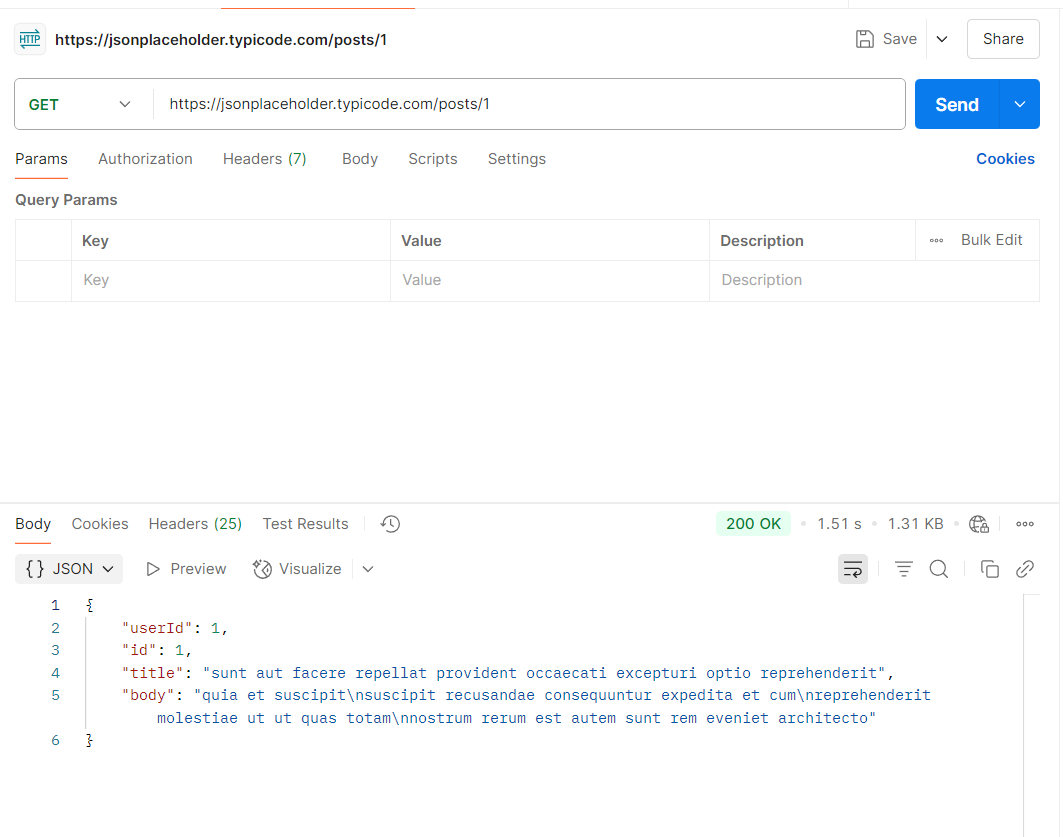
Заголовки ответа:

* Location: https://youtube.com/ URL для перенаправления
* CacheControl: nostore запрет на сохранение в кэше
* StrictTransportSecurity гарантия использования HTTPS в течение года

2. С помощью Postman отправьте GET и HEAD-запрос к публичному API, например https://jsonplaceholder.typicode.com/posts/1, и сравните результаты. Далее выполните POST, PUT, PATCH и DELETE-запросы — объясните, что делает каждый метод, в чём их отличие и как они влияют на ресурсы. Также определите, какие из этих методов являются идемпотентными — то есть при повторном вызове не изменяют состояние сервера дополнительно.

**1. GET vs HEAD-запросы**

**GET**



Статус: 200 OK

Тело: Есть (полные данные поста).

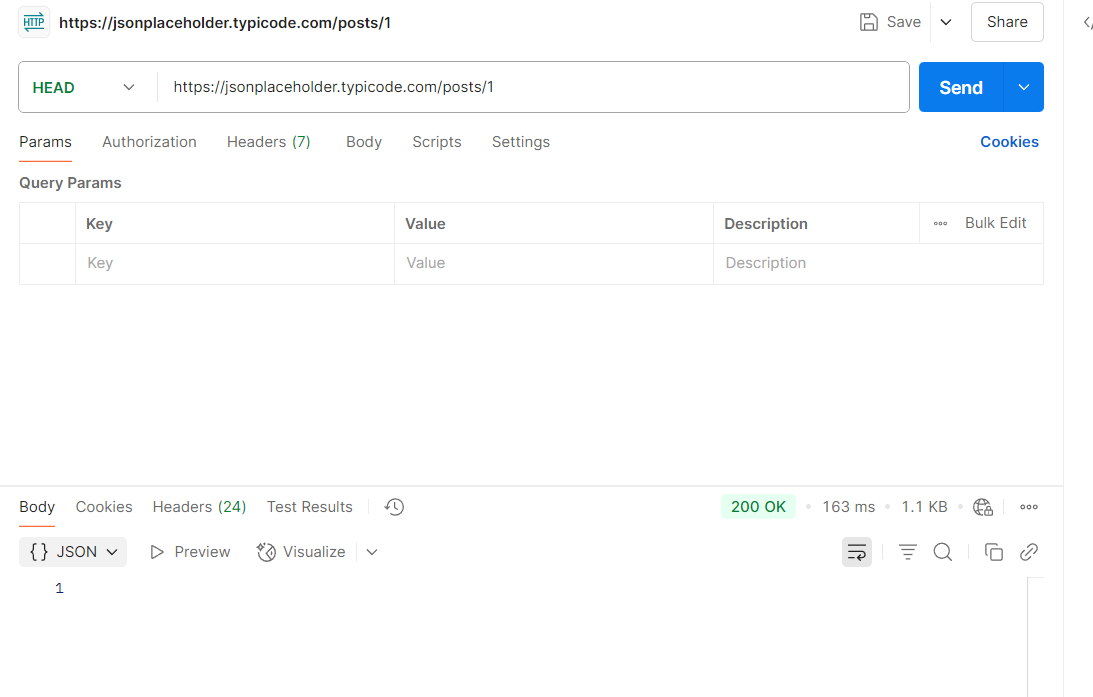
Назначение: Получение ресурса.

Запрашивает данные с сервера. Не изменяет состояние ресурса - только читает информацию.

**Идемпотентность:** Да  
Повторные одинаковые запросы не изменяют сервер и возвращают один результат.

HEAD

Аналогичен GET, но сервер возвращает только заголовки ответа без тела. Используется для проверки доступности ресурса и его метаданных.



Статус: 200 OK

Тело: Отсутствует.

Заголовки: Те же, что у GET (Content-Type: application/json).

Назначение: Проверка существования ресурса и метаданных (без загрузки тела).

**POST**

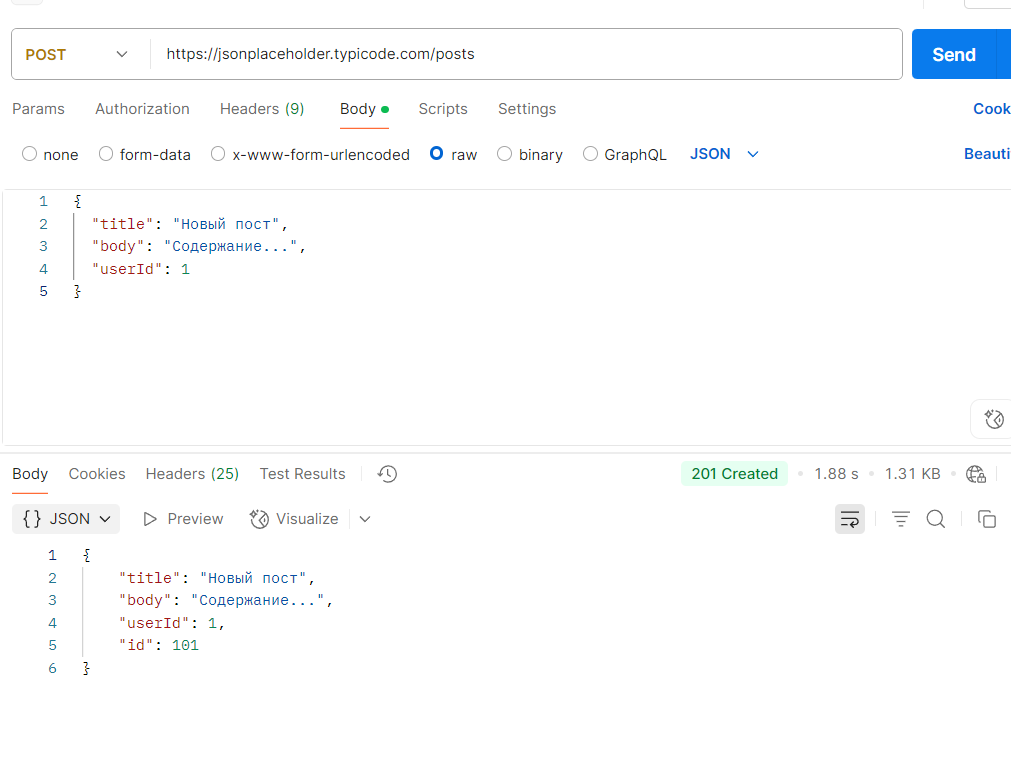
Создает новый ресурс. Тело запроса содержит данные для создания.

**Идемпотентность:** Нет  
Каждый запрос создает новый ресурс. Повторный идентичный запрос создаст дубликат.

Статус: 201 Created

Действие: Создание нового ресурса.

Идемпотентность: Нет. Повторный запрос создаст дубликат.



**PUT (Полное обновление)**

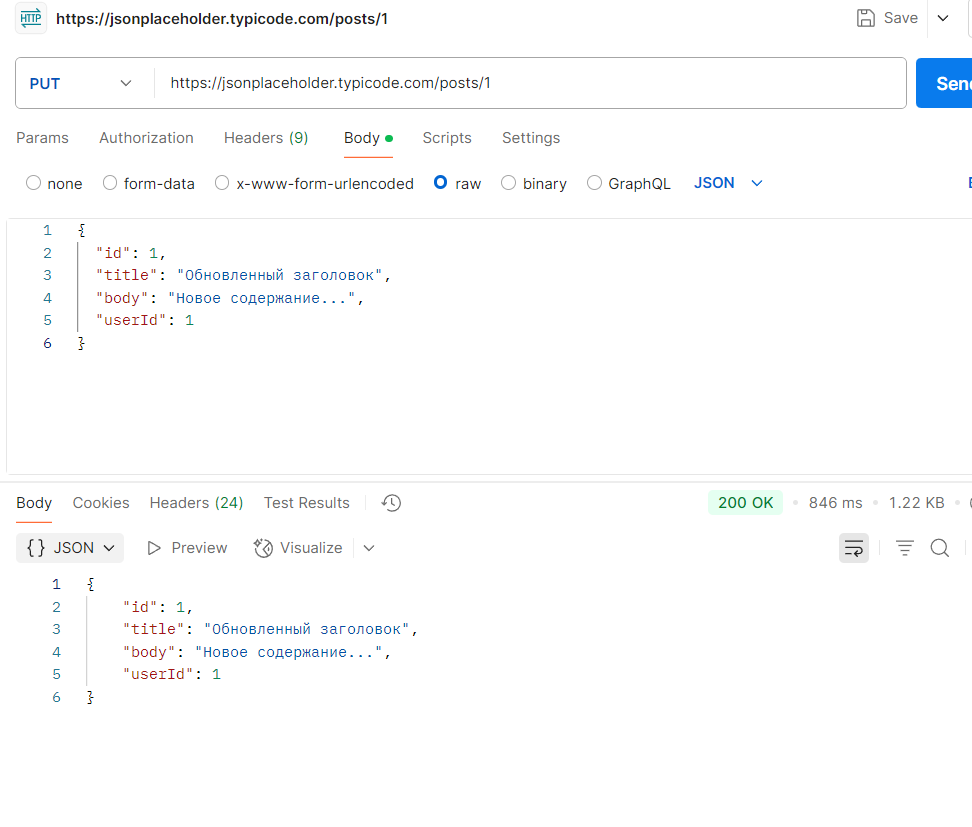
Полностью заменяет ресурс новыми данными. Если ресурса нет - может создать его (зависит от API).

**Идемпотентность:** Да  
Повторные одинаковые запросы не изменяют результат после первого выполнения.

Статус: 200 OK

Действие: Полная замена ресурса. Если ресурса нет, может создать его (зависит от API).

Идемпотентность: Да. Повторный запрос не изменит результат.

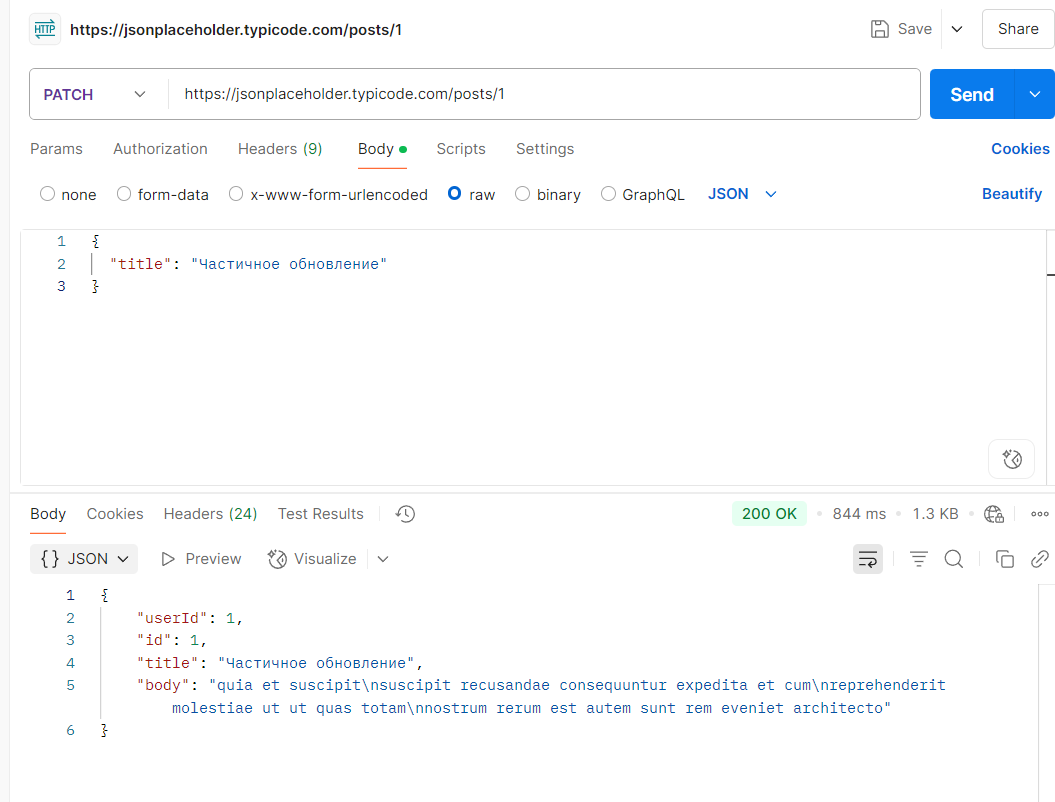


**PATCH (Частичное обновление)**

Обновляет только указанные поля ресурса, а не весь ресурс целиком.

Идемпотентность: Зависит от реализации

Теоретически может быть идемпотентным, но на практике часто не является таковым, так как зависит от логики обновления.



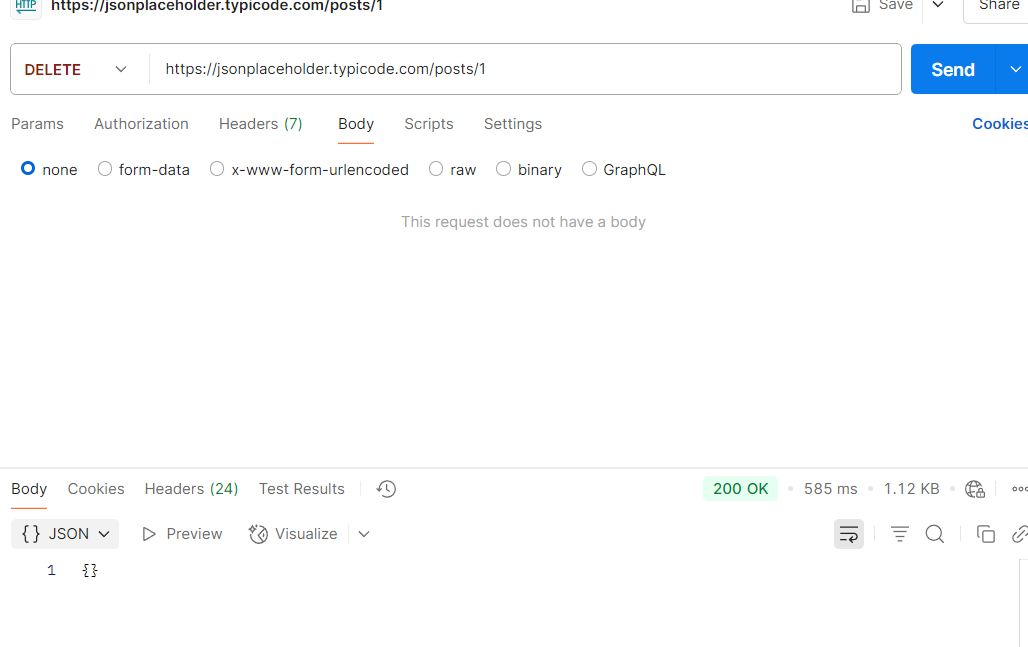
**DELETE (Удаление)**

**Идемпотентность:** Да  
После первого удаления ресурса последующие запросы не изменяют состояние сервера (хотя могут возвращать разные статусы).

Статус: 200 OK (или 204 No Content).

Действие: Удаление ресурса.

Идемпотентность: Да. Повторный запрос вернет 404, но состояние сервера не изменится.



**Сравнение методов**

**По изменению состояния сервера:**

* **Не изменяют (безопасные):** GET, HEAD
* **Изменяют:** POST, PUT, PATCH, DELETE

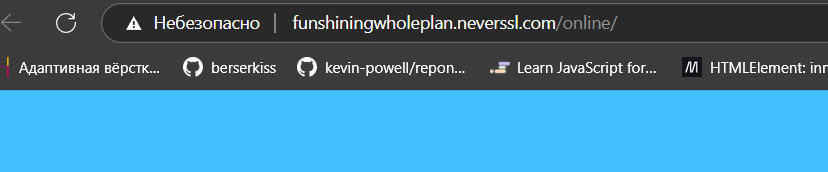
**По идемпотентности:**

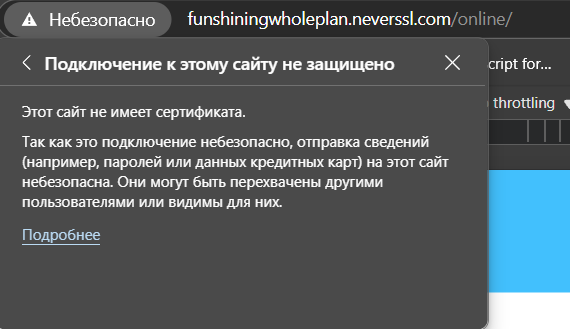
* **Идемпотентные:** GET, HEAD, PUT, DELETE
* **Не идемпотентные:** POST, PATCH

**По действию на ресурс:**

* **Создание:** POST
* **Чтение:** GET, HEAD
* **Полное обновление:** PUT
* **Частичное обновление:** PATCH
* **Удаление:** DELETE

3 Перейдите на сайты http://neverssl.com и https://example.com, затем сравните соединения. Проверьте наличие сертификата, наличие шифрования (замочек в адресной строке), а также откройте вкладку Security в DevTools, чтобы увидеть протокол. Объясните, зачем нужен HTTPS, и как он защищает пользователя от угроз (например, перехвата данных и подмены контента).





Протокол: HTTP/1.1 или HTTP/2 (без шифрования).

Данные:

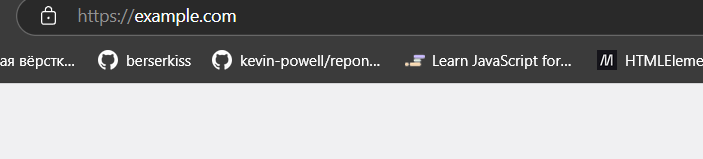
Передаются в открытом виде (текстом).

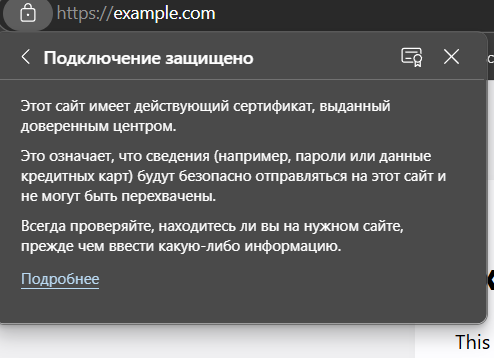
Уязвимы к перехвату (например, в публичных Wi-Fi).

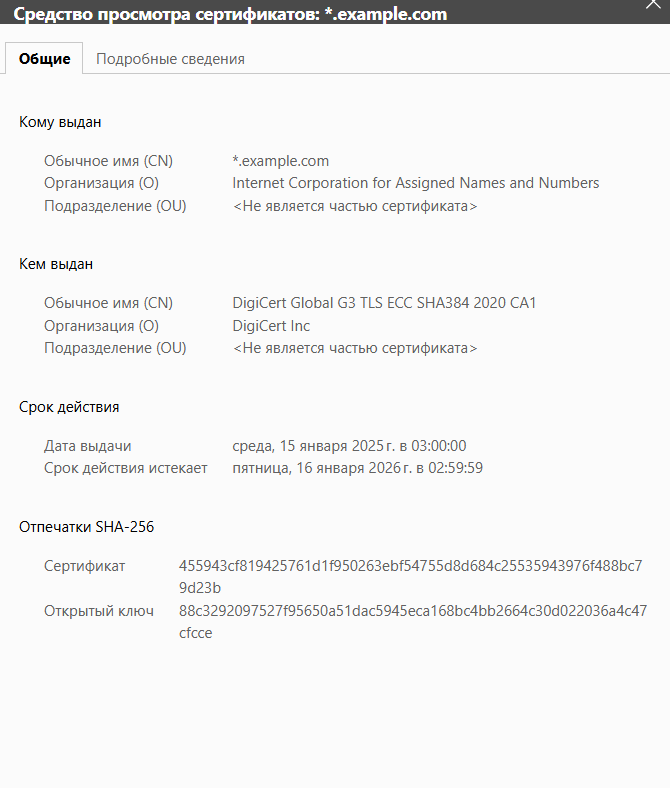
Пример угроз:

Злоумышленник может подменить контент (например, вставить рекламу или фишинговую форму).

Перехватить логины/пароли, если сайт использует формы без HTTPS.







1. Прямые указатели HTTPS

* Request URL:
* https://example.com/ - явное указание протокола HTTPS в URL.
* :scheme заголовок:
* https в секции псевдо-заголовков HTTP/2.
* Порт 443:
* В Remote Address: 23.192.228.80:443 порт 443 стандартно используется для HTTPS.

2. Технические маркеры шифрования

* alt-svc (Alternative Services):
* Поддержка HTTP/3 через QUIC (h3=":443"), который работает только поверх шифрованных соединений.
* quic-version:
* Наличие этого заголовка подтверждает использование QUIC (основа HTTP/3), требующего обязательного шифрования.

**Зачем нужен HTTPS?**

1. **Шифрование данных**

Превращает информацию в нечитаемый код при передаче. Даже если данные перехватят, расшифровать их без ключа невозможно.

1. **Аутентификация сервера**

Сертификат подтверждает, что вы подключены к настоящему сайту (а не к поддельному).

Проверяется центром сертификации (Let's Encrypt, DigiCert и др.).

1. **Защита от атак:**

**MITM (Man-in-the-Middle):** Злоумышленник не может подменить данные.

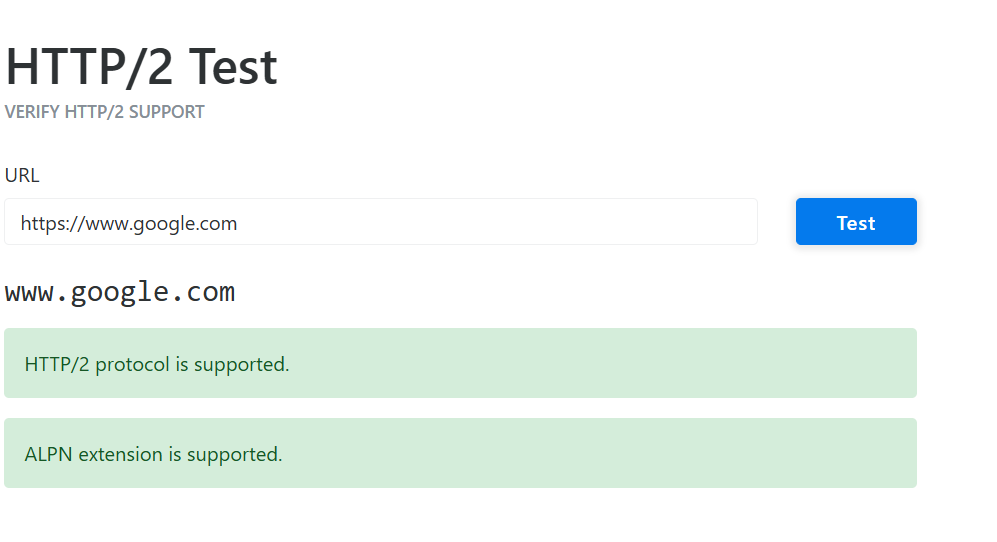
**Сниффинг:** Перехватчики увидят только мусор вместо паролей.

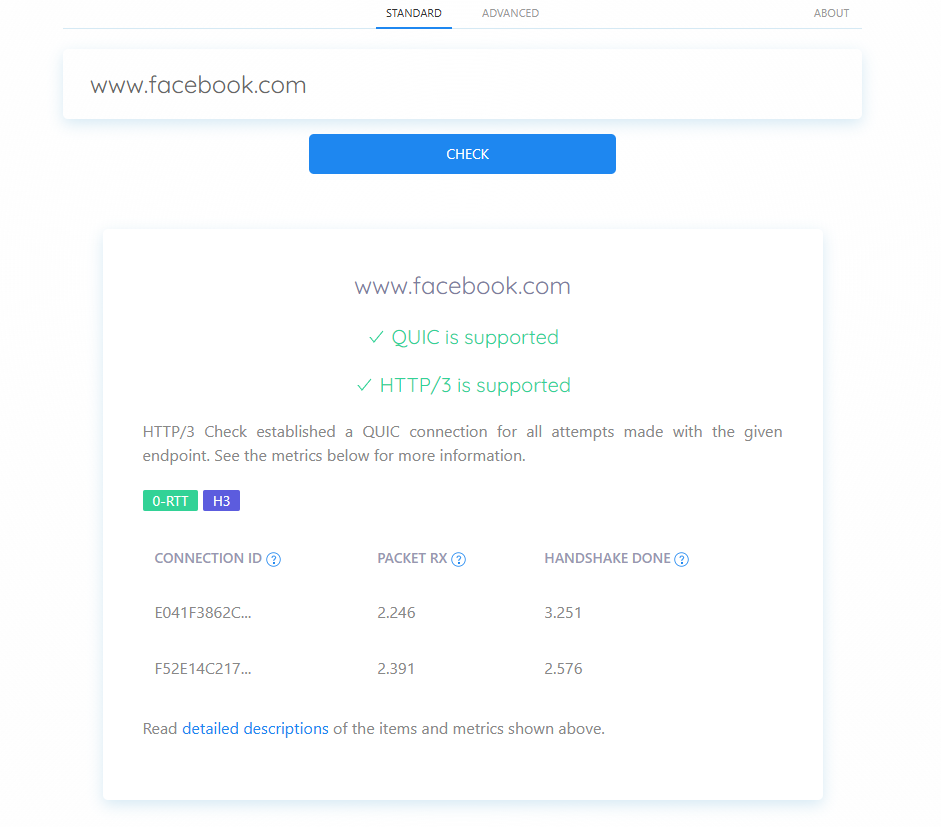
1. **Доверие пользователей**

Браузеры помечают HTTP-сайты как Не защищенные (может отпугнуть посетителей).

4. Используйте инструмент https://tools.keycdn.com/http2-test или DevTools (вкладка Network → Protocol), чтобы определить, какие сайты используют HTTP/1.1, HTTP/2 и HTTP/3. Найдите примеры таких сайтов и сравните, какие ключевые различия между версиями: например, использование мультиплексирования в HTTP/2, уменьшение задержек, бинарный формат протокола и поддержка push-ресурсов.







**Ключевые различия между HTTP/1.1, HTTP/2 и HTTP/3**

HTTP/1.1

* Текстовый формат протокола
* Заголовки и данные передаются в виде текста (ASCII), что увеличивает объем передаваемой информации.
* Отсутствие мультиплексирования
* Каждый ресурс (CSS, JS, изображение) требует отдельного TCP-соединения (или очереди в одном соединении).
* Возникает проблема Head-of-Line Blocking (блокировка очереди запросов).
* Нет сжатия заголовков
* Заголовки передаются в исходном виде, что повышает нагрузку.
* Отсутствие Server Push
* Сервер не может proactively отправлять ресурсы клиенту до их запроса.
* Высокие задержки
* Множественные TCP-соединения увеличивают время загрузки страницы.

HTTP/2

* Бинарный формат протокола
* Данные передаются в бинарном виде (не текстом), что уменьшает объем передаваемой информации и ускоряет парсинг.
* Мультиплексирование
* Возможность передавать несколько ресурсов параллельно в рамках одного TCP-соединения.
* Устраняет проблему Head-of-Line Blocking на уровне приложения.
* Сжатие заголовков (HPACK)
* Заголовки HTTP сжимаются, что уменьшает накладные расходы.
* Server Push
* Сервер может отправлять ресурсы клиенту до того, как они будут запрошены (например, CSS/JS для страницы).
* Приоритизация потоков
* Клиент может указывать приоритеты для загрузки ресурсов (например, сначала CSS, потом изображения).
* Обязательное использование TLS
* Большинство реализаций HTTP/2 работают только через HTTPS.

HTTP/3 (QUIC)

* Работа поверх UDP вместо TCP
* Устраняет задержки, связанные с TCP-рукопожатиями и восстановлением после потерь пакетов.
* Встроенное шифрование (TLS 1.3)
* Шифрование является обязательной частью протокола (нет незашифрованного HTTP/3).
* Улучшенное мультиплексирование
* Полностью устранена проблема Head-of-Line Blocking (даже на транспортном уровне).
* Быстрое восстановление соединения
* Сжатие заголовков (QPACK)
* Улучшенная версия HPACK, адаптированная для работы с QUIC.
* Поддержка Server Push
* Аналогично HTTP/2, но с оптимизациями для UDP.
* Сниженные задержки

QUIC уменьшает latency за счет:

* Отсутствия TCP-рукопожатий.
* Возможности начать передачу данных до завершения установки соединения.
* Независимость от операционной системы

1. Итоговое сравнение технологий
2. HTTP/1.1: Устаревший, медленный, без оптимизаций.
3. HTTP/2: Значительное ускорение за счет мультиплексирования и бинарного формата.
4. HTTP/3: Максимальная производительность за счет QUIC и UDP, идеально для современных сетей.