

Веб-технологии: уязвимости и безопасность

# Введение в веб

Введение в веб-технологии. Основные сетевые протоколы и программное обеспечение, которое мы будем использовать на курсе.

## Оглавление

[Введение](#)

[Видеоурок 1](#)

[Пример веб-клиентов](#)

[Пример веб-сервера](#)

[Вопросы для самоконтроля](#)

[Ссылки](#)

[Итоги](#)

[Видеоурок 2](#)

[Пример frontend-веб-сервера](#)

[Пример backend-веб-сервера](#)

[Настроим Nginx и модуль php-fpm](#)

[Вопросы для самоконтроля](#)

[Ссылки](#)

[Итоги](#)

[Видеоурок 3](#)

[Логи в Nginx](#)

[Установка Burp Suite](#)

[Инструменты разработчика в браузере](#)

[Вопросы для самоконтроля](#)

[Ссылки](#)

[Итоги](#)

## [Видеоурок 4](#)

[Прокси-сервер](#)

[Настройка браузера для Burp Suite](#)

[Proxy и HTTP history в Burp Suite](#)

[Intercept в Burp Suite](#)

[Repeater в Burp Suite](#)

[Target и Scope в Burp Suite](#)

[Сайты с HTTPS-шифрованием](#)

[Вопросы для самоконтроля](#)

[Ссылки](#)

[Итоги](#)

# Введение

Первый урок будет посвящен введению в веб-технологии, основным сетевым протоколам и программному обеспечению, которое мы будем использовать на курсе.

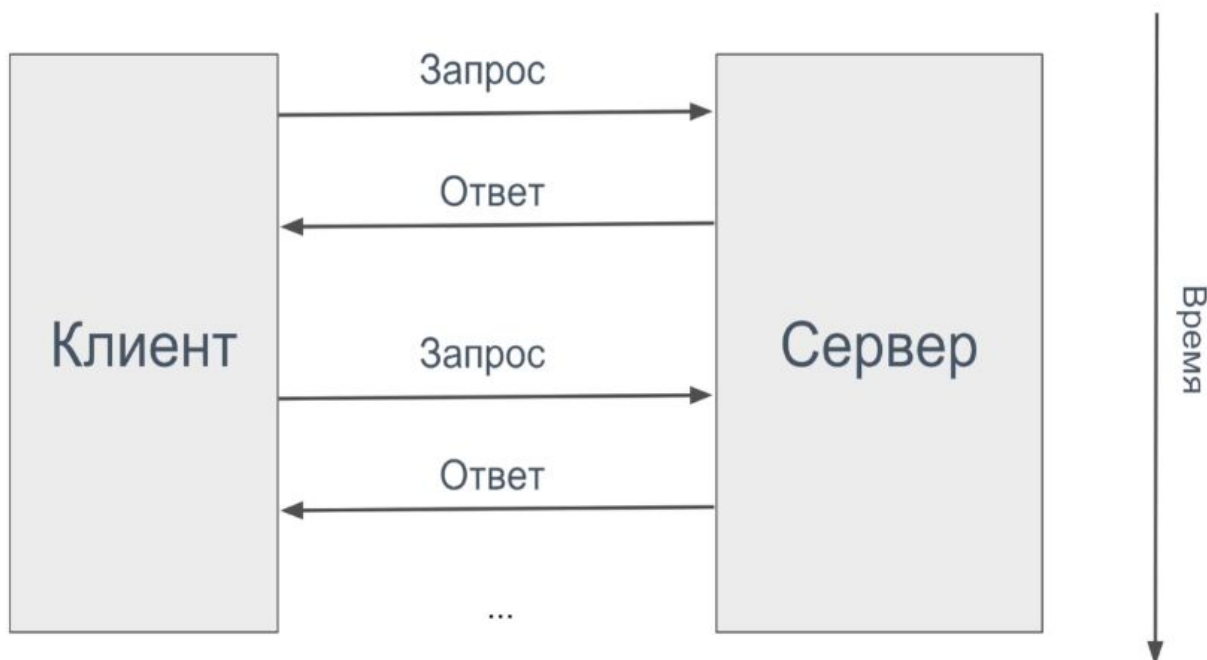
1. В первом видео мы рассмотрим общую схему веб-приложений, познакомимся с понятиями «клиент» и «сервер», узнаем, как работает протокол HTTP.
2. Во втором видео мы подробно рассмотрим, как устроена серверная часть веб-приложения, т. н. Server Side.
3. В третьем видео мы научимся пользоваться программным обеспечением, которое будет необходимо нам по ходу курса.
4. Из четвёртого видео мы узнаем, что такое прокси и зачем они нужны, и познакомимся с частным случаем прокси Burp Suite.

К концу урока вы будете понимать основные концепции работы веб-приложений, изучите, что такое клиент, сервер, прокси, HTTP, как устроена Server Side, что такое инструменты разработчика Chrome и как пользоваться прокси Burp Suite. Также вы сможете настроить инфраструктуру для дальнейшего курса.

# Видеоурок 1

В первом видеоуроке мы рассмотрим, как устроено взаимодействие в веб, различные примеры этого взаимодействия, узнаем, что такое клиент и что такое сервер.

В этом курсе под веб мы будем подразумевать всю сеть Интернет и связанные с ней технологии. Основной протокол, на котором общаются машины в веб, — HTTP. Подробно мы разберём этот протокол в следующих уроках, сейчас нам нужно понять общую концепцию его работы. Протокол HTTP основан на принципе «запрос-ответ», то есть клиент — тот, кто хочет получить ресурс — делает запрос на сервер. Сервер возвращает ответ клиенту с документом, данными, картинками или другими файлами. Клиент может отправлять несколько запросов на сервер, получать несколько ответов, таким образом и происходит взаимодействие по протоколу HTTP: клиенты делают запросы, сервер посылает им ответы на эти запросы. Протокол — набор правил, которые клиент и сервер соблюдают, чтобы общаться и понимать друг друга.

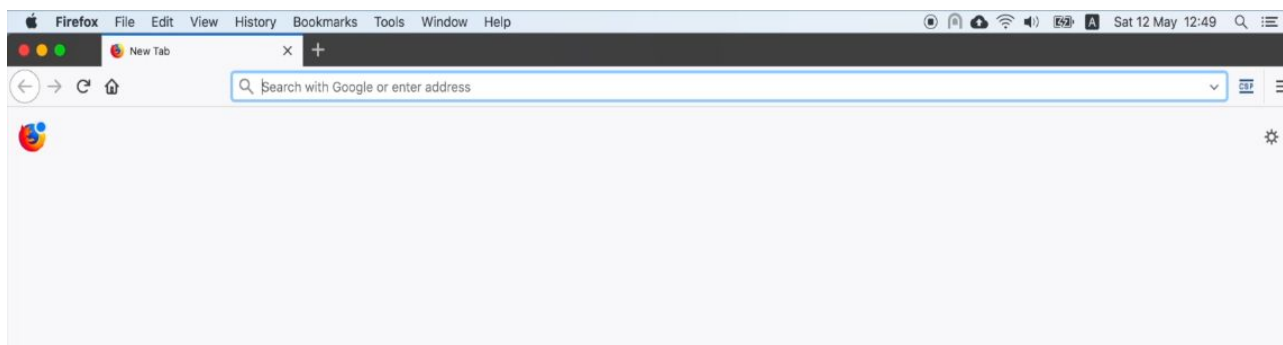


Большое количество протоколов в вебе и сетях описываются диаграммами, похожими на ту, которую вы видите. Здесь по вертикальной оси вниз идет время, соответственно, запрос, который выше, был первым по времени. На него через какое-то время пришел ответ и так далее. Потом опять был запрос, позже был получен ответ, и так далее.

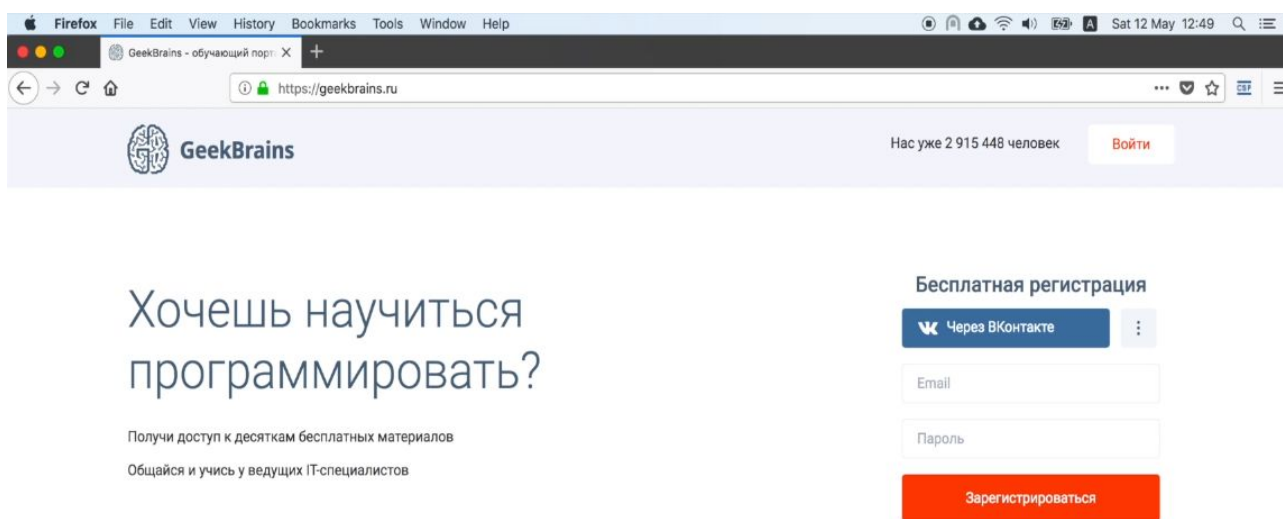
## Пример веб-клиентов

Давайте рассмотрим несколько примеров веб-клиентов. Первый пример вам всем хорошо знаком — это веб-браузер, через который вы зашли на этот сайт и смотрите этот курс — скорее всего, это

Firefox. Пример браузера Firefox в Ubuntu:



Давайте выполним HTTP-запрос серверу <https://geekbrains.ru> к его главной странице:



Мы успешно выполнили запрос страницы <https://geekbrains.ru> и сервер вернул нам ее. Браузер показал эту страницу так, как она должна выглядеть.

Но существуют и другие клиенты, напριме, программа командной строки `curl`. Откроем терминал и выполним программу `curl`. Если у вас еще не установлена эта программа, то ее можно установить на Linux через пакетный менеджер `apt`-командой:

```
sudo apt install curl
```

Пакетный менеджер — это программа, которая позволяет удобно устанавливать программы в командной строке или в графической оболочке. Например, на Mac используется другой пакетный менеджер — `brew`. Поэтому для установки `curl` на Mac используется команда:

```
brew install curl
```

Если вы уже установили `curl`, сделаем запрос к сайту из него:

```
curl https://geekbrains.ru
```

Мы получим много текста, он представлен в формате HTML:

```
geek@ubl6:~$  
geek@ubl6:~$ curl https://geekbrains.ru  
<!DOCTYPE html><html><head><script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':new  
Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],j=d.createElement(s),dl=l!='  
dataLayer'?&l='+l:'';j.async=true;j.src='https://www.googletagmanager.com/gtm.js?id='+i+dl;f.par  
entNode.insertBefore(j,f);})(window,document,'script','dataLayer','GTM-TZ45BH');</script><!--new  
user registration event for Google tag manager!--><script charset="UTF-8" src="//cdn.sendpulse.c  
om/9dae6d62c816560a842268bde2cd317d/js/push/61838643890e03d22aa7e81129f3eef2_1.js" async></script  
><script>if (!window.carrotquest) {  
  (function(){  
    function Build(name, args){return function(){window.carrotquestasync.push(name, arguments);}}
```

Что такое HTML, мы подробно изучим в дальнейших уроках. Сейчас нам важно понимать, что HTML — это некоторый способ представления веб-страницы. В браузере он отображается нормально в виде страницы, т.к. браузер умеет его отображать. А командная строка не умеет его по-другому отображать, поэтому она пишет текст как он есть.

Но клиентом могут быть не только программы, которые написаны кем-то еще, но и те, которые написали мы сами. Давайте посмотрим ниже на исходный код скрипта request.py:

```
#!/usr/bin/env python  
import requests  
  
r = requests.get("https://geekbrains.ru")  
print(r.text)
```

Здесь всего несколько строчек: мы используем библиотеку request и делаем запрос через функцию requests.get. Подробнее о языке Python вы узнаете на следующих курсах, сейчас нам важно понимать, что языки программирования тоже умеют делать HTTP-запросы к серверу.

Давайте теперь запустим Python-скрипт request.py, который сделает запрос на наш сайт:

```
./request.py
```

Как мы видим, наш скрипт выводит результат в терминал, и он тоже не умеет отображать страницу так, как в браузере, поэтому мы увидим простой текст.

Но веб-клиентами могут быть не только браузер, утилита командной строки или программа на любом языке программирования — также клиентами могут быть приложения на компьютере, телефоне, а в последнее время — телевизоры, холодильники и даже лампочки. Устройства интернета вещей — то есть лампочки, холодильники — тоже умеют делать запросы и получать ответы, то есть являются полноценными участниками веб-взаимодействия.

## Пример веб-сервера

Давайте разберемся, что же такое сервер. В нашем курсе под сервером мы будем понимать программное обеспечение, которое обрабатывает запросы и выдает ответы клиенту. Давайте посмотрим на пример сервера.

Рекомендуется сразу поставить виртуальную машину и устанавливать на нее любые пакеты, чтобы экспериментировать внутри нее, а не на основной системе. Откроем виртуальную машину с Ubuntu Linux. Если вы еще не установили Nginx, вы можете также сделать это на Ubuntu командой:

```
sudo apt-get install nginx
```

Либо на Mac:

```
brew install nginx
```

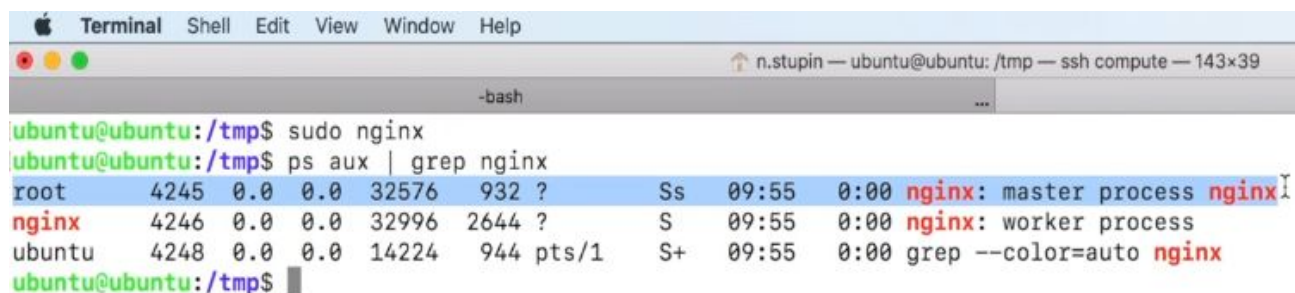
Теперь давайте запустим сервер Nginx. Для этого введем команду:

```
sudo nginx
```

Команда `sudo` означает, что следующая команда будет выполнена с привилегиями суперпользователя `root`, а Nginx запускает сам веб-сервер. Если все хорошо и никаких сообщений об ошибках мы не увидим, то значит скорее всего наш веб-сервер запустился. Мы можем проверить это с помощью команды `ps`:

```
ps aux | grep nginx
```

Команда `ps` выводит все программы, которые в данный момент запущены, а программа `grep` выберет для нас только те строки, в которых встречается слово Nginx:



```
Terminal Shell Edit View Window Help
n.stupin — ubuntu@ubuntu: /tmp — ssh compute — 143x39
-bash
ubuntu@ubuntu:/tmp$ sudo nginx
ubuntu@ubuntu:/tmp$ ps aux | grep nginx
root      4245  0.0  0.0 32576  932 ?    Ss   09:55   0:00 nginx: master process nginx
nginx     4246  0.0  0.0 32996 2644 ?    S    09:55   0:00 nginx: worker process
ubuntu    4248  0.0  0.0 14224  944 pts/1  S+   09:55   0:00 grep --color=auto nginx
ubuntu@ubuntu:/tmp$
```

Как мы можем видеть, Nginx успешно запущен, и это значит, что мы можем сделать запрос к веб-серверу.

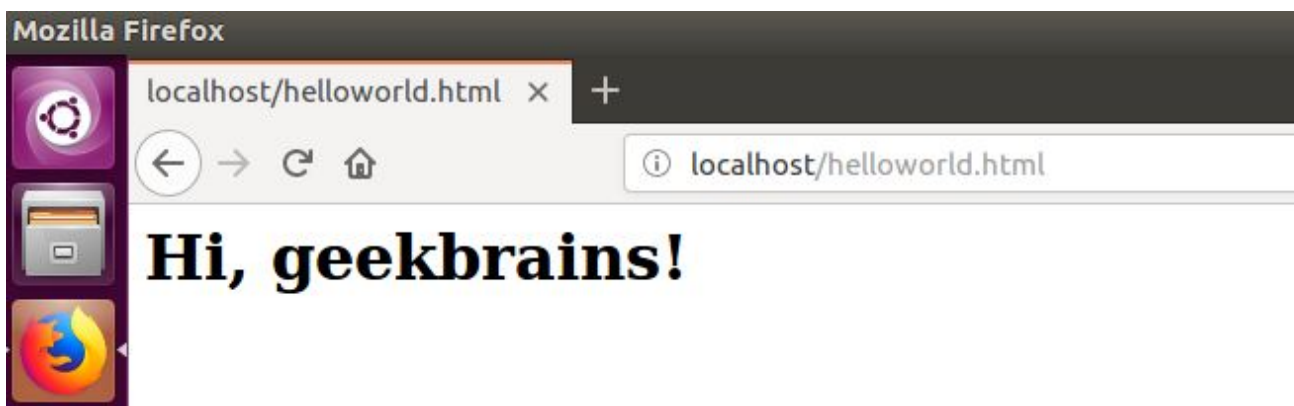
Давайте выполним запрос к нему через браузер на адрес <http://127.0.0.1> или <http://localhost>:



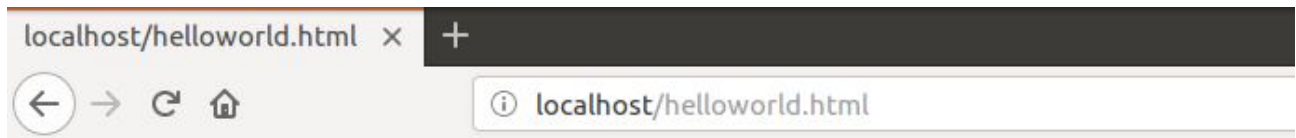
Теперь давайте разместим файл helloworld.html в домашнем каталоге веб-сервера. Например, в Nginx — это каталог /var/www/html, по умолчанию:

```
<h1>Hi, geekbrains!</h1>
```

Затем мы сможем запросить этот файл через браузер:



Давайте посмотрим, как этот файл будет выглядеть, если мы получим его через curl:

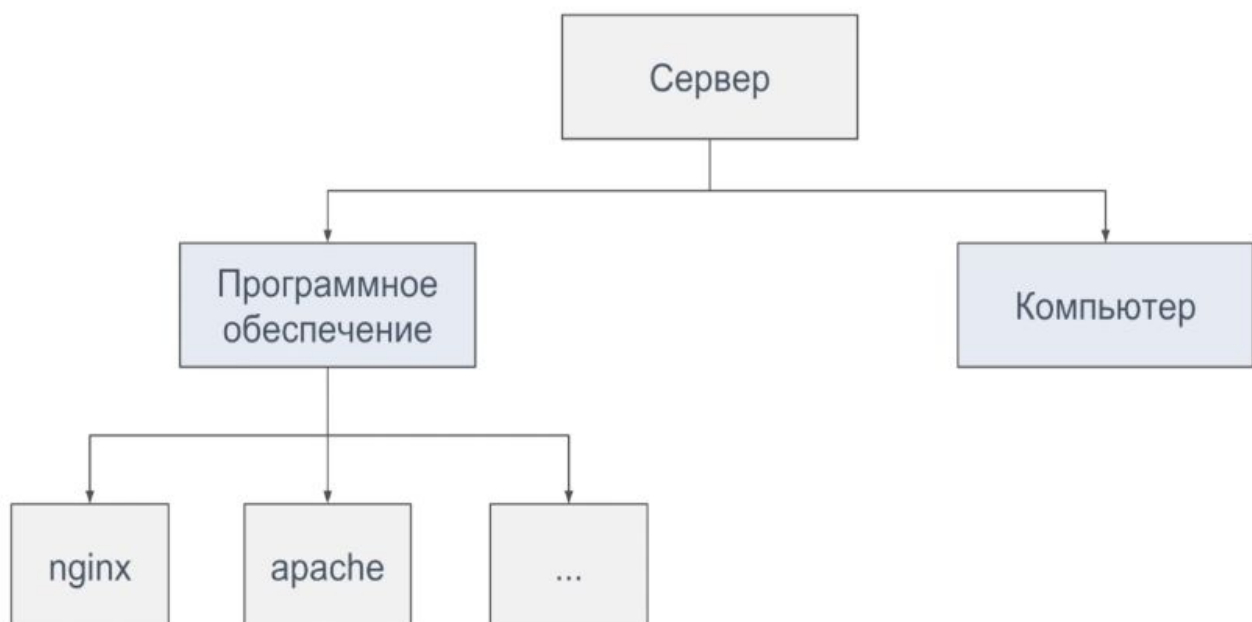


# Hi, geekbrains!

```
geek@ub16: ~  
geek@ub16:~$ curl http://localhost/helloworld.html  
<h1>Hi, geekbrains!</h1>  
geek@ub16:~$
```

Как мы видим, жирного шрифта уже нет, потому что curl выводит файл в консоль, а она не умеет распознавать HTML и поэтому отображает его просто как текст.

Чаще всего в нашем контексте мы будем говорить, что сервер — это программное обеспечение. Но на самом деле слово «сервер» применяется еще и к целому компьютеру, на котором установлено программное обеспечение сервера. Чаще всего термин «сервер» в значении «компьютер» применяется на больших сайтах типа geekbrains.ru, где сервер действительно устанавливается на отдельную машину с мощным «железом», чтобы можно было обрабатывать тысячи и сотни тысяч запросов одновременно.



Сервер как программное обеспечение не ограничивается одним только Nginx. Существует множество других серверов, например, Apache2, Lighttpd, Node.js и так далее, их очень-очень много. Подробнее веб-серверы мы разберем во втором видео этого урока.



## Вопросы для самоконтроля

1. Что такое протокол HTTP?
2. Что такое веб-клиент и сервер?
3. Какие виды веб-клиентов и веб-серверов вы знаете?

## Ссылки

1. [О модели взаимодействия «клиент-сервер» простыми словами.](#)
2. [Что нужно знать про HTTP-протокол.](#)
3. [5 лучших открытых веб-серверов.](#)
4. [Что лучше, Nginx или Apache?](#)
5. [Установка Nginx на Ubuntu 16.04.](#)

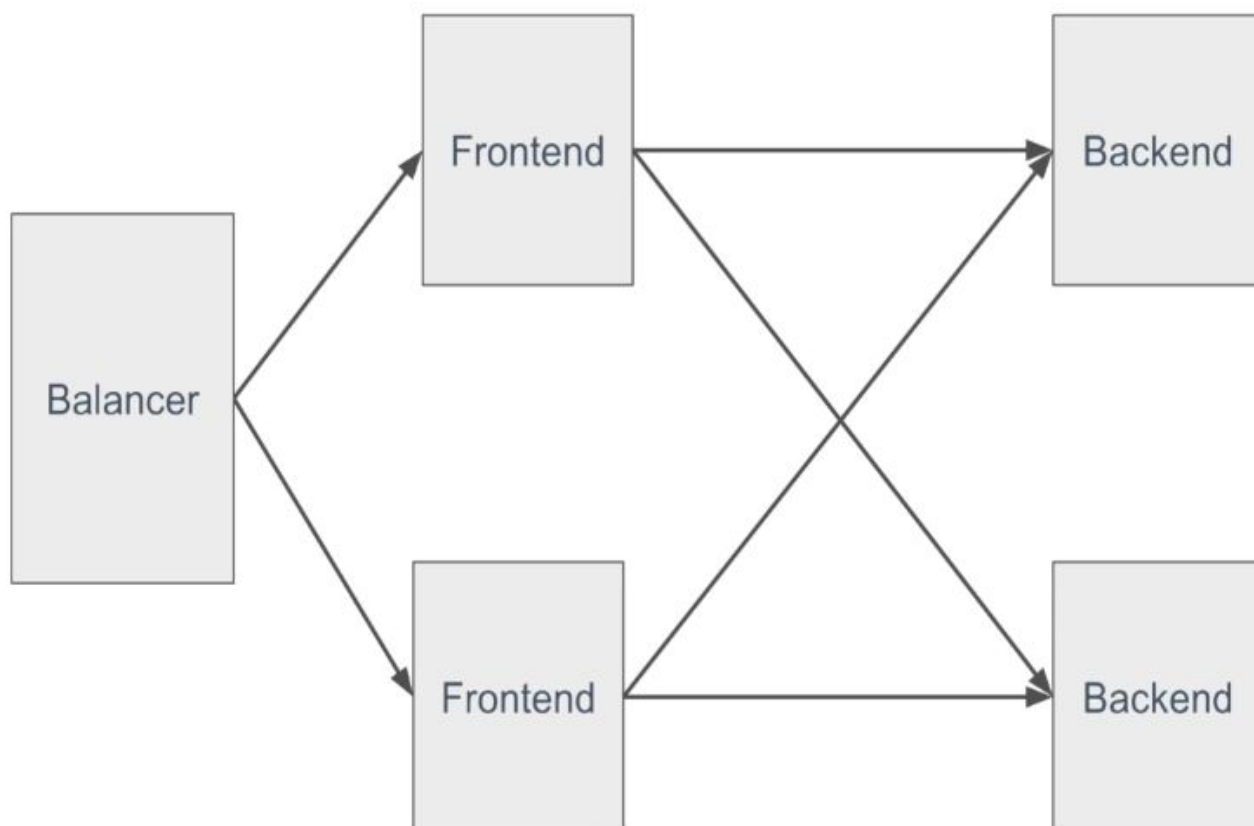
## Итоги

Итак, в первом видео вы узнали, как работают веб-приложения, что такое протокол HTTP, как он устроен на верхнем уровне, почему происходит деление на сервер и клиента, как они между собой взаимодействуют, а также познакомились с основными понятиями веба и увидели примеры клиентов и серверов.

## Видеоурок 2

В этом видеоуроке мы рассмотрим с вами серверы и Server-side-часть веб-приложений. В этом видео мы поймем, что сервер на самом деле не так уж и прост, и что чаще всего это даже не одна машина, а несколько. Мы рассмотрим различные типы серверов: frontend-сервер (FE), backend-сервер (BE) и балансировщик Balancer (NLB). Мы посмотрим, для чего они нужны, что они умеют и как мы на практике можем их применить.

На слайде ниже вы видите общую схему связи различных веб-серверов. Давайте разберемся, как же запрос клиента обрабатывается, когда он приходит на сервер?



Первым делом запрос клиента попадает на так называемый [балансировщик](#). Из названия понятно, что Balancer (или, как еще говорят, NLB — Network Load Balancer) выполняет распределение нагрузки — иными словами, балансировку между несколькими серверами. В данном случае Balancer отправляет запрос либо на верхний, либо на нижний [frontend-сервер](#). Запрос приходит на один из FE-серверов, а далее FE решает, что же сделать с этим запросом. Если это простой запрос, и он не требует много времени на обработку (например, картинки, стиля CSS или статической HTML-страницы), сервер FE может сразу вернуть эту страницу пользователю. Если же запрос требует чего-то более сложного, например, выполнить какие-то действия с помощью скрипта или же посмотреть в базе данных, сколько денег у клиента на счету, FE-сервер перенаправляет запрос на [backend-сервер](#). Серверы BE, в свою очередь, выполняют более сложные действия: исполняют скрипты, делают сложные вычисления и по результатам возвращают ответ на FE-сервер. Фронтенд-сервер, в свою очередь, — на балансир, и с него уже ответ отправляется запросившему его пользователю.

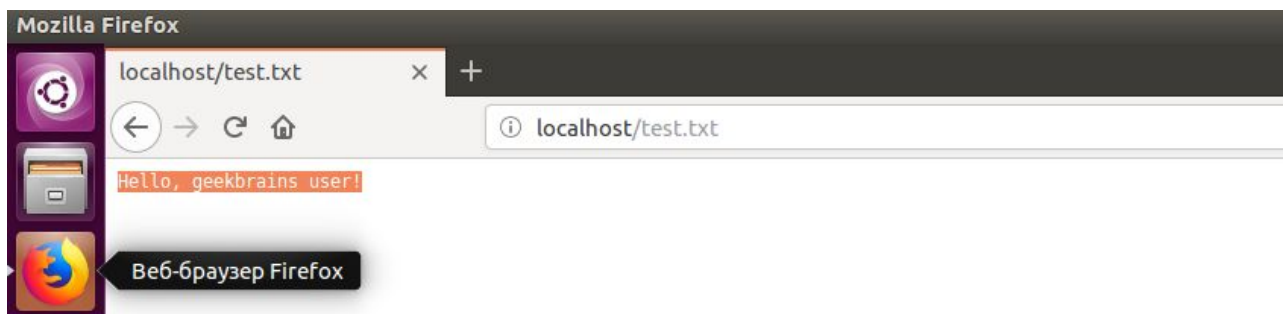
## Пример frontend-веб-сервера

Рассмотрим возможности фронтенд- и бэкенд-серверов. Используем Nginx в качестве демонстрации того, что может фронтенд-сервер. Для этого нам необходимо добавить какой-нибудь статический файл в директорию, из которой Nginx отдает файлы. По умолчанию это директория `/var/www/html`. Давайте создадим файл `test.txt`, напомним в нем **Hello, GeekBrains user!**:

```
echo "Hello, GeekBrains user!" | sudo tee test.txt
```

Вам необходимо выполнить эту команду от имени суперпользователя: директория `/var/www/html` требует прав суперпользователя, чтобы создавать, изменять или удалять файлы. Также вы можете использовать любой удобный вам текстовый редактор, например Nano или Vi.

Теперь попробуем получить этот файл с сервера:



Мы получили именно то содержимое, которое записывали. Но, когда файлы статичные, одинаковые для всех пользователей, это не удовлетворяет потребности бизнеса. Поэтому появилась необходимость получать динамические файлы, которые, как правило, генерируются бэкенд-сервером.

Рассмотрим пример такого файла, в качестве backend-языка программирования используем PHP. Чтобы настроить взаимодействие Nginx и PHP, нам потребуется изменить конфигурационный файл Nginx.

## Пример backend-веб-сервера

Откроем терминал и перейдем в директорию с конфигурационными файлами Nginx. Для этого выполним команду `cd`:

```
cd /etc/nginx/sites-available/
```

Это директория, в которой находятся конфигурационные файлы Nginx. Выполним команду `ls`, чтобы посмотреть, какие конфигурационные файлы у нас есть сейчас. Мы увидим, что есть только конфигурационный файл по умолчанию — `default`. Чтобы не создавать свой файл с нуля, скопируем конфигурационный файл по умолчанию.

Действия с файлами в директории `/etc/nginx` нужно выполнять от суперпользователя, потому что только он имеет право на изменение, удаление и добавление новых файлов в этой директории. Командой `cp` мы скопируем файл `default` в файл `test.conf`:

```
r@r:/etc/nginx/sites-available$ sudo cp default test.conf
```

В Nginx есть одна особенность — конфигурационные файлы веб-сайтов находятся в папке `sites-available`, но они не активируются просто так. Для этого необходимо добавить ссылку на этот файл в папке `sites-enabled`. Это сделано, чтобы можно было включать и отключать конфиг-файлы по мере необходимости, при этом не перемещая и не удаляя сам файл конфигурации.

Создадим ссылку на новый файл `test.conf`, для этого выполним команду:

```
r@r:/etc/nginx/sites-available$ ll
-rw-r--r-- 1 root root 2074 фев 12 2017 default
-rw-r--r-- 1 root root 2048 апр 2 05:54 test.conf
r@r:/etc/nginx/sites-available$ cd ../sites-enabled/
r@r:/etc/nginx/sites-enabled$ sudo ln -s ../sites-available/test.conf test.conf
r@r:/etc/nginx/sites-enabled$ ll
lrwxrwxrwx 1 root root 28 апр 2 05:57 test.conf ->
../sites-available/test.conf
```

Как вы увидите, ссылки в терминале подсвечиваются другим цветом — это нужно для вашего удобства.

## Настроим Nginx и модуль php-fpm

Теперь мы отредактируем файл `test.conf` так чтобы он удовлетворял необходимым требованиям. Мы должны убрать директиву `default_server`, поменять `server_name` на `localhost` и включить поддержку PHP. У вас должно получиться вот так:

```
r@r:/etc/nginx/sites-enabled$ cat test.conf
# Default server configuration
#
server {
    listen 80;
    listen [::]:80;

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;

    server_name localhost;

    location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =404;
    }

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    location ~ \.php$ {
```

```

        include snippets/fastcgi-php.conf;
#
#   # With php7.0-cgi alone:
#   fastcgi_pass 127.0.0.1:9000;
#   # With php7.0-fpm:
fastcgi_pass unix:/run/php/php7.0-fpm.sock;
    }
}

```

Запишем файл и выйдем из него.

Теперь перезапускаем Nginx, чтобы изменения вступили в силу. Выполним команду:

```
sudo nginx -s reload
```

Если сообщений об ошибке нет, то все хорошо и конфигурационный файл был написан без синтаксических ошибок.

Для поддержки PHP версии 7.0 (обратите внимание на версию: у вас может быть другая, поэтому проверяйте правильность написания команды) нужно установить модуль PHP-FPM:

```
sudo apt install -y php7.0-fpm
```

Теперь создадим новый файл `hellouser.php` в папке `/var/www/html`:

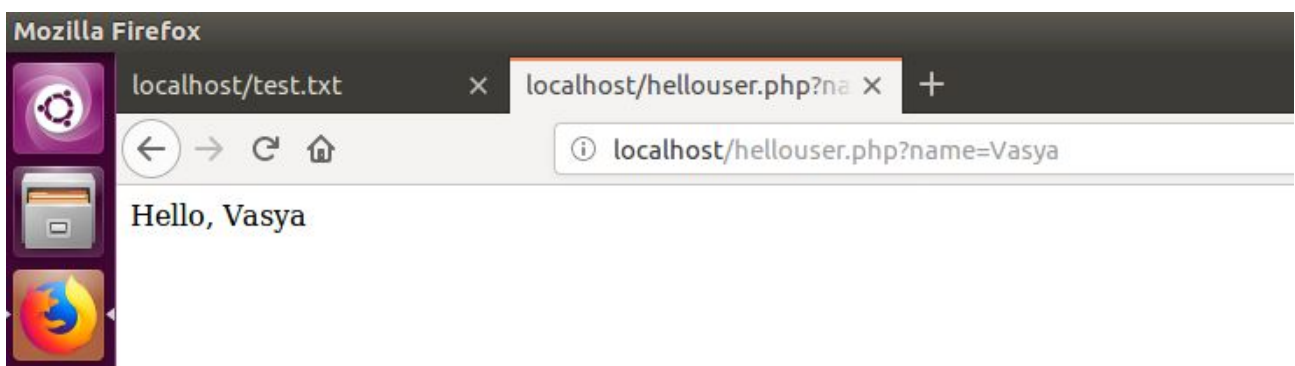
```
sudo nano /var/www/html/hellouser.php
```

```

<?php
    echo 'Hello, ' . $_GET['name'];
?>

```

Затем откроем его в браузере: <http://localhost/hellouser.php?name=Vasya>:



Это и есть динамическая генерация страницы, за которую отвечает бэкенд-сервер. Давайте обобщим наши знания о frontend- и backend-серверах.

Frontend-сервер нужен для отдачи картинок стилей и других файлов, которые редко меняются — обычно такие файлы называются статическими. Также frontend-сервер используют для балансировки нагрузки между бэкенд-серверами. Еще одно из применений frontend-сервера — это ограничение доступа к ресурсам и об этом мы поговорим подробнее в следующих уроках.

Не путайте frontend-сервер и frontend в браузере — это две разные вещи хоть и называются очень похоже. Frontend в браузере — это про язык JavaScript, про HTML и про то, что происходит у пользователя в браузере.

Бэкенд-сервер нужен, чтобы проводить большие вычисления, войти в базу данных, выполнять скрипты, например PHP или Python, и тому подобные вещи.

## Вопросы для самоконтроля

1. Что такое frontend- и backend-сервер, Balancer?
2. Для чего нужны frontend- и backend-сервер?
3. Какие виды frontend- и backend-веб-серверов вы знаете?

## Ссылки

1. [Простыми словами о «фронтенде» и «бэкенде»: что это такое и как они взаимодействуют.](#)
2. [Учимся работать с запросами GET и POST.](#)
3. [«Hello, \(real\) world!» на PHP в 2017 году.](#)

## Итоги

Вы узнали общую архитектуру серверов и как они связаны между собой, узнали, что такое frontend- и backend-сервер, Balancer. Вы узнали, для чего они нужны. Мы также смогли на живом примере настроить frontend-сервер nginx, и написали скрипт на PHP для backend-сервера, а затем успешно выполнили его в браузере.

## Видеоурок 3

В этом видеоуроке мы рассмотрим, как настроить и установить программное обеспечение, которое будем в дальнейшем применять на курсе:

1. Сначала мы уделим внимание еще нескольким функциями Nginx.
2. Установим Burp Suite.

3. Изучим инструменты разработчика в Chrome и Firefox (на самом деле инструменты разработчика в них очень похожи).

## Логи в Nginx

Nginx, как и любая другая программа, записывает сообщение о том, что она в данный момент делала, какие запросы обрабатывала и какие ошибки у нее возникают. Нужно это в случае, если что-то не получается, чтобы можно было всегда обратиться в лог и посмотреть, что именно пошло не так.

Давайте посмотрим, где располагаются логи Nginx. Откроем терминал Ubuntu. Обычно логи в Linux хранятся в директории в /var/log. Перейдем в нее командой `cd` и посмотрим, какие файлы есть директории /var/log/nginx:

```
r@r:~$ cd /var/log/nginx
r@r:/var/log/nginx$ ls
access.log  error.log
```

Мы видим, что сейчас есть только два файла: `access.log` и `error.log`. В `error.log` у нас хранятся сообщения об ошибках, а в `access.log` — о запросах, которые приходили на сервер. Как только эти два файла станут слишком большими, Nginx автоматически переименует их и создаст новые `access.log` и `error.log`.

Посмотрим файл лога командой `tail access.log`:

```
r@r:/var/log/nginx$ tail access.log
::1 - - [05/Apr/2019:10:37:17 +0300] "GET /helloworld.html HTTP/1.1" 200 25 "-"
"curl/7.47.0"
::1 - - [05/Apr/2019:10:40:25 +0300] "GET /helloworld.html HTTP/1.1" 200 55 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0"
::1 - - [05/Apr/2019:10:40:26 +0300] "GET /favicon.ico HTTP/1.1" 404 152 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0"
192.168.1.8 - - [05/Apr/2019:10:50:29 +0300] "GET /favicon.ico HTTP/1.1" 404 209
 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/71.0.3578.99 YaBrowser/19.1.2.258 Yowser/2.5 Safari/537.36"
::1 - - [05/Apr/2019:10:53:58 +0300] "GET /test.txt HTTP/1.1" 200 24 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0"
::1 - - [05/Apr/2019:11:11:35 +0300] "GET /hellouser.php?name=Vasya HTTP/1.1"
200 43 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0"
::1 - - [05/Apr/2019:11:15:26 +0300] "GET /hellouser.php?name=Vasya HTTP/1.1"
200 43 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0"
::1 - - [05/Apr/2019:11:15:26 +0300] "GET /favicon.ico HTTP/1.1" 404 152 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0"
::1 - - [05/Apr/2019:21:05:52 +0300] "GET / HTTP/1.0" 200 21 "-"
"Lynx/2.8.9dev.8 libwww-FM/2.14 SSL-MM/1.4.1 GNUTLS/3.4.9"
192.168.1.100 - - [05/Apr/2019:21:06:34 +0300] "GET / HTTP/1.1" 200 52 "-"
"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
```

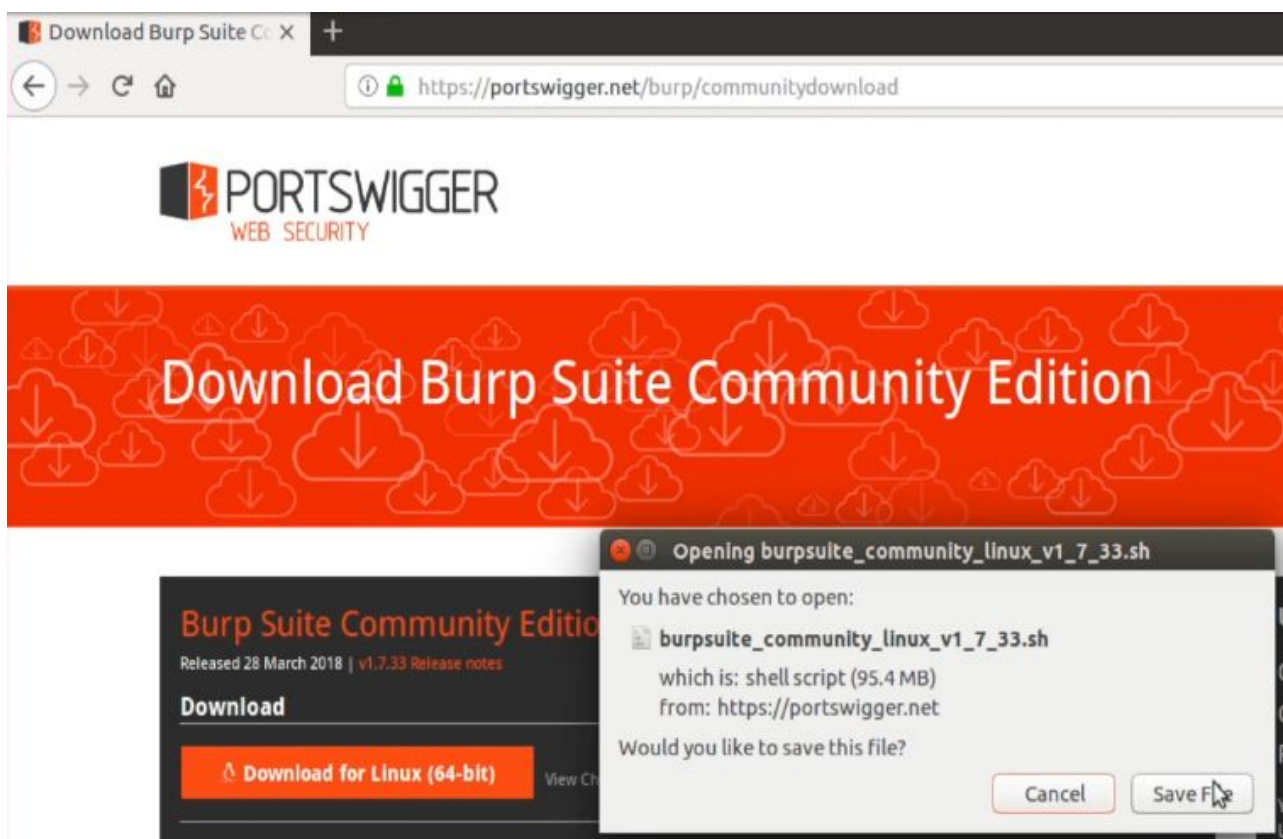
```
Chrome/72.0.3626.121 YaBrowser/19.3.1.828 Yowser/2.5 Safari/537.36"
r@r:/var/log/nginx$
```

Команда `tail` по умолчанию выводит последние десять строк указанного файла, что очень полезно, когда необходимо просмотреть файл в несколько гигабайт и в обычном текстовом редакторе мы бы долго его открывали. Здесь находятся запросы из предыдущего видео: есть IP-адрес клиента, который сделал запрос. IP-адрес — адрес машины, подробнее об IP-адресах мы узнаем в следующих уроках. Также здесь есть дата, когда был сделан запрос, сам запрос, код ответа и user-agent клиента. Обо всех этих параметрах мы также узнаем в следующих видео, а сейчас нам необходимо запомнить, что если нам нужно посмотреть, что происходило с Nginx, мы должны проверить эти лог-файлы.

## Установка Burp Suite

Теперь давайте разберемся, как установить Burp Suite. Для этого откроем браузер, введем в поисковую строку `burp suite`, перейдем на сайт <https://portswigger.net> — это разработчики Burp — и выбираем Community Edition (бесплатная версия).

Далее нажимаем Download for Linux. Если у вас Windows, тут будет предложение скачать установочный пакет Burp под Windows, если Mac, то для Mac. Выбираем «Сохранить файл»:

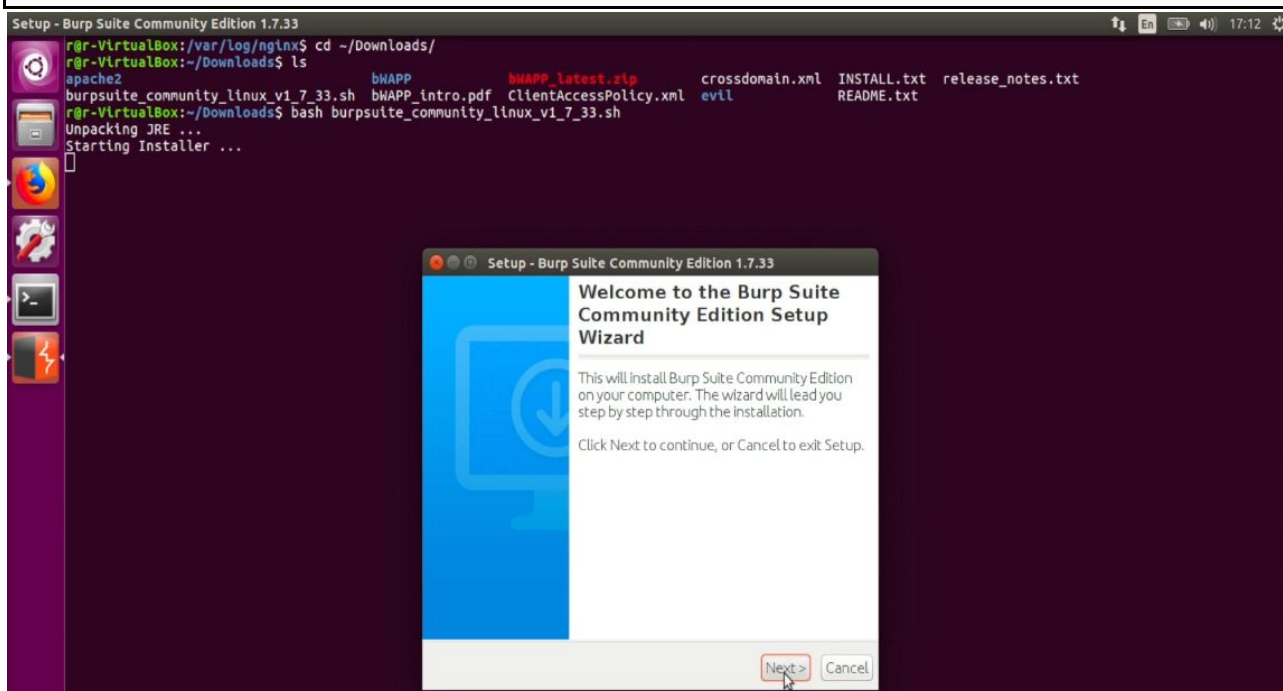


Теперь запустим установочный файл. Для этого откроем терминал и выполним команду `cd ~/Downloads/` (тильда ~ это обозначение нашей домашней директории). Так мы перейдем в

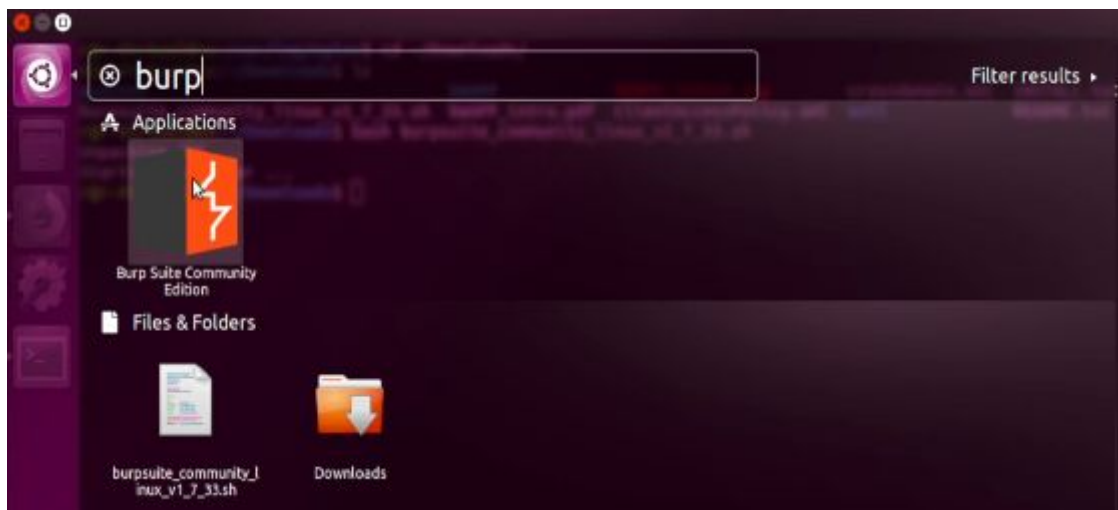


«Загрузки» и посмотрим файлы, которые там есть. Запустим установку командой (обратите внимание, что у вас может быть более новая версия, поэтому проверяйте правильность написания команды):

```
bash burpsuite_community_linux_v1_7_33.sh
```

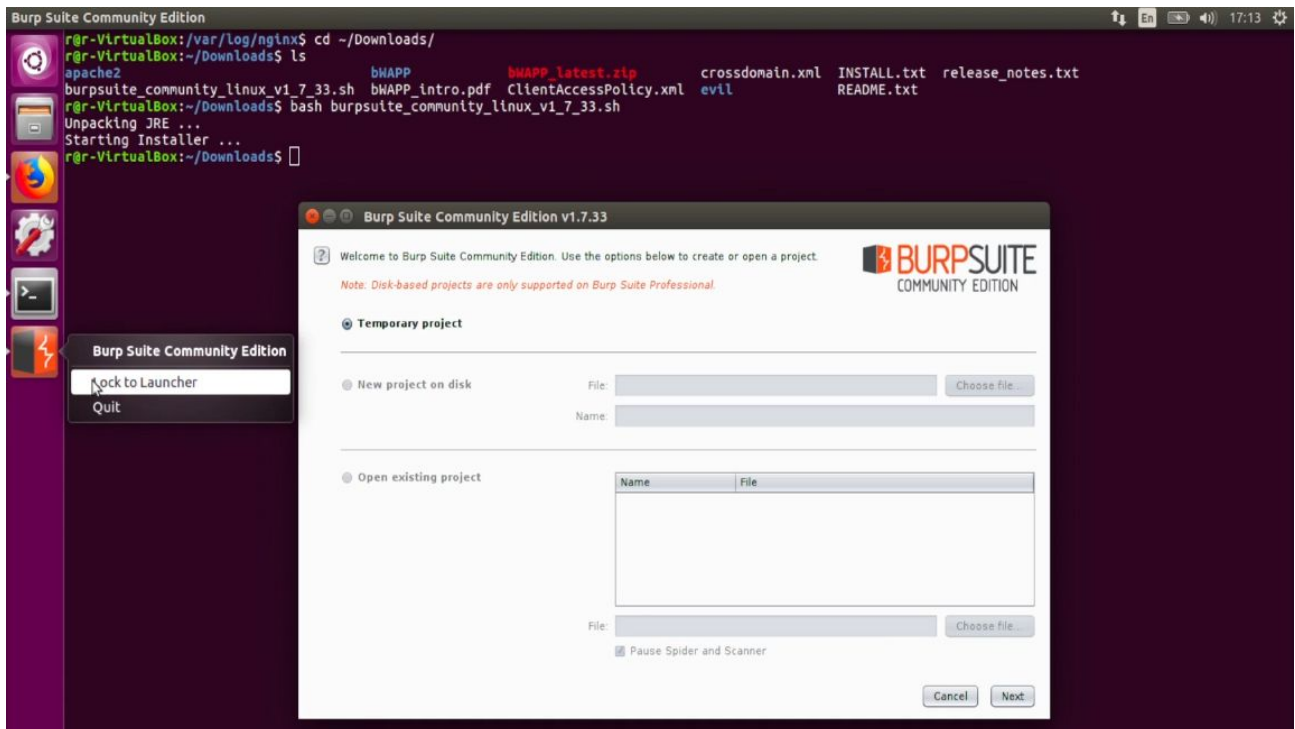


Теперь мы введём в поиске Ubuntu burp и увидим, что он успешно установился:

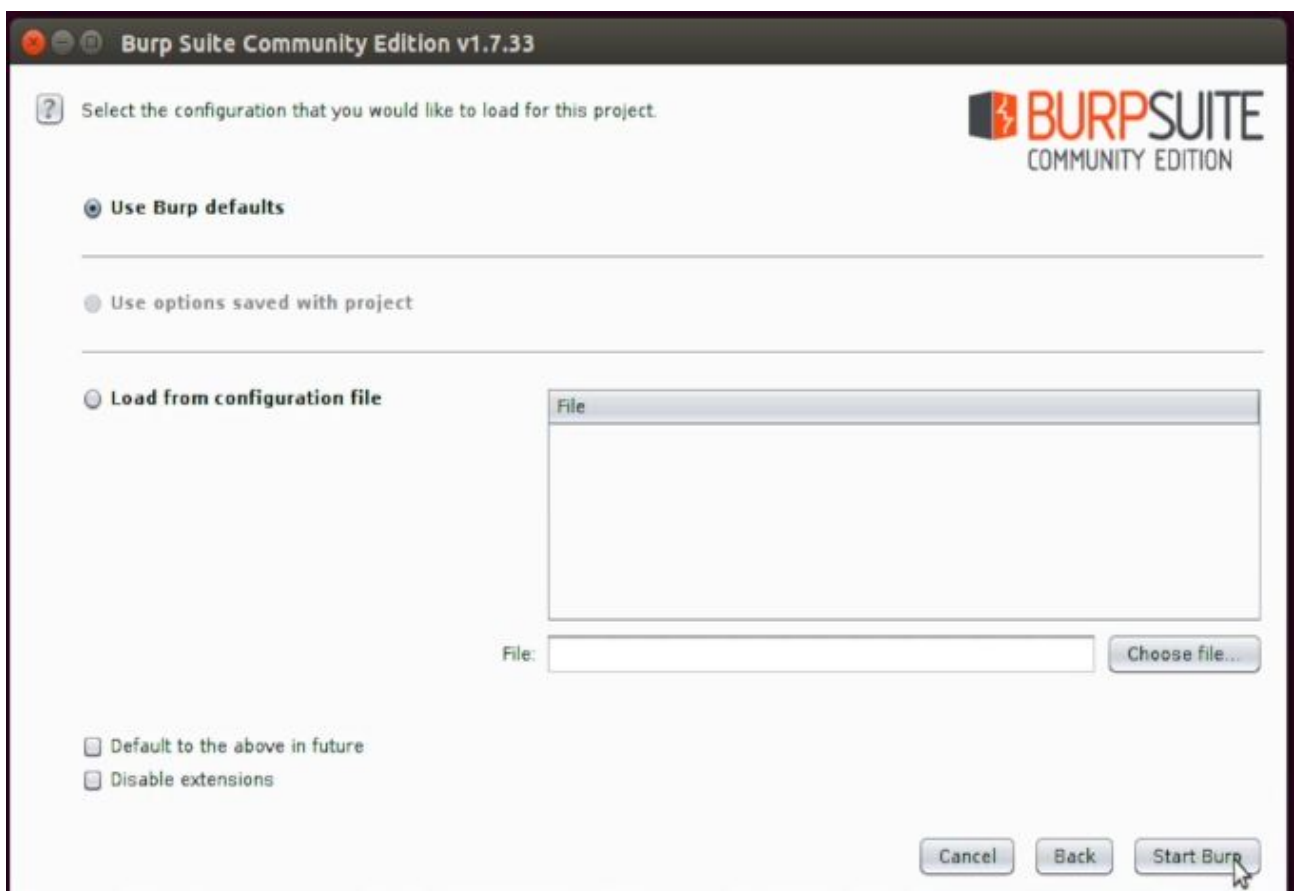


Открываем его и принимаем лицензионное соглашение. Burp Suite установлен, теперь для удобства рекомендуется закрепить его в панели быстрого запуска. После этого даже после закрытия Burp Suite не придется искать через поиск. Для этого щелкнем правой кнопкой мыши по иконке и нажмем Lock to

Launcher:



Далее мы создаем временный проект (Temporary project) и выбираем стандартные настройки (Use Burp defaults):



Мы успешно запустили Burp Suite. Подробнее про его функции и нужные нам возможности мы узнаем в следующем видеоуроке, а сейчас перейдем к инструментам разработчика.

## Инструменты разработчика в браузере

Откроем Firefox и нажмем правую кнопку мыши на странице Inspect Element (Q). Мы открыли инструменты разработчика. Из этих инструментов нам потребуется:

- Inspector — то, где мы сможем посмотреть документ, который вернул сервер.
- Console — консоль будет необходима для взаимодействия с JavaScript и понимания ошибок в JavaScript (об этом подробнее узнаем в следующих уроках).
- Также будет полезна вкладка Network. На ней будут отображаться все запросы клиента к серверу и от сервера к нам.

В целом инструменты разработчика в разных браузерах очень похожи, поэтому нет большой разницы, используете вы Chrome или Firefox — по сути они будут различаться внешним видом.

## Вопросы для самоконтроля

1. Какие лог-файлы есть у Nginx, где они находятся?
2. Для чего нужен Burp Suite?
3. Что такое инструменты разработчика и какие из них вы знаете? Имеются в виду браузерные инструменты.

## Ссылки

1. [Nginx-логирование, уровни логирования.](#)
2. [Burp Suite. Курс молодого CTF-бойца v 1.5.](#)
3. [Обзор инструментов разработки в браузерах.](#)

## Итоги

Итак, вы уже узнали, что такое логи, зачем они нужны и как можно посмотреть логи в Nginx. Вы научились устанавливать Burp Suite и использовать инструменты разработчика Chrome/Firefox, узнали про вкладку Network, Inspector и Console. В дальнейшем эти знания нам очень сильно помогут для исследования веб-сайтов на уязвимости.

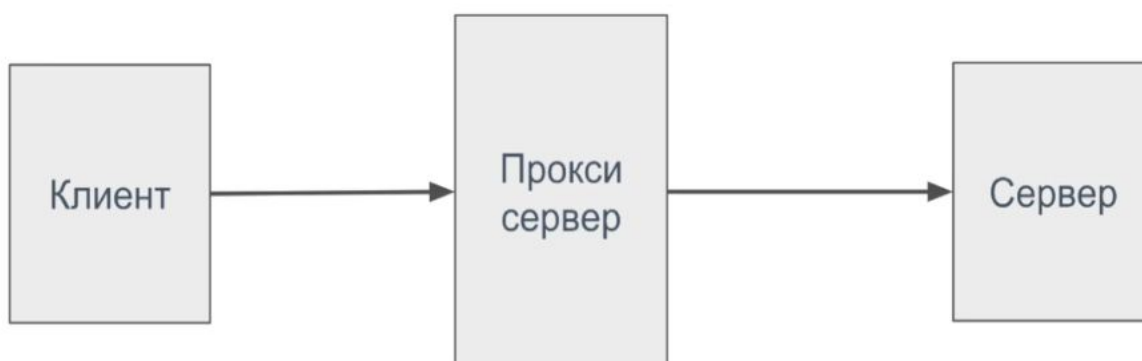
# Видеоурок 4

В этом видео мы рассмотрим:

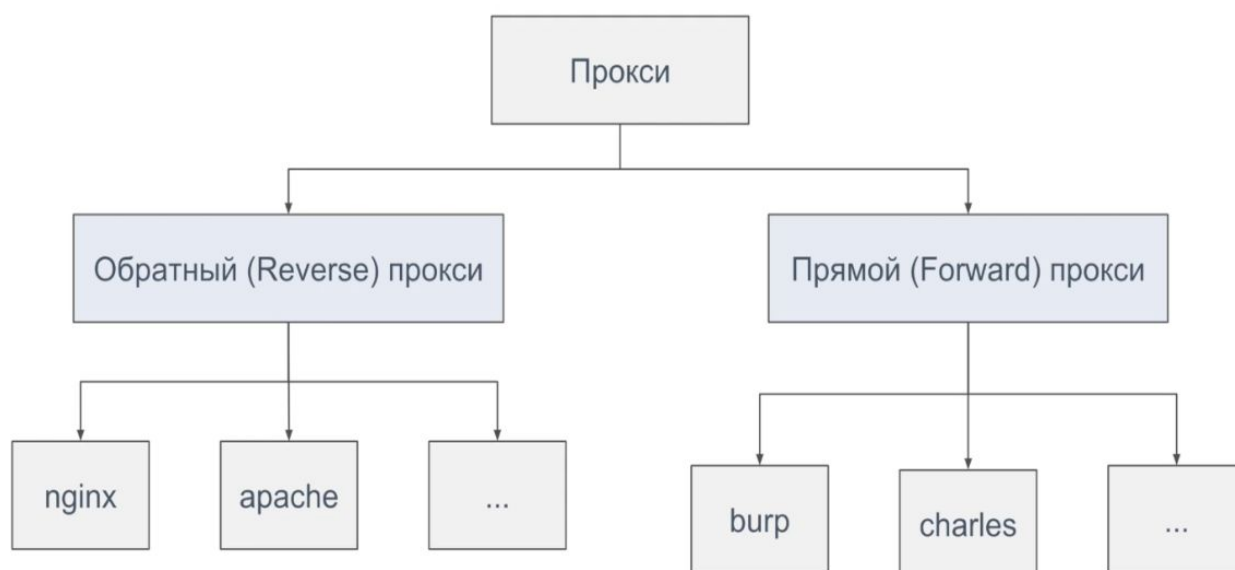
1. Что такое прокси-сервер, какие прокси бывают, зачем они нужны.
2. Также мы подробно рассмотрим, как можно пользоваться Burp Suite.

## Прокси-сервер

Прокси — сервер, который принимает запрос от клиента и передает его на сервер:



Применения у прокси бывают разные. Бывают прокси, которые нужны, чтобы посмотреть, что находится в запросе. Бывают прокси, которые нужны, чтобы модифицировать запрос. А бывают прокси такие, как, например, Balancer-сервер или frontend-сервер, которые необходимы, чтобы равномерно распределять нагрузку.



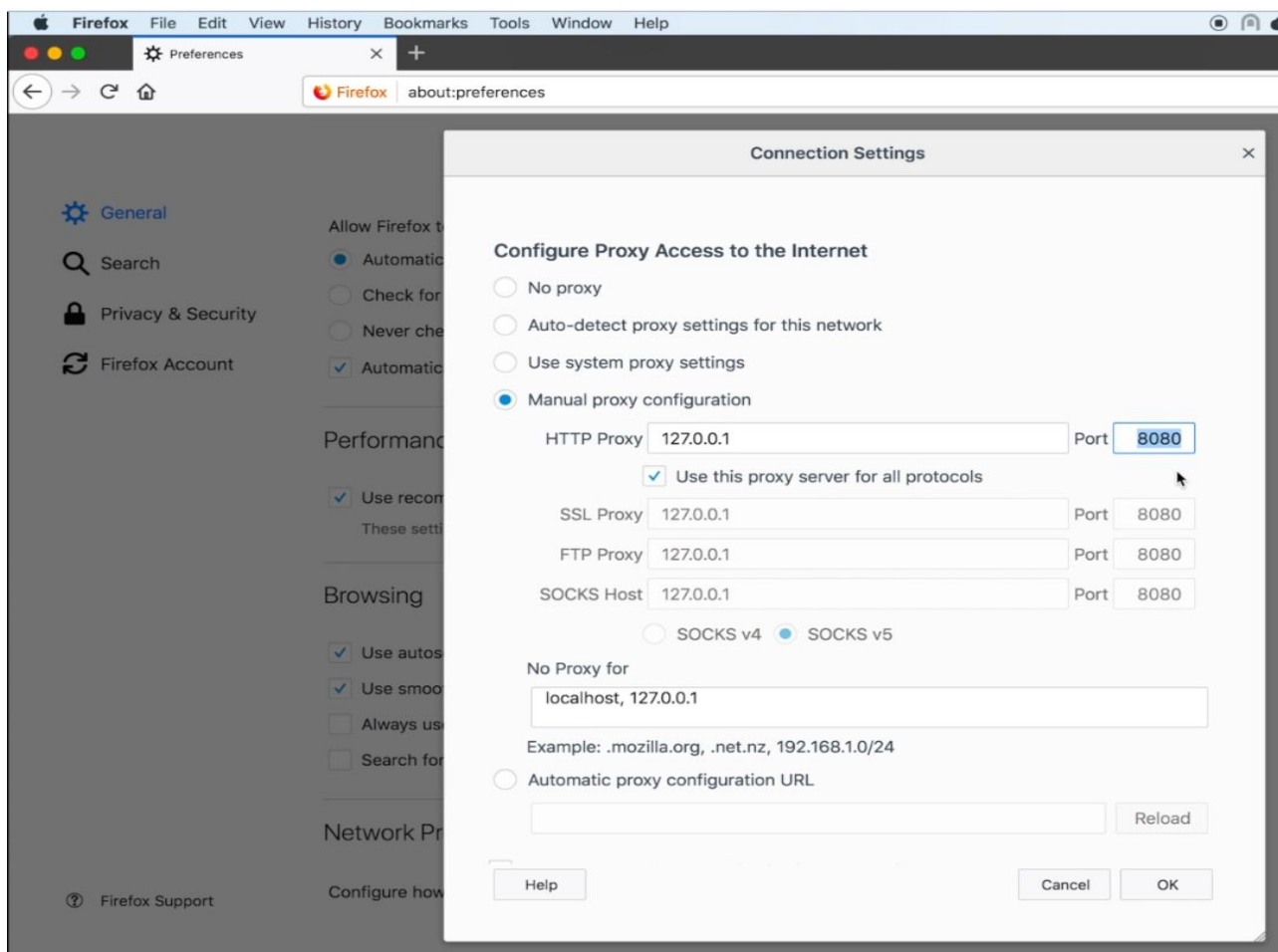
Один из видов классификации — это обратный прокси и прямой прокси. С обратными прокси мы уже знакомы — это, например, Nginx, Apache и другие веб-серверы. С прямым прокси мы познакомимся в

этом уроке на примере Burp Suite. Также существует множество других прямых прокси, но Burp — это фактически стандарт для тестирования веб-уязвимостей, поэтому именно его мы и будем изучать.

## Настройка браузера для Burp Suite

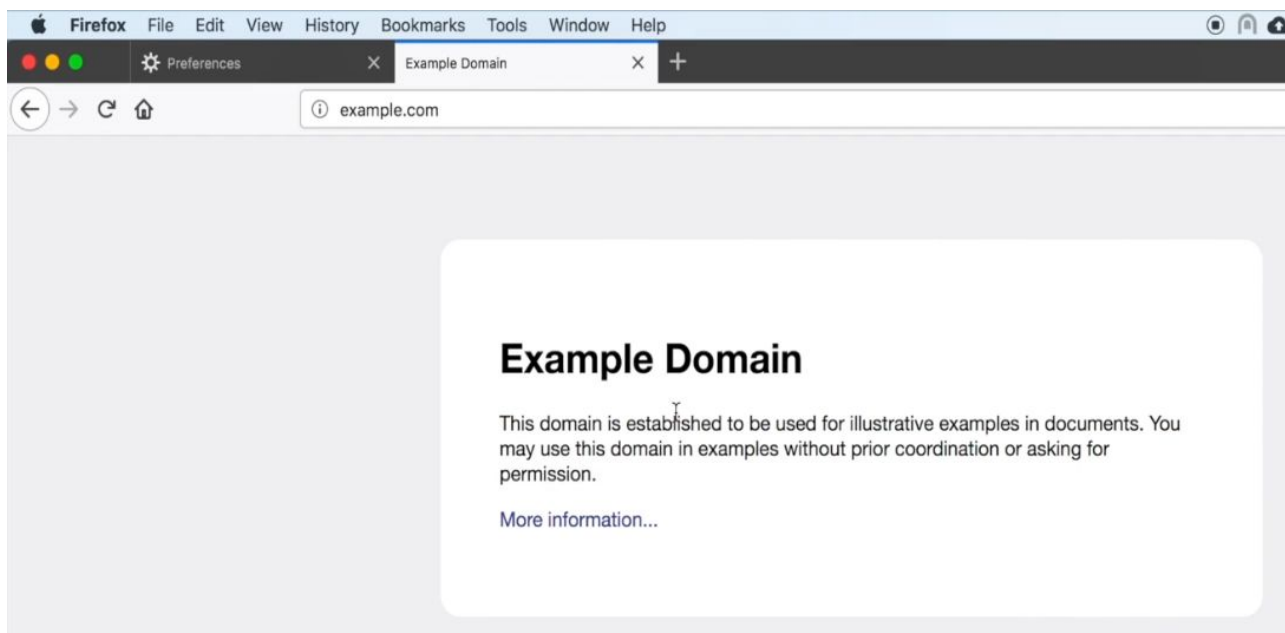
Рано или поздно встанет вопрос, как анализировать запросы и ответы браузера. Можно сделать это старыми способами, например открыть инструменты разработчиков в Chrome и посмотреть запросы-ответы там. Такой подход может применяться, когда запросов и ответов немного, или когда нам нет необходимости повторять один и тот же запрос с разными параметрами, менять его или менять ответ от сервера. То есть для простых вещей в принципе подойдут инструменты разработчиков браузера, но, если нам нужно более детальное и более глубокое исследование на уязвимость, стоит воспользоваться инструментарием Burp Suite.

Давайте откроем Firefox и настроим его так, чтобы он использовал Burp в качестве прокси. Мы должны зайти в меню Preferences (Настройки), пролистать закладку General (Основные настройки) в самый низ — здесь будет вкладка Network Proxy (Прокси). Нужно нажать Settings.. (Настройки..) :

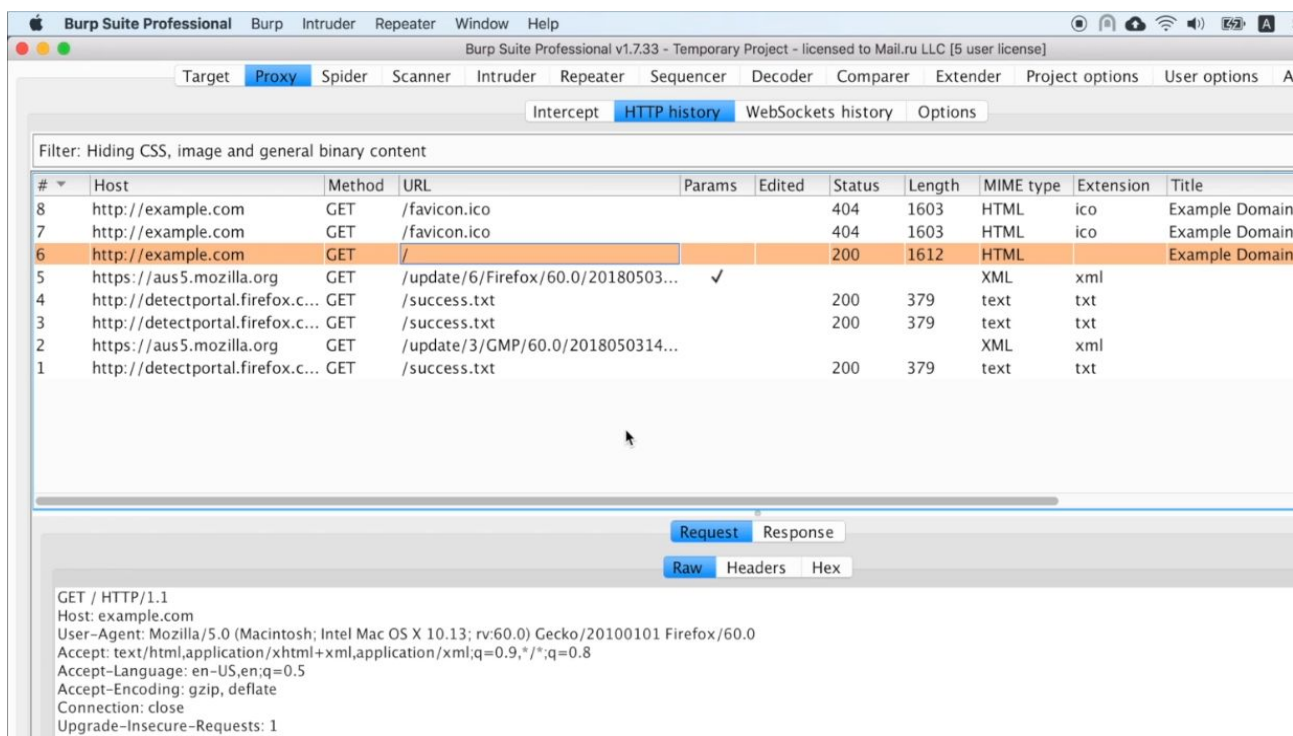


Изначально будет установлено No proxy (Без прокси), а вам будет необходимо выбрать Manual proxy configuration (Ручная конфигурация прокси) и здесь ввести адрес 127.0.0.1 и выставить порт 8080 — это стандартный порт, на котором Burp слушает сетевые подключения к нему. Подробнее о портах и адресах мы узнаем в следующих уроках.

Далее нажимаем ОК и проверяем, что у нас все работает. Для этого откроем сайт <http://example.com>:



И, как мы видим, в Вурп был сделан запрос и получен ответ:



## Proxy и HTTP history в Burp Suite

Теперь рассмотрим его возможности. Основной инструмент Вурп — прокси. В этой вкладке есть Intercept и HTTP history — то, что нас больше всего интересует. В HTTP history отображаются все запросы, которые делает браузер.

Давайте выберем какой-либо запрос, например, запрос за основной страницей сайта [example.com](http://example.com), чтобы посмотреть, что в нем находится, и проверить его ответ:

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
8	http://example.com	GET	/favicon.ico			404	1603	HTML	ico	Example Domain
7	http://example.com	GET	/favicon.ico			404	1603	HTML	ico	Example Domain
6	http://example.com	GET	/			200	1612	HTML		Example Domain
5	https://aus5.mozilla.org	GET	/update/6/Firefox/60.0/20180503...		✓			XML	xml	
4	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt	
3	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt	
2	https://aus5.mozilla.org	GET	/update/3/GMP/60.0/2018050314...					XML	xml	
1	http://detectportal.firefox.c...	GET	/success.txt			200	379	text	txt	

Request Response

Raw Headers Hex HTML Render

```

HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Sat, 12 May 2018 14:38:39 GMT
Etag: "1541025663"
Expires: Sat, 19 May 2018 14:38:39 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (lga/13A2)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1270
Connection: close

<!doctype html>
<html>
<head>

```

? < + > Type a search term

Также здесь можно посмотреть отдельно HTTP-заголовки:

Request Response

Raw Headers Hex HTML Render

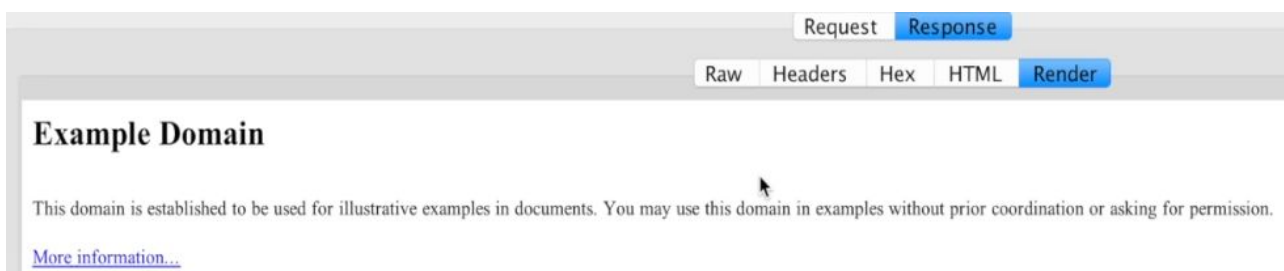
Name	Value
HTTP/1.1	200 OK
Accept-Ranges	bytes
Cache-Control	max-age=604800
Content-Type	text/html
Date	Sat, 12 May 2018 14:38:39 GMT
Etag	"1541025663"
Expires	Sat, 19 May 2018 14:38:39 GMT
Last-Modified	Fri, 09 Aug 2013 23:54:35 GMT
Server	ECS (lga/13A2)
Vary	Accept-Encoding
X-Cache	HIT
Content-Length	1270
Connection	close



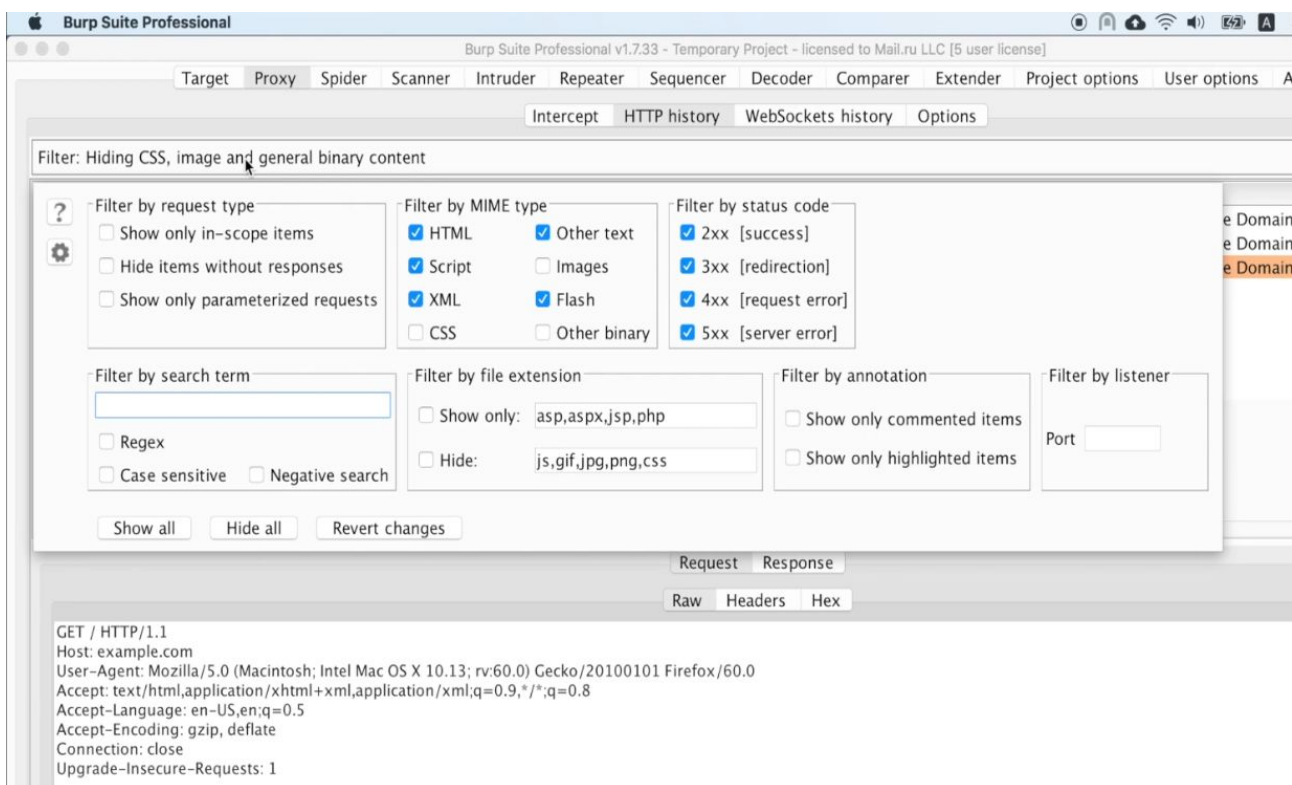
Или отдельно тело ответа:



Также можно рендерить страницу и показать ее как в браузере. При этом простые страницы в Burp будут показываться нормально, но страницы посложнее, где больше стилей и JavaScript, будут отображаться некорректно:

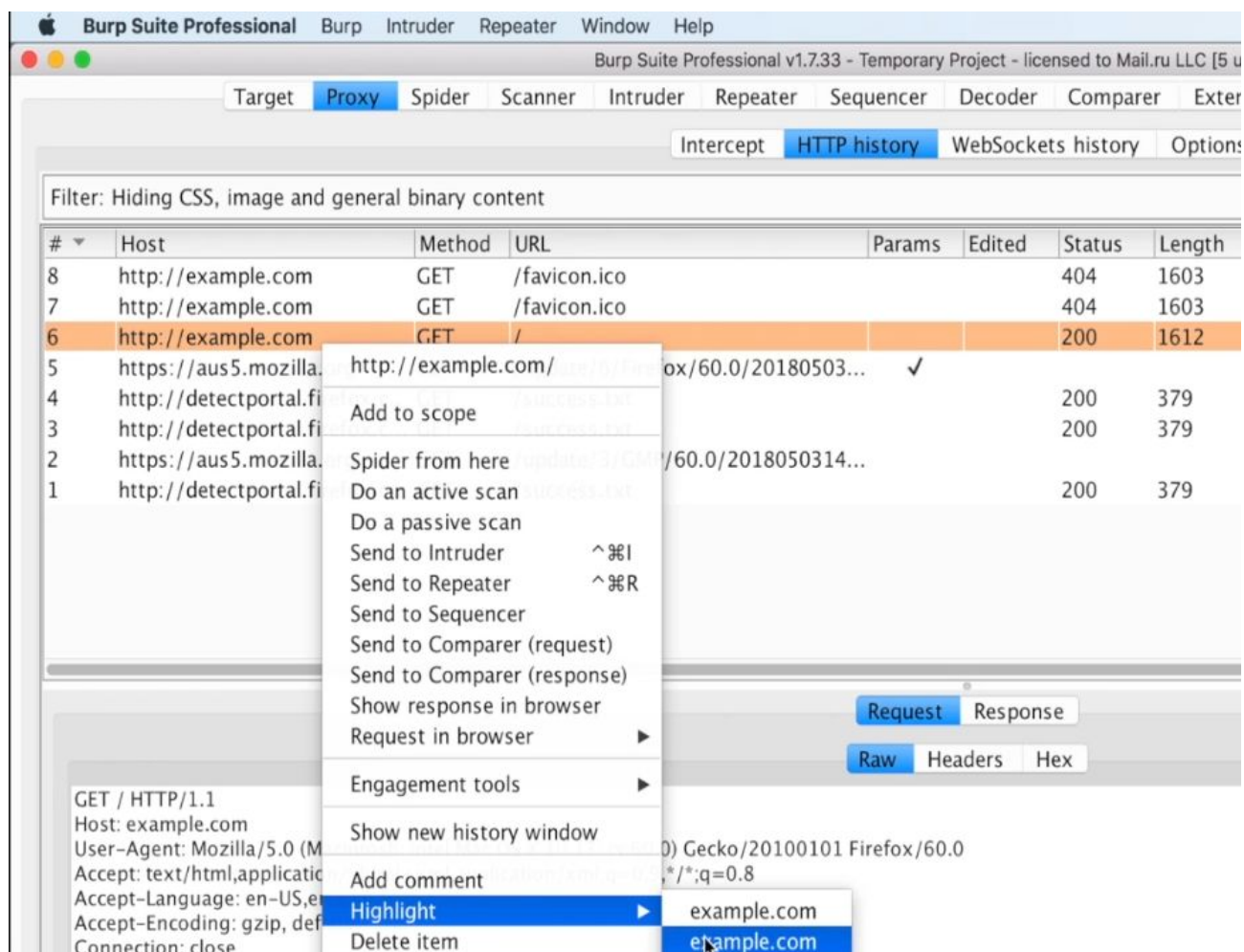


Также в этой вкладке есть фильтры, о них мы узнаем позже, когда будет необходимость показывать какие-то конкретные запросы:

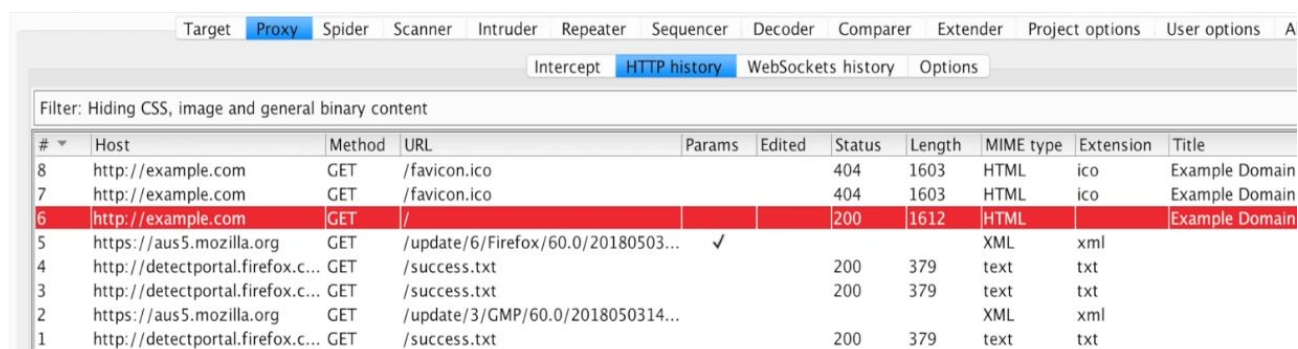




Также возможно выбрать любой запрос и нажать правую кнопку мыши:

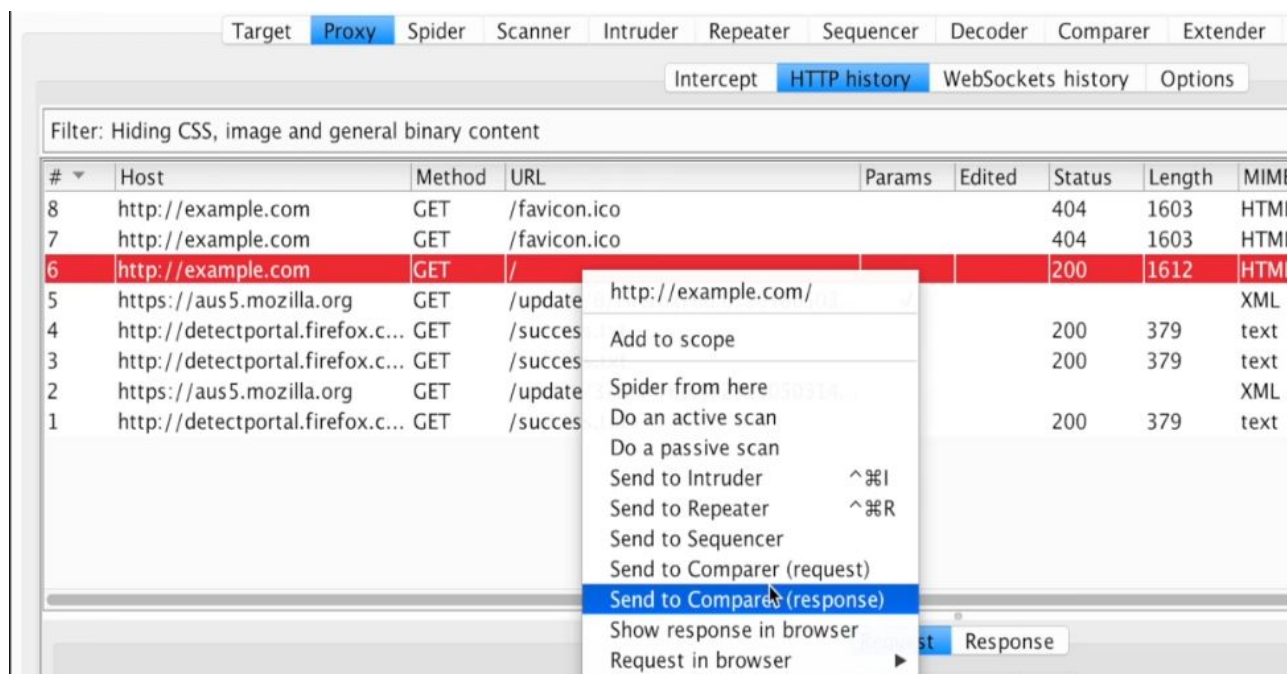


И затем подсветить его для наглядности:



Это очень полезные функции для повседневного тестирования: как правило, на реальных сайтах уходит очень много запросов, и подсвечивать и выделять нужные удобно для работы.

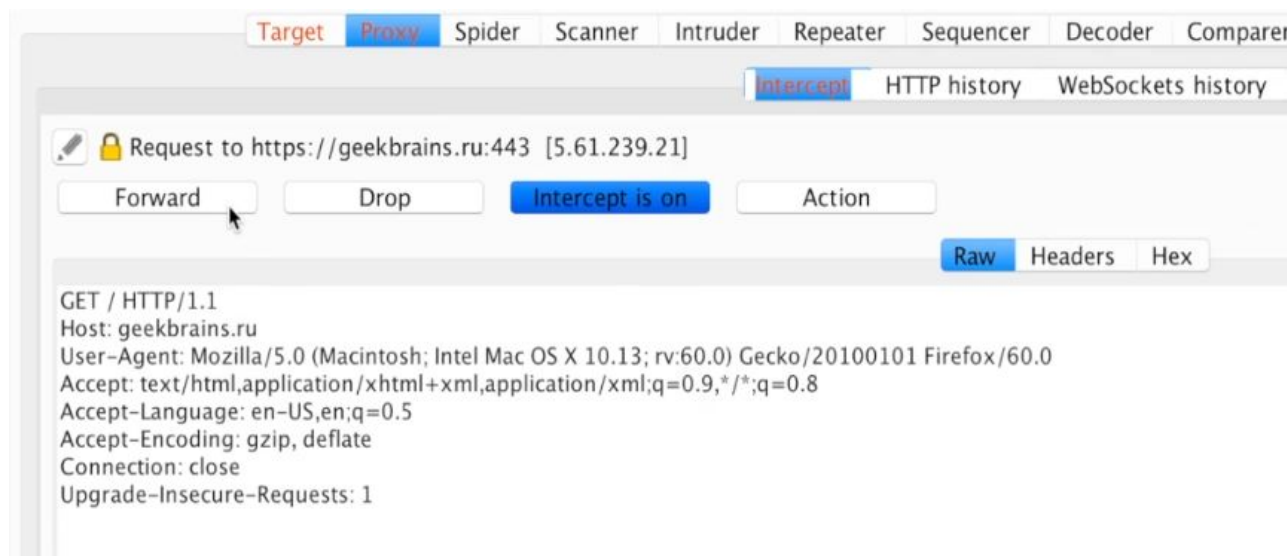
Также правой кнопкой мыши мы можем отправить этот запрос в различные инструменты: Intruder, Repeater или сравнивать запрос и ответ:



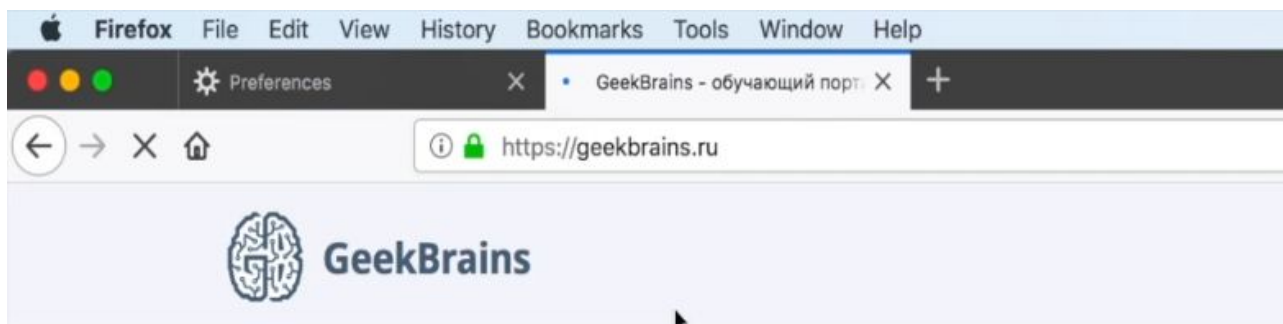
Мы рассмотрим Repeater, а Intruder и сравнение запросов — на следующих видео.

## Intercept в Burp Suite

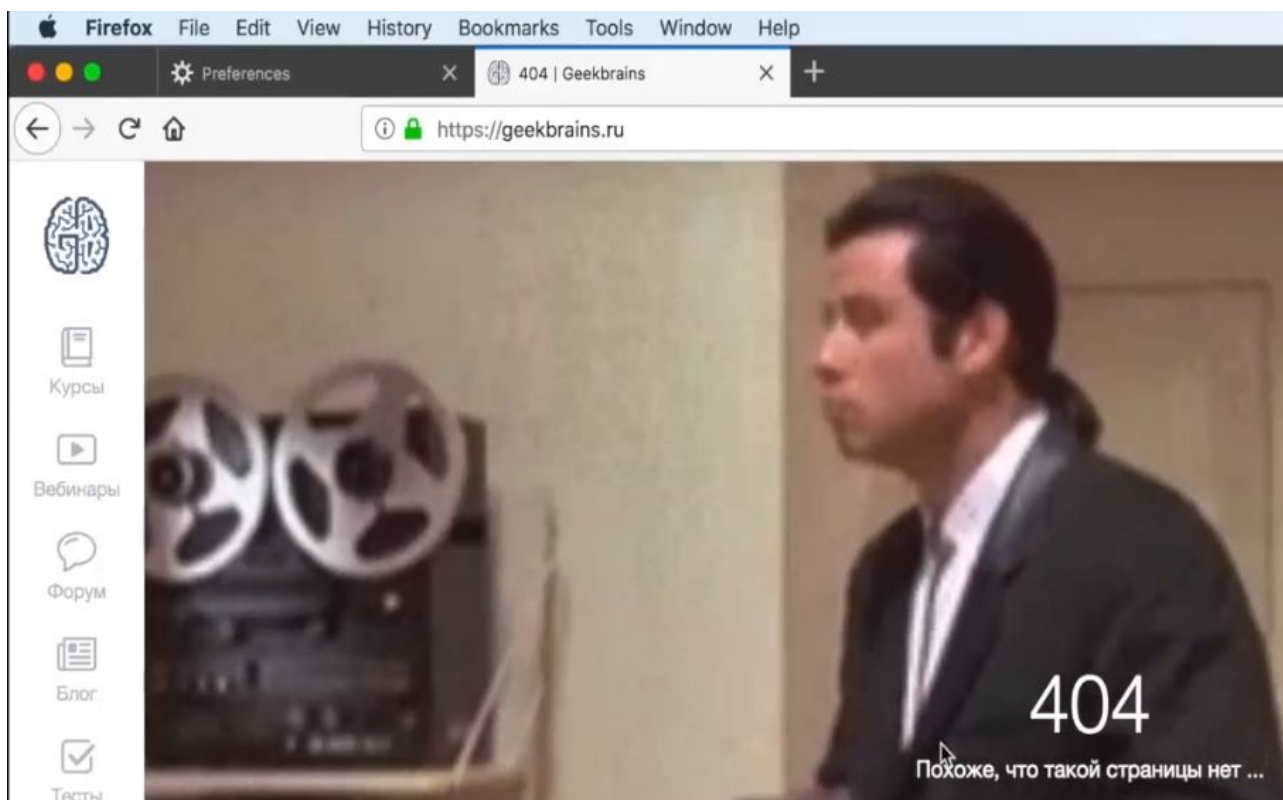
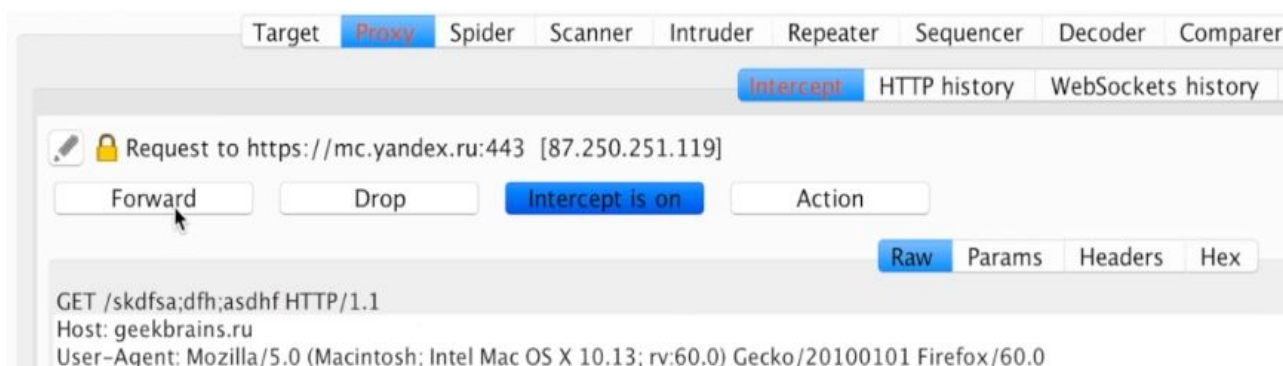
Давайте посмотрим, что же такое вкладка Intercept (Перехват). Она нужна, чтобы останавливать запросы к серверу и модифицировать их перед отправкой. Рассмотрим это на примере: включим перехват кнопкой Intercept is on, откроем Firefox и попробуем открыть страницу GeekBrains — <https://geekbrains.ru>, нажимаем Enter и видим, что ничего не происходит. Вернемся в Burp и увидим, что ничего не происходит из-за того, что все запросы, которые посылает браузер, останавливаются в Burp и ждут, пока мы их переправим кнопкой Forward либо отменим кнопкой Drop:



Отправим этот запрос и выключим на время перехват — Intercept is off. И теперь мы видим, что после этого страница успешно загрузилась:



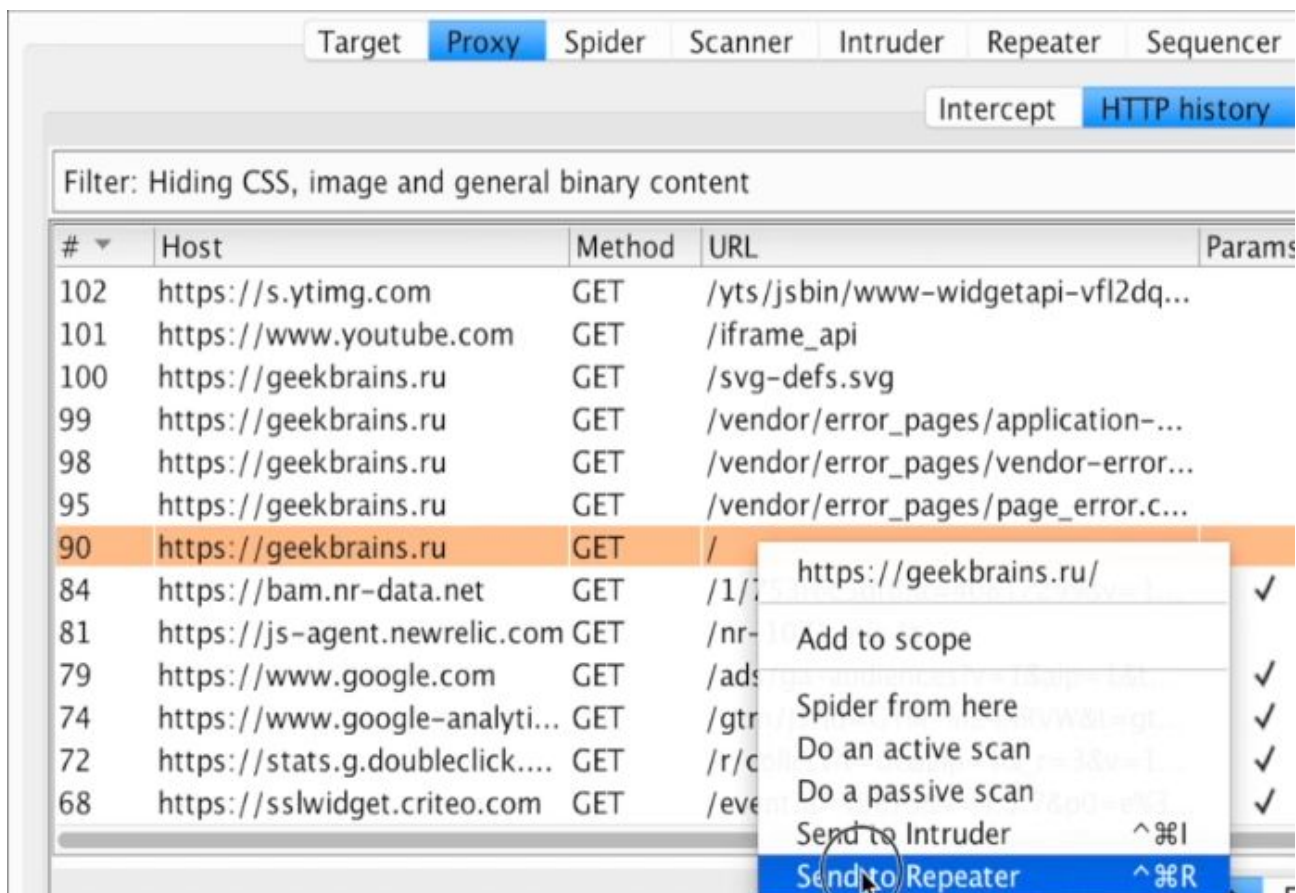
Теперь попробуем модифицировать запрос. Снова включим Intercept is on, вернемся в браузер и обновим страницу. Увидим запрос на GeekBrains, допишем сюда что-нибудь после GET / и перенаправим этот запрос:



Такой страницы нет, потому что мы ввели не валидный URL и GeekBrains не смог его найти.

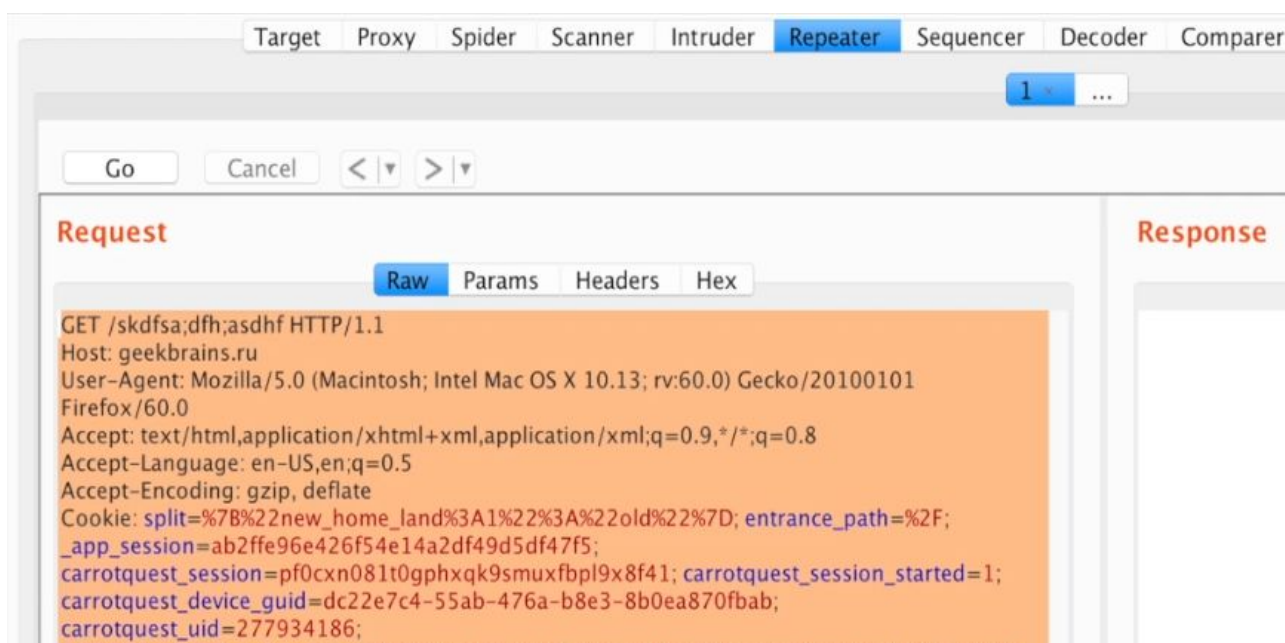
## Repeater в Burp Suite

Теперь посмотрим что дает нам Repeater. Выберем запрос GET / geekbrains.ru, нажмем правую кнопку мыши и выберем Send to Repeater:





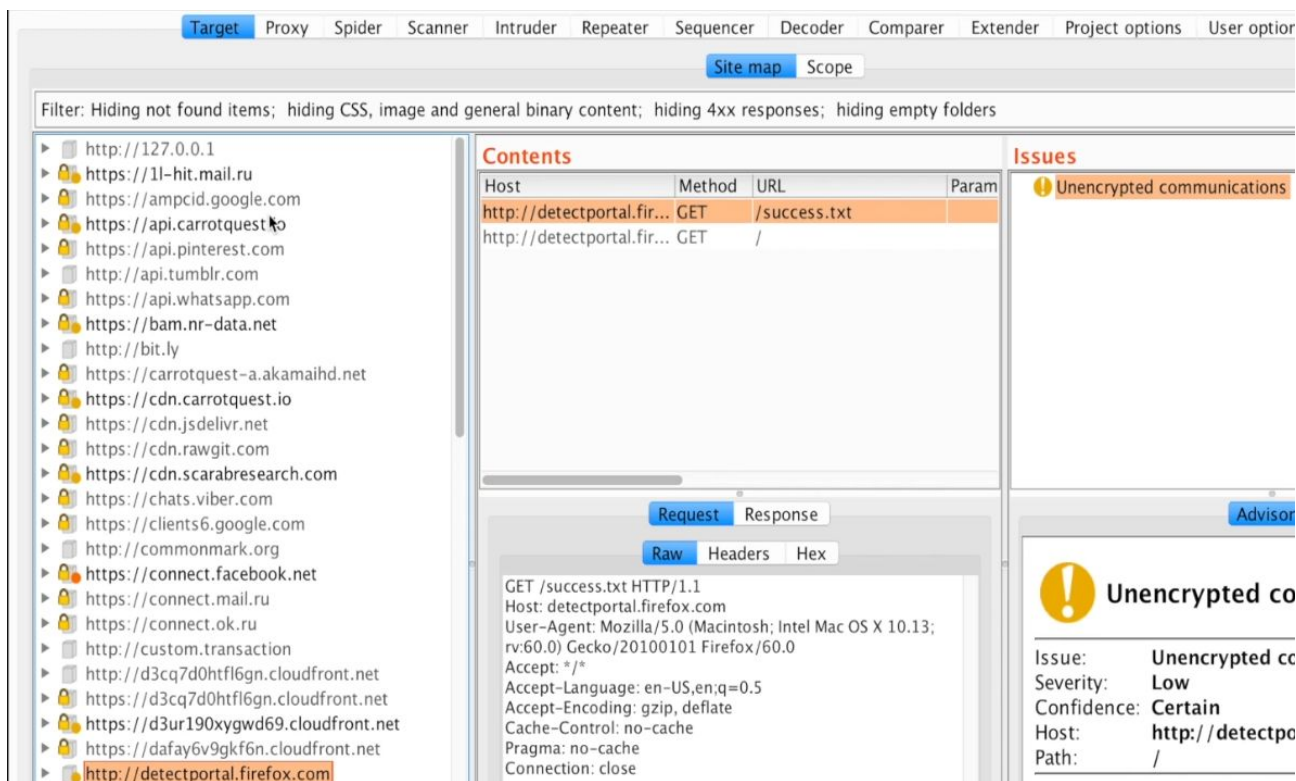
Repeater отображает запрос со всеми заголовками. Не стоит их пугаться: хоть и кажется, что здесь много текста, но мы позже разберемся с заголовками и вы будете понимать, что здесь происходит.



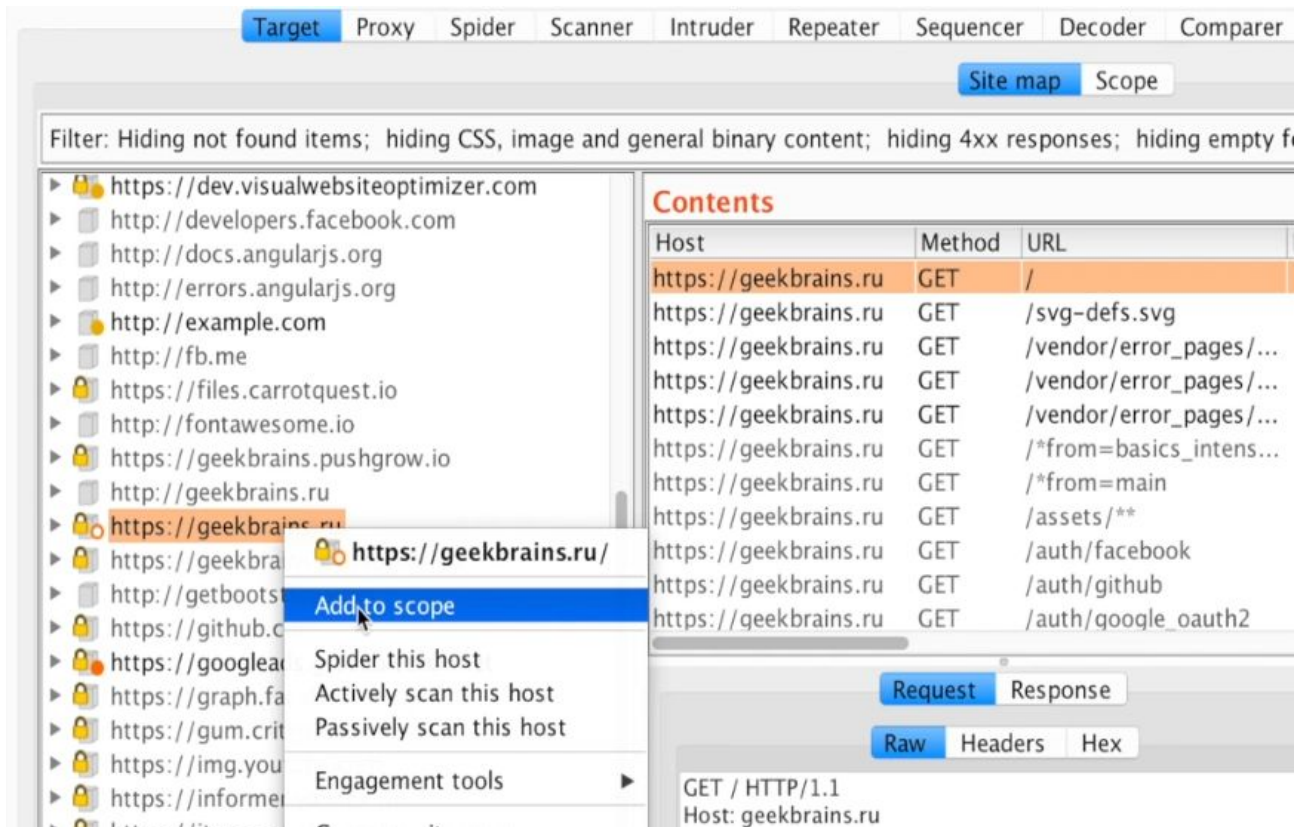
Можно изменять запрос и смотреть ответы от сервера — это очень полезный функционал, когда приходится смотреть определённый адрес и тестировать его.

## Target и Scope в Burp Suite

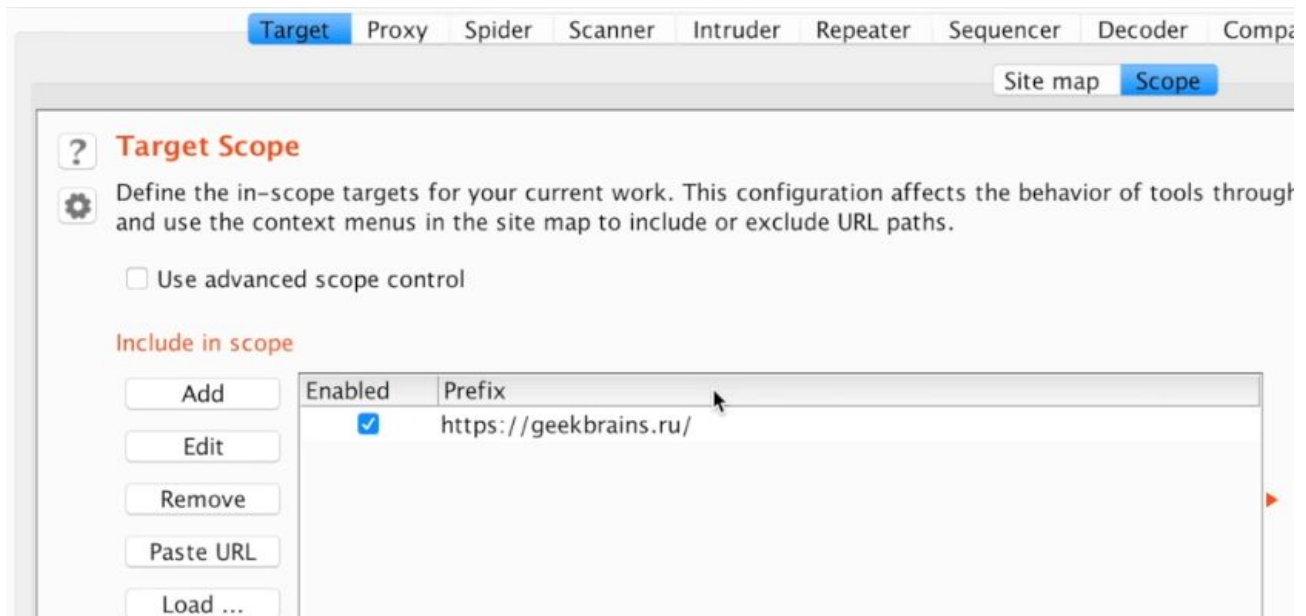
Во вкладке Target отображаются сайты, которые мы посетили, а также те, на которые Burp нашёл ссылки на странице:



Делается это с помощью механизма Spider. Подробнее об этом вы узнаете позже, а сейчас важно то, что мы можем выбрать, например, `geekbrains.ru`, нажать правой кнопкой мыши и выбрать `Add to scope`:

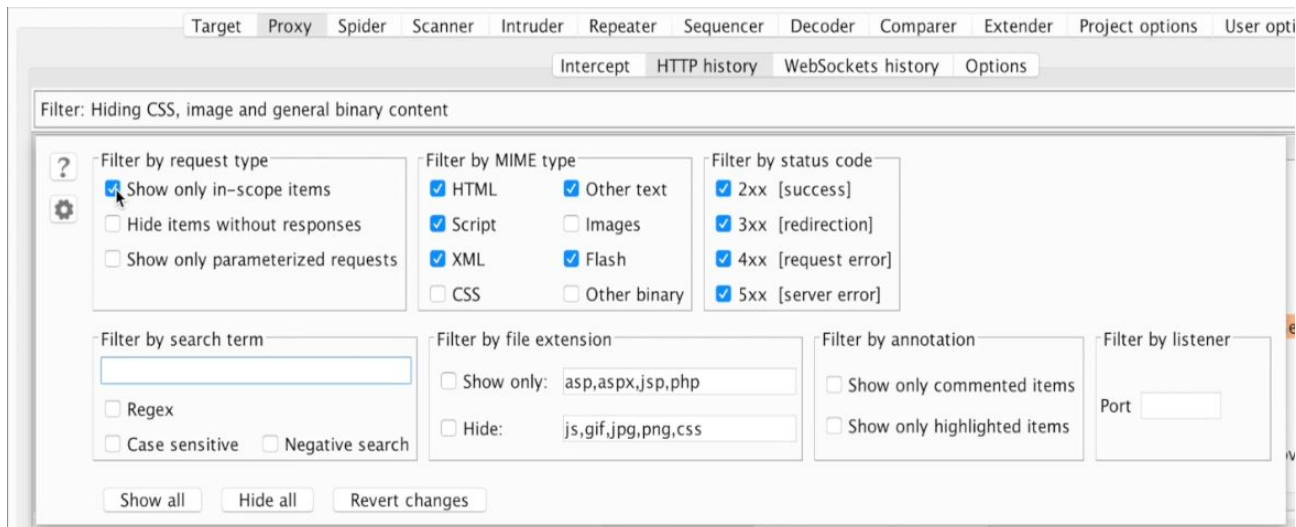


После этого мы переходим во вкладку **Scope** и видим, что добавили сайт `geekbrains.ru`:

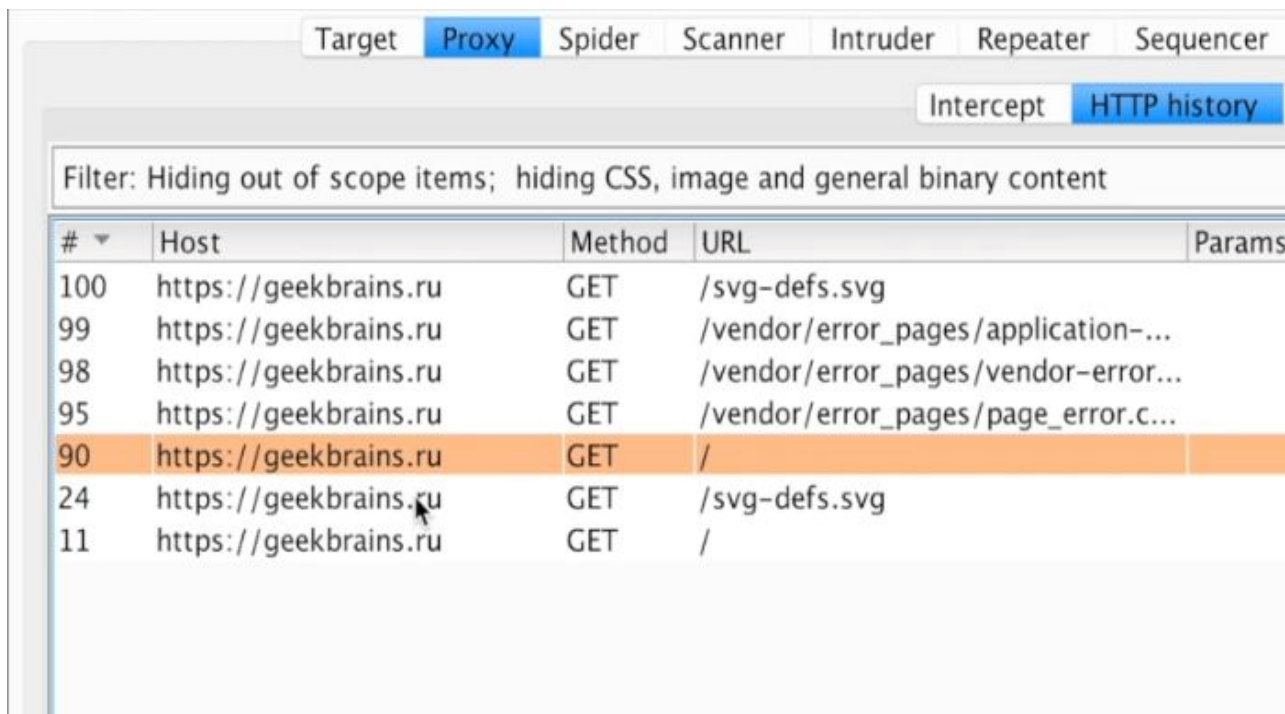


Это означает, что мы заинтересованы в этом домене и хотим его посмотреть.

Что нам это дает? Например, во вкладке Proxy -> HTTP history мы можем применить фильтр, который будет показывать только запросы-ответы, которые входят в выбранный Scope. Отметим эту галочку Show only in-scope items и посмотрим, что произойдет:



Мы увидим, что все запросы, кроме тех, которые сделаны к <https://geekbrains.ru>, были отфильтрованы и остались только эти:

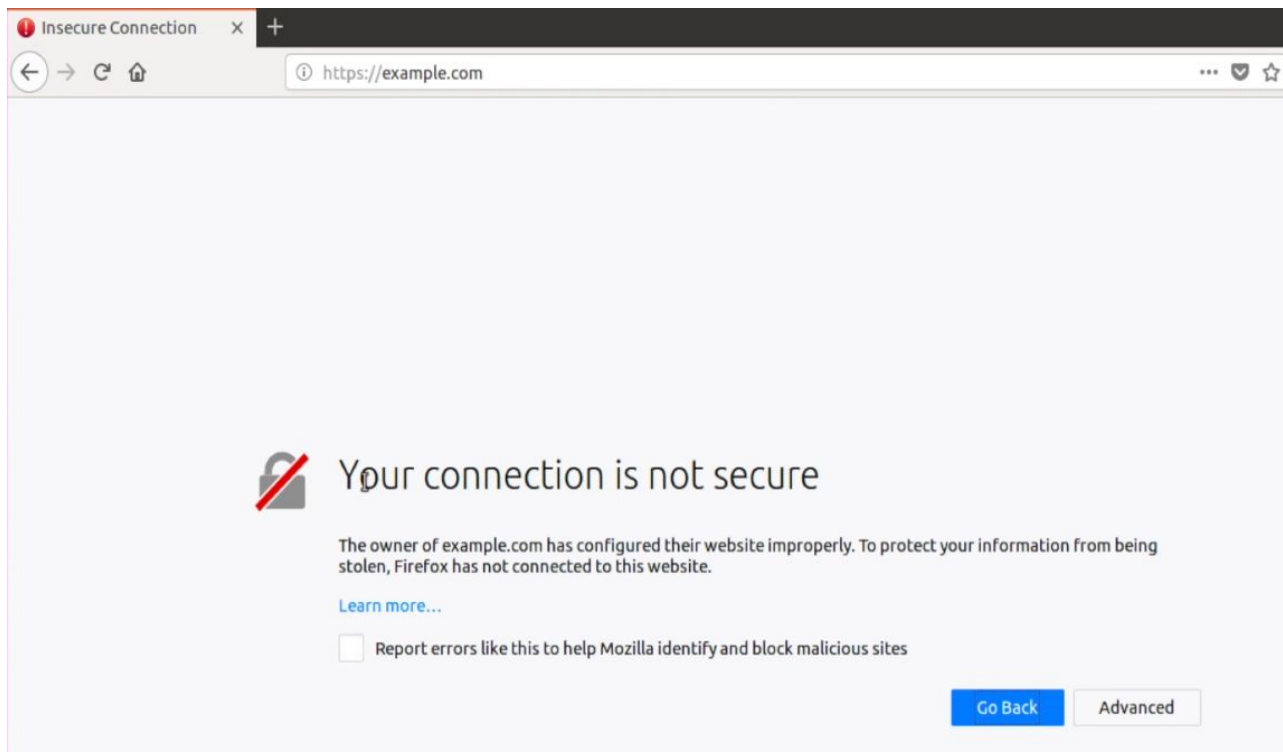


При огромном количестве запросов на реальных сайтах эта функция очень полезна. То есть, если мы грамотно настраиваем Scope, у нас не будет много рекламных запросов, запросов за картинками и так далее.

## Сайты с HTTPS-шифрованием

Также у вас могут возникнуть затруднения, когда вы будете пытаться просмотреть зашифрованный трафик. Чтобы просматривать его, вам нужно установить корневой сертификат Burp.

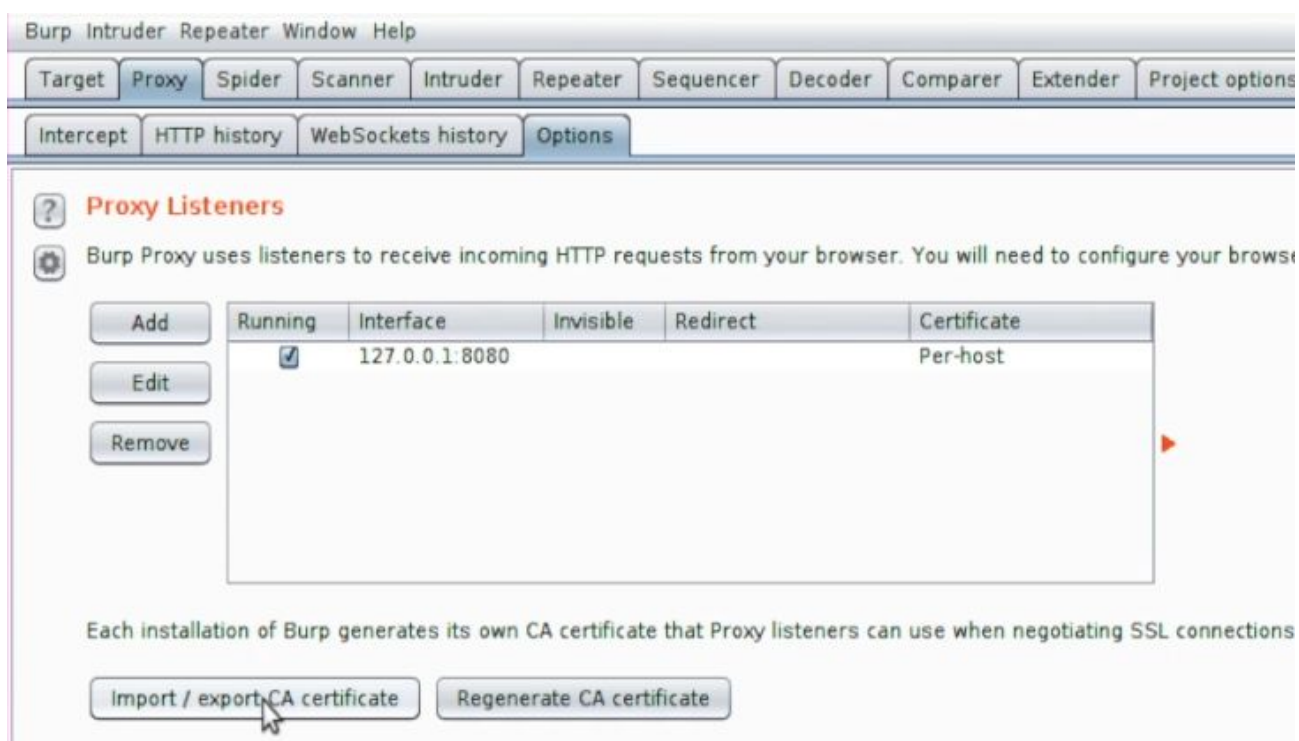
Сайт <http://example.com> открывается без проблем, но, если мы выберем шифрованное соединение https, браузер скажет, что соединение небезопасно:



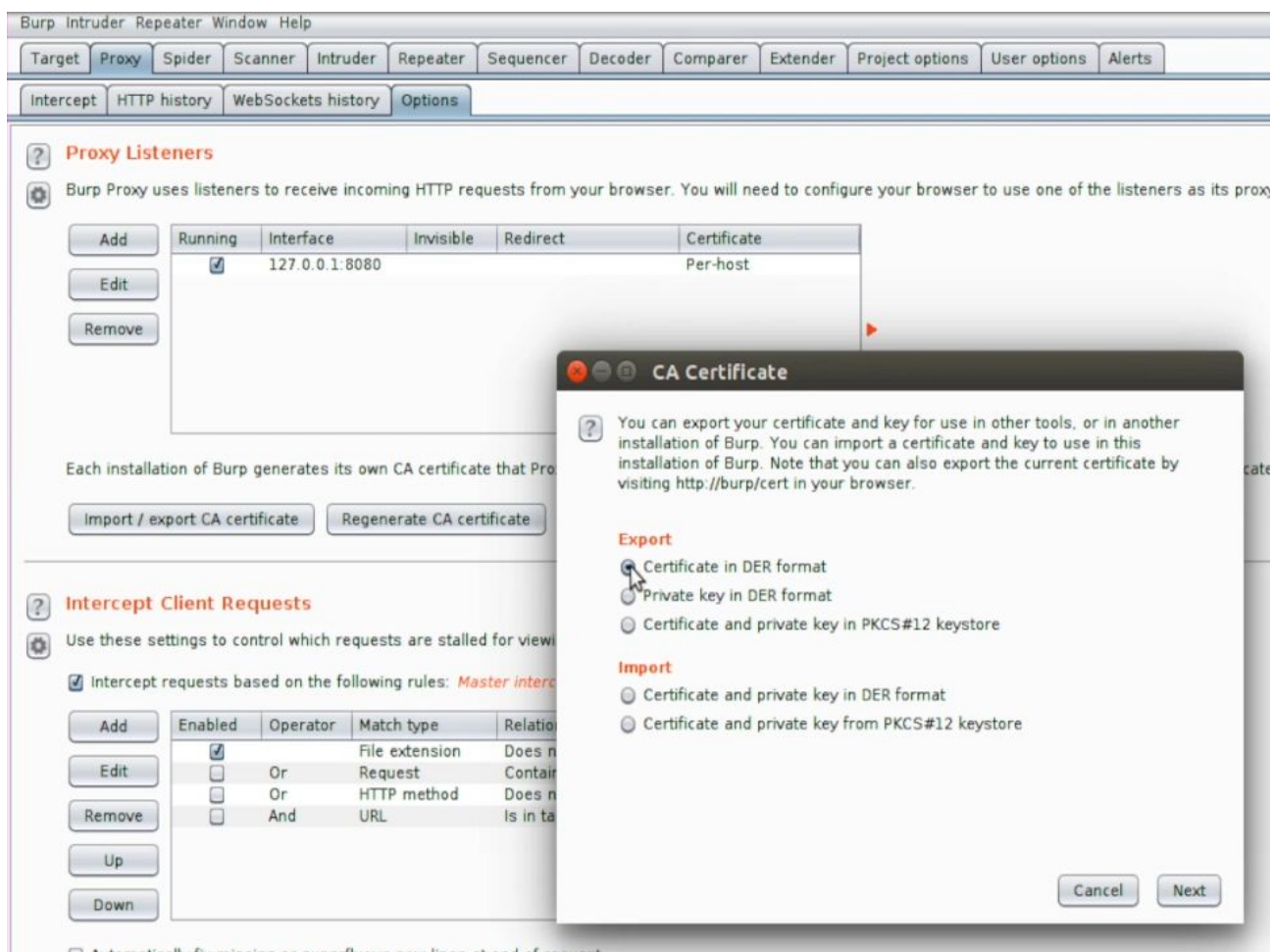
Это происходит из-за того, что мы подключаемся не к серверу example.com, а к Burp. Чтобы обойти ограничение, и пользоваться Burp как на HTTP, так и на HTTPS, добавим сертификат Burp в браузер Firefox.



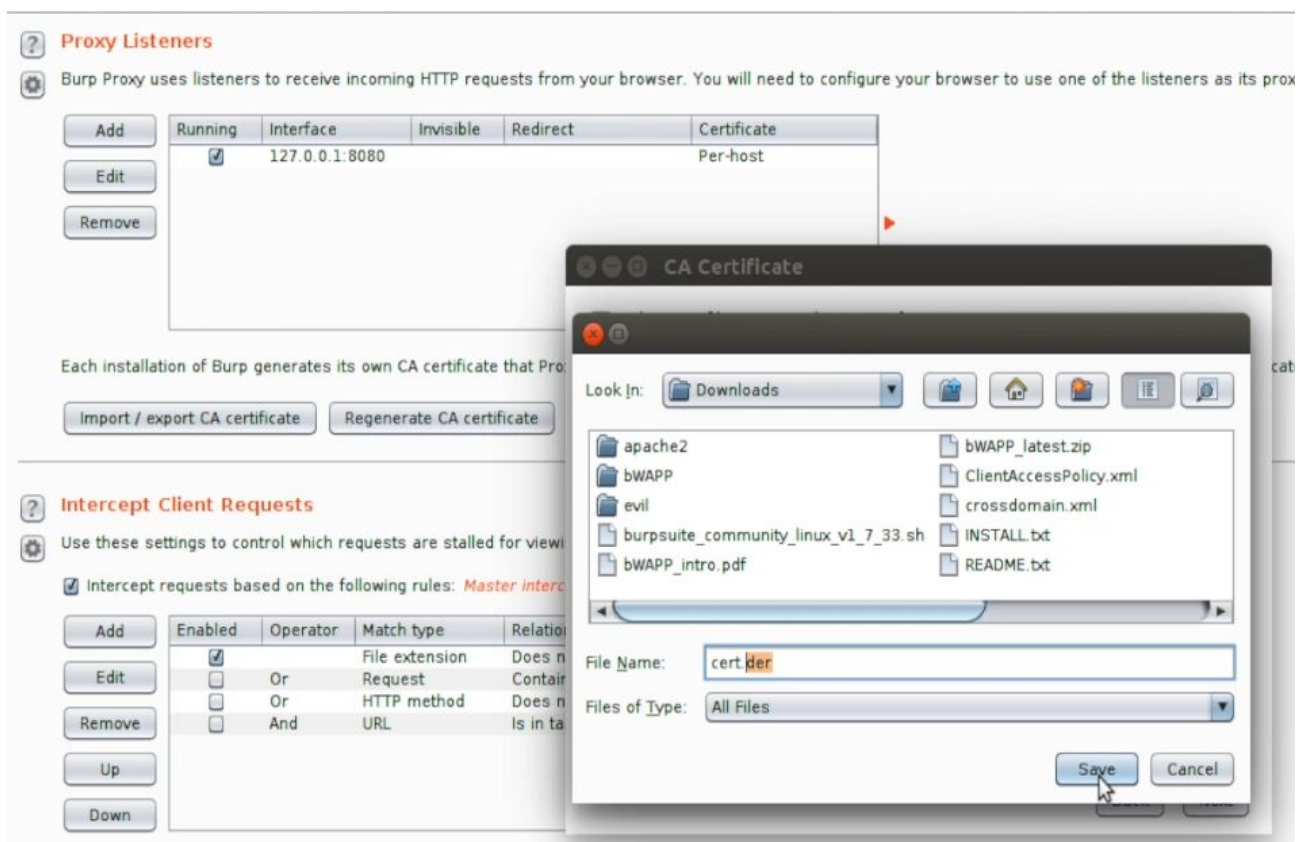
Для начала нужно экспортировать сертификат из Burp. Заходим в Burp -> Proxy -> Options (Опции) -> Import/export CA certificate:



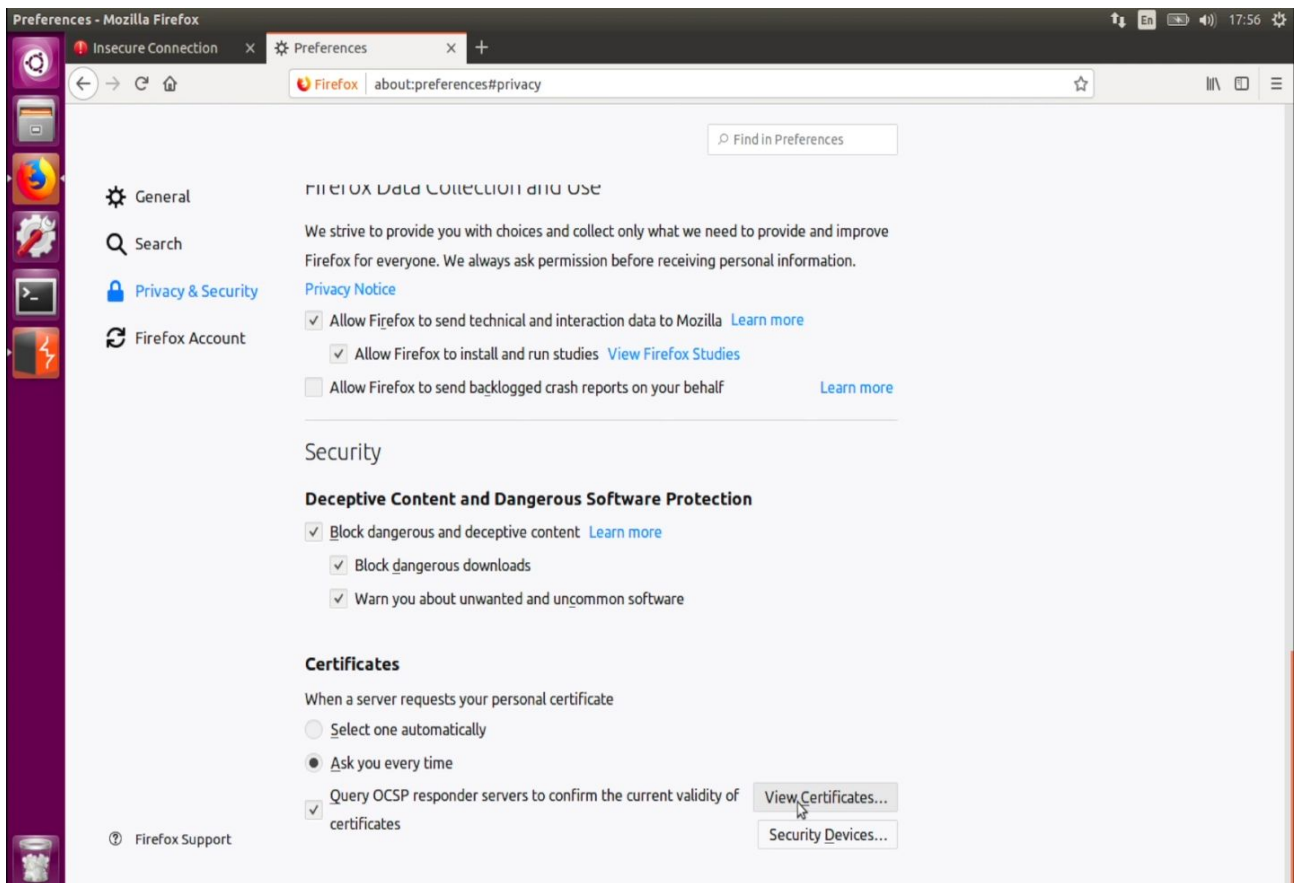
Выбираем Export -> Certificate in DER format:



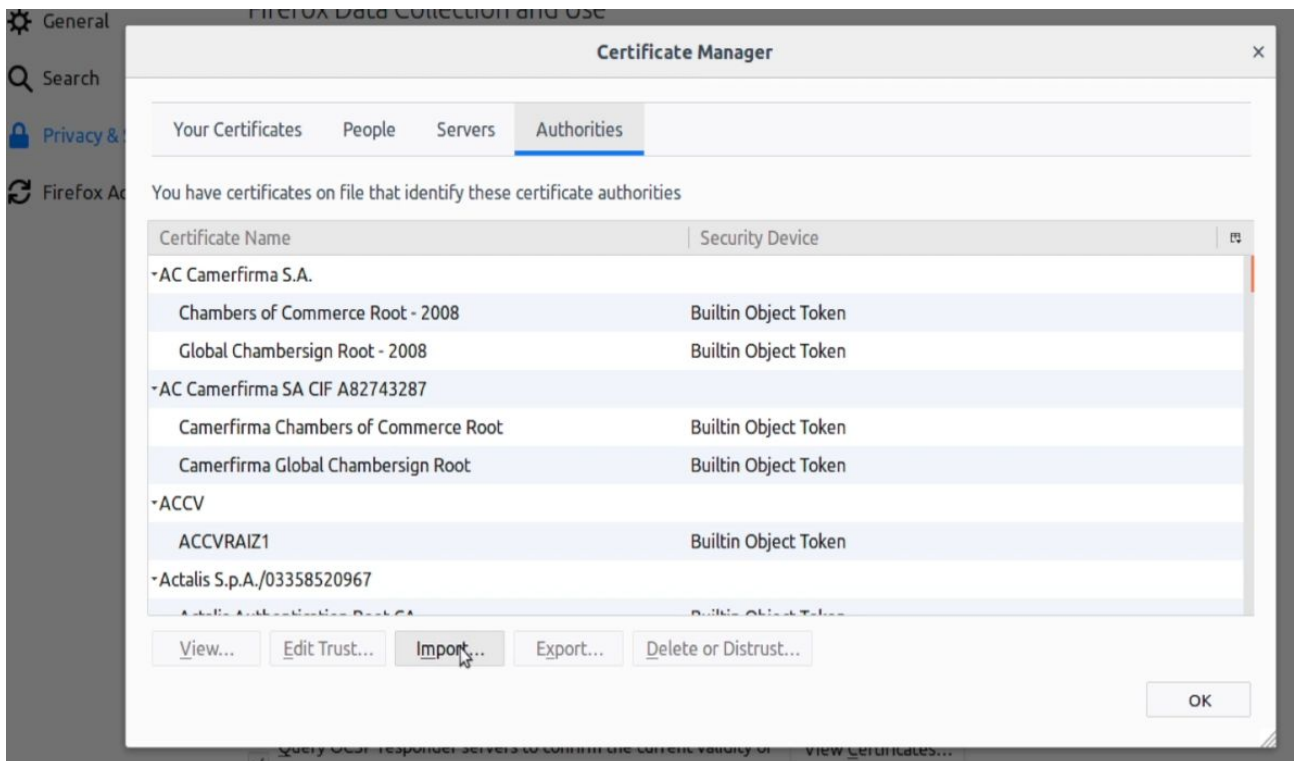
Жмем Next и выбираем, куда экспортируем. Пусть это будет файл cert.der — имя файла не особо важно, нам важен формат .DER, чтобы Firefox смог распознать его как сертификат:



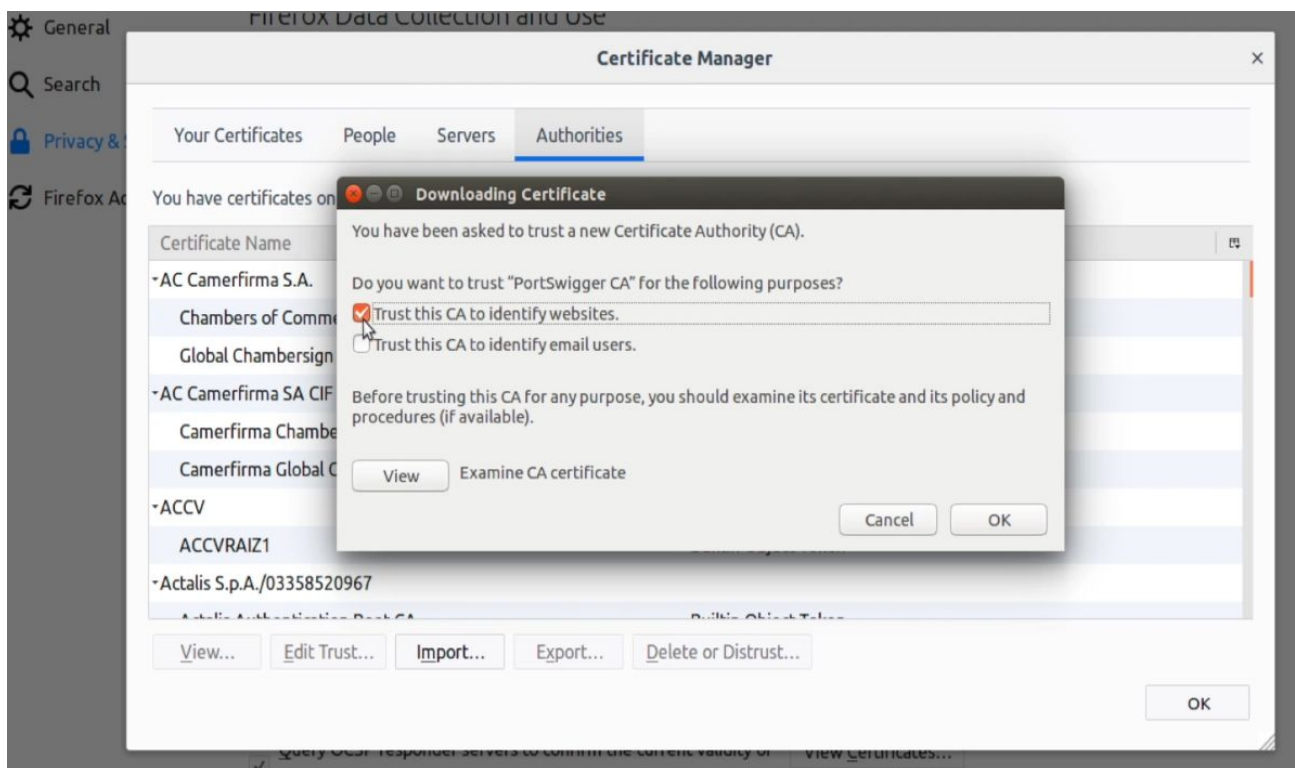
Далее нажимаем Save (Сохранить) -> Next -> Close. Заходим в Firefox и выбираем меню Preferences (Настройки) -> Privacy & Security -> View Certificates... :



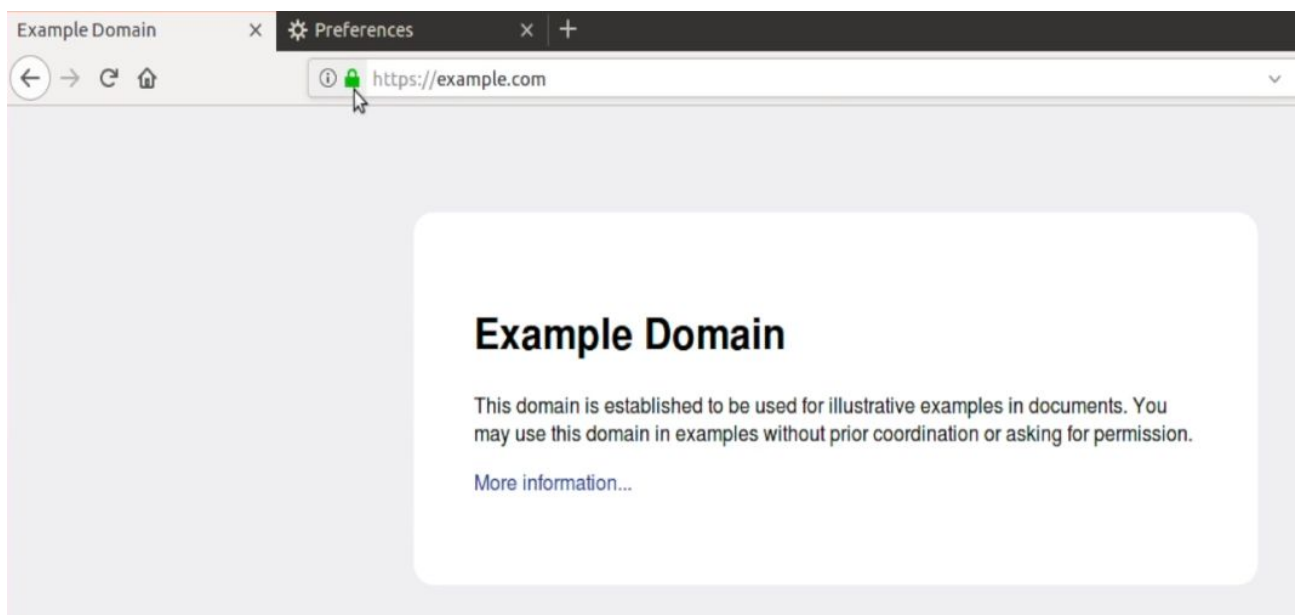
Нажимаем импорт и выбираем наш сертификат:



И ставим галочку Trust this CA to identify websites (Доверять этому центру сертификации для идентификации веб-сайтов):



Теперь попробуйте перезагрузить страницу:



Как мы видим, сертификат успешно установлен и можно посещать HTTPS-версии сайтов.

## Вопросы для самоконтроля

1. Что такое Burp Proху, какие режимы его работы вы знаете?
2. Что нужно для работы в Burp с защищенными соединениями https?

## Ссылки

1. [Что такое Burp Proxy?](#)
2. [BURP SUITE TUTORIAL — WEB APPLICATION PENETRATION TESTING \(PART 1\).](#)
3. [BURP SUITE TUTORIAL — WEB APPLICATION PENETRATION TESTING \(PART 2\).](#)

## Итоги

В этом видео мы узнали, зачем нужен Burp Suite, как его настроить для работы с HTTPS. Подробно рассмотрели, как в Burp Suite использовать функции History, Scope, Intercept и Repeater.