

Курс «Веб-технологии: уязвимости и безопасность»

# URL

Структура URL, схема доступа к ресурсу, доменные имена DNS, порт, путь, запрос и якорь

## Оглавление

[Введение](#)

[Видеоурок 1](#)

[URL](#)

[Виды URL](#)

[Практика](#)

[Вопросы для самоконтроля](#)

[Итоги](#)

[Видеоурок 2](#)

[Виды схем в URL](#)

[Схема file://](#)

[Схема javascript:](#)

[Схема mailto:](#)

[Схема data:](#)

[Авторизация в Nginx](#)

[Задание для самоконтроля](#)

[Итоги](#)

[Видеоурок 3](#)

[Доменные имена](#)

[IP-адреса](#)

[Система DNS](#)

[Полезные утилиты](#)

[Команда host](#)

[Команда whois](#)

[Файл hosts](#)

[Punycode](#)

[Итоги](#)

#### [Видеоурок 4](#)

[Сетевые порты](#)

[Путь к ресурсу](#)

[URL Encode](#)

[Таблица ASCII](#)

[Итоги](#)

#### [Видеоурок 5](#)

[Запрос в URL](#)

[Якорь](#)

[Практика](#)

[URL и уязвимости](#)

[Итоги](#)

[Ссылки к уроку](#)

## Введение

Добро пожаловать на второй урок курса «Веб-технологии: уязвимости и безопасность».

На первом уроке мы узнали основные понятия и концепции веба. Теперь настало время перейти к более детальному освоению каждого из компонентов. Начнем мы с URL.

1. В первом видео мы узнаем, что такое URL, зачем он нужен, разберем его общую структуру, научимся различать абсолютный и относительный URL.
2. Во втором видео мы поговорим про такие составные части URL, как схема, логин и пароль для доступа к ресурсу. Также мы рассмотрим, зачем нужны схемы сторонних приложений и некоторые примеры из них.
3. Третий урок будет посвящен доменным именам и системе DNS. Также мы поговорим о Punycode и связанных с ним уязвимостях.
4. Четвертое видео посвящено таким составным частям URL, как порт и путь.

5. Пятое видео завершит описание структуры URL, мы узнаем, что такое запрос и якорь.

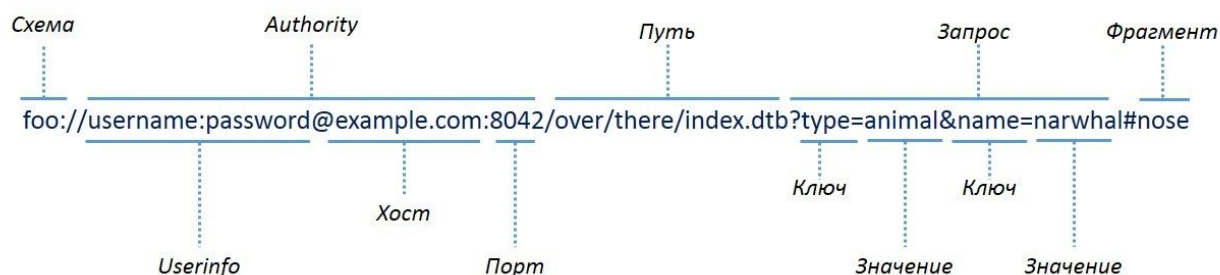
К концу урока мы будем уметь составлять URL, а также разделяют его на отдельные части и понимать зачем каждая из частей нужна. Также мы будем знать что такое DNS, как он работает, научимся изменять его настройки и освоим несколько новых программ в командной строке.

## Видеоурок 1

К понятию URL можно прийти, если задаться вопросом, как серверу однозначно понять, какой конкретный ресурс запрашивает клиент. Чтобы решить эту проблему, и придумали URL. Он позволяет однозначно идентифицировать ресурс на каком-либо сервере, а также передать параметры для сервера или клиентов.

### URL

На слайде представлена общая схема устройства URL:



URL состоит из следующих частей: схема, логин и пароль для доступа к ресурсу, адрес сервера, порт, путь до ресурса на сервере, строка запроса и фрагмент, он же якорь. Соответственно, обязательными являются схема, адрес и путь до ресурса. Остальные параметры URL опциональны.

### Виды URL

Давайте разберемся, чем отличается абсолютный и относительный URL:

scheme://	login:password@	address:port	/path/to/resource	?query_string	#fragment
-----------	-----------------	--------------	-------------------	---------------	-----------

/path/to/resource	?query_string	#fragment
-------------------	---------------	-----------

Абсолютный URL начинается с имени хоста и чаще всего со схемы. Относительный же URL не содержит имени сервера, то есть адрес у него отсутствует.

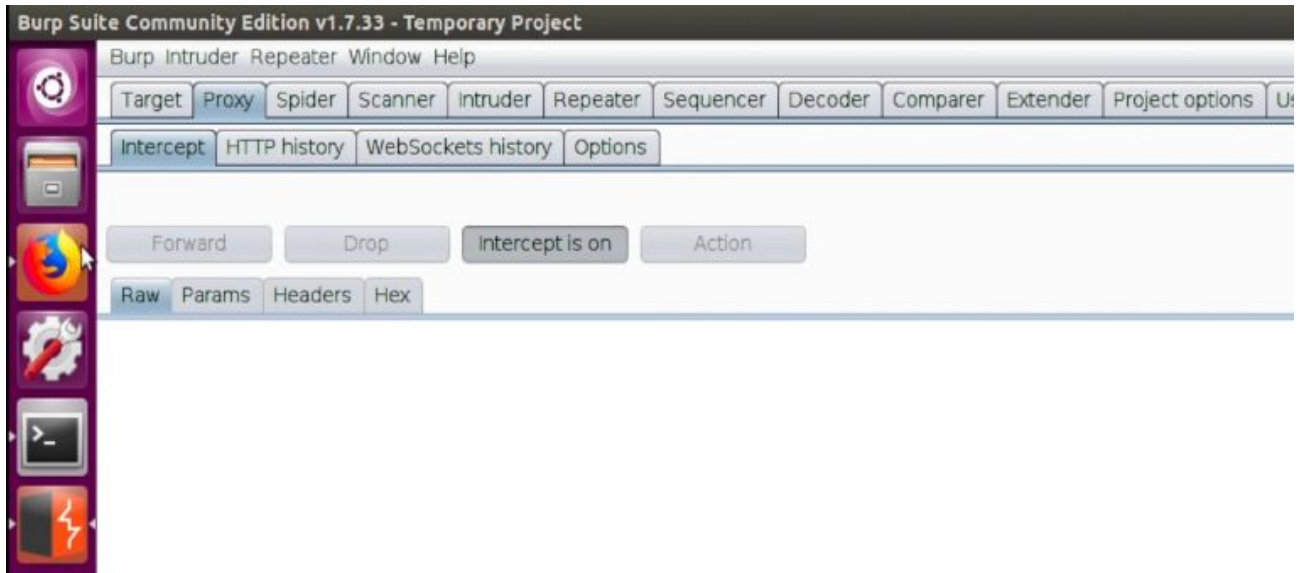
Давайте рассмотрим конкретные примеры абсолютных и относительных узлов:

- `http://geekbrains.ru` -- абсолютный
  - `//mail.ru` -- абсолютный (schemeless)
  - `/admin/index.html` -- относительный (в пределах домена)
  - `user/1` -- относительный (в пределах документа)
  - `?name=vasya` -- относительный (в пределах документа)
  - `#chapter=3` -- относительный (в пределах документа)
- 
- <http://geekbrains.ru> — абсолютный URL, так как у него есть схема и доменное имя.
  - <//mail.ru> — тоже абсолютный URL, но так называемым schemeless, то есть без схемы. Это означает, что такой URL, если его открыть, будет наследовать схему из родительской страницы. Например, если где-то на странице <http://geekbrains.ru> встретится ссылка <//mail.ru>, она преобразуется в <http://mail.ru>.
  - URL </admins/index.html> относительный в пределах домена. Это означает, что, если где-то на странице <http://geekbrains.ru> встретится ссылка </admins/index.html>, в результате будет составлен абсолютный полный URL — <http://geekbrains.ru/admins/index.html>.
  - URL <user/1> — относительный в пределах документа. Это означает, что этот путь будет прикреплен к остальному в URL.
  - Относительные URL с параметром и якорем тоже являются относительными в пределах документа, а значит, будут прикреплены после основного пути документа.

Давайте посмотрим, как ведут себя относительные URL на примере.

## Практика

Откроем виртуальную машину с Ubuntu, запустим в ней Burp и включим Intercept:



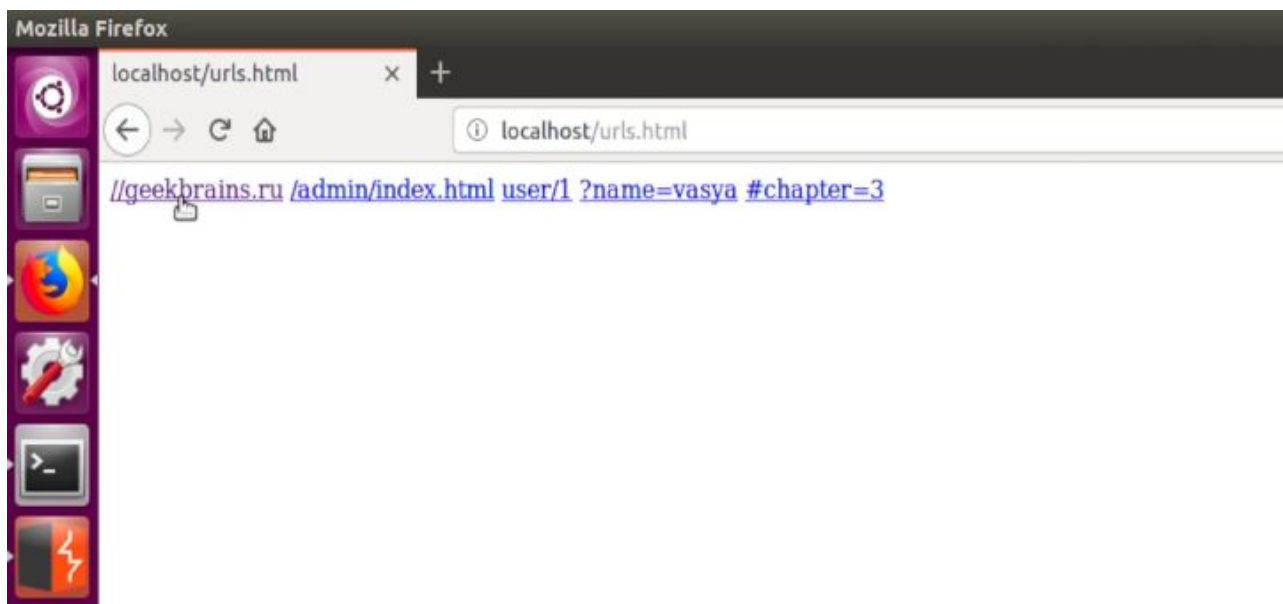
Создадим страничку urls.html, которую будет отдавать сервер:

```
r@r:/var/www/html$ cd /var/www/html && nano urls.html

<a href="//geekbrains.ru">//geekbrains.ru</a>
<a href="/admin/index.html">/admin/index.html</a>
<a href="user/1">user/1</a>
<a href="?name=vasya">?name=vasya</a>
<a href="#chapter=3">#chapter=3</a>
```

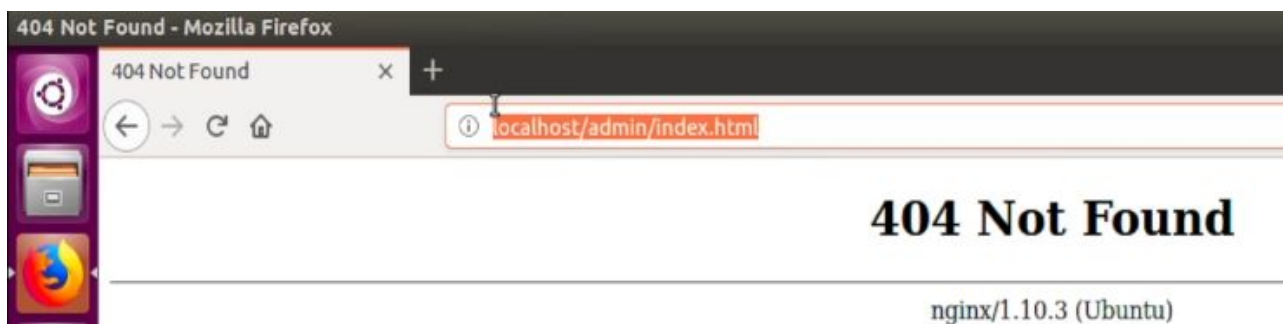
Это содержимое странички urls.html. Вы можете создать ее и повторить все действия на своей машине.

Откроем Firefox и перейдем в созданный файл:



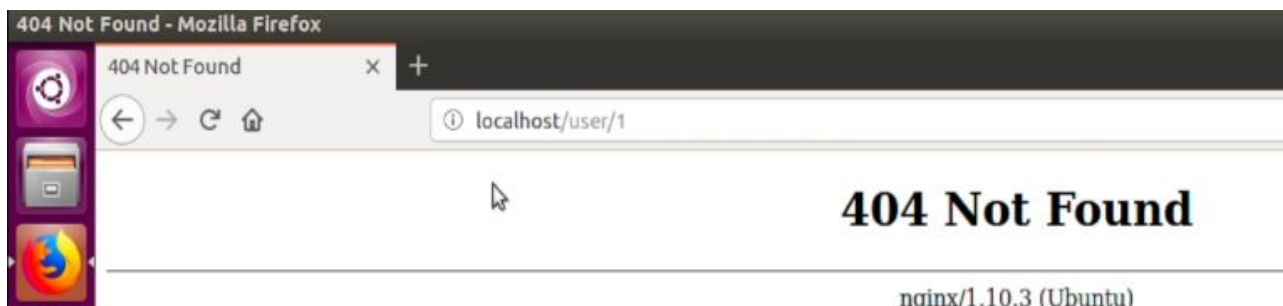
Попробуем открыть [//geekbrains.ru](http://geekbrains.ru). Мы увидим, что запрос пошел по <http://geekbrains.ru>, то есть схема унаследовалась от <http://localhost>. Сработало, как мы и ожидали.

Вернемся на страницу и посмотрим, куда пойдет </admin/index.html> — он должен попасть на <http://localhost/admin/index.html>:



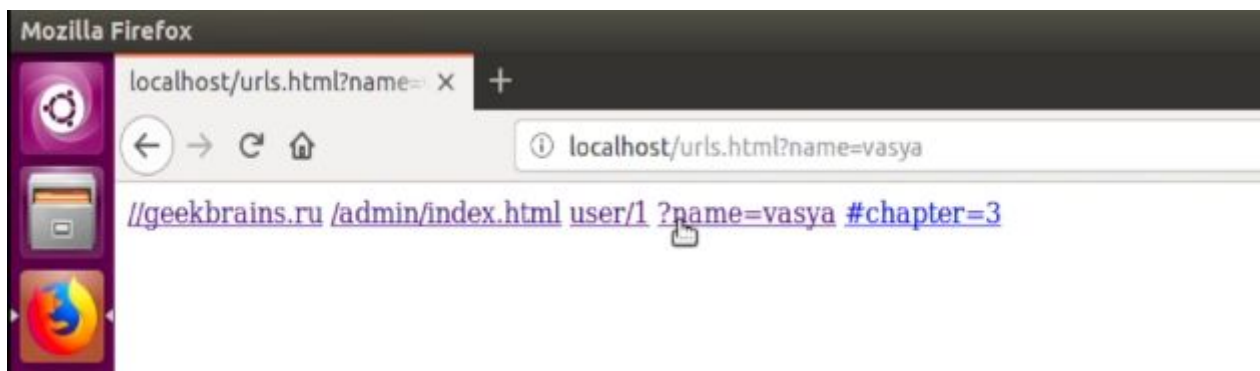
Так и есть, мы пошли именно по этому пути.

Попробуем оставшиеся. Как вы думаете, куда попадет user/1? Давайте нажмем и проверим:

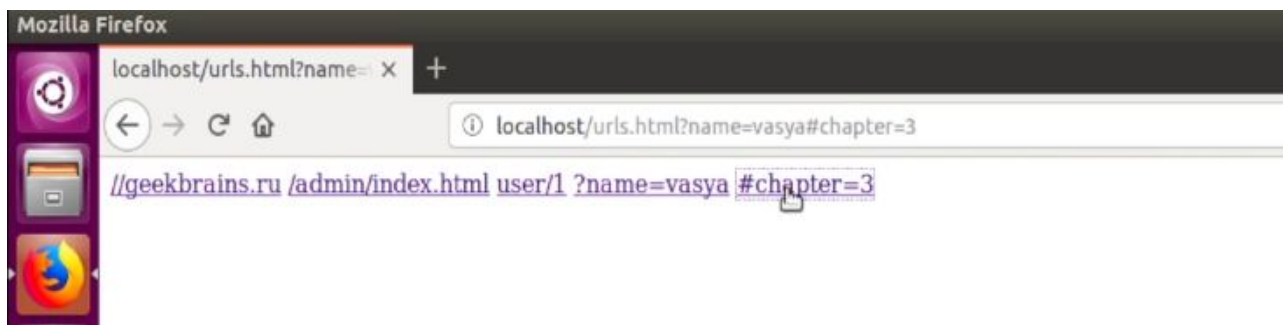


Как и ожидалось, он попал в <http://localhost/user/1>.

Ну и [?name=vasya](#) добавляет параметр к URL:



И [#chapter=3](#) добавляет якорь к URL:



## Вопросы для самоконтроля

А теперь небольшое упражнение: попробуйте угадать, в какие URL превратятся вот эти относительные URL, с учетом того, что они будут открыты на <https://geekbrains.ru>?

- `http://example.com` → ?
- `//mail.ru` → ?
- `/admin/index.html` → ?
- `?name=vasya` → ?
- `#chapter=3` → ?

## Итоги

В этом уроке вы узнали:

1. Что такое URL, для чего он применяется.
2. Из каких частей состоит URL.
3. Чем отличаются абсолютный и относительный URL.

# Видеоурок 2

В этом уроке мы разберем:

1. Что такое схема, какие схемы бывают, для чего они нужны, посмотрим несколько примеров схем.
2. Логин и пароль для доступа к ресурсу, как он реализуется в сервере Nginx.

## Виды схем в URL

Когда клиент делает запрос к серверу, сервер должен понять, по какому протоколу, то есть, на каком языке говорить с клиентом. Для этого и придумали схему. В схеме указывается протокол — http или https, file, javascript, data:

### Схема

1. http:
2. https:
3. file:
4. javascript:
5. data:
6. ...

Этот протокол говорит, как нужно общаться серверу и клиенту. Или, если это специальная схема, — как, например, file, javascript, data — у них есть специальное назначение.

Как работают схемы http и https, мы уже видели и пробовали в предыдущих уроках. Теперь научимся работать со схемой file.

### Схема file://

Схема file необходима, чтобы можно было открыть файл в браузере с локального компьютера. Рассмотрим на примере.



Запустим виртуальную машину Ubuntu, попробуем открыть файл urls.html из предыдущего видео с помощью схемы file и посмотрим что из этого получится:

```
r@r-VirtualBox: /var/www/html
r@r-VirtualBox:/var/www/html$ ls
bWAPP  hellousre.php  index.nginx-debian.html  test.txt  topsecret  urls.html
r@r-VirtualBox:/var/www/html$
```

Когда мы используем схему file, запрос на сервер не посылается, а браузер напрямую открывает файл с нашего компьютера. Когда мы пишем file, двоеточие — это схема, два слеша указывают, что это иерархический URL, еще один слеш говорит, что это корень файловой системы:



Как видим, открылся тот же самый файл что и в первый раз, но теперь внизу экрана каждый относительный URL наследуется уже от абсолютного, который мы открыли:



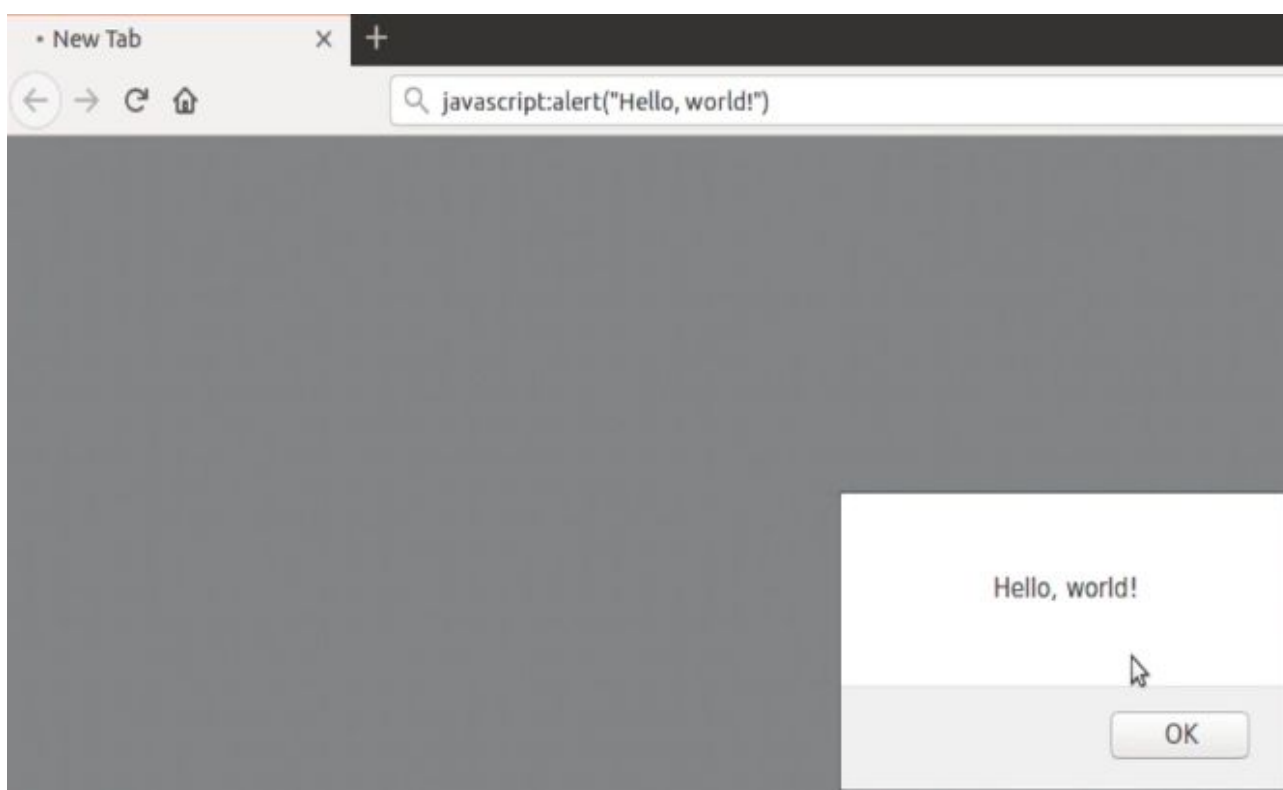
Если нажать на </admin/index.htm> и открыть <file:///admin/index.htm>, браузер выдаст ошибку File not found, так как его нет на компьютере.

Тему file:// можно применять, чтобы открывать страницы, которые мы сохранили через браузер или получили с помощью программы curl или скриптов языка программирования.

## Схема javascript:

Про язык программирования JavaScript мы поговорим подробнее в одном из следующих уроков, а сейчас нам нужно понять что схема javascript:// исполняет javascript в браузере. Теперь посмотрим, как это работает.

Напишем в строку запрос в браузер [javascript:alert\("Hello, world!"\)](javascript:alert('Hello, world!')):

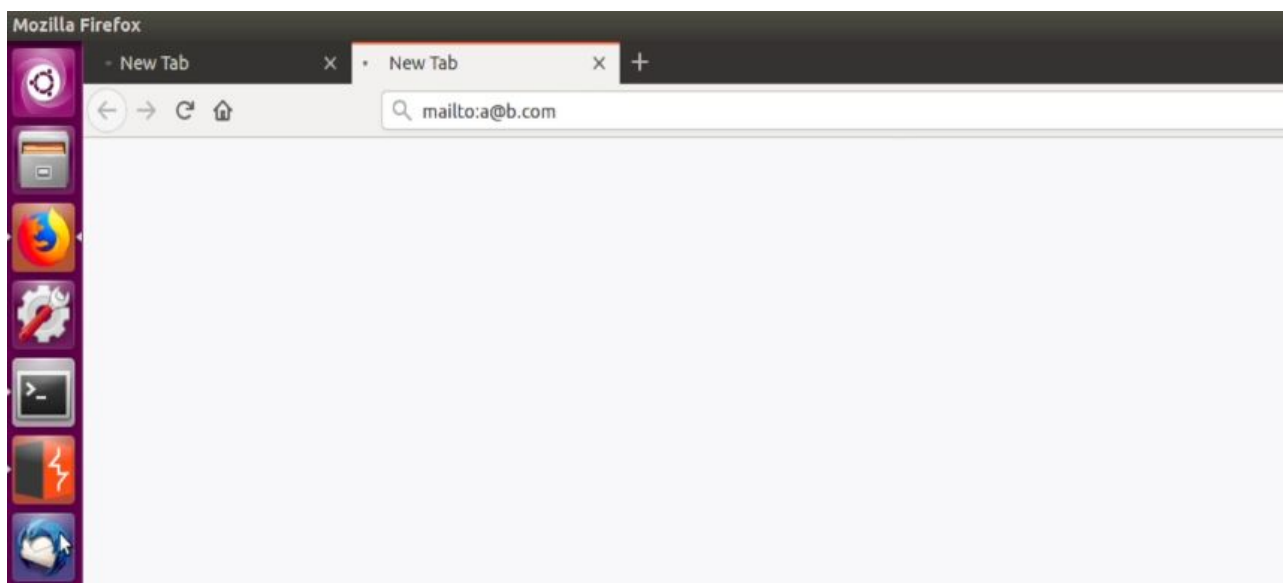


Мы получили диалоговое окно с текстом Hello, world! и написали первую программу на JavaScript.

## Схема mailto:

Давайте теперь попробуем открыть схему mailto:, которая нужна, чтобы отправить письмо.

Введем в браузере адрес <mailto:a@b.com>:



Когда мы переходим по этой схеме, у нас открывается почтовый клиент по умолчанию, в данном случае — Thunderbird.

После того, как мы открыли этот URL, мы запустили программу и можем написать письмо на адрес [a@b.com](mailto:a@b.com).

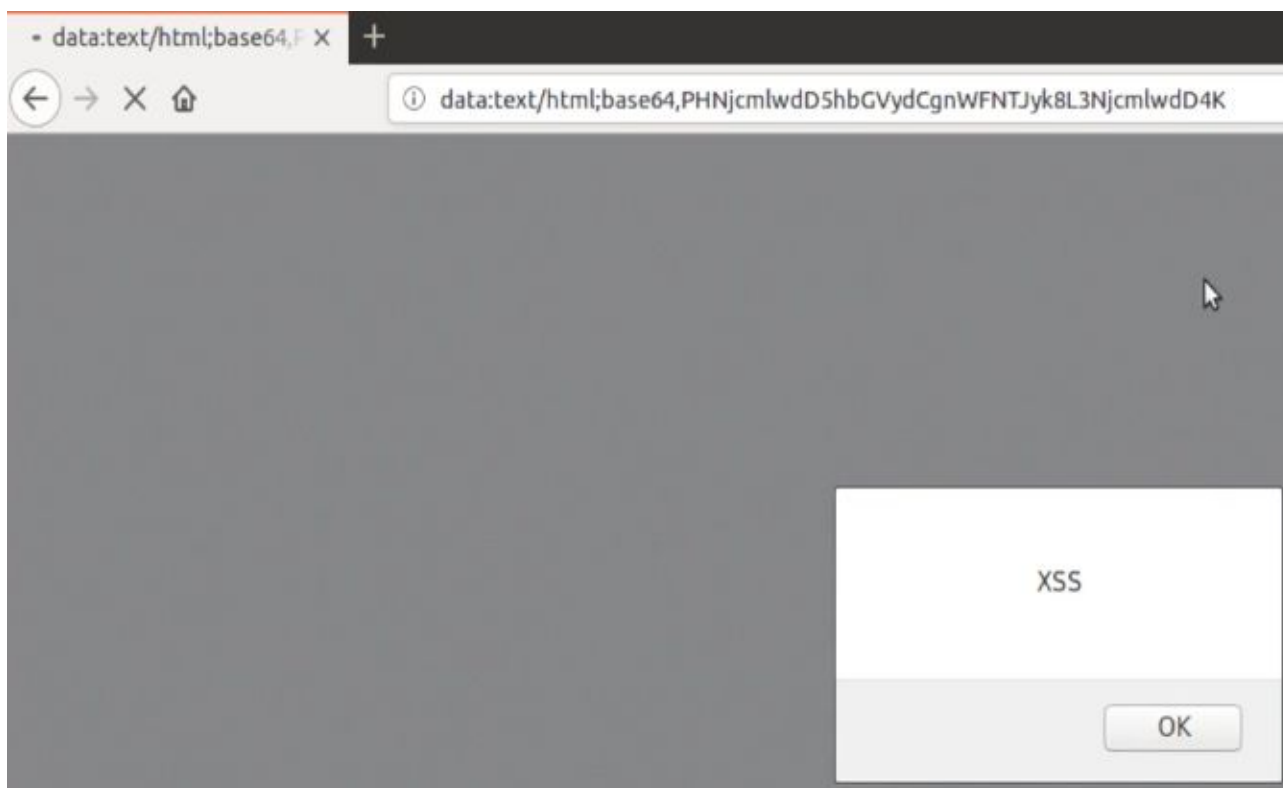
## Схема data:

Рассмотрим пример схемы <data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K>:



Мы видим схему data:, дальше идет text/html — указание типа, который находится внутри. Дальше идет тип кодировки — в данном случае содержимое закодировано в кодировку base64. Таким образом, текст на картинке закодирован в base64 и внутри находится обычный text/html.

Давайте посмотрим, что произойдет, если открыть такой URL:



Сейчас нам важно понимать, что схема `data:` позволяет в некоторых случаях обойти фильтры безопасности.

Давайте подробнее посмотрим, как можно увидеть содержимое ссылки. Для этого мы скопируем полезную нагрузку (payload) и декодируем ее из base64:

```
echo "PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K" | base64 -d
```

Получим на выходе такой ответ:

```
<script>alert('XSS')</script>
```

Как мы видим, здесь полезной нагрузкой в payload находится функция `alert('XSS')` внутри HTML-тегов, описывающих javascript-код. Как это работает, мы узнаем чуть позже, сейчас главное — понимать, что base64 — просто кодировка, которая позволяет представить данные в другом формате. В данном случае, например, удобном для передачи с сервера на клиент и с клиента на сервер.

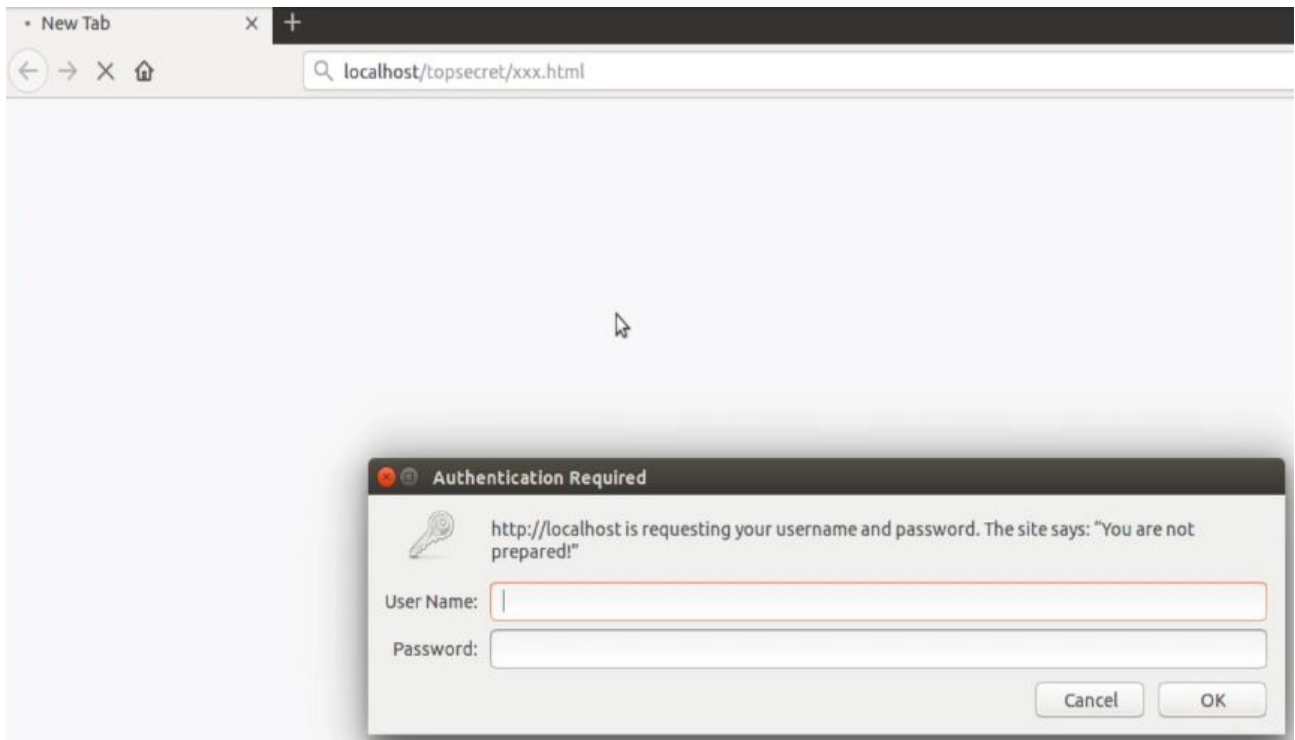
Также существует множество других схем, которые мы постепенно будем изучать и понимать их особенности.

## Авторизация в Nginx

Поговорим, как, ограничить доступ к ресурсу, к которому не должны иметь доступ все пользователи, например к админке портала - /admin.

Для этого в Nginx есть специальная функция, которая включает базовую авторизацию. Базовая авторизация — авторизация протокола HTTP. Давайте посмотрим это на примере.

Откроем виртуальную машину с Ubuntu, перейдём на <http://localhost/topsecret/xxx.html> и попробуем открыть этот файл:



Мы видим, что файл не открылся и браузер запрашивает аутентификацию, то есть логин и пароль.

Давайте разберемся, как устроена базовая аутентификация и найдем пароль. Откроем конфиг Nginx:

```
nano /etc/nginx/sites-available/test.conf
```

Мы видим, что в конфигурационном файле добавилась одна директива location специально для пути /topsecret/:

```
root@ub16:/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;
}

location /topsecret/ {
    auth_basic "You are not prepared!";
    auth_basic_user_file /home/r/login_password.txt;

    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;
}
}

```

В ней есть директива `auth_basic`, которая включает базовую аутентификацию, здесь можно добавить сообщение, которое будет высвечено:

```
auth_basic "You are not prepared!";
```

Также есть `auth_basic_user_file`, в котором располагаются логины и пароли всех пользователей. Мы видим, что он располагается по пути `/home/r/login_password.txt`:

```
auth_basic_user_file /home/r/login_password.txt;
```

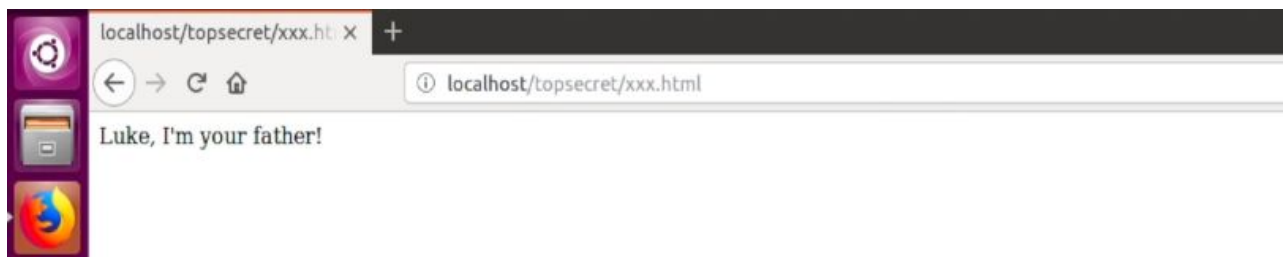
Давайте откроем файл по этому пути:

```
cat /home/r/login_password.txt

admin:{PLAIN}strongpassword
```

Здесь есть логин и пароль, и они записаны в формате `login:{PLAIN}` — это означает, что пароль написан так, как есть. Это небезопасно: когда вы будете использовать подобный функционал не в учебных целях, необходимо применить хэширование (что такое хэширование, мы поговорим позже). После фигурных скобок записан искомый пароль.

Скопируем его и попытаемся войти с его помощью на ресурс:

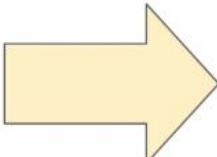


Логин/пароль `admin/strongpassword` подошли и мы получили секретный файл.

В целом, базовая авторизация устроена следующим образом: когда мы вводим логин и пароль и нажимаем ОК, браузер преобразовывает логин и пароль в base64-строку, например:

## Базовая авторизация

login: admin  
password: sjdgksyoscbk



YWRtaW4uc2pkZ2tzeW9zY2JrCg==

Давайте попробуем разобраться, как логин и пароль превратились в такую абракадабру — откроем терминал и напишем:

```
echo "YWRtaW4uc2pkZ2tzeW9zY2JrCg==" | base64 -d  
admin.sjdgksyoscbk
```

Это не что иное, как логин и пароль, соединенные двоеточием. Протокол HTTP, когда передает логин и пароль, соединяет `логин.пароль` и кодирует это в base64. Только после этого он отправляет их на сервер. Безопасно это или нет, зависит от конкретной ситуации. Если есть шифрование HTTPS, это

вполне безопасно. А если нет шифрования, тогда логин и пароль сможет получить любой злоумышленник, который находится в одной сети с жертвой.

## Задание для самоконтроля

Настройте базовую HTTP-аутентификацию у себя на сервере. Для этого в конфигурационный файл нужно будет добавить 2 директивы. Обязательно, после того, как измените конфигурационный файл, не забудьте перезагрузить Nginx (командой `sudo nginx -s reload`), чтобы Nginx перечитал конфигурационный файл.

Естественно, у вас будет путь до файла с паролями и логинами, на это тоже обратите внимание.

## Итоги

В этом видео мы узнали:

1. Что такое схема, для чего она применяется.
2. Примеры основных схем, которые используются в вебе.
3. Базовую авторизацию и ее применение для защиты ресурсов.

## Видеоурок 3

В этом уроке мы разберем:

1. Что такое доменное имя и для чего оно нужно.
2. Устройство DNS и как компьютер с ним взаимодействует.
3. Несколько программ, связанных с IP-адресами и DNS, и узнаем, зачем нужен файл hosts.
4. Punycode и уязвимости, связанные с ним.

## Доменные имена

Мы умеем однозначно идентифицировать ресурс на сервере, но сам сервер идентифицируется доменными именами или адресом сервера.

На слайде ниже показаны примеры доменных имен: [example.com](http://example.com), [geekbrains.ru](http://geekbrains.ru) и [e.mail.ru](mailto:e@mail.ru). В общем виде доменное имя состоит из латинских букв, цифр, символов тире и каждый из уровней доменного имени соединяется точкой:



# Доменное имя и IP-адрес

- example.com
  - geekbrains.ru
  - e.mail.ru
  - ...
- 93.184.216.34
  - 127.0.0.1
  - 2606:2800:220:1:248:18  
93:25c8:1946
  - ...

Под уровнем здесь понимаем, например, [com](#) — это домен верхнего уровня; [example](#) — домен нижнего уровня. В случае с [e.mail.ru](#), [e](#) — домен еще более низкого уровня, чем [mail.ru](#). Почему они так делятся, мы узнаем чуть позже.

Доменные имена необходимы, чтобы идентифицировать сервер, и каждое из них должно соответствовать IP-адресу (подробнее об этом мы узнаем, когда будем говорить про систему доменных имен DNS).

Доменное имя может содержать латинские символы от а до z, цифры от 0 до 9 и дефисы. Итоговый набор слов из такого алфавита, соединенных точкой, будет называться доменным именем.

## IP-адреса

Также сервер может идентифицироваться IP-адресом, они бывают двух видов: IPv4 и IPv6. IPv4 — два верхних примера, IPv6 — третий пример на рисунке выше.

Как вы видите, IPv6 длиннее IPv4, а значит, в него можно записать гораздо больше адресов.

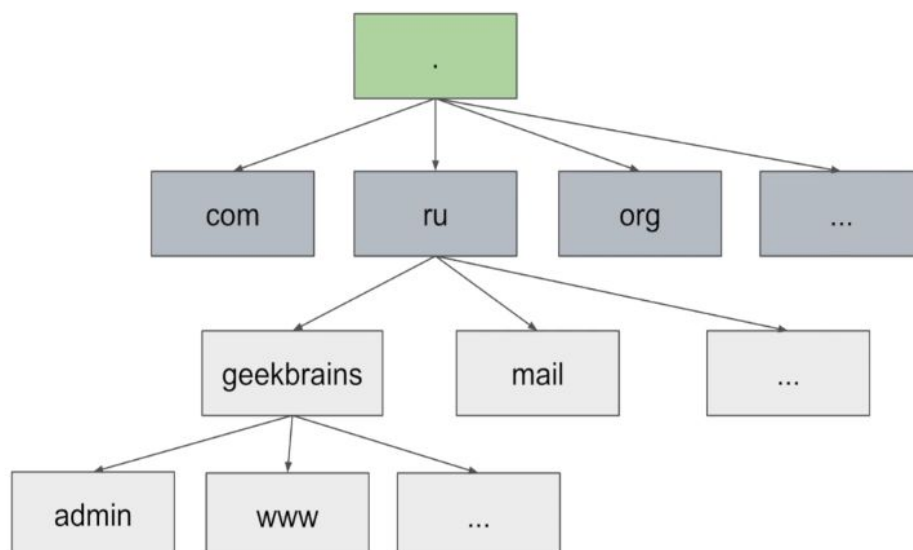
IPv4-адрес состоит из 4 чисел, соединенных точкой, и каждое число может быть в диапазоне от 0 до 255:

IPv4-адрес  
**0-255.0-255.0-255**

Общеизвестно, что сайтов в интернете очень много и если бы все доменные имена хранились на одном сервере, скорее всего, он не выдержал бы нагрузки, потому что на него приходили бы сразу все пользователи интернета. Разберемся, как устроена система доменных имен.

## Система DNS

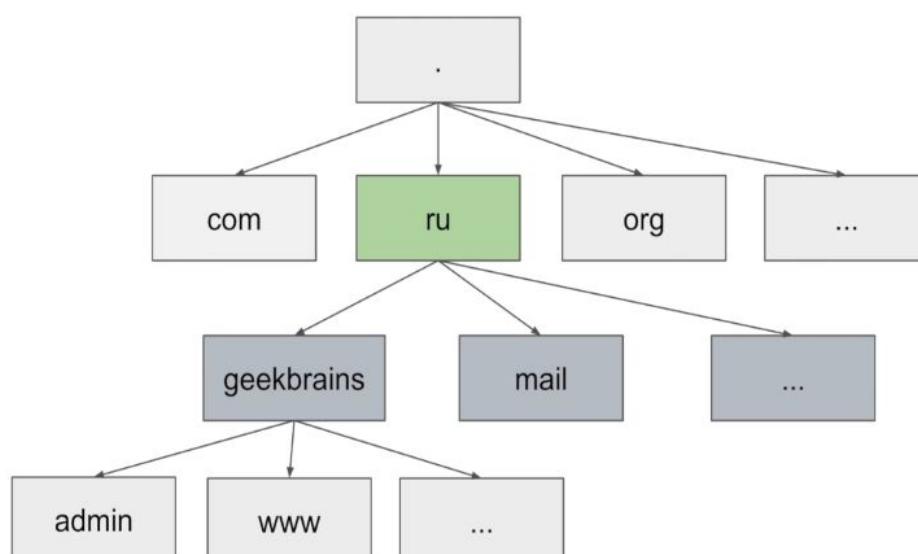
Система доменных имен, или коротко — DNS, устроена в виде дерева:



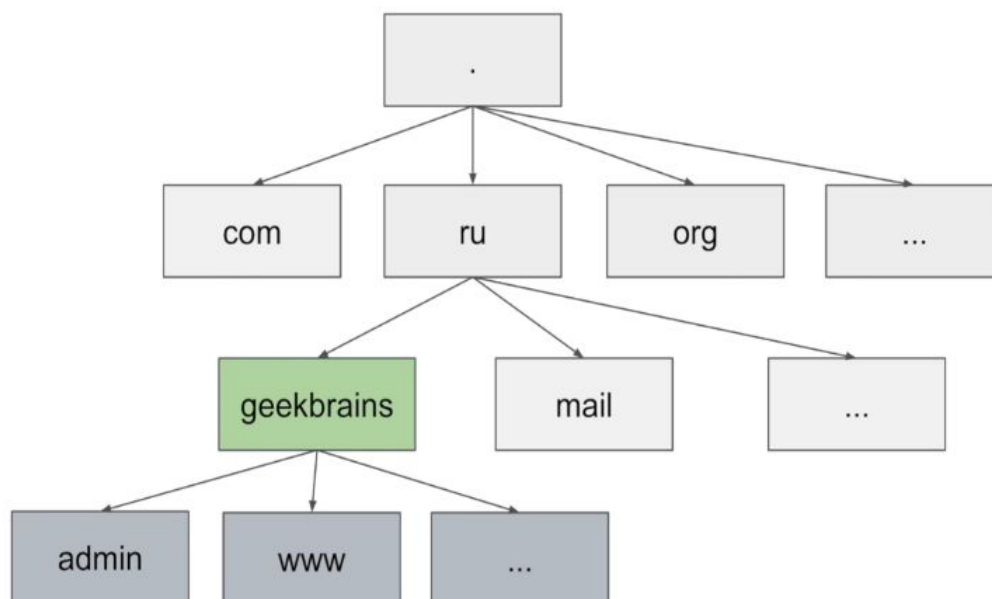
Дерево — такая структура данных, у которой есть узлы или вершины и соединяющие их линии — рёбра. По-другому еще говорят что дерево — это структура типа графа.

В данном случае узлы [.](#), [com](#), [ru](#), [org](#), [geekbrains](#), [mail](#) — все вершины дерева. И рёбра (или ветви) — стрелки, которых их соединяют. В дальнейшем вы часто будете сталкиваться с деревьями и видеть такую структуру данных. Древоподобная структура удобна тем, что можно разграничить зоны ответственности, распределить, какой сервер DNS про что знает. Например, корневой домен [.](#) знает, где находятся DNS-сервера [com](#), [ru](#), [org](#) и так далее (то есть все DNS-сервера верхнего уровня).

Соответственно, DNS-сервер зоны [ru](#) знает, где находятся все DNS-сервера из поддомена [ru](#), например: [geekbrains.ru](#), [mail.ru](#), [yandex.ru](#) и так далее:

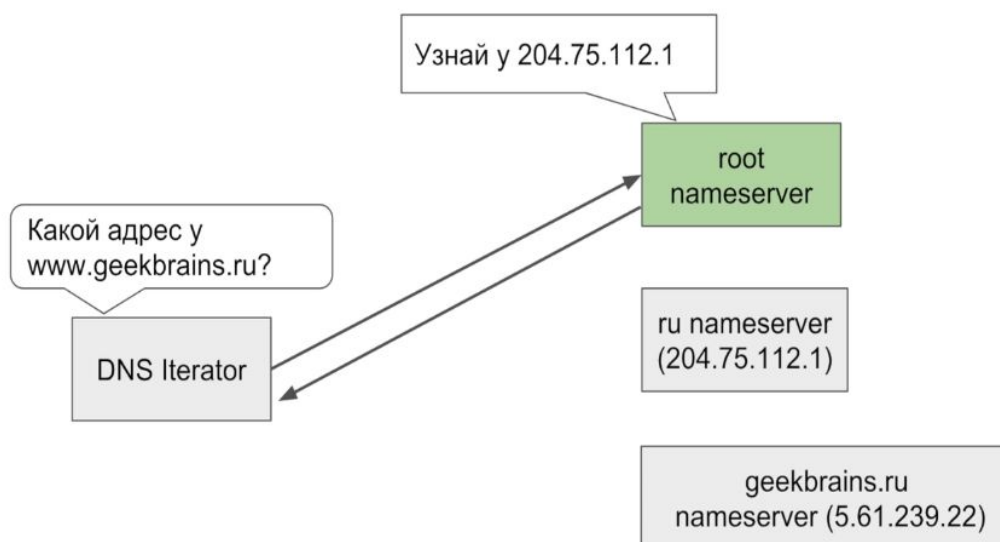


Поддомен — это домен, располагающийся ниже текущего:



Например, [admin.geekbrains.ru](http://admin.geekbrains.ru) — поддомен [geekbrains.ru](http://geekbrains.ru), в то же время [admin.geekbrains.ru](http://admin.geekbrains.ru) — поддомен домена [.ru](http://.ru).

Компьютер узнает, какой адрес соответствует доменному имени, с помощью DNS-итератора и это происходит следующим образом:

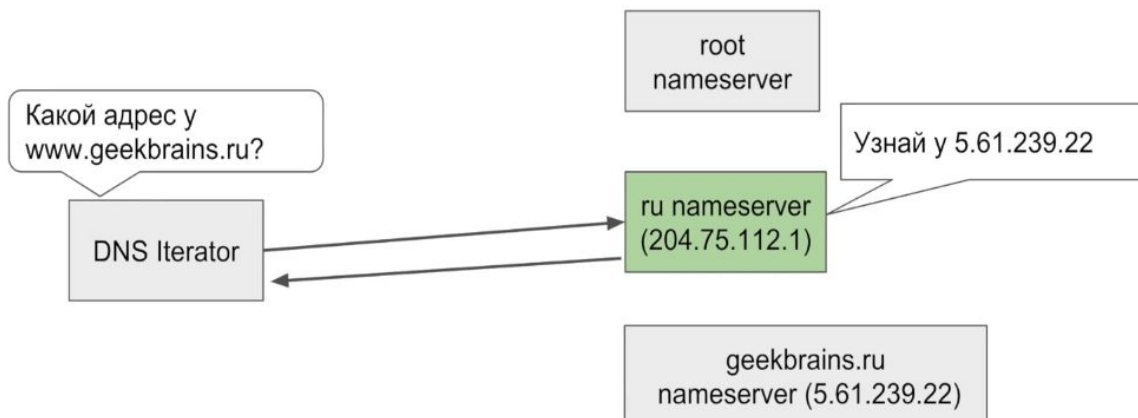


Итератор спрашивает у корневого DNS-сервера, то есть у того сервера, который отвечает за домен . (точка), какой адрес у [geekbrains.ru](http://geekbrains.ru).

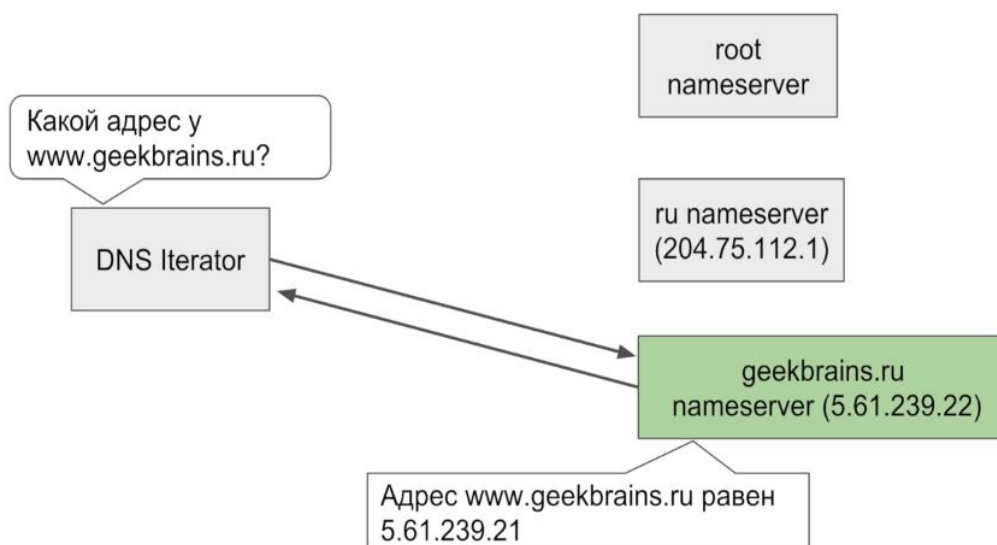
На самом деле каждый домен должен заканчиваться точкой. Например, в данном случае после [www.geekbrains.ru](http://www.geekbrains.ru) должна стоять еще одна точка: [www.geekbrains.ru](http://www.geekbrains.ru). Но, так как все доменные имена содержат эту точку в конце, для простоты ее опускают и не пишут.

DNS-итератор спрашивает у корневого сервера адрес [geekbrains.ru](http://geekbrains.ru). Но предположим, что корневой сервер не знает этого адреса, но знает, у кого можно спросить.

Он отвечает: узнай у DNS-сервера с адресом 204.75.112.1, отвечающего за зону [.ru](#). DNS-итератор идет с тем же самым вопросом на указанный DNS-сервер зоны [.ru](#):



Но этот сервер тоже не знает адрес [geekbrains.ru](#), зато знает, у кого можно спросить. Он отвечает: узнай у DNS-сервера с адресом 5.61.239.22 (допустим, это адрес DNS-сервера [geekbrains.ru](#)). DNS-итератор идет к этому серверу:



Сервер [geekbrains.ru](#) уже знает адрес [www.geekbrains.ru](#) и отвечает: [www.geekbrains.ru](#) равен 5.61.239.21.

После этого компьютер понимает, куда необходимо пойти, чтобы получить страницу [geekbrains.ru](#). Таким образом, клиенты подключаются к серверу не по имени, а по IP-адресу, используя для этого подключение по протоколу IP из стека протоколов TCP/IP (понятие стек здесь — набор чего-то).

Подробнее, что это такое и как происходит подключение, вы сможете узнать в специальном курсе по сетям. Сейчас же нужно понимать, что для того, чтобы подключиться к серверу, необходимо сначала узнать его адрес и потом зайти по нему на сервер.

# Полезные утилиты

## Команда host

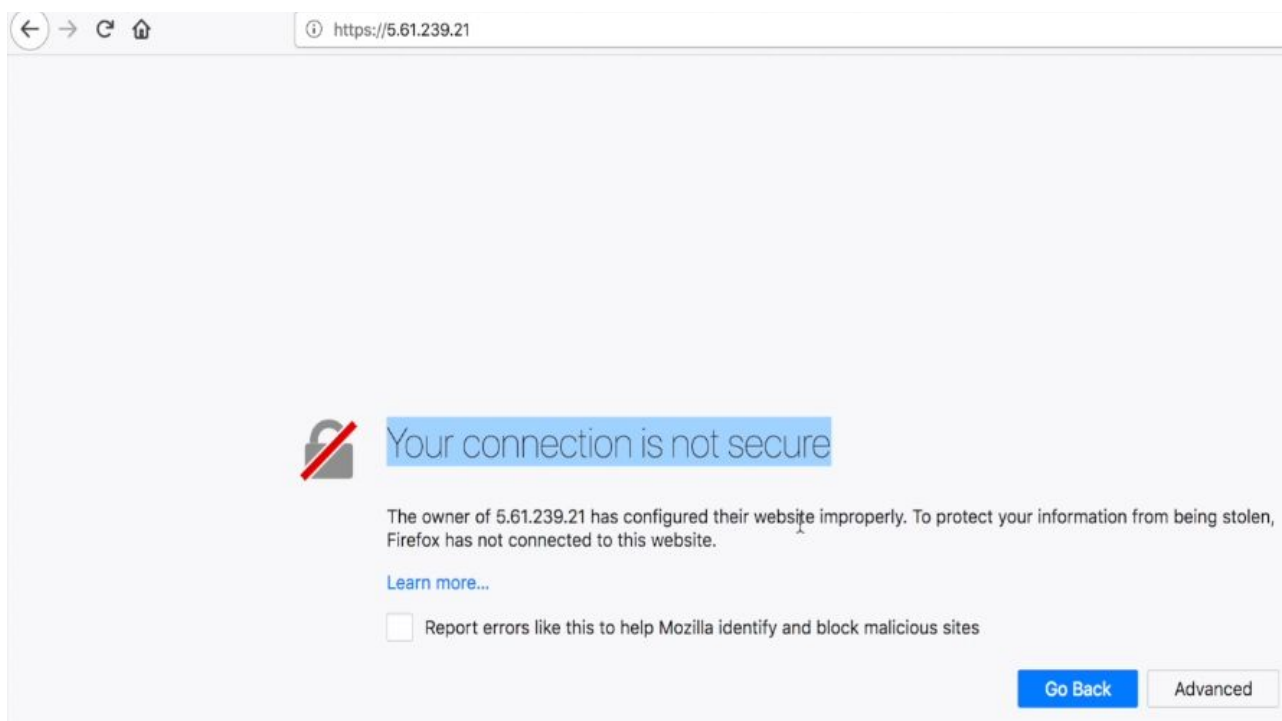
Посмотрим, как узнать IP-адрес на практике и чем это может быть нам полезно. Откроем терминал и воспользуемся командой `host`, чтобы узнать адрес [geekbrains.ru](https://geekbrains.ru):

```
host geekbrains.ru

geekbrains.ru has address 5.61.239.21
geekbrains.ru has address 5.61.239.22
geekbrains.ru mail is handled by 10 mx.mail.ru.
```

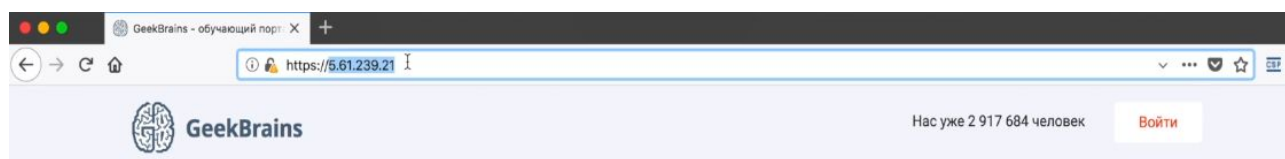
Видим, что [geekbrains.ru](https://geekbrains.ru) имеет 2 адреса — 5.61.239.21 и 5.61.239.22. Это означает то, что оба сервера — как 21, так и 22 — могут обработать запросы для [geekbrains.ru](https://geekbrains.ru).

В некоторых случаях, но не во всех, мы можем обратиться к серверу не через доменное имя, а напрямую через IP-адрес. Выделим первый адрес, скопируем, перейдем в браузер, вставим его и выберем схему `https`:



Браузер сообщил, что соединение небезопасно, но на самом деле это из-за того, что полученный сертификат не совпал с IP-адресом, на который мы перешли — такое иногда бывает. Про сертификаты мы подробнее узнаем в следующем уроке. Пока же нажмем `Advanced`, добавим адрес в исключение и подтвердим это.

В итоге как мы увидим откроется обычный сайт [geekbrains.ru:](https://geekbrains.ru/)



Но мы обратились к нему напрямую по адресу, а не через доменное имя.

## Команда whois

Еще есть очень полезная утилита `whois`, с ее помощью мы можем узнать, кому принадлежит IP-адрес, а также другую полезную информацию. Выполним запрос `whois` на IP-адрес [geekbrains.ru](https://geekbrains.ru/):

```
whois 5.61.239.22

% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

% Note: this output has been filtered.
%       To receive output for a database update, use the "-B" flag.

% Information related to '5.61.239.0 - 5.61.239.31'

% Abuse contact for '5.61.239.0 - 5.61.239.31' is 'abuse@corp.mail.ru'

inetnum:        5.61.239.0 - 5.61.239.31
netname:        M100-COLO
descr:          M100 Colocation
country:        RU
admin-c:        EY1327-RIPE
admin-c:        VG659-RIPE
tech-c:         MAIL-RU
status:         ASSIGNED PA
mnt-by:         MNT-M100
created:        2018-11-19T16:51:39Z
last-modified:  2018-11-27T13:02:47Z
source:         RIPE

role:           MAIL.RU NOC
address:        Limited liability company Mail.Ru
```

```

address:      Leningradskiy prospect, 39/79
address:      125167 Moscow Russia
phone:        +7 495 7256357
fax-no:       +7 495 7256359
remarks:      -----
admin-c:      VG659-RIPE
admin-c:      EY1327-RIPE
tech-c:       DBF3-RIPE
tech-c:       IS13
remarks:      -----
remarks:      General questions: noc@corp.mail.ru
remarks:      Spam & Abuse: abuse@corp.mail.ru
remarks:      Search Abuse: search-abuse@corp.mail.ru
remarks:      Routing inquiries: ncc@corp.mail.ru
remarks:      Peering issues: ncc@corp.mail.ru
remarks:      -----
remarks:      ----- A T T E N T I O N !!! -----
remarks:      Please use abuse@corp.mail.ru e-mail
remarks:      address for spam and abuse complaints.
remarks:      Mails for other addresses will be ignored!
remarks:      -----
mnt-by:       MNT-NETBRIDGE
abuse-mailbox: abuse@corp.mail.ru
nic-hdl:      MAIL-RU
created:      2010-11-29T12:03:47Z
last-modified: 2018-11-14T20:30:14Z
source:       RIPE # Filtered

person:       Elena Yakupova
address:      39/79, Leningradsky prospect
address:      Moscow, Russia,125167
phone:        +7 495 725 6357
nic-hdl:      EY1327-RIPE
mnt-by:       MNT-NETBRIDGE
created:      2018-11-14T11:06:34Z
last-modified: 2018-11-14T11:06:34Z
source:       RIPE # Filtered

person:       Vladimir Gabrelyan
address:      47, Leningradsky prospect, Moscow, Russia
fax-no:       +7 495 7256357
phone:        +7 495 7256357
nic-hdl:      VG659-RIPE
mnt-by:       MNT-NETBRIDGE
remarks:      modified for Russian phone area changes
created:      2004-06-08T13:21:36Z
last-modified: 2010-11-29T13:30:08Z
source:       RIPE # Filtered

% Information related to '5.61.232.0/21AS47764'

route:        5.61.232.0/21
descr:        Mail.Ru Network

```

```
origin:          AS47764
mnt-by:          MNT-NETBRIDGE
created:         2012-06-13T10:00:20Z
last-modified:   2018-11-27T13:02:47Z
source:          RIPE

% This query was served by the RIPE Database Query Service version 1.93.2
(HEREFORD)
```

Как мы видим, если есть претензии или что-то нужно узнать про [geekbrains.ru](https://geekbrains.ru), можно сразу писать на [noc@corp.mail.ru](mailto:noc@corp.mail.ru) — то есть мы понимаем, что [geekbrains.ru](https://geekbrains.ru) принадлежит [mail.ru](https://mail.ru). Тут много разной информации про эти адреса и она очень полезна на этапе разведки, когда мы еще ничего не знаем о сервисе, который проверяем на безопасность, и ищем адреса, доменные имена этого сервиса для того, чтобы в дальнейшем их исследовать.

## Файл hosts

Также давайте поговорим про файл `hosts` — этот файл очень полезен, когда мы хотим поменять соответствие доменного имени и IP-адреса на своей машине, но при этом не хотим настраивать для этого DNS-сервер. Особенно это полезно на первых порах, когда настроить выделенный DNS-сервер довольно затруднительно.

Давайте откроем виртуальную машину с Ubuntu, зайдём в директорию `/etc` и проверим файл `hosts`. Это важно: на Linux-подобных системах, таких как Ubuntu или Mac, хост-файл находится в каталоге `/etc`, а в Windows — в каталоге `C:\Windows\System32\drivers\etc`.

```
cat /etc/hosts

127.0.0.1      localhost
127.0.1.1      r-VirtualBox

# The following lines are desirable for IPv6 capable hosts
::1           ip6-localhost ip6-loopback
fe00::0       ip6-localnet
ff00::0       ip6-mcastprefix
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters
```

Как мы видим, файл устроен следующим образом: здесь есть две колонки — слева адрес, а справа доменное имя. Адрес может быть и IPv4, и IPv6, но сейчас мы остановимся на IPv4-адресах. Как мы видим, в этом файле поставлено соответствие домена [localhost](https://localhost) и адреса 127.0.0.1

Если мы захотим, например, чтобы домен [geekbrains.ru](https://geekbrains.ru) резолвился или по-другому разрешался, или ставился в соответствие нашей локальной машине, необходимо изменить файл `hosts`:



```
nano /etc/hosts

127.0.0.1    localhost
127.0.0.1    geekbrains.ru
127.0.1.1    r-VirtualBox

# The following lines are desirable for IPv6 capable hosts
::1         ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
```

Теперь попробуем открыть в браузере [geekbrains.ru](http://geekbrains.ru):



Открылся не [geekbrains.ru](http://geekbrains.ru), а наш сайт, где написано Welcome to nginx.

## Punycode

Чтобы добавить в доменное имя интернациональные символы, например, русские буквы или китайские символы, используется Punycode.

Он выглядит так:

xn--gkbrains-c8ga.ru  
==  
geekbrains.ru

На первой строчке представлен сам Punycode, а на второй — то, что он на самом деле представляет. Буквы «ee» выделены не случайно: это не английские, а русские буквы. И хотя по написанию они выглядят максимально похожими, но на самом деле домены с русскими «ee» и домен [geekbrains.ru](http://geekbrains.ru) с английскими буквами «ee» — разные домены.

Угроза безопасности здесь в том, что Punycode дает большие возможности для фишинга. Например, если злоумышленник отправляет жертве сообщения и хочет, чтобы она перешла по ссылке, он может

использовать Punycode и отправить ссылку как будто на [geekbrains.ru](http://geekbrains.ru), но на самом деле это будет зарегистрированный им домен [geekbrains.ru](http://geekbrains.ru) с русскими «ее» и там будет вредоносный сайт.

С этим борются несколькими способами:

1. Регистраторы доменных имен должны проверять, регистрируется ли домен [geekbrains.ru](http://geekbrains.ru) с русскими «ее» самими [geekbrains.ru](http://geekbrains.ru) или каким-то третьим лицом. Если это третье лицо, такой запрос должен быть отклонен.
2. Со стороны сайта можно защититься следующим образом: все Punycode-домены можно отображать как в первой строчке, то есть латинскими символами с xn--.. и т. п. вместо того, чтобы показывать их с теми символами, которые есть изначально.

## Итоги

В этом видеоуроке вы узнали:

1. Как однозначно идентифицировать сервер с помощью доменного имени.
2. Что такое доменное имя, для чего оно нужно, как устроены и какая связь между доменными именами и IP-адресами.
3. Как устроен DNS, как он работает, как делают запросы. Освоили несколько утилит для работы с DNS.
4. Что такое Punycode и какие проблемы безопасности с ним связаны.

## Видеоурок 4

В этом видео мы разберем:

- 1) Что такое порт, зачем он нужен, для чего применяется и как порты можно разделить на различные классы.
- 2) Что такое путь в URL.
- 3) Что такое URL Encode, для чего он нужен и как производится.

## Сетевые порты

Как правило, на сервере запущена не одна программа, а несколько, и зачастую нескольким программам нужен доступ в интернет. Допустим, на нашем сервере есть 2 программы, к которым клиенты хотят обращаться. Чтобы клиент мог обратиться к программам, запущенным на сервере, существуют порты.

Говорят, что сервисы «слушают» или «сидят» на порту (от английского listen — слушать, прослушивать). Например, Nginx слушает на сетевом порту 80 по протоколу TCP — то есть, когда мы запускаем Nginx, он занимает порт tcp/80 и ждет, пока на этот сетевой порт не придут соединения или

не произойдут какие-то события. После того, как событие или соединение приходят, Nginx обрабатывает их так, как в нём заложено.

Есть некоторые стандартные порты, то есть порты, на которых слушают определенные сервисы:

## Стандартные порты

- 80 -- http (`http://geekbrains.ru:80 == http://geekbrains.ru`)
- 443 -- https (`https://geekbrains.ru:443 == https://geekbrains.ru`)
- 22 -- ssh

Как правило, порт 80 отдан под запросы HTTP и поэтому, когда мы набираем <http://geekbrains.ru>, — это эквивалентно обращению к <http://geekbrains.ru:80>.

То же самое касается HTTPS — если мы настроим HTTPS в Nginx, то по умолчанию HTTPS будет на 443 порту. Чтобы зайти на [geekbrains.ru](https://geekbrains.ru) по протоколу HTTPS, нам нужно будет подключиться к порту 443.

На 22 порту, как правило, слушает сервер SSH (Secure Shell). SSH — это удаленный терминал, то есть со своего компьютера можно управлять сервером, который находится где-то очень далеко и это взаимодействие осуществляется как раз по протоколу SSH.

Давайте посмотрим на такой пример и попробуем понять, что же будет открыто, если мы обратимся по протоколу HTTP, но по порту 443 и что будет, если мы обратимся по HTTPS, но на порт 80:

- <http://geekbrains.ru:443>
- <https://geekbrains.ru:80>

Вы можете сначала представить результат, а потом попробовать сделать это в браузере и посмотреть, что получится. Откроем Ubuntu и проверим, что будет, когда мы перейдем по этим адресам.

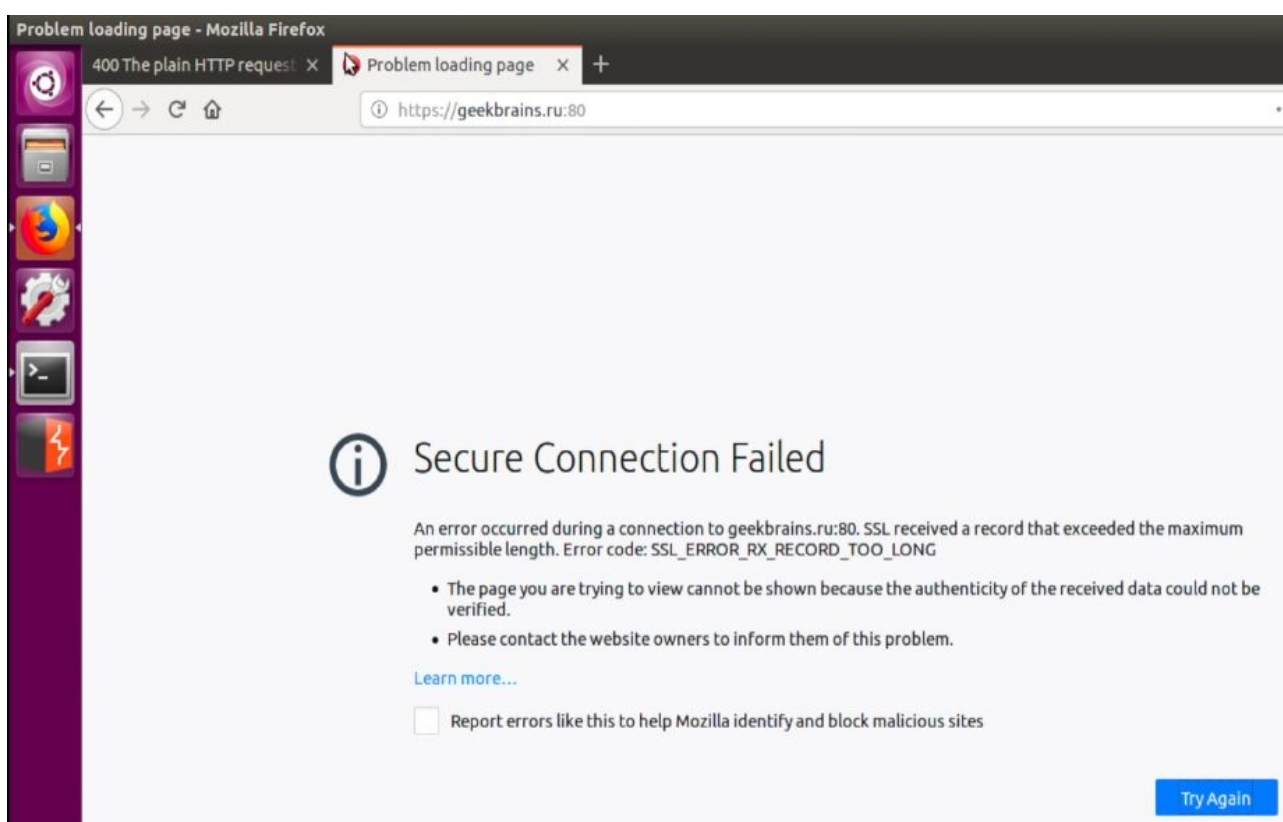
В URL порт находится после адреса сайта и двоеточия:

Попробуем зайти на <http://geekbrains.ru> по HTTP-протоколу, но при этом на порт 443, то есть тот, на котором слушает HTTPS:



Nginx выдал ошибку и не выдал нам страницу [geekbrains.ru](https://geekbrains.ru). Это произошло потому, что протоколы разные, соответственно, клиент и сервер не могут общаться на разных языках.

Теперь попробуем проверить, что будет в случае адреса HTTPS и порта 80:



Произошло ровно то же самое — клиент попытался установить защищенное соединение, но на порту 80 Nginx слушает без шифрования, поэтому сервер и клиент не смогли договориться, на каком языке им общаться, и соединение было разорвано.

Для портов 0-1024 существует особенность: без прав суперпользователя на них слушать нельзя. Именно по этой причине Nginx и запускается от суперпользователя, т. к. Nginx по умолчанию слушает на порту 80. Вы убедитесь в этом, если откроете конфигурационный файл Nginx.

Зайдем в терминал и выполним две команды:

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

```
less default

# 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;
    }
}
```

По умолчанию Nginx слушает на порту 80, но мы можем указать любой другой. Сделаем это в нашем конфиге: откроем его в редакторе nano и пусть Nginx слушает на порту 31337:

```
nano default

# Default server configuration
#
server {
    listen 31337;
    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;
}
}

```

Когда мы меняем порт, недостаточно просто перезагрузить конфигурационный файл, необходимо полностью остановить Nginx и запустить его заново. Для этого мы выполним команду:

```
sudo nginx -s quit
```

И потом запустим Nginx заново:

```
sudo nginx
```

Откроем браузер и проверим:



Действительно, теперь Nginx слушает на этом порту.

## Путь к ресурсу

Теперь обсудим, что такое путь.

Путь — то, что располагается после порта или доменного имени, начиная со слеша: [/topics/5200](#) — это путь и [/login](#) — тоже путь.

Путь необходим, чтобы на конкретном сервере однозначно идентифицировать ресурсы. Например, ресурс или файл [/login](#) — форма логина, и когда мы переходим к пути [/login](#), нам отдается форма логина, если говорить в упрощенной форме.

## URL Encode

Все, что находится дальше домена и порта, как правило, подвержено URL Encode:

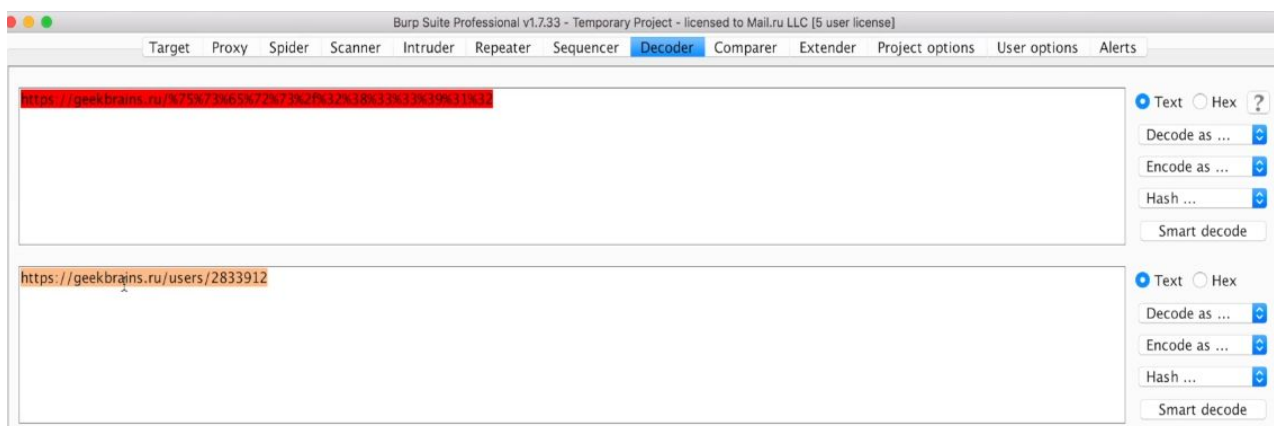
`https://geekbrains.ru/%75%73%65%72%73%2f%32%38%33%33%39%31%32`

То, что вы сейчас видите на экране — не ошибка. Это называется URL encode. Чтобы понять, что находится в этой ссылке, используем Burp, где есть специальный инструмент Decoder, который нужен для декодирования:



Вставим ссылку и выберем Decode as ...

Burp Decoder умеет как декодировать, так и кодировать обратно — выбираем кодировку URL:



В окне ниже появился результат декодирования и мы видим, что это на самом деле ссылка на пользователя.

Также в Вигр можно закодировать обратно, правда, URL encode в Вигр работает так, что он кодирует сразу весь текст:

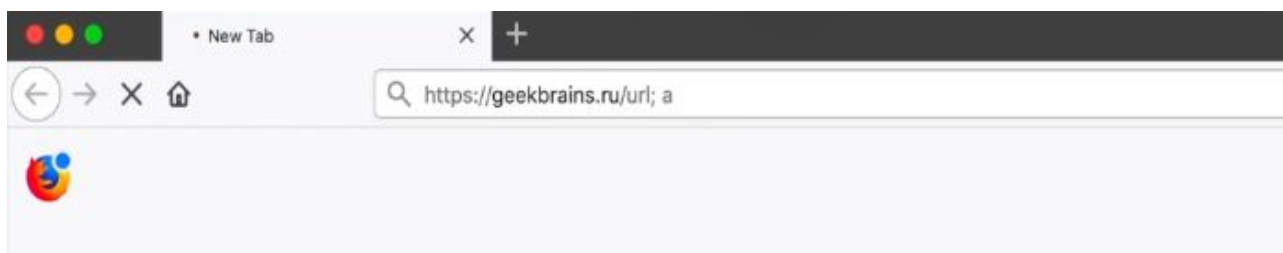


При этом, если мы такую строку ставим в браузер, он не поймет, что с ней нужно делать.

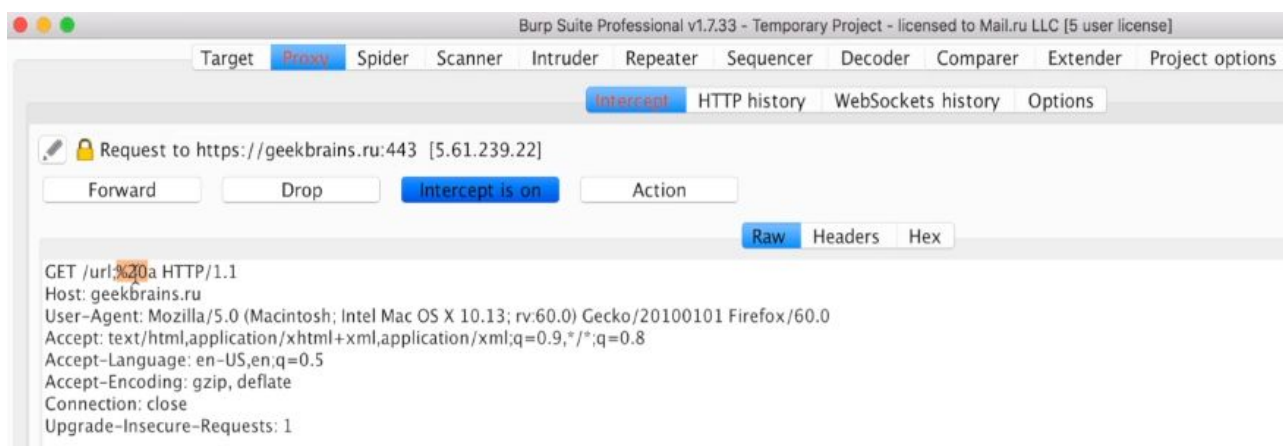
URL encode применяется только к данным, которые находятся после слеша / — к пути, параметру и якорю. Еще важно знать, что браузер применяет URL encode перед тем, как отправить данные на сервер. Рассмотрим это на следующем примере.

Откроем Burp, Proxu, включим Intercept. Откроем Firefox, попробуем зайти на <https://geekbrains.ru> и здесь дополнительно введём **/url; a** и нажмем Enter:





И мы увидим, что некоторые символы браузер сам переводил в URL encode:



Так как пробел завершает URL, то браузер, чтобы передать его на сервер, перевел его в URL encode. В данных этот пробел представляется не как пробел, а как %20. То же происходит и с другими символами, которые имеют специальное значение в синтаксисе URL.

Разберемся, почему именно %20, а не %30 или %100. Это идет от того, как именно происходит URL encode. Чтобы это понять, познакомимся с таблицей ASCII.

## Таблица ASCII

# ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Каждый столбец содержит в себе представление символа в десятичной системе счисления и в Hex — шестнадцатеричной системе. Hex называется так потому что английское слово, обозначающее шестнадцатеричную систему счисления — hexadecimal.

Числа 32 и 20 значат одно и то же, просто в десятичной системе счисления это будет 32, а в шестнадцатеричной 20.

Эти символы обозначают пробел, то есть %20 в URL появился, потому что это hex-представление пробела. Соответственно, чтобы перевести в URL encode какой-то символ, в таблице ASCII необходимо взять его hex-представление и добавить символ процента. С символами, которые не входят в таблицу ASCII, мы разберемся следующих уроках.

Важно понять и запомнить, что URL encode производится так: сначала мы ищем соответствие символа и его hex-представления в таблице ASCII и потом добавляем процент. Автоматизировано это в Burp в инструменте Decoder.

URL encode нужен, чтобы передавать символы без учета их значения в контексте синтаксиса URL. То есть, например, в URL, если мы поставим пробел, это означает, что после него символы не будут восприниматься, т. к. на пробеле URL заканчивается. Но если мы хотим передать пробел как часть пути к ресурсу, необходимо написать вместо пробелов %20 или по-другому сделать URL encode для пути к ресурсу.

## Итоги

Подведем итоги по этому видео. Мы узнали:

1. Что такое путь в URL, для чего он нужен.
2. Как производится URL encode и где его можно применять.
3. Что такое порт, зачем он нужен в URL, какие порты бывают и как их использовать.

## Видеоурок 5

В этом видео мы узнаем:

1. Что такое запрос в URL и для чего он нужен.
2. Понятие якоря и его применение в URL.

## Запрос в URL

Чтобы передать данные на сервер, существуют запросы.

Запрос в URL пишется после знака вопроса и состоит из пар Имя=Значение. Слева от равно стоит имя, то есть та переменная, в которую мы записываем значение. Справа — значение переменной:

1. `https://geekbrains.ru/courses?tab=professions`
2. `https://e.mail.ru/search/?q_query=mail&from_suggest=0&from_search=1&q_threads=1`

В первом примере `tab` — имя переменной, параметр `professions` — значение параметра `tab`.

В стандартах явно не указано, как должен выглядеть запрос, и там может быть произвольный текст, но в веб принято, что символ `&` — разделитель между параметрами:

```
q_query=mail&from_suggest=0&from_search=1&q_threads=1
```

**q\_query** -- параметр

**mail** -- значение параметра

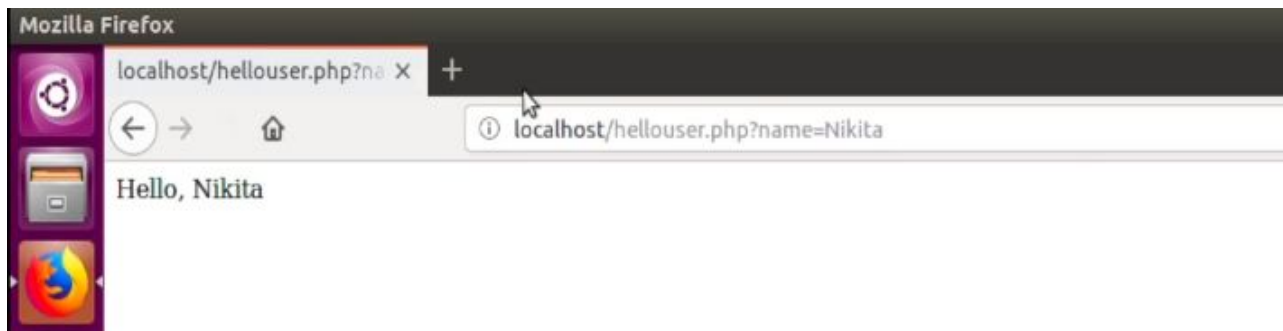
**&** -- разделитель

Обычно разделителем является **&**, но некоторые парсеры принимают за разделитель и **;**

Стоит отметить, что некоторые парсеры — программа, которая принимает на вход текст в некотором формате и разбирает этот текст на составные части. Вместо & можно использовать ; (точку с запятой). Некоторые парсеры это воспринимают, и иногда может возникнуть недопонимание, если вы не знаете этой особенности.

Посмотрим, как применяются запросы, на примере. Мы уже применяли запросы в предыдущих видео, когда использовали функцию `get()` в PHP.

Откроем Firefox и ведем в браузере <http://localhost/hellouser.php?name=Nikita>:



Мы передали параметр в URL и PHP отображает нам этот URL-параметр на странице. Код:

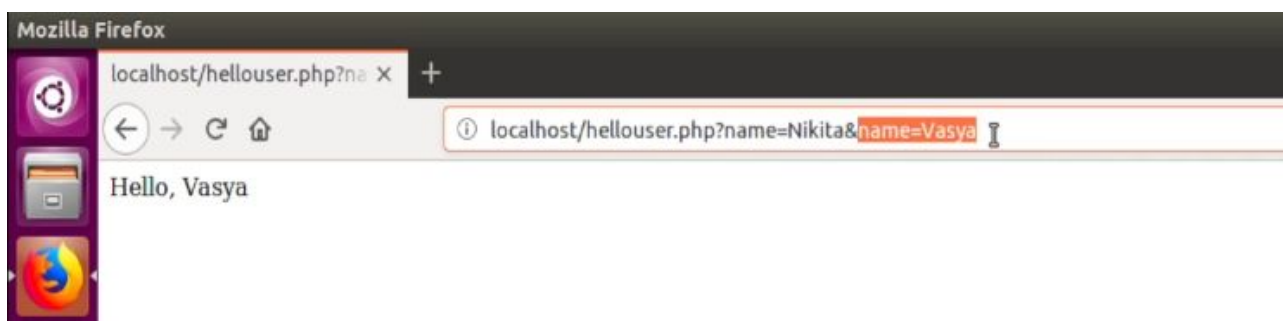
```
cat /var/www/html/hellouser.php

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

В PHP для этого используется специальная переменная `_GET`. В других языках программирования, где нет встроенных функций и переменных для работы с HTTP и URL, используются специальные встроенные функции.

Если в данном случае мы передадим несколько параметров, то те, которые не используются, будут игнорироваться.

Попробуем передать один и тот же параметр несколько раз:



Как мы видим, парсер срабатывает так, что выбирает в качестве аргумента последний из параметров — тут это был `?name=Vasya`, поэтому оно и отобразилось.

## Якорь

Если страница очень большая, а нам нужно переместиться на конкретный участок так, чтобы не перелистывать ее всю, а попасть сразу в нужное место, используем якорь (то, что находится после решетки):

1. `https://geekbrains.ru/#page1`
2. `https://geekbrains.ru/#page2`
3. `https://geekbrains.ru/#page3`
4. **Данные в якорь не уходят на сервер!**

Например, если вы зайдете на сайт <https://geekbrains.ru/> и введёте `#page1`, то попадете на верх первой страницы с якорем `page1`, и т. д.

Сервер в URL может явно указать браузеру, что необходимо делать, куда следует переместиться на данной странице — чаще всего якорь используется именно так. И очень редко он нужен для передачи параметров клиенту. Важно понимать, что якорь никогда не применяют, чтобы передать параметры сервера — для этого используются GET-запрос и его параметры.

Якорь никогда не уходит на сервер. Давайте посмотрим, как он может использоваться, на примере.

## Практика

Откроем Ubuntu, возьмем уже подготовленный файл <http://localhost/anchor.html>:



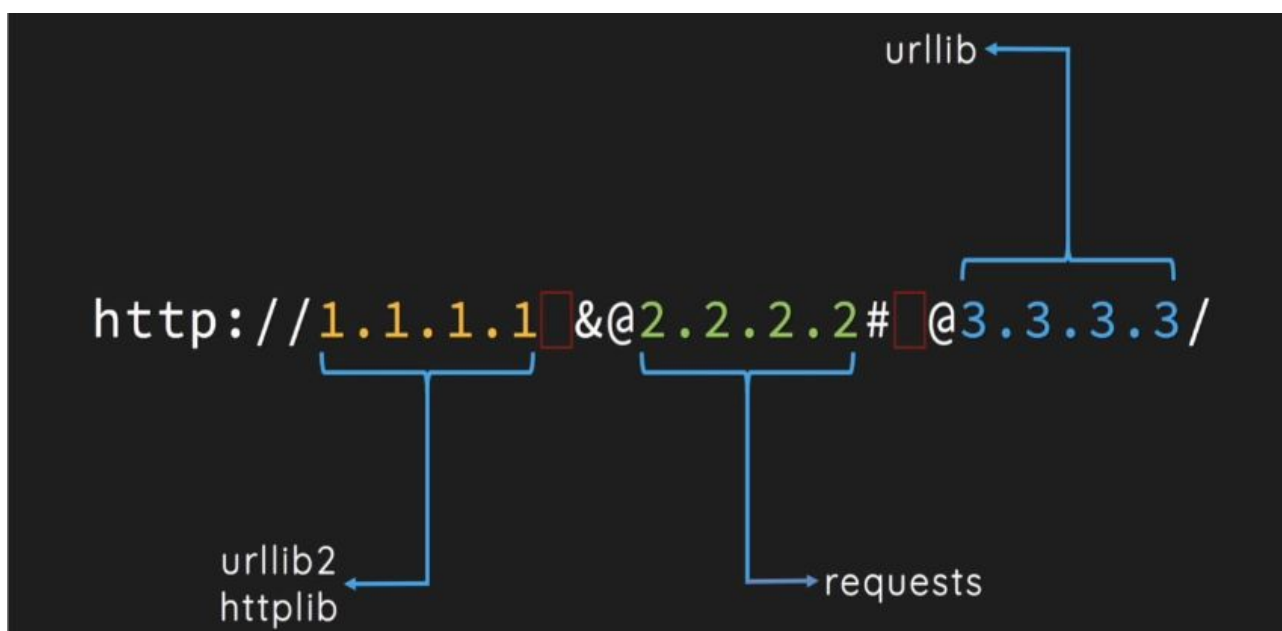
Если мы открываем просто так, появляется самое начало страницы. Но если мы введём <http://localhost/anchor.html#page2>, то сразу перескочим на вторую часть. То, что вводится после символа решетки, задается HTML-атрибутом `id`. Подробнее HTML-атрибуты разберем на следующем уроке, а сейчас нам нужно понимать, что мы можем любому элементу на странице присвоить идентификатор и обращаться к нему по идентификатору через якорь, чтобы быстро переходить к этому элементу.

## URL и уязвимости

Теперь, когда мы разобрались со структурой URL, мы можем задаться вопросом: когда программы, которая разбивает URL на составные части, парсят URL, делают они это все одинаково или есть расхождения?

Одна из фундаментальных проблем веба — в том, что каждый производитель и разработчик по-разному интерпретирует тонкие моменты и из-за этого зачастую возникают проблемы информационной безопасности.

Рассмотрим пример на слайде:



Здесь изображен URL и показано, как разные библиотеки определяют, что из этих адресов будет хостом, а что — адресом сервера. Мы видим, что библиотека `urllib2` и `httpLib` — из Python, они думают, что хост — то, что после `//`. Библиотека `requests` думает, что хост — то что после `&@`. А `urllib` думает, что хост — то, что после `#@`. И в такой ситуации очень опасно бывает полагаться на фильтры безопасности (например Web Application Firewall (WAF)). Фильтр безопасности — программа, которая проверяет запросы от клиента и смотрит, есть ли в этом запросе данные, которые могут быть потенциально вредоносными.

Такие расхождения в парсинге URL могут приводить к тому, что можно обойти фильтры защиты. Допустим, что фильтр использует библиотеку `urllib`. Она подумает, что хост, на который указывает этот URL — `3.3.3.3`, и пропустит его. А когда он придет на backend, хостом окажется `1.1.1.1`, потому что на бэкенде используется, например, библиотека `urllib2`, и такое расхождение в парсинге приведет к уязвимости. С ее помощью можно будет обходить фильтр защиты для эксплуатации Server-Side request Forgery (SSRF). Что такое SSRF, мы узнаем в дальнейших курсах. А сейчас важно понимать, что в вебе иногда даже в стандартах явно не прописаны тонкие моменты: как разбирать, как парсить

те или иные ситуации. Из-за этого у производителей происходит расхождение в том, как это делается и за счет этого случаются ошибки.

## Итоги

Давайте подведем итоги видео, в нем вы узнали:

- 1) Что такое запрос и на примере увидели как применяется запрос.
- 2) Понятие якоря — в каких случаях его нужно использовать, а в случаях — нельзя или не получится.
- 3) URL, его составные части, научились с ними работать.

В следующем уроке нас ждёт протокол HTTP — мы научимся с ним взаимодействовать, посмотрим, как с ним работает браузер и узнаем его синтаксис.

## Ссылки к уроку

1. [Что такое URL.](#)
2. [Схемы \(протоколы\) URL.](#)
3. [Basic HTTP: авторизация для Nginx.](#)
4. [Что такое DNS?](#)
5. [URL-кодирование и декодирование строк.](#)
6. [Тип взлома: внедрение через URL.](#)