GeekBrains

Курс «Веб-технологии: уязвимости и безопасность»

HTTP

Синтаксис HTTP, методы запросов GET и POST, коды ответов, Cookie, HTTPS и сертификаты

Оглавление

Введение

Добро пожаловать на третий урок курса «Веб-технологии: уязвимость и безопасность». Мы успешно освоили основу веба и подробно разобрали устройство URL, теперь переходим к протоколу HTTP. На этом уроке мы обсудим, как общаются клиент и сервер:

1) Первый видеоурок будет посвящен базовому синтаксису протокола HTTP.
2) Во втором видео уделим внимание методам запроса HTTP и кодам ответа от сервера.
3) Потом мы обсудим заголовок Referer и выясним, какие уязвимости могут быть с ним связаны.
4) В четвертом видео обсудим, как устроены Cookies и для чего они нужны.
5) В конце урока мы узнаем, что такое HTTPS, сертификаты и для чего нужны центры сертификации.

К концу урока вы будете знать основной синтаксис HTTP, уметь составлять и отправлять HTTP-запросы, читать и понимать HTTP-ответ, поймете, что такое Cookies, зачем они нужны, научитесь проставлять их на различные домены, добавлять для них атрибуты, понимать, что они

делают. Также поймете, как работает и для чего нужен HTTPS, узнаете, как устроены центры выдачи сертификатов.

Видеоурок 1

Мы уже знаем, что, чтобы получить ресурсы сервера, пользователю необходимо сделать HTTP-запрос.

HTTP-запрос и метод GET

Посмотрим, как происходит этот запрос и как именно его можно выполнить.

Команда telnet

Откроем виртуальную машину с Ubuntu и воспользуемся новой для нас командой (программой) telnet. Программа telnet подключается к серверу по заданному порту и после этого мы можем посылать ему произвольные данные. Подключимся посредством telnet к машине по порту 80:

```
telnet localhost 80

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Как мы видим, подключение успешно, так как нет сообщений об ошибке. Чтобы прервать выполнение программ в терминале, нажмите Ctrl+C.

Заново подключимся к порту 80 и сделаем запрос `GET` к главной странице, укажем путь / и версию протокола `HTTP/1.1` . Затем нажмём Enter и введём `Host: localhost`:

```
telnet localhost 80

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

GET / HTTP/1.1
Host: localhost
```

Самое первое слово GET называется методом HTTP-запроса и говорит, что мы хотим сделать с этим запросом. Запрос методом GET указывает, что мы хотим получить определенный ресурс, то есть как бы говорит, верни мне этот ресурс.

Вы можете спросить: зачем еще раз указывать localhost ведь мы указали его, когда писали команду telnet. Но когда мы подключались командой telnet, то выбирали сервер, к которому нужно подключиться и где нужно открыть соединение. HTTP еще не начал работать в это время, и чтобы протокол и веб-сервер поняли, куда мы хотим попасть, нужно указать еще раз localhost.

Есть интересная особенность при подключении. Если вы не успели за определенное время набрать запрос и отправить его, сервер может закрыть соединение. Это происходит из-за того, что время ожидания при подключении к серверу ограничено, как правило, это 30 секунд или минута, по истечении которых незавершенное соединение закрывается. Эту настройку на сервере можно изменять, она нужна, чтобы не было такого, что клиент подключился, но ничего не шлет, а занимает ресурсы сервера.

Версия протокола HTTP/1.1 — самая распространенная версия, т. к. HTTP/1.0 и 0.9 уже устарели, а HTTP/2.0 еще не так широко распространена. В конце HTTP-запроса должно стоять два перевода строки: то есть нам нужно нажать Enter, чтобы один раз привести строку и потом еще раз Enter, чтобы сказать то, что это конец:

```
telnet localhost 80

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sun, 28 Apr 2019 17:14:57 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 31 Jan 2017 15:01:11 GMT
Connection: keep-alive
ETag: "5890a6b7-264"
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

```
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Вернулась HTML-страница Welcome to nginx!. Так как терминал не умеет отрисовывать HTML-страницы, она отображается просто как текст.

Структура запроса

В общем случае HTTP-запрос начинается со строки, которая состоит из следующих элементов:

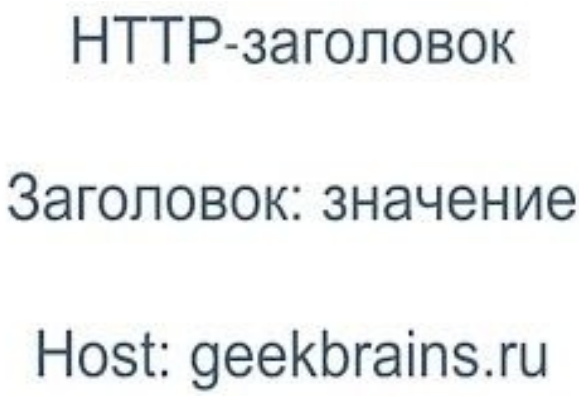
1. Метод HTTP-запроса. Например: `GET`, `POST`, `PUT`, `DELETE`.
2. Путь до ресурса и параметры. Например: `/index.html`, `/info.php?param=value`.
3. Версия протокола HTTP. Например: `HTTP/1.1`, `HTTP/1.0`.

Все три элемента обязательны.

HTTP-заголовки

Как браузеру передать дополнительную информацию серверу? Мы говорили про GET-параметры в строке URL-запроса, и вы можете предположить, что информацию возможно передать через них. Но те GET-параметры, как правило, используются, чтобы передавать информацию в скрипты и в приложение. А для передачи информации на сервер, существуют специальные HTTP-заголовки.

HTTP-заголовок — пара «Заголовок: Значение», то есть переменная и ее значение. Это почти так же, как и GET-параметр в URL, но, в отличие от него, здесь разделитель — не знак равенства (=), а двоеточие и пробел (**:**). Например, хост geekbrains.ru:



Здесь Host — заголовок, а его значение — geekbrains.ru.

Заголовок Host указывает, какой сайт сервера мы хотим получить. Дело в том, что некоторые сервера содержат несколько сайтов — это называется виртуальный хостинг. На больших высоконагруженных сайтах такое, как правило, не применяется, но если сайт небольшой, вполне можно на одном сервере разместить их несколько, и тогда обращаться к ним можно будет по разным заголовкам Host.

HTTP-ответ

HTTP-заголовки бывают и у запросов, и у ответов. Посмотрим на примере.

Команда curl

Откроем Ubuntu и воспользуемся командой curl. Если мы добавим флаг -i, curl выведет заголовки запросов и ответов. Сделаем запрос к https://geekbrains.ru:

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

Если тело ответа очень большое, как в данном случае, мы не сможем увидеть заголовки и нам придется долго листать терминал. Чтобы было проще, можем добавить команду `| less`, таким образом, мы передадим вывод команды `curl` в команду `less` и начнем с начала вывода команды:

```
curl -i https://geekbrains.ru | less

HTTP/1.1 200 OK
Server: nginx
Date: Sun, 28 Apr 2019 18:14:30 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 45388
Connection: keep-alive
Vary: Accept-Encoding
Vary: Accept-Encoding
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
ETag: W/"caab0787fc959aa5a7453443c1ea8475"
Cache-Control: max-age=0, private, must-revalidate
```

```
Set-Cookie: utm_source=geekbrains.ru; path=/; expires=Tue, 28 Apr 2020 18:14:30
-0000
Set-Cookie: utm_medium=referral; path=/; expires=Tue, 28 Apr 2020 18:14:30 -0000
Set-Cookie: entrance_path=%2F; path=/; expires=Sun, 09 Jun 2019 18:14:30 -0000
Set-Cookie: _app_session=7890452f234d887e0c423bd041ffe895; path=/; HttpOnly
X-Request-Id: a353af52beef00407827423b2c8bc8c1
X-Runtime: 0.033673
Strict-Transport-Security: max-age=15724800

<!DOCTYPE html><html><head><script>

..

</head>
</html>
```

Сервер ответил нам очень большим количеством заголовков. Давайте остановимся на некоторых из них. Например, заголовок server показывает, какой сервер обслужил нас — в данном случае Nginx. Заголовок Content-Type показывает, какой тип документа идет ниже, то есть что в теле ответа — в данном случае text/html.

Cookies

Тело ответа отделяется от заголовков переводом строки, все, что идет после этого, называется телом ответа. Если мы делаем HTTP-запрос, это называется телом запроса. Также мы видим заголовки, которые относятся к кэшу и Cookies. Кэш — механизм, который позволяет браузеру или серверу запомнить данные на определенный период времени, чтобы в дальнейшем их можно было быстро достать из памяти, не запрашивая снова и не тратя на это много времени.

Также мы можем увидеть заголовки запроса и ответа. Можем попросить curl выводить больше информации с флагом -v — это называется подробный режим verbose. Команда curl в данном случае выводит заголовки запроса и ответа в лог ошибок, а значит, необходимо перенаправить лог ошибок на стандартный вывод. Делается это вот так:

```
curl -v https://geekbrains.ru 2>&1 | less

* Rebuilt URL to: https://geekbrains.ru/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
^M  0      0     0      0     0      0       0       0 --:--:-- --:--:-- --:--:--
0*   Trying 5.61.239.22...
* Connected to geekbrains.ru (5.61.239.22) port 443 (#0)
* found 148 certificates in /etc/ssl/certs/ca-certificates.crt
* found 594 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* SSL connection using TLS1.2 / ECDHE_RSA_AES_128_GCM_SHA256
*        server certificate verification OK
```

```
*        server certificate status verification SKIPPED
*        common name: *.geekbrains.ru (matched)
*        server certificate expiration date OK
*        server certificate activation date OK
*        certificate public key: RSA
*        certificate version: #3
*                        subject:   OU=Domain   Control   Validated,OU=EssentialSSL
Wildcard,CN=*.geekbrains.ru
*        start date: Mon, 17 Dec 2018 00:00:00 GMT
*        expire date: Wed, 16 Dec 2020 23:59:59 GMT
*                    issuer:   C=GB,ST=Greater   Manchester,L=Salford,O=COMODO   CA
Limited,CN=COMODO RSA Domain Validation Secure Server CA
*        compression: NULL
* ALPN, server accepted to use http/1.1
> GET / HTTP/1.1
> Host: geekbrains.ru
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sun, 28 Apr 2019 18:43:12 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 45388
< Connection: keep-alive
< Vary: Accept-Encoding
< Vary: Accept-Encoding
< X-Frame-Options: SAMEORIGIN
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< ETag: W/"132b1c1d9c2bb54be3f3c9cce008cb33"
< Cache-Control: max-age=0, private, must-revalidate
<  Set-Cookie:  utm_source=geekbrains.ru;  path=/;  expires=Tue,  28  Apr  2020
18:43:12 -0000
< Set-Cookie: utm_medium=referral; path=/; expires=Tue, 28 Apr 2020 18:43:12
-0000
< Set-Cookie: entrance_path=%2F; path=/; expires=Sun, 09 Jun 2019 18:43:12 -0000
< Set-Cookie: _app_session=1fafbbf0f4b87286e83dae835b5781de; path=/; HttpOnly
< X-Request-Id: 8255b31e6db40f95f565f2d313582c46
< X-Runtime: 0.032352
< Strict-Transport-Security: max-age=15724800
<
{ [3276 bytes data]
^M   7 45388     7   3276     0      0   13101       0  0:00:03 --:--:--  0:00:03
13104<!DOCTYPE html><html><head><script>

..

</head>
</html>
```

Мы видим, что есть наш запрос GET, Host и заголовок User-Agent. User-Agent — заголовок, означающий, что данный запрос был сделан из какого-то конкретного браузера или браузера на телефоне или, как в данном случае, программой curl. Этот заголовок используется, чтобы сайт понимал, отдать вам мобильную или десктопную версию, то есть ту, которую поймет запросившая ее программа. Чтобы вы зашли, например, на e.mail.ru с телефона и не мучились с крупным интерфейсом для компьютера, а получили сразу телефонный. Разработчики не должны доверять этому заголовку, т. к. хакер его полностью контролирует. Полагаться на него стоит только в тех случаях, когда это не влияет на безопасность.

Также мы видим уже знакомые заголовки ответа.

Итоги

Давайте подведем итоги. В этом видеоуроке вы узнали:

1. Как работает протокол HTTP и научились делать простейший HTTP-запрос с помощью команды telnet.
2. Что такое GET-запрос, научились делать простейший HTTP-запрос методом GET.
3. Примеры HTTP-заголовков, увидели их в запросе и в ответе.
4. Что такое тело запроса и ответа.

Видеоурок 2

В этом уроке мы рассмотрим:

1. Что такое HTTP-метод и зачем он нужен.
2. Использование основных HTTP-методов.
3. Основные коды ответа веб-сервера и их значение.

Метод POST

Мы уже знаем некоторые способы отправки данных на сервер, например, через параметры GET-запроса или HTTP-заголовки. Но, как правило, данные в вебе отправляют через так называемые параметры POST-запроса. Разберемся, что это такое.

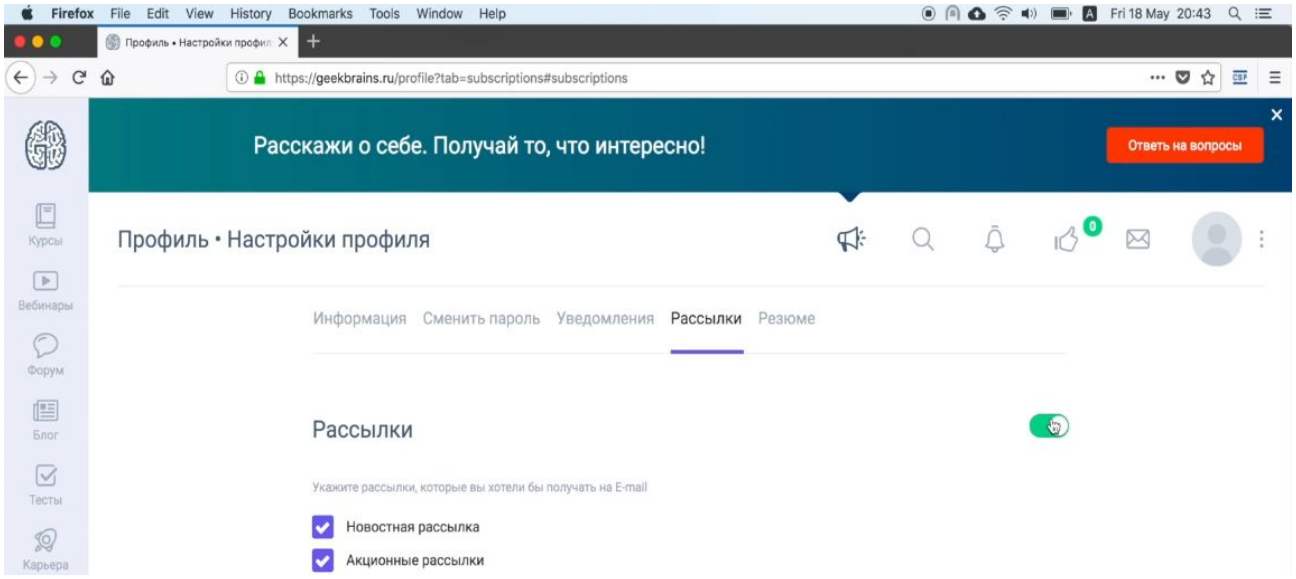
Допустим, мы хотим передать логин и пароль на сервер, например, через GET-запрос:

```
https://geekbrains.ru?login=vasya&password=superparol
```

В этом случае логин и пароль останутся в логах промежуточных и даже наших серверов. Это плохо тем, что любой, кто имеет доступ к логам или истории браузера, увидит логин и пароль и

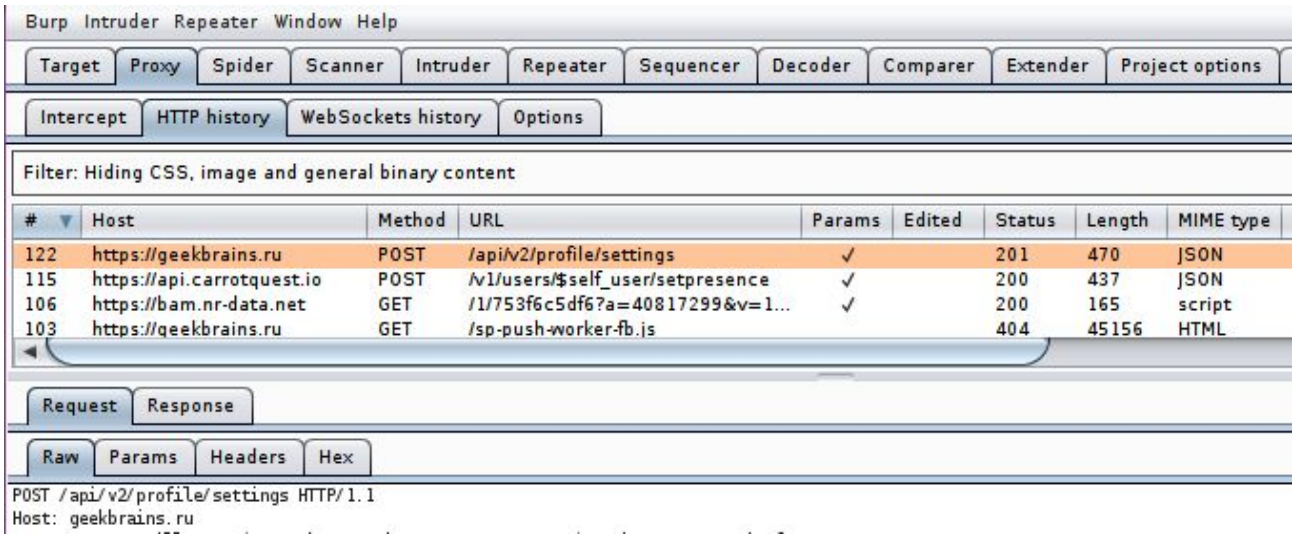
соответственно, сможет украсть аккаунт у пользователя. Именно поэтому важные данные нельзя передавать в параметрах GET-запроса. Специально для решения этой проблемы придумали POST-запрос, то есть HTTP-запрос методом POST.

Посмотрим на примере, как выглядит POST-запрос. Откроем браузер Firefox и страницу https://geekbrains.ru/profile?tab=subscriptions#subscription:



Так мы перейдем в настройки профиля («Профиль» — «Редактировать профиль» — «Рассылки»). Эта страница выбрана для примера потому, что здесь ничего не нужно заполнять, следует включать и выключать галочку; будет видно, что отправляется POST-запрос.

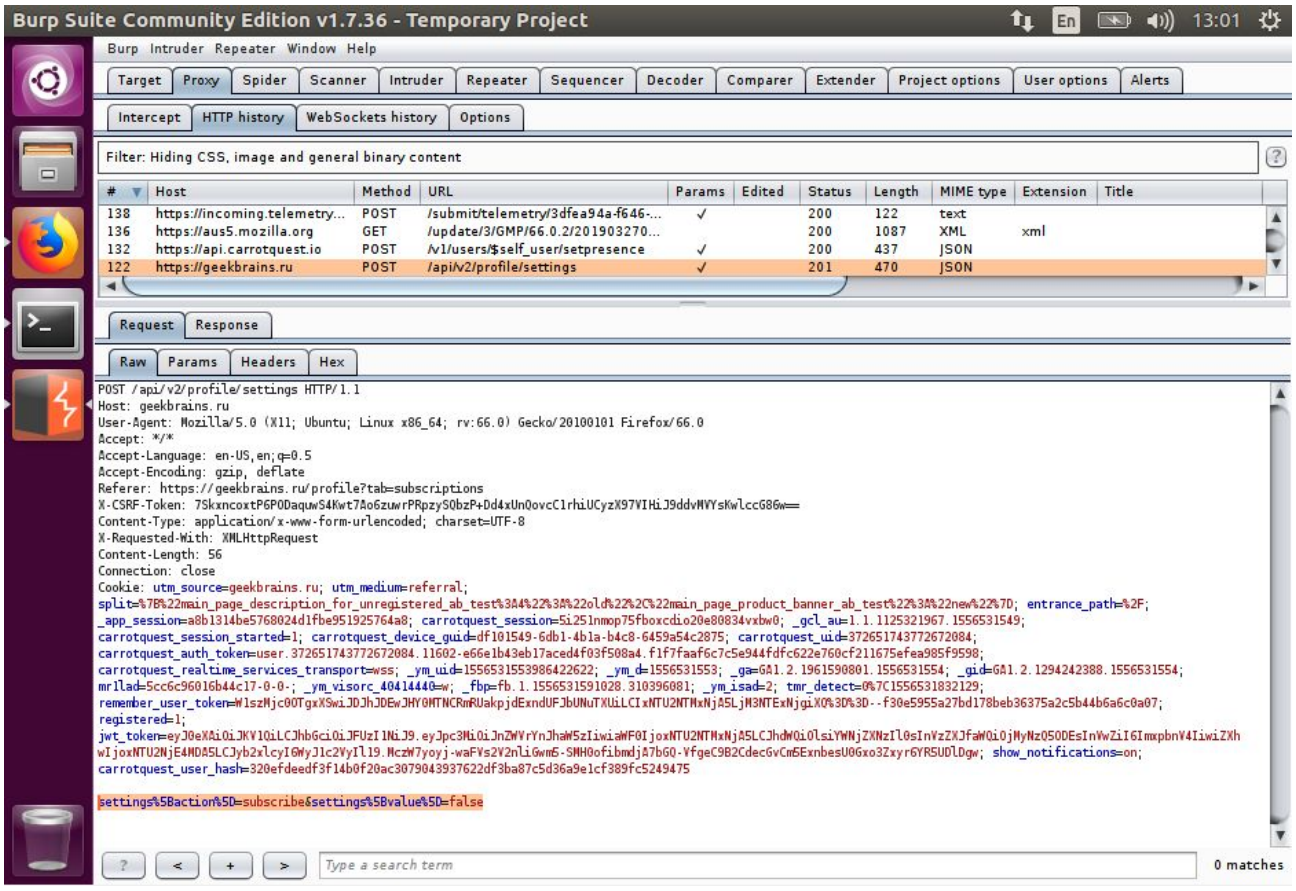
Попробуем их отключить и затем откроем Burp:



Третья колонка показывает метод запроса, мы видим GET и POST. С первым методом, GET, мы уже знакомы — это запрос на получение страницы. Запрос POST — запрос на отправку данных, то есть тот, который мы разберем подробнее.

Параметры POST-запроса

Откроем в списке запрос POST к URL /api/v2/profile/settings:



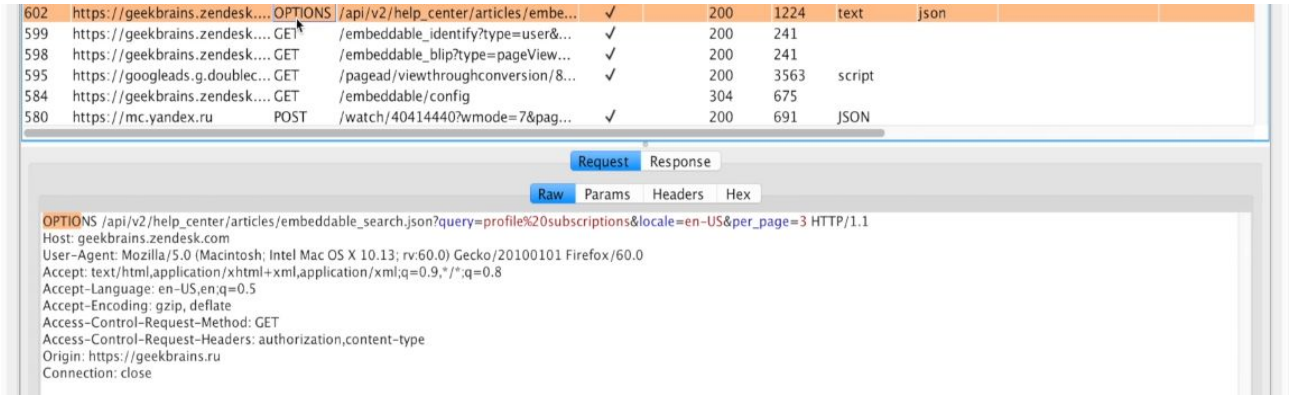
В запросе нет GET-параметров, нет знака вопроса (?) и данных после него. Но зато есть так называемые POST-параметры — они идут в самом низу запроса и отделяются двумя символами переноса строки от заголовков. Тело POST-запроса очень похоже на параметры GET-запроса: также используется знак ampersand (&), чтобы соединять различные элементы.

Отдельные элементы POST-параметров, как правило, пишутся в формате «имя=значение». Но здесь может быть и другой формат: закодированный JSON, base64 и все что угодно. Подробнее о JSON мы узнаем в следующих видео, когда будем разбирать JavaScript.

Методы OPTIONS, PUT, DELETE

Метод POST нужен, чтобы отправлять данные на сервер. Особенно это касается данных, которые могут быть значимы для пользователя — логин, пароль, текст письма и тому подобное.

Также существуют другие методы HTTP-запросов, например OPTIONS.



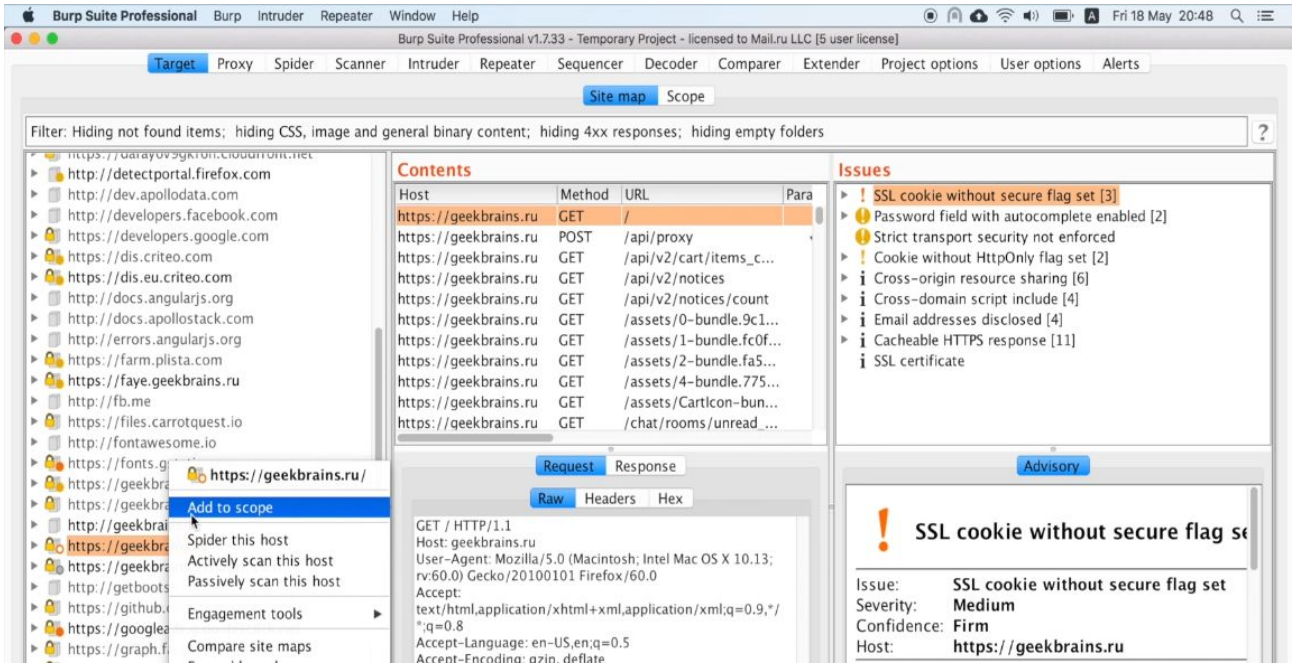
Здесь мы видим пример такого запроса.

Также существуют методы PUT и DELETE — они применяются не очень часто. PUT нужен, чтобы загрузить данные на сервер, то есть он используется вместо POST. А метод DELETE нужен, чтобы удалять данные, загруженные методом PUT. Чаще всего используются GET- и POST-запросы. Запрос OPTIONS используется в некоторых протоколах, чтобы они могли корректно работать.

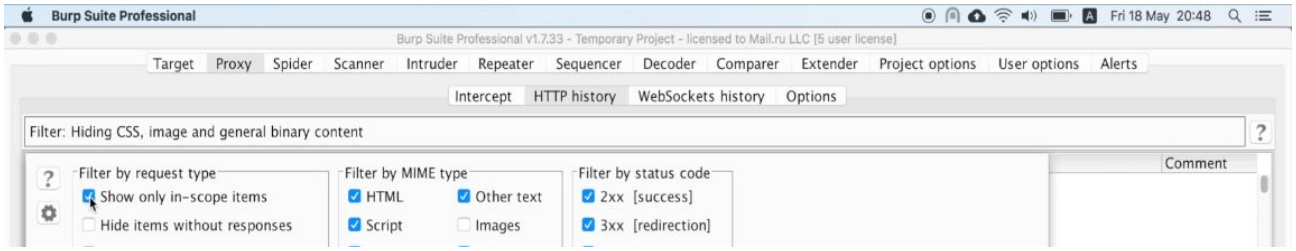
Коды ответа на HTTP-запрос

Чтобы сервер сообщил браузеру об ошибке (например, браузер запросил HTML-страницу, которой нет на сервере), существуют коды ответов.

Сделаем запрос к https://geekbrains.ru и отфильтруем в Burp лишние через функционал Target->Scope, — мы видим множество разных запросов в Burp Proxy и искать geekbrains неудобно. Добавим в Scope https://geekbrains.ru:

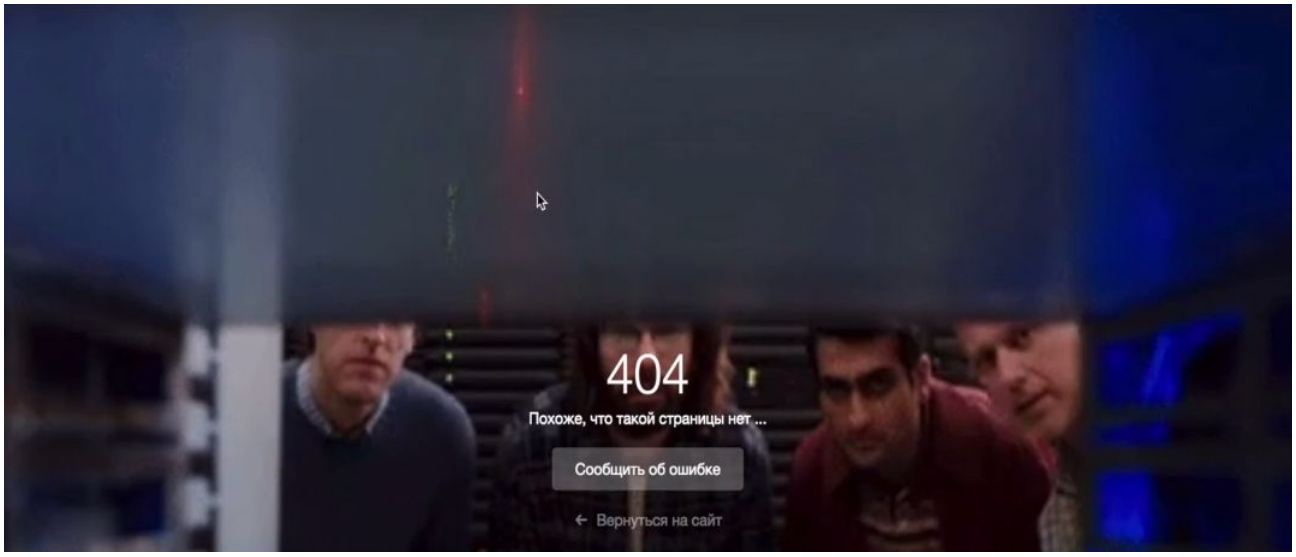


Возвращаемся в раздел Proxy и просим показать только запросы из Scope:

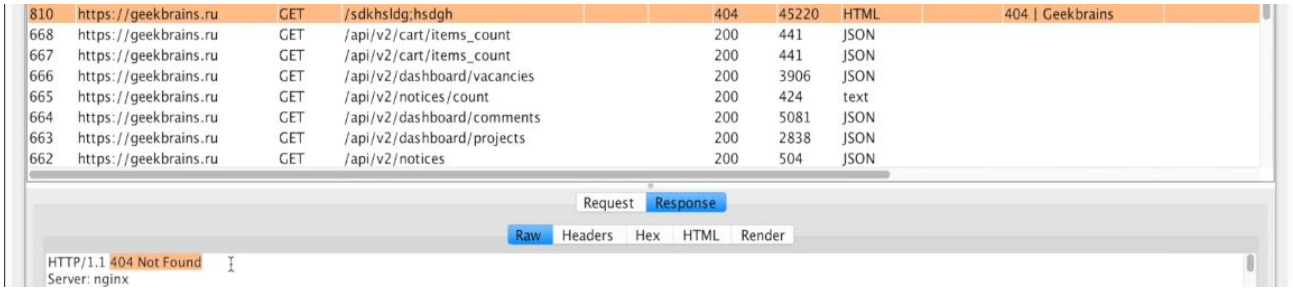


Ищем запрос главной страницы и видим, что в ответе был возвращен код **200 OK**. Это означает, что запрос был обработан корректно и завершён успешно. Помните, что не всегда статус ответа совпадает с реальным положением вещей. Например, сервер может ответить **200 OK**, но в теле ответа написать, что произошла ошибка — сам по себе запрос был обработан корректно, но в скрипте — неверно. Скрипт в теле запроса конкретизирует ошибку, но на уровне протокола HTTP запрос был обработан успешно.

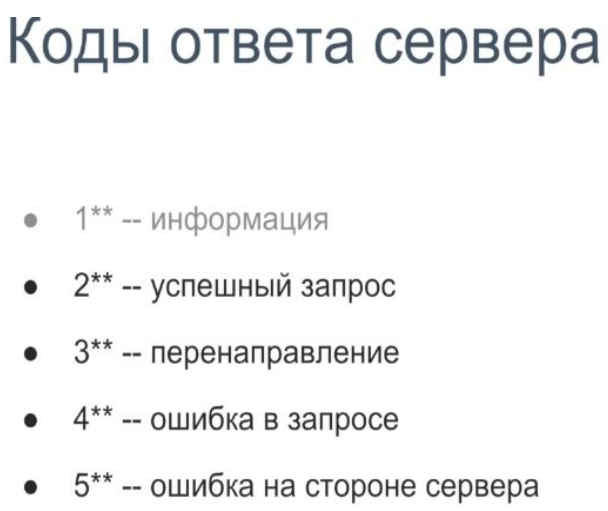
Посмотрим на код статуса запроса: если мы введём невалидный URL, которого нет на сервере, например, https://geekbrains.ru/sdnjdjjncjn, увидим:



Страница пишет код ошибки 404. Вернёмся в Burp — в ответе видим 404 Not Found:



Статус 404 означает, что ресурс не найден на сервере. Есть и другие коды ответов сервера. Например, 301 и 302 обозначают перенаправление — ресурс находится в другом месте, на другом URL. А 500 — ошибка на стороне сервера.



Коды ответов разделены на 5 классов: это ошибки, которые начинаются с единицы, то есть 100, 101, 102, 150 и так далее с 2, 3, 4 и 5.

Ошибки, которые начинаются с единицы — информационные сообщения, например, что поменялся протокол. Коды ответа, которые начинаются с двойки, означают, что запрос был успешно выполнен. Коды, начинающиеся с тройки, означают перенаправление запросов на другой ресурс или, например, в кэш. Коды 400 обозначают ошибки со стороны клиента — например, ошибку в запросе и они подробно указывают на ошибку в теле ответа на запрос. Коды, которые начинаются с пятерки, означают ошибку на стороне сервера.

Итоги

В этом видео мы узнали:

1. Что такое HTTP-метод, зачем он нужен и как применяется, какие основные HTTP-методы существуют.
2. Как применять HTTP-метод POST. Узнали, что такое POST-параметры и чем они лучше GET-параметров, почему они используются для передачи важной информации.
3. Что означают коды ответа от сервера.

Видеоурок 3

В этом видео мы:

1. Познакомимся с заголовком Referer, узнаем, как он работает.
2. Разберем проблемы безопасности, связанные с этим заголовком.
3. Узнаем, как можно отключить данный заголовок.

Заголовок Referer

Чтобы сервер мог узнать, сам ли пользователь ввел https://geekbrains.ru в строку браузера или он нажал на ссылку https://geekbrains.ru на другом портале, необходим заголовок Referer.

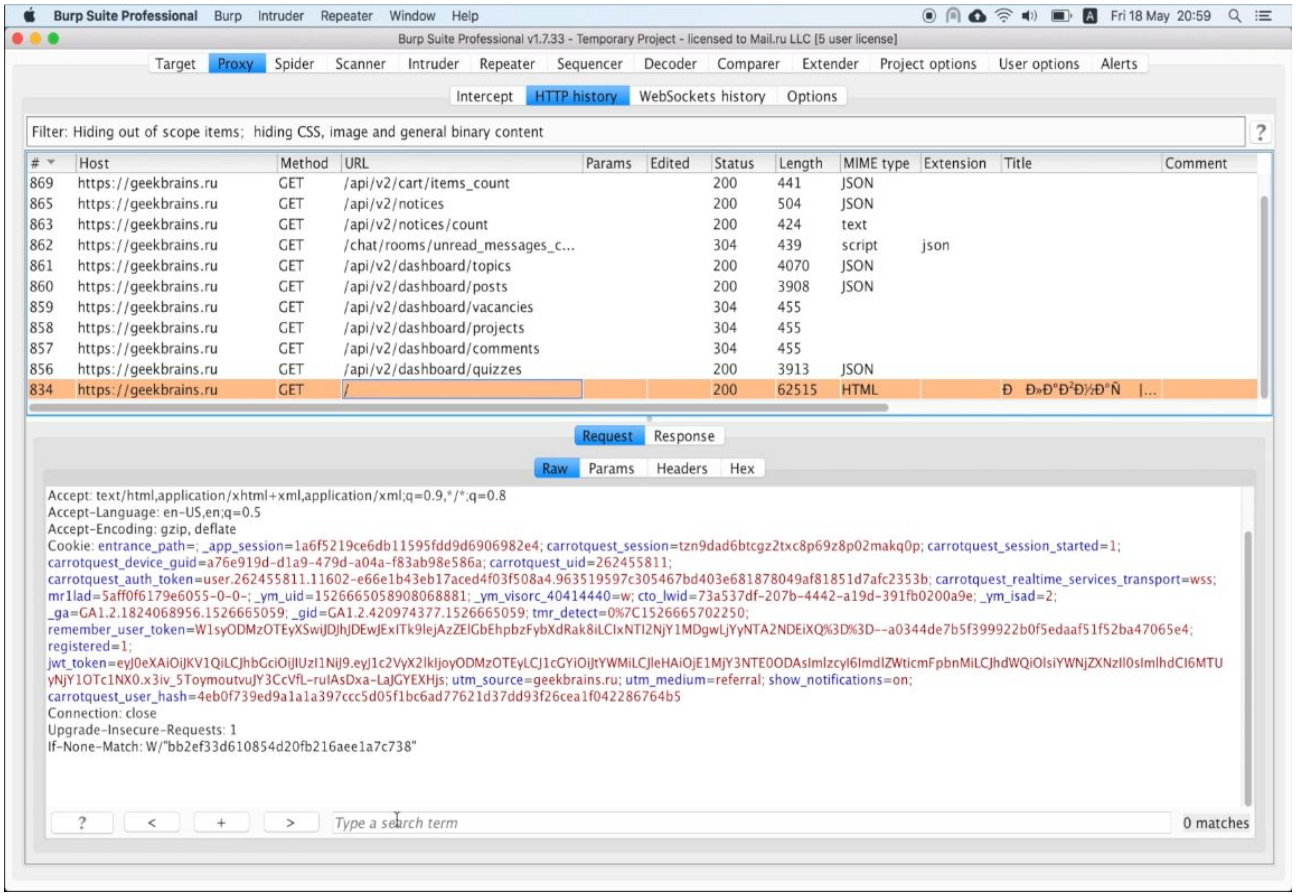
Этот заголовок устроен так, что он отправляется практически с каждым HTTP-запросом, то есть все современные браузеры, когда делают запрос, отправляют заголовок Referer. Исключения — случаи, когда заголовок Referer специально отключен, либо когда его просто не может быть. Например, если вы только открыли новую вкладку в браузере и вбили туда https://geekbrains.ru, у ссылки не будет заголовка Referer. Важно помнить, что Referer отправляется практически с каждым HTTP-запросом.

Все это необходимо для статистики, рекламы и других подобных вещей.

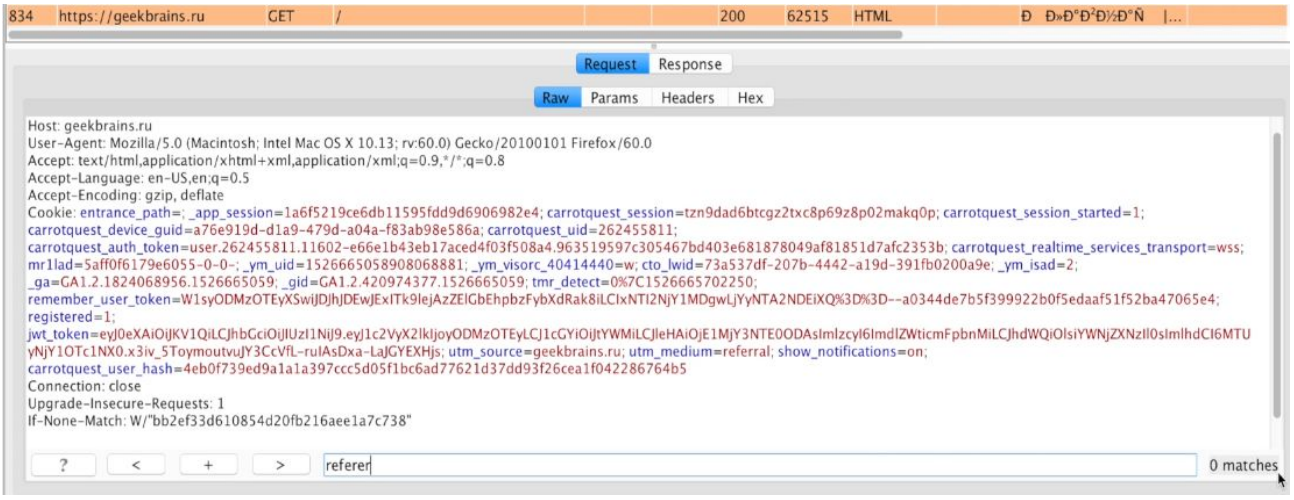
Посмотрим, как отправляется заголовок Referer, на примере.

Практика

Откроем в Firefox сайт https://geekbrains.ru, затем вернемся в Burp и посмотрим на первый запрос:

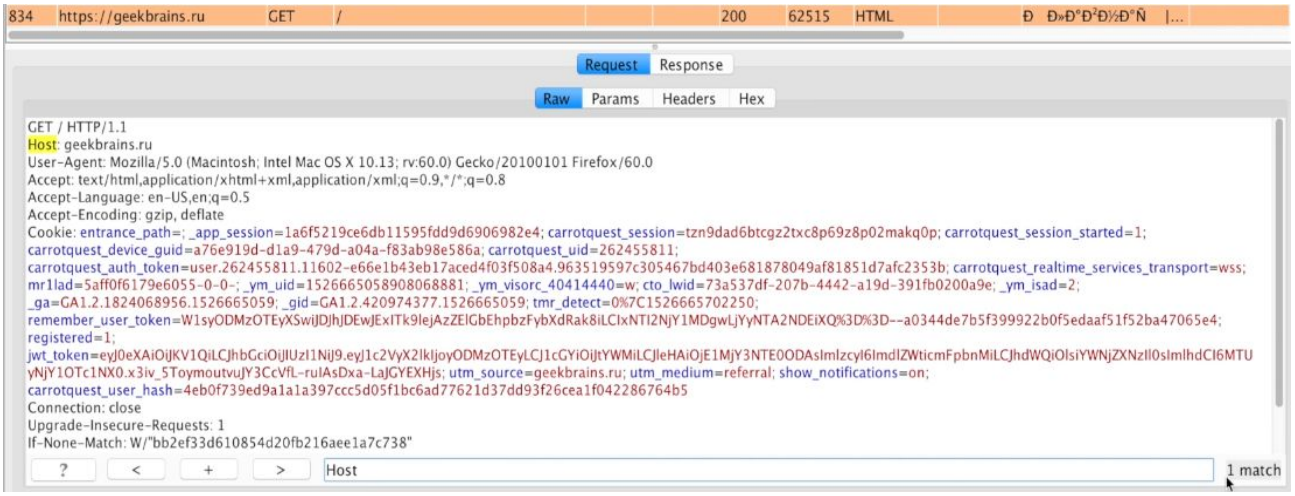


Мы видим, что заголовка Referer нет, и убедимся в этом, если попробуем сделать поиск по запросу. Это очень удобный элемент Burp, потому что мы можем не искать глазами какую-то сущность, а ввести ее в поиск и он ее подсветит:



Как видим, Referer нет — у нас 0 совпадений.

Чтобы проверить, что поиск действительно работает, попробуем найти Host:

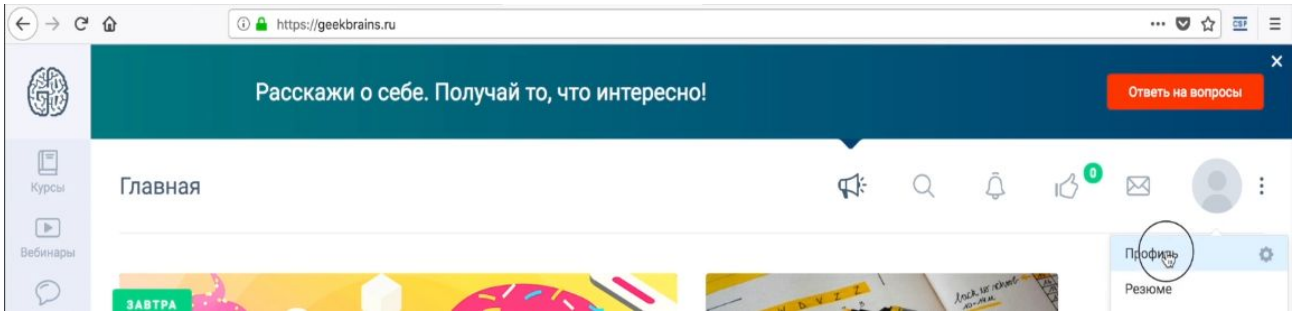


Как мы видим есть одно совпадение и оно подсвечивается желтым - Host.

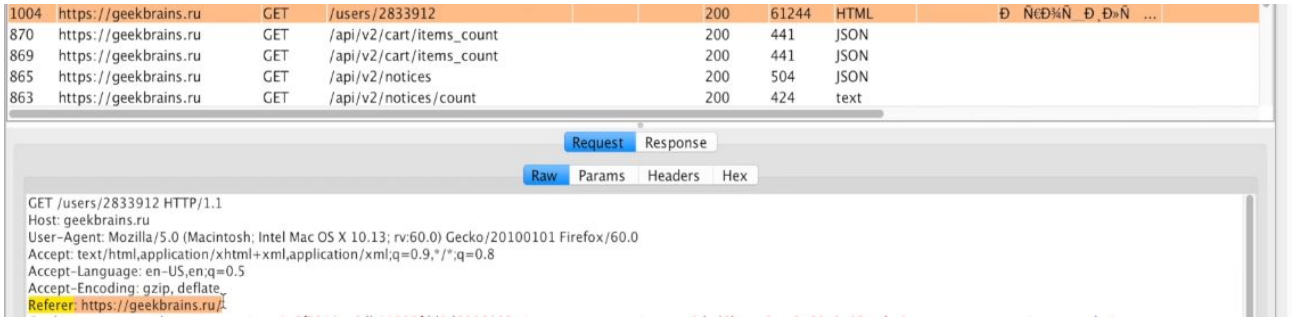
Теперь посмотрим на следующие HTTP-запросы:

Как мы видим, здесь заголовок Referer есть и он очевидным образом имеет значение https://geekbrains.ru, потому что запрос был сделан уже с GeekBrains автоматически.

Даже в автоматических запросах, то есть тех, которые подгрузились за счёт того, что ссылка на эти ресурсы была размещена на странице, поле Referer все равно проставляется. Убедимся, что Referer проставляется, когда мы с вами переходим куда-то:



Откроем Firefox и нажмем на профиль:

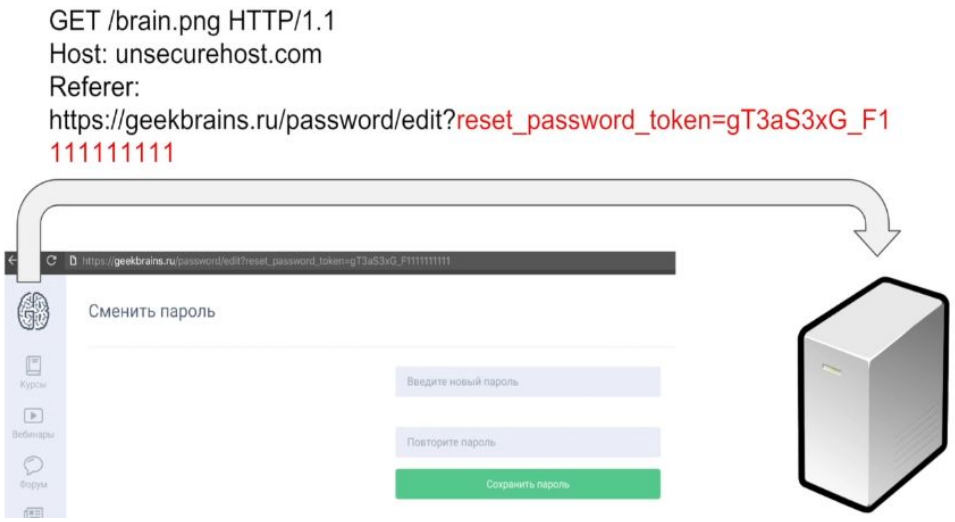


Кроме профиля, видим, что здесь Referer — тоже https://geekbrains.ru. Все последующие запросы будут уже с Referer https://geekbrains.ru/users/2833912 (ID юзера).

Теперь разберем реальный случай, когда утечка заголовка Referer повлекла за собой проблему и уязвимости информационной безопасности.

Referer и проблемы безопасности

Представим функционал смены пароля. Когда мы меняем пароль на сайте, мы получаем ссылку на почту, и, как правило, переходим по ней и меняем пароль. Теперь представим, что логотип GeekBrains загружается с домена unsecurehost.com — уже его название говорит, что он небезопасен:



Как мы знаем, с каждым HTTP-запросом, даже запросом на картинку, будет отправляться заголовок Referer. Мы видим, что текущий URL включает в себя ссылку на сброс пароля:

Таким образом, в Referer будет полный URL и Token, который необходим для сброса пароля. Это, в свою очередь, означает, что если злоумышленник взломает unsecurehost.com, он сможет собирать такие URL, переходить по ним, менять пароль пользователям и угонять их аккаунты.

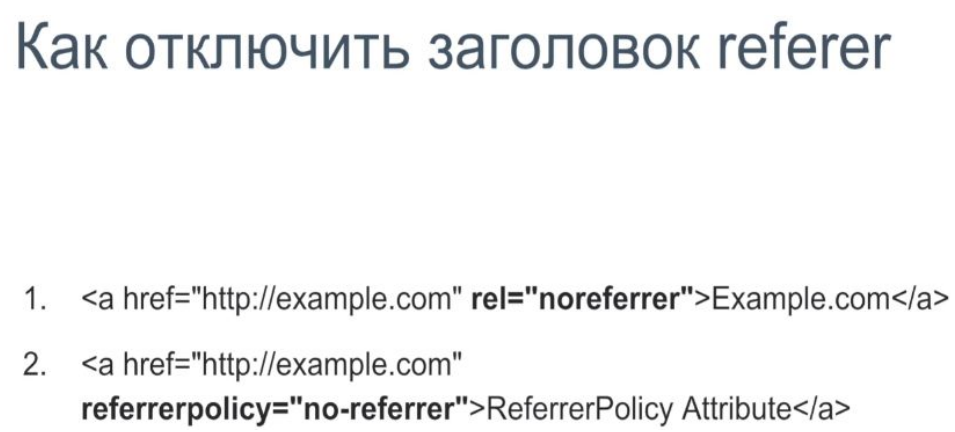
Это очень критичная уязвимость, на самом деле на GeekBrains такого нет, ситуация специально смоделирована, чтобы было понятно, чем опасен заголовок Referer. Но точно такая же уязвимость с другим сайтом была в реальности. Будьте внимательны и следите, куда уходит заголовок Referer. Хороший пример того, что заголовок Referer не всегда полезен, — чтение письма.

Как правило, у письма есть идентификатор, по которому к нему можно постучаться. Если дополнительной защиты нет, то, когда пользователь читает письмо, а в нем есть картинка от злоумышленника, она подгружается с URL, в котором есть идентификатор письма. Таким образом, злоумышленник получает как минимум URL письма, который может использовать в дальнейших атаках. Но об этом мы поговорим подробнее в следующих курсах, когда будем разбирать практические примеры атак.

Как отключить Referer

Заголовок Referer можно отключить. Это полезно, когда мы не хотим, чтобы кто-то узнал, откуда пришел ресурс, или не хотим показывать URL.

Чтобы отключить заголовок Referer, можно воспользоваться специальными HTML-атрибутами rel=noreferrer или referrerpolicy=no-referer:



Полный список можно посмотреть в интернете, смысл в том, что такие атрибуты существуют. Например, указанные атрибуты отключают заголовок Referer для именно этой ссылки и для этого тега .

Подробнее про теги HTML мы узнаем в следующем уроке, сейчас важно понимать, что, когда мы перейдем по ссылке http://example.com, Referer не передастся. Если мы перейдем на любую другую ссылку на этой странице, Referer уйдет, потому что атрибут noreferrer применяется только для этой ссылки.

Также существует специальный мета-тег noreferrer, который может отключить Referer на всей странице целиком.

Итоги

Подведем итоги. Вспомним, что заголовок Referer уходит почти с каждым HTTP-запросом к веб-серверу. Даже если вы разместите картинку на странице с восстановлением пароля, с ней утечет Referer; чаще всего ссылки на восстановление пароля содержат в URL уникальный идентификатор и этот идентификатор окажется у злоумышленника. Поэтому нужно следить, чтобы на критически важных страницах не было контента других сайтов и оценивать риски, чтобы сделать безопасно.

В этом видеоуроке вы узнали:

1. Зачем нужен заголовок Referer, как он работает.
2. Какие проблемы безопасности связаны с заголовком Referer.
3. Как можно отключить Referer на всей странице или на конкретные ссылки.

Видеоурок 4

Добро пожаловать в третий урок курса, где мы узнаем:

1) Что такое Cookie, зачем они нужны.
2) Как работают Cookie.

3) Как проставляются Cookie, с каких на какие домены их можно проставлять.

4) Какие атрибуты бывают у Cookie, зачем каждый атрибут нужен, и разберемся с наиболее важными атрибутами.

Cookies

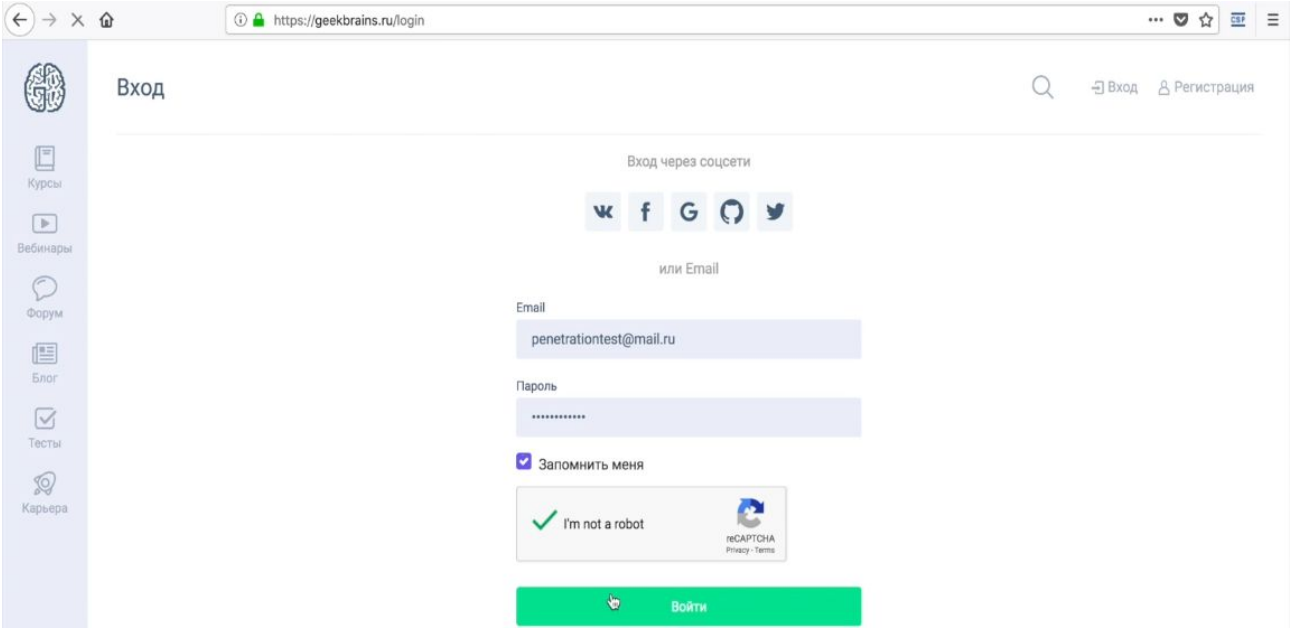
Как серверу проверить, кто пришел? Например, на GeekBrains вы вводите логин и пароль, и сервер понимает, что пришли именно вы, а не кто-то другой. Процесс подтверждения, что это действительно вы, называется аутентификация. Чаще всего аутентификация происходит по логину и паролю, но есть и другие возможности для этого.

Представим, что после каждого перехода на сайте GeekBrains нужно каждый раз вводить логин и пароль, чтобы представиться серверу. Согласитесь это неудобно. Именно поэтому в браузере существует механизм Cookie. В этой ситуации Cookie можно использовать в качестве идентификатора вашей сессии. Это означает, что мы отправляем логин и пароль серверу, а тот на своей стороне создает случайную строку, сохраняет ее и отправляет обратно. По выданной строке сервер может понять, кто к нему обращается, то есть, когда пользователь идет на сервер, он отправляет эту строку в HTTP-запросе. А затем сервер сверяет, та ли это строка, которую он выдал. Если все совпадает, то сервер идентифицирует пользователя.

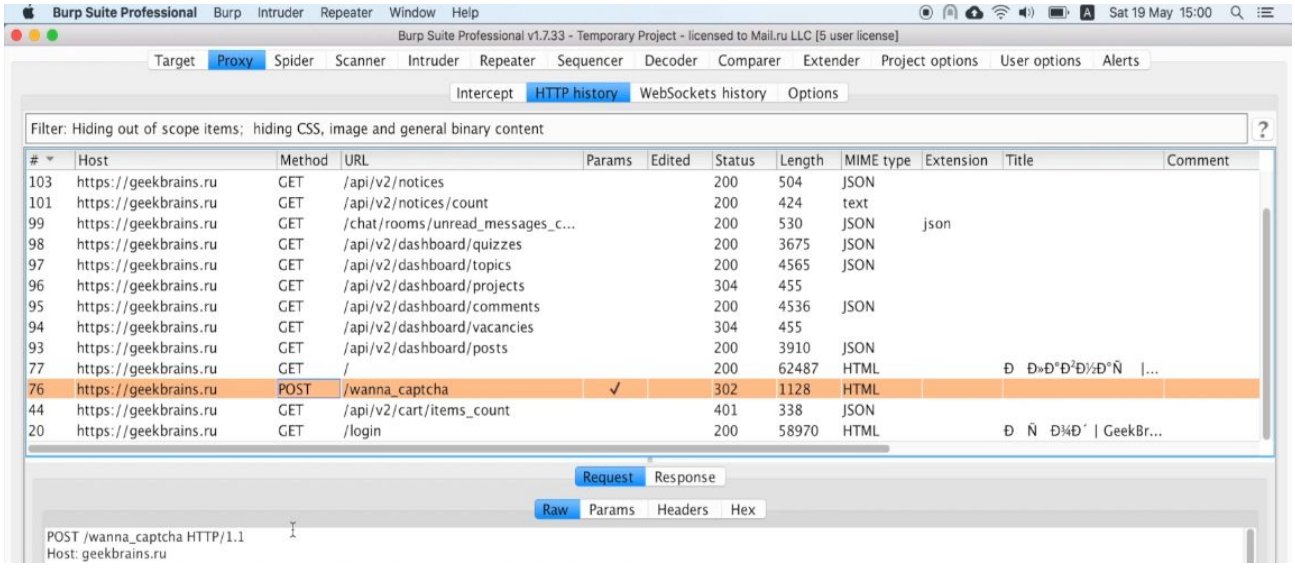
В общем случае Cookie (куки) нужны, чтобы сохранить данные на стороне клиента и его браузера. Важно, что Cookie отправляются с каждым запросом. Если мы поставили Cookie на https://geekbrains.ru, она будет отправляться с каждым запросом к https://geekbrains.ru.

Практика

Давайте рассмотрим Cookie на примере. Откроем Firefox и войдем в аккаунт на GeekBrains:

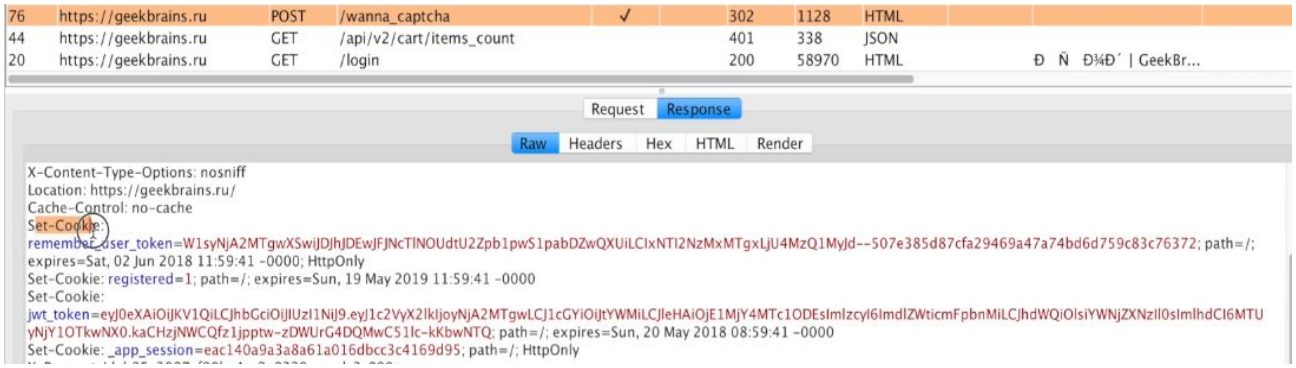


Теперь откроем Burp и настроим Scope на сайт https://geekbrains.ru, затем найдем запрос на вход:



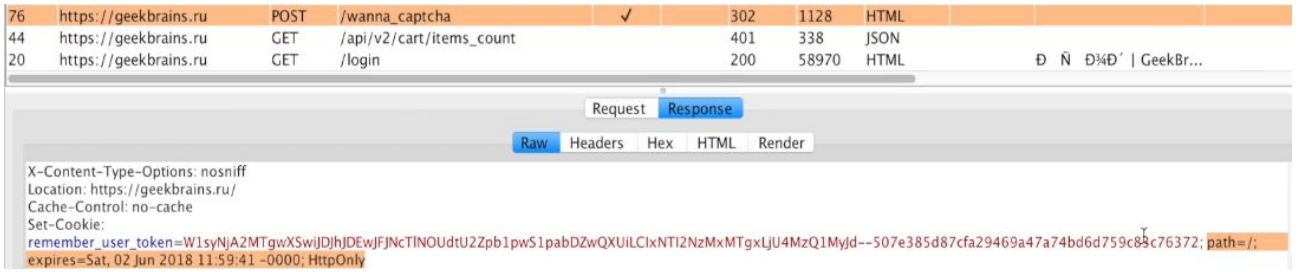
Как мы видим, это метод POST для отправки логина и пароля в теле запроса.

Посмотрим ответ от сервера:



Мы видим, что сервер проставляет Cookie с помощью заголовка Set-Cookie.

Собственно Cookie в браузере выглядят как пары «ключ-значение» и очень похожи на GET-параметры:

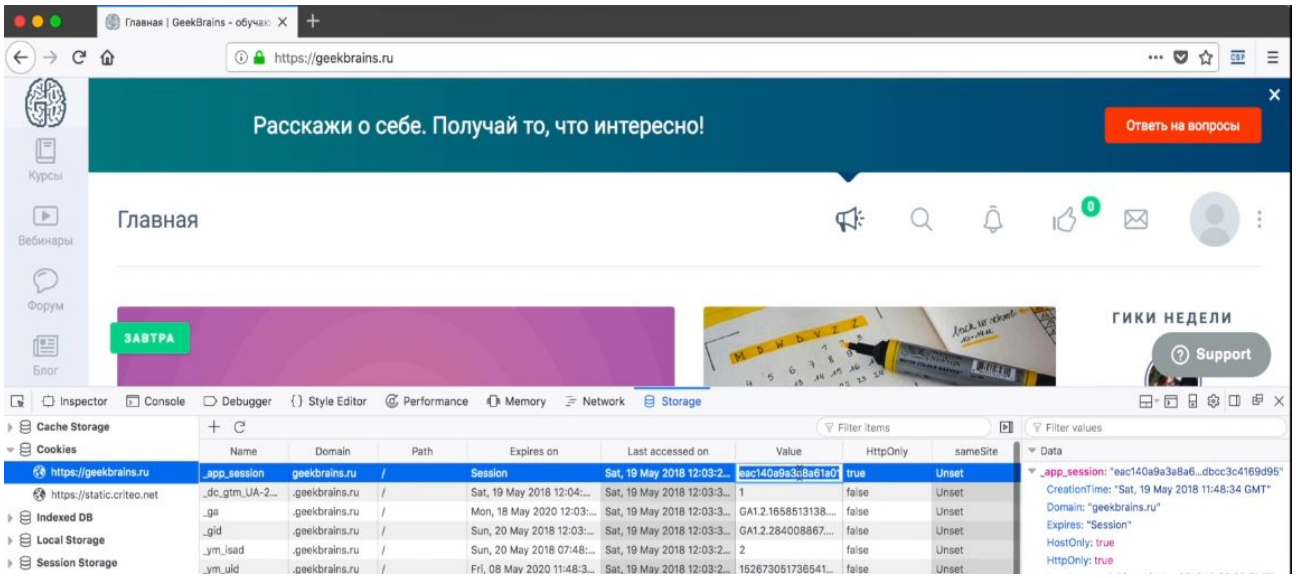


У Cookie есть различные атрибуты, но о них мы поговорим чуть позже.

В данном случае, после запроса на логин-пароль одна из Cookie, которую выставил сервер — сессионная, то есть та, которая подтвердит серверу, что это снова мы к нему пришли, а не какой-то другой пользователь.

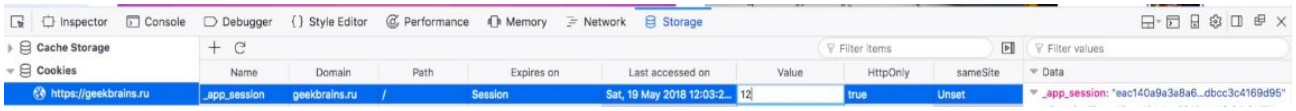
Сессионная Cookie эквивалентна логину и паролю. Если мы знаем логин и пароль, то мы можем получить сессионную Cookie и доступ к аккаунту пользователя. И наоборот, если мы знаем сессионную Cookie, мы можем получить доступ к аккаунту пользователя без знания его логина и пароля.

Также Cookie можно увидеть в браузере и там же их, если нужно, проставить. Посмотрим где это находится в Firefox. Жмем правой кнопкой мыши и открываем инструменты разработчика, затем выбираем вкладку Storage (хранилище):



Мы видим, что здесь сохраняются все Cookie для домена https://geekbrains.ru.

Если мы поменяем значение Cookie _app_session и попробуем перезагрузить страницу, то увидим не аккаунт пользователя, а страницу входа, потому что сессии 12 не существует на сервере:



Точно так же, если мы введём в Cookie _app_session, правильное значение сессии другого пользователя, то войдём под его аккаунтом на сайт.

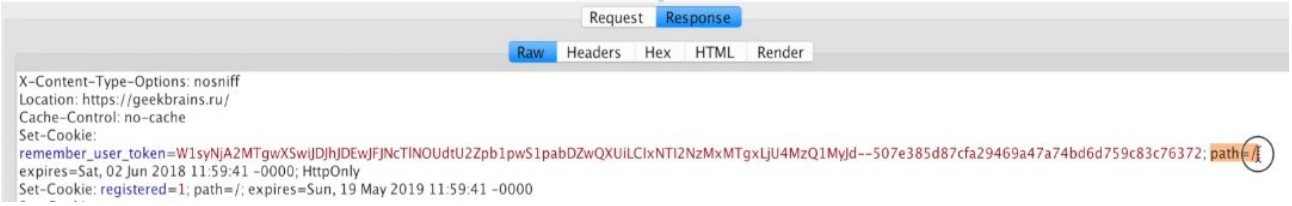
Атрибуты Cookie

Рассмотрим атрибуты Cookie — некоторую дополнительную информацию. И она может включать в себя, например, время жизни Cookie, на какой домен они проставляются, можно ли отправлять именно их через незащищенные соединения.

Атрибуты cookie

- Expires -- время жизни куки
- Domain -- на какой домен проставляется кука
- Secure -- отправлять куки только через защищенное соединение
- ...

Мы можем посмотреть все эти флаги в Burp. Например, remember_user_token располагается по пути path=/, что значит, что она поставлена на весь домен https://geekbrains.ru:



Эта Cookie будет отправлена по любому пути на этом домене.

Дальше мы видим, что эта Cookie будет действовать до 2 июня 2018 года:

После этого времени эта Cookie работать уже не будет.

Еще мы видим, что она HttpOnly — то есть JavaScript не сможет получить её через свои методы:

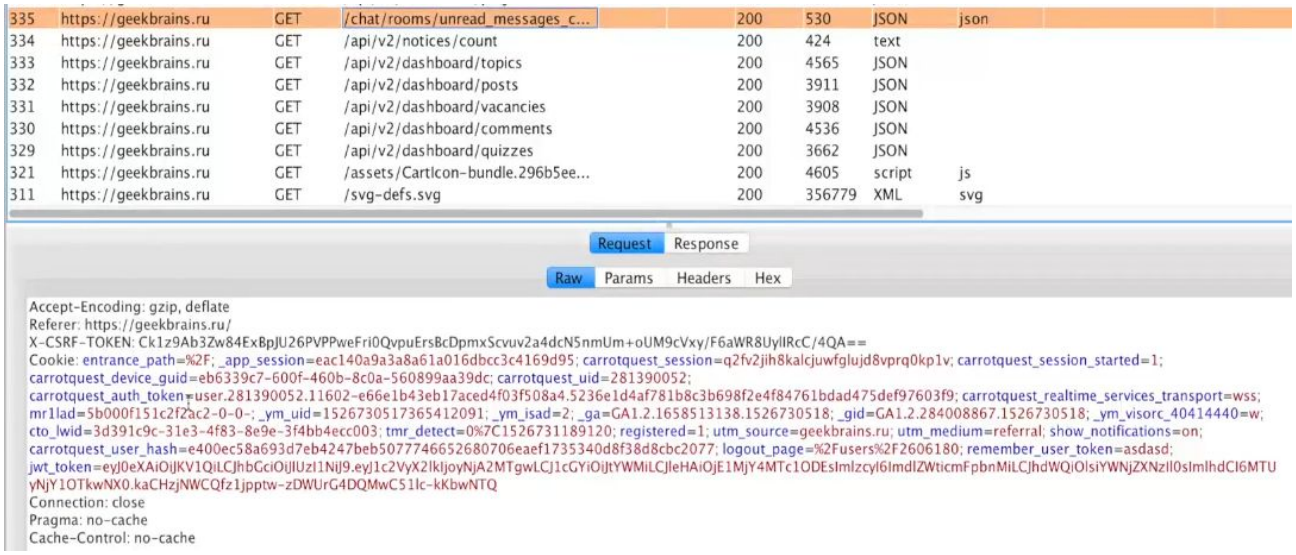
Подробнее о JavaScript и о том, зачем нужна HttpOnly, мы узнаем на следующем уроке.

Проставление Cookie

Еще один очень важный момент, связанный с Cookie — любой поддомен сайта сможет выставлять их на свой родительский домен.

Если, например, домен https://app.geekbrains.ru будет взломан, он сможет проставлять Cookie на https://geekbrains.ru и таким образом, например, подменять сессию пользователя или делать атаку Session Fixation (см. ссылки к уроку).

Cookie уходят вместе со всеми запросами к домену. Рассмотрим это на примере. Откроем Firefox: у нас есть некоторый набор Cookie для домена https://geekbrains.ru и каждый запрос к https://geekbrains.ru содержит их все:



Вы убедитесь в этом, если откроете свои запросы в Burp и посмотрите их.

Cookie и вопросы безопасности

Рассмотрим один момент, связанный с безопасностью при использовании Cookie.

После того как вы нажимаете кнопку выхода, все Cookie этого сайта должны становиться недействительными. Так они станут недействительными еще до того, как выйдет время их жизни. Все это делается на стороне сервера.

Подставив сессионную куку в запрос к серверу мы можем получить доступ к аккаунту пользователя и выполнять все действия от его имени, и при этом сервер будет считать, что мы и есть пользователь, чью сессионную Cookie мы поставили.

Именно поэтому алгоритмы генерации Cookie должны быть очень надежными, то есть генерировать Cookie, которые будет тяжело подобрать. Например, если сессионная Cookie будет состоять из числа 100 или 200, или даже 10 миллионов, ее будет легко подобрать, потому что компьютер сможет быстро перебрать такой диапазон чисел.

Подробнее о критериях, которым должен будет удовлетворять токен пользовательской сессии, мы поговорим в курсе по криптографии.

Итоги

В этом видео мы узнали:

1) Что такое Cookie, зачем они нам нужны.

2) Как работают Cookie, с помощью каких заголовков сервер их проставляет.

3) Куда можно проставлять Cookie, когда они отправляются на сервер, с каких на какие домены их можно проставлять.

4) Основные атрибуты Cookie, связанные с безопасностью и с их использованием.

Видеоурок 5

На уроке мы узнаем:

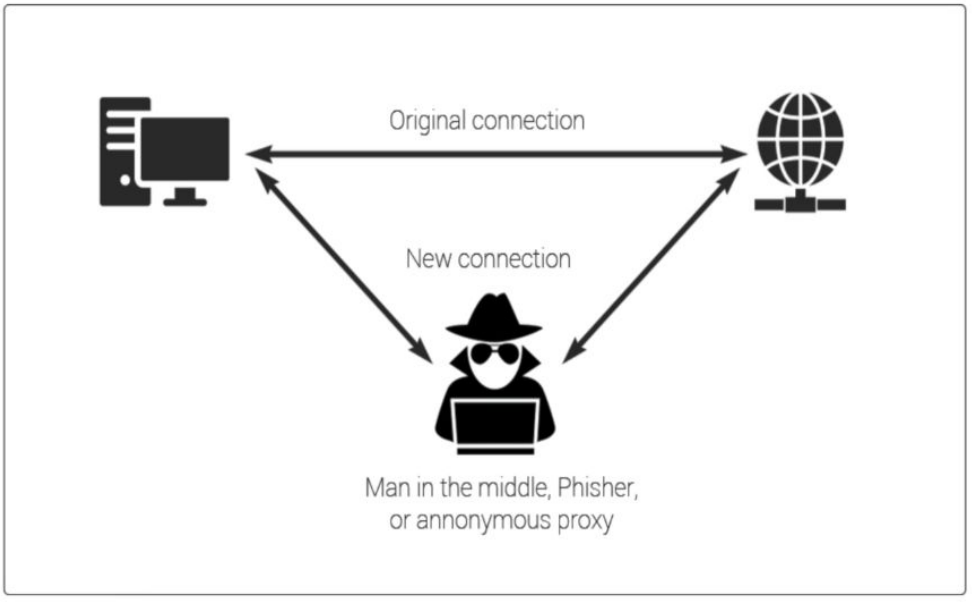
1) Для чего используется HTTPS и от чего защищает, поймем общую схему работы протокола HTTPS.

2) Что такое сертификаты, научимся смотреть сертификаты, поймём, зачем они нужны.

3) Зачем нужны центры сертификации.

Протокол HTTPS

Изначально веб не был рассчитан на то, что по нему будут передаваться данные, которые нужно будет от кого-то прятать. Изначально он состоял из открытых страниц, которые размещали для общего просмотра, как телефонный справочник или публичные информационные ресурсы.



Но со временем появлялось все больше и больше функциональности, потом — пользовательские данные: логины, пароли, письма, Cookie и т. п. И тогда потребовалось защищать трафик от так называемых атак Man in the middle (дословно — «человек посередине»); коротко обычно называют MITM по первым буквам.

Суть атаки в том, что злоумышленник может подслушивать или перехватывать и изменять данные, которые отправляет клиент серверу и которые сервер возвращает клиенту. Происходит это, когда злоумышленник и жертва или злоумышленник и сервер находятся в одной и той же локальной сети. Подробнее, почему так происходит, вы узнаете в курсе по сетям, сейчас важно понимать, что, если злоумышленник может подключиться к одной и той же точке доступа Wi-Fi, что и вы, и ваш трафик

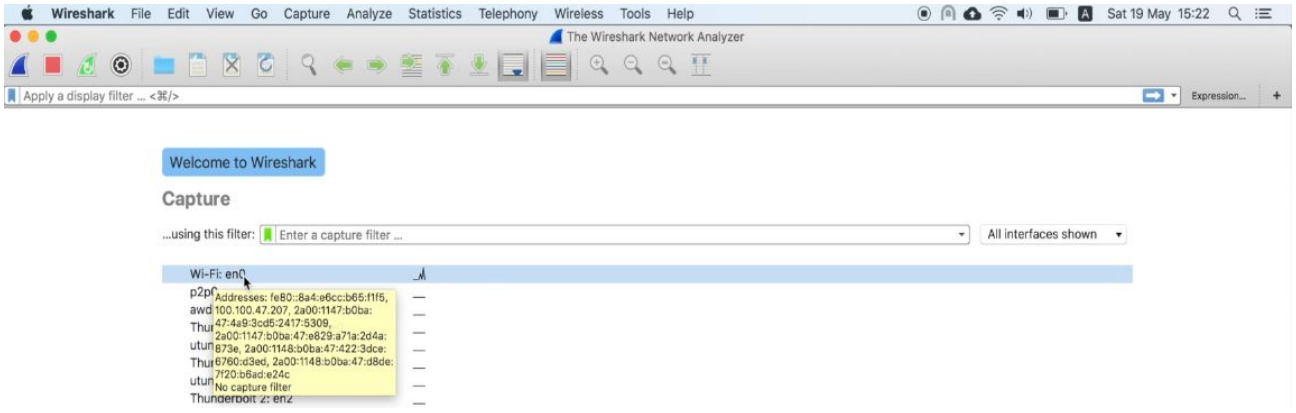
будет идти по нешифрованному HTTP-протоколу, то злоумышленник сможет увидеть или даже изменить весь трафик.

Протокол HTTPS решает эту проблему за счет шифрования. Он шифрует весь трафик от клиента до сервера, и когда злоумышленник пытается перехватить что-то, он видит абракадабру и не понимает, что находится в самих данных. Более того, у злоумышленника нет возможности понять, что находится внутри. Подробнее об этом мы узнаем в курсе по криптографии, а сейчас посмотрим, как это выглядит.

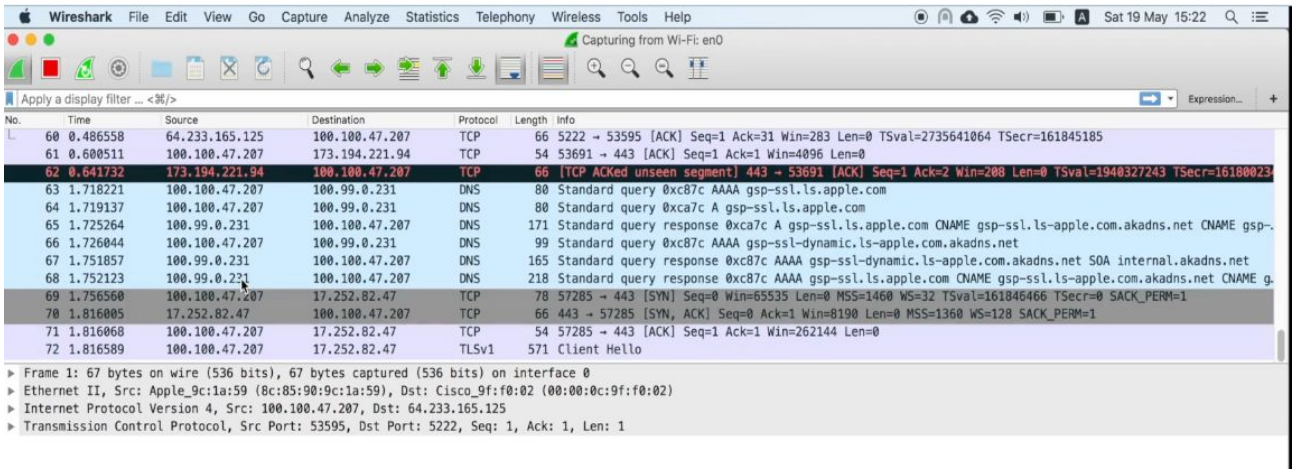
Wireshark

Откроем Firefox и новую для нас программу Wireshark. Она необходима, чтобы перехватывать пакеты на уровне сети. Об этом вы узнаете подробнее в курсе по сетям, но сейчас мы увидим, как выглядит зашифрованный, а как — незашифрованный трафик.

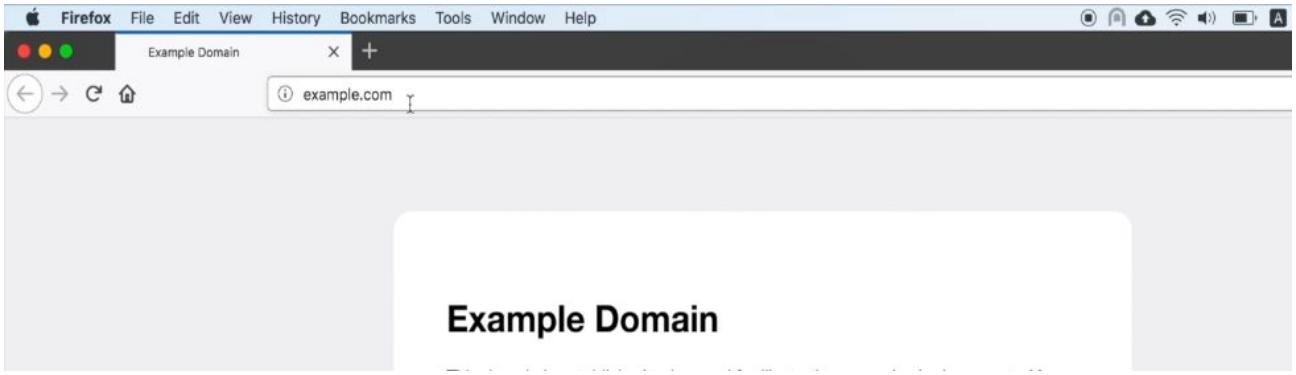
Откроем Wireshark и выберем сетевой интерфейс, на котором идет соединение в основной машине. Например, выберите Wi-Fi, если вы подключены к беспроводному соединению, или локальную сеть, если вы используете Ethernet-кабель:



Здесь данных еще больше и они непонятны, в отличие от тех, что в Burp, и поэтому без фильтрации не обойтись:

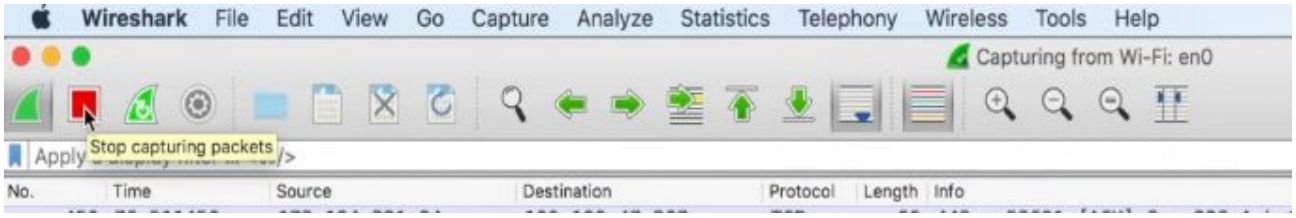


Сначала сделаем запросы к http://example.com:



Если мы не указываем протокол, по умолчанию браузер делает запрос по HTTP. Мы сделали запрос по HTTP, запросим данные и по HTTPS:

Запрос по HTTPS тоже был успешно выполнен. Вернёмся в Wireshark и остановим прослушивание трафика:

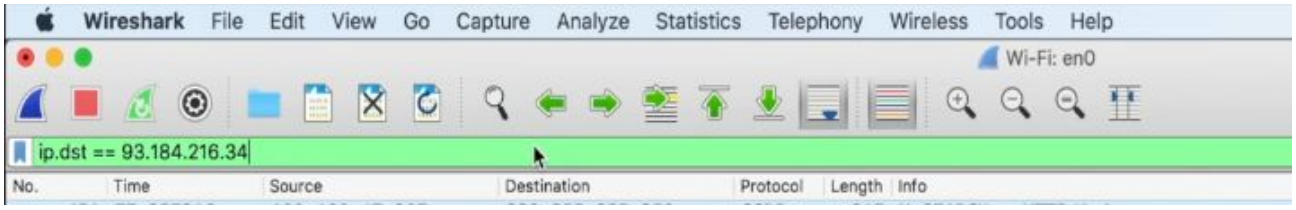


Теперь воспользуемся фильтром и выделим адреса, которые относятся к сайту example.com — для этого узнаем адрес сайта через программу host:

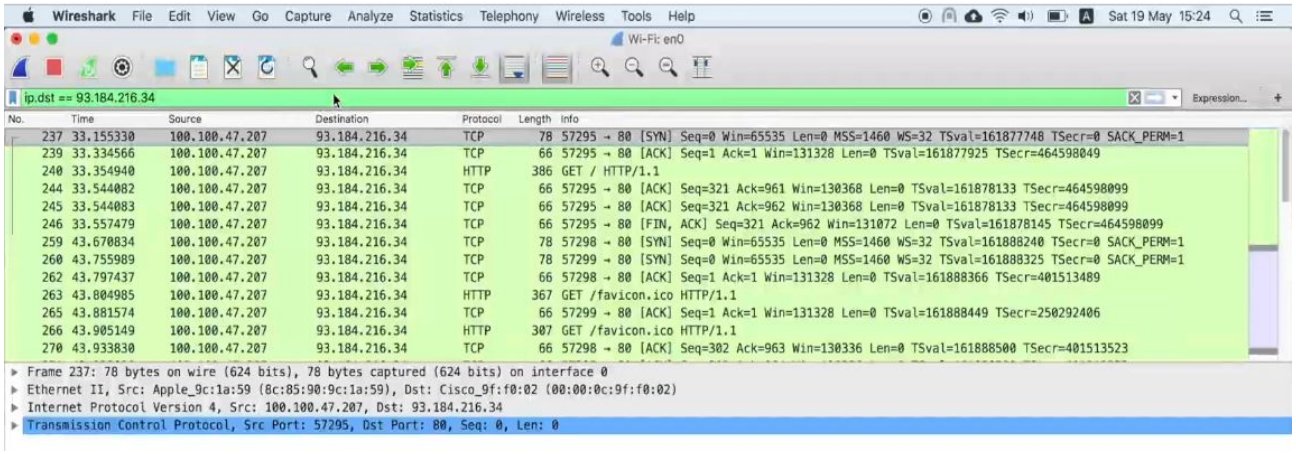
```
host example.com

example.com has address 93.184.216.34
```

```
example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
```

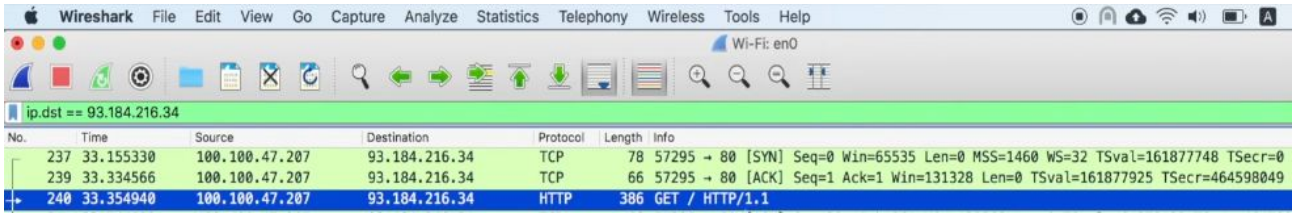
Затем скопируем адрес и применим фильтр:



Так мы выберем только те пакеты, которые уходят на сервер example.com, то есть наши запросы.

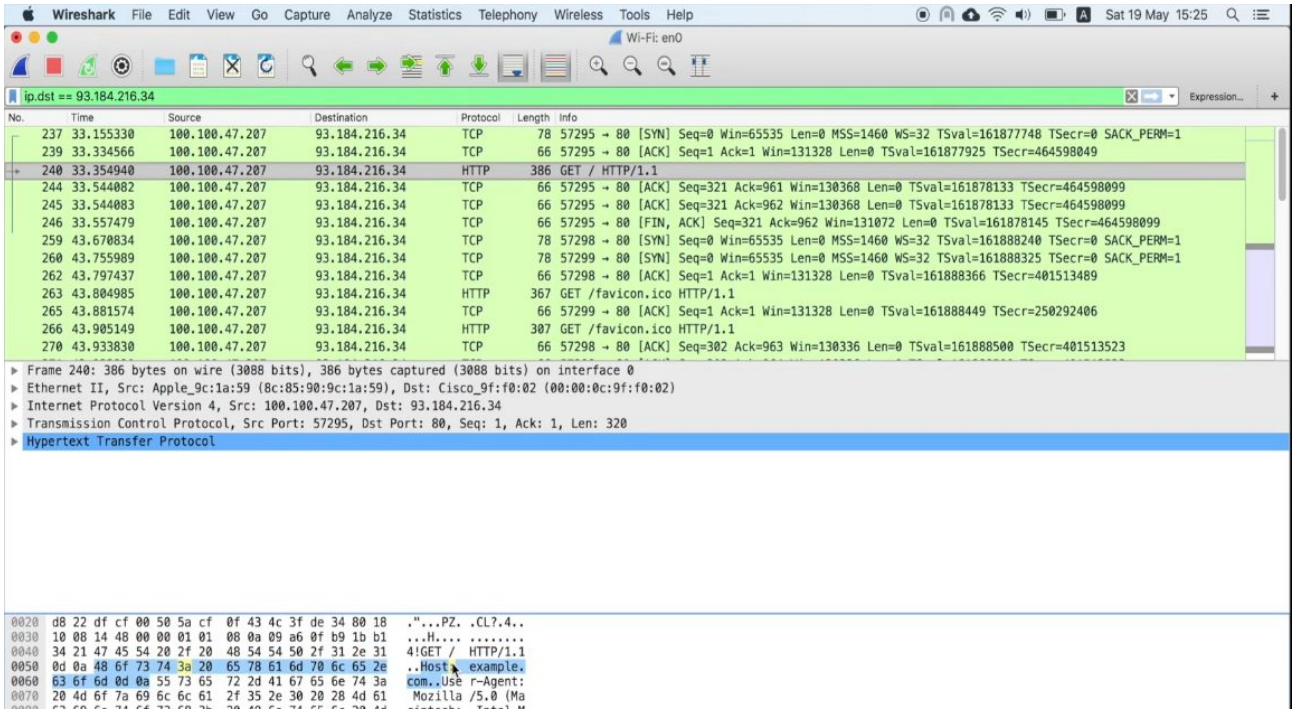


Фильтр был успешно применен и мы сразу можем видеть, что есть запрос с нашего IP-адреса (в столбце Source=100.100.47.207) на IP-адрес сайта example.com Destination=93.184.216.34:

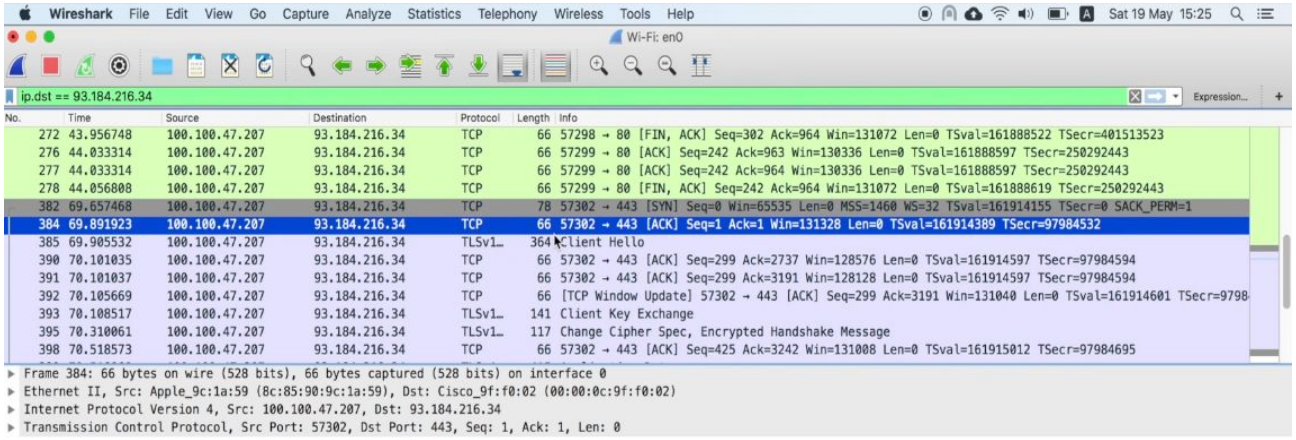


Wireshark расшифровывает, что это GET-запрос за главной страницей.

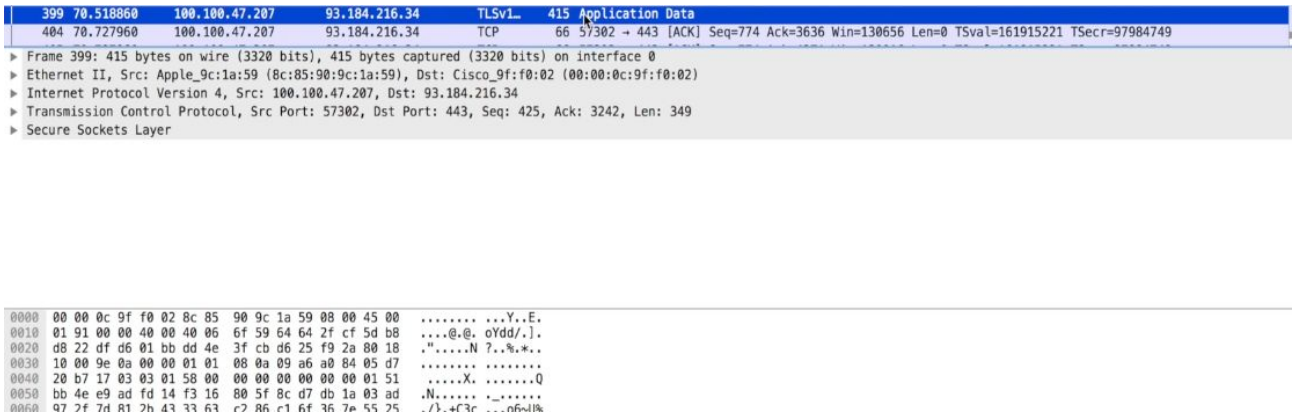
Мы видим здесь все заголовки, они представлены в байтах — GET, Host: example.com:



Пролистаем ниже, и вот здесь уже идет зашифрованный трафик:



Это трафик, который пошел по HTTPS, и, какой запрос бы ни был открыт, мы увидим абракадабру:



В информации есть Application Data — это Wireshark подсказывает, что запрос был сделан, скорее всего, с какими-то данными. В данном случае мы делали запрос главной страницы по HTTPS. Вы уже

не увидите GET / HTTP или Host — здесь что-то непонятное и поэтому восстановить изначальный запрос просто нереально.

Получается, что сам HTTPS шифрует трафик, который идет от сервера клиенту и от клиента к серверу. Таким образом, прочитать этот трафик могут только клиент и сервер, потому что они знают секретные ключи шифрования. Шифрование и заключается в том, что мы применяем некоторый алгоритм к данным и они превращаются в нечто нечитаемое или невосстанавливаемое в исходный вид без ключа. Мы будем называть это нечто шифротекстом в курсе по криптографии. Мы отправляем шифротекст серверу. У сервера уже есть ключ, который позволяет превратить нечитаемые данные в читаемые, или, как он называется в курсе по криптографии, в открытый текст. Так сервер может понять, что было изначально написано, и отдать ответ, также зашифровав его. Это одна из двух задач, для которых нужен протокол HTTPS.

Сертификаты HTTPS

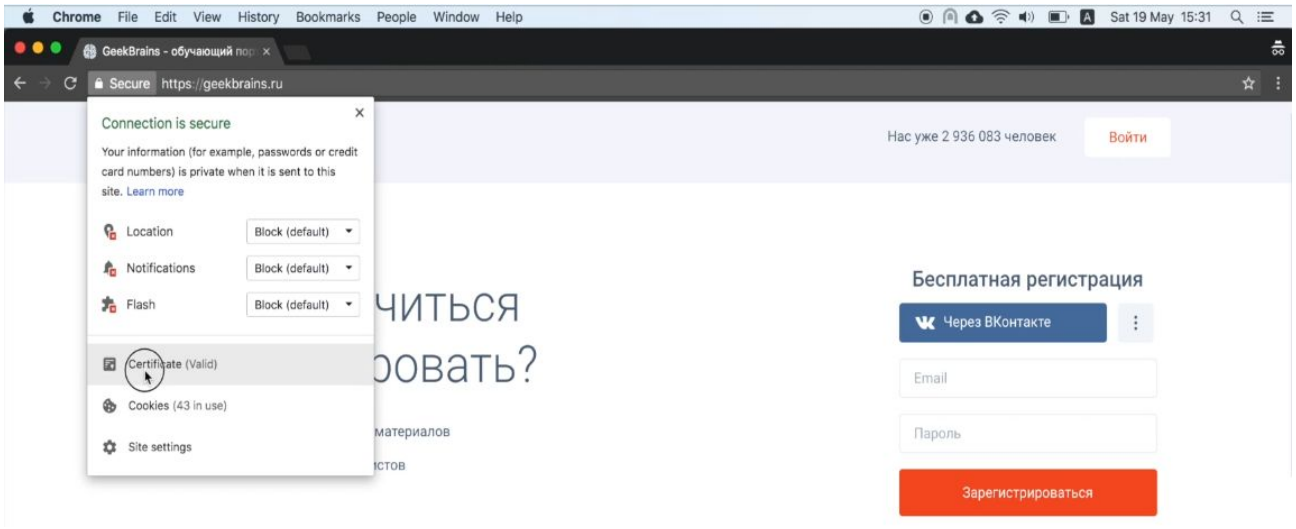
Как клиенту понять, что он пришел на настоящий сервер, а не его дубликат, который разместил злоумышленник? Ведь возможно, что кто-то подменил файл hosts и мы попали на сервер злоумышленника за счет того, что сайт (например GeekBrains) был в записях файла hosts перенаправлен на сервер злоумышленника.

Для решения этой проблемы и нужна вторая возможность протокола HTTPS, которая относится к сертификатам.

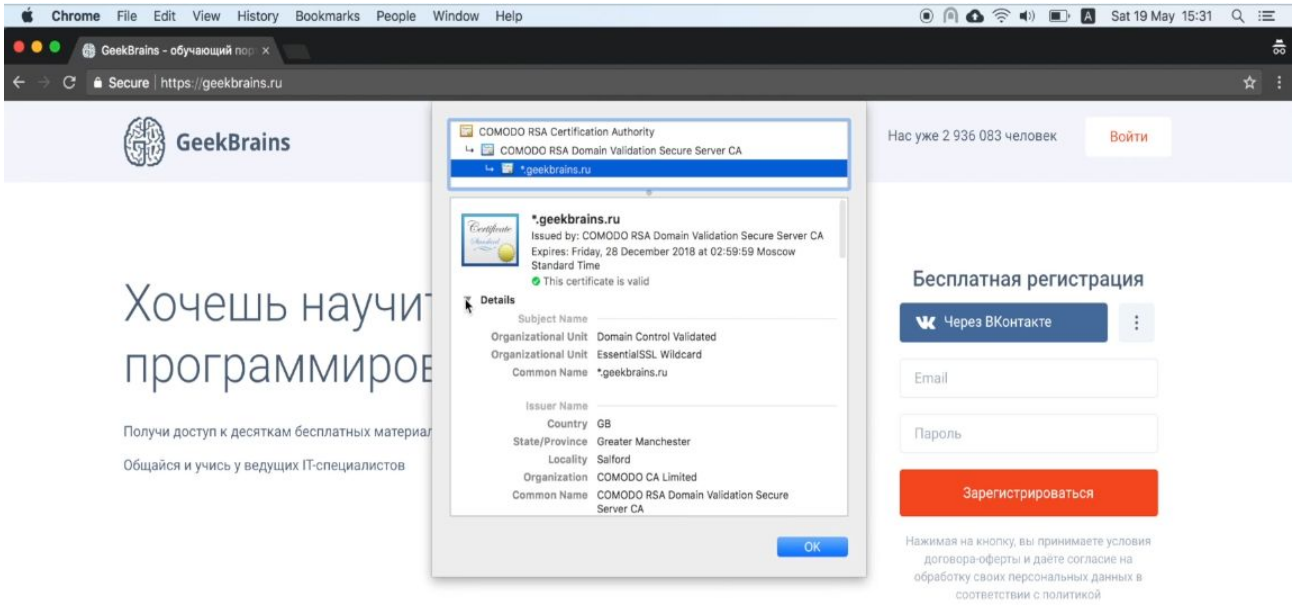
Не будем углубляться в определение сертификата. Нам важно понять, что это файл или строка, которые позволяют подтвердить подлинность сервера. Подробнее с устройством сертификатов и их работой мы разберемся уже на курсе криптографии. Это довольно сложная тема, нам нужно узнать для начала некоторые базовые алгоритмы, которые используются как фундаментальные криптографические протоколы. На следующем учебном курсе по криптографии вы разберетесь с этим и поймете, что происходит, когда браузер устанавливает HTTPS-соединения.

На примере посмотрим, где мы можем увидеть сертификат в браузере. Откроем Chrome — не Firefox, который сейчас настроен с Burp-прокси, на всех сайтах будет отображаться именно он: ведь сначала Firefox идет в Burp, а лишь потом Burp сам уже идет на сервер.

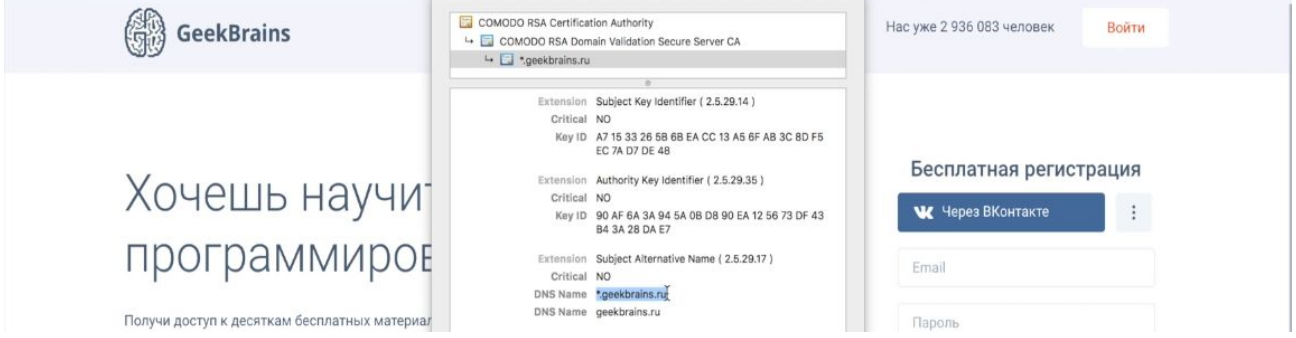
Нам нужен сайт https://geekbrains.ru. Нажмем на значок замка и выберем сертификат:



Мы получаем информацию о сертификате, можно посмотреть больше информации:



Тут есть имя сертификата, домены, для которых он валиден:

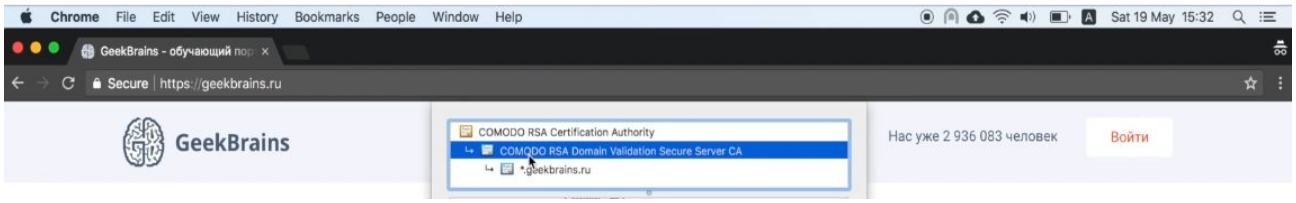


Если мы разместим этот сертификат на любом сервере или поддомене сайта https://geekbrains.ru, браузер будет считать, что этот домен и сайт правильный. А если, наоборот, мы разместим этот сертификат на домене mail.ru, браузер придет туда и скажет, что это небезопасный сайт и вас

пытаются обмануть. Это происходит потому, что в сертификате доменное имя и домен сайта, по которому мы пришли, не совпадает.

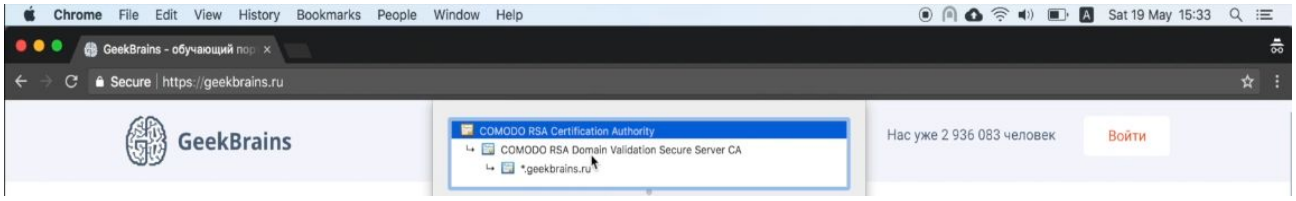
Удостоверяющие центры сертификации

Как происходит алгоритм проверки сертификата? Браузер приходит на https://geekbrains.ru и запрашивает сертификат. Сайт https://geekbrains.ru показывает сертификат и браузер проверяет его специальным системным вызовом. Чтобы проверить валидность, он обращается к другому сертификату, который находится выше:



В данном случае это Comodo RSA. Он уточняет у удостоверяющего центра, которому принадлежит этот сертификат, валиден ли *.geekbrains.ru. Если это так, удостоверяющий центр подтверждает, что сертификат действительный.

Мы поднимаемся еще выше, спрашиваем у корневого центра сертификации, валиден ли Comodo RSA или нет.



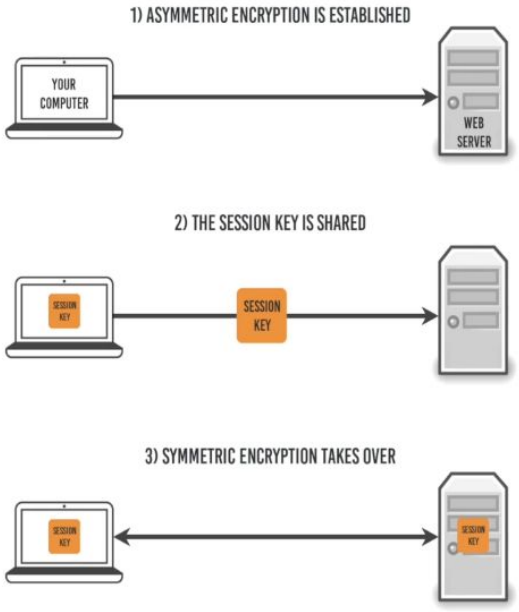
Comodo RSA Certifiaction Authority — самый верхний корневой центр сертификации — подтверждает или опровергает валидность сертификата. Подробнее, как они ее проверяют, вы узнаете в курсе по криптографии.

Основная проблема сертификатов — все в итоге упирается в корневой сертификат с неким корневым центром сертификации и мы вынуждены доверять ему и считать, что он хорошо ведет учет своих сертификатов.

Другая проблема — центром сертификации может стать любой. Все алгоритмы, которые генерируют сертификаты — открыты, вы сами можете создать сертификат и положить на сервер. Браузер не воспримет его как валидный, потому что он не подписан ни одним корневым центров сертификации. Но в целом каждый может стать сам себе корневым центром сертификации и выпускать сертификаты. Вопрос только в том, чтобы потом вас поддержали браузеры — так пользователи будут доверять вашему центру сертификации и переходить туда.

Центр сертификации — организация, которая генерирует сертификаты, выдает их клиентам, таким, например, как https://geekbrains.ru, ведет их учет, следит за безопасностью.

Обобщим то, что мы узнали о центрах сертификации и HTTPS как шифровании, и алгоритме того, как работает HTTPS на верхнем уровне.



Первым шагом компьютер делает запрос на веб-сервер, чтобы установить защищенное соединение. Дальше компьютер проверяет, что это действительно тот сервер, куда мы хотели попасть и нас никто не обманул. Потом клиент и сервер договариваются о секретном ключе, на котором будут производиться дальнейшие коммуникации. После того, как они договорились, включается симметричное шифрование и весь трафик между клиентом и сервером шифруется. После этого HTTPS считается установленным.

Итоги

Вы узнали:

1) Что такое HTTPS, как он используется и общую схему его работы.
2) Каково устройством сертификатов, как они проверяются.
3) Что такое центр сертификации, как он выдает сертификаты и какая проблема лежит в основе всех центров сертификации.

Ссылки к уроку

1. Коды ответа HTTP.
2. Список кодов состояния HTTP.
3. Что такое JSON?
4. HTTP. Методы GET и POST.
5. Referer.
6. Уязвимость CSRF. Скрываем Referer при редиректе.
7. Что такое Cookie-файлы и как их очистить в современных браузерах.
8. Whitepaper: Security Cookies.
9. Атака «Фиксация сессии».
10. Wireshark: приручение акулы.
11. Протокол HTTPS и как он защищает вас в интернете.

Домашнее задание

Задания необходимо сдавать в формате скриншотов, которые сопровождаются комментариями. Отчет должен быть в формате PDF. Пожалуйста, пишите в названии файла домашнего задания своё имя.

1. Создать файл test.txt в корневом каталоге сервера. Получить этот файл через браузер. Установить в терминале программу curl, получить тот же файл с помощью этой программы. Установить telnet или netcat, получить тот же файл с помощью одной из этих программ.
2. Создать на сервере файл sensitive_info.txt. Добавить базовую HTTP-авторизацию для этого файла. Получить этот файл через браузер. Получить тот же файл с помощью curl и telnet или netcat.
3. Открыть инструменты разработчика, вкладку Network (сеть). Зайти на сайт https://geekbrains.ru. Проанализировать Cookie каждого запроса HTML и картинок. Какие запросы уходят с Cookie, а какие — без? Почему в каждом из случаев происходит именно такое поведение?

4. (*) Для выполнения этого задания вам потребуется:

4.1. Настроить домены attacker.com, sub.attacker.com, sub.sub.attacker.com, victim.com. Каждый из этих доменов должен указывать на 127.0.0.1

4.2. Настроить установку Cookie для доменов. Добавьте следующий конфигурационный файл nginx (изменив root сервера на свой):

```
$ sudo nano /etc/nginx/sites-available/cookie-research.conf

server {
    listen 80;
    server_name attacker.com;
    root /var/www/html;

    location / {
                                        add_header     "Set-Cookie"
"test1=attacker-com_sub-attacker-com; Domain=sub.attacker.com";
           add_header "Set-Cookie" "test2=attacker-com_victim-com;
Domain=victim.com";
        add_header Cache-Control no-cache;

        try_files $uri $uri/ =404;
    }
}

server {
    listen 80;
    server_name victim.com;
    root /var/www/html;

    location / {
        try_files $uri $uri/ =404;
    }
}

server {
    listen 80;
    server_name sub.attacker.com;
    root /var/www/html;

    location / {
                                        add_header     "Set-Cookie"
"test3=sub-attacker-com_attacker-com; Domain=attacker.com";
        try_files $uri $uri/ =404;
    }
}

server {
    listen 80;
    server_name sub.sub.attacker.com;
```

```
    root /var/www/html;

    location / {
                                            add_header     "Set-Cookie"
"test4=sub-sub-attacker-com_attacker-com; Domain=attacker.com";
        try_files $uri $uri/ =404;
    }
}
```

Проведите исследование механизма проставления Cookie, для этого попробуйте установить следующие Cookie:

- С attacker.com на sub.attacker.com
- С attacker.com на victim.com
- С sub.attacker.com на attacker.com
- С sub.sub.attacker.com на attacker.com

По каждому пункту ответьте на вопросы:

- Куда установились Cookie?
- Если не установились, то почему?

Обобщите полученные знания и напишите вывод в формате: «Домен может проставлять куки для себя, для ... и ..., но не может проставлять куки для ..., ... и ...».

5. (*) Сгенерировать самоподписанный сертификат и разместить его на своем сервере. Для этого добавьте в конфигурационный файл nginx параметры для ssl-сертификатов, например так:

```
$ sudo nano /etc/nginx/sites-available/cookie-research.conf

server {
    listen 80;
    server_name your-ssl-site-here.com;

    root /var/www/html;
    listen 443 ssl;
    ssl_certificate /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

    location / {
     try_files $uri $uri/ =404;
    }
}

server {
    listen 80;
    server_name attacker.com;
```

```
    root /var/www/html;

    listen 443 ssl;

    location / {
                                            add_header     "Set-Cookie"
"test1=attacker-com_sub-attacker-com; Domain=sub.attacker.com";
           add_header "Set-Cookie" "test2=attacker-com_victim-com;
Domain=victim.com";
        add_header Cache-Control no-cache;

        try_files $uri $uri/ =404;
    }
}

server {
    listen 80;
    server_name victim.com;
    root /var/www/html;

    location / {
        try_files $uri $uri/ =404;
    }
}

server {
    listen 80;
    server_name sub.attacker.com;
    root /var/www/html;

    location / {
                                            add_header     "Set-Cookie"
"test3=sub-attacker-com_attacker-com; Domain=attacker.com";
        try_files $uri $uri/ =404;
    }
}

server {
    listen 80;
    server_name sub.sub.attacker.com;
    root /var/www/html;

    location / {
                                            add_header     "Set-Cookie"
"test4=sub-sub-attacker-com_attacker-com; Domain=attacker.com";
        try_files $uri $uri/ =404;
    }
}
```