HTTP

HTTP протокол нужен чтобы ты мог видеть эту страницу. Когда ты отправляешь запрос (нажатие кнопки Enter в адресной строке), то первым делом браузер отправляет DNS запрос, чтобы получить IP адрес сервера, после этого этот IP указывается в запросе HTTP и отправляется, а принимает его уже веб сервер (apache, nginx, etc), обрабатывает и дает ответ. Т.е. обмен данными с сервером идет через отправку и получения HTTP пакетов. В этих пакетах указывается тип запроса, внутренний путь ресурса, версия HTTP, заголовки и тело запроса. В ответе почти тоже самое только еще добавляется код состояния.

Основные типы запросов (методы).

GET - метод гет это обычный запрос. Ты открываешь браузер, вводишь URL и нажимаешь <Enter> и выполняется GET запрос. GET запрос может передавать параметры, обычно url от параметров отделяет ?, а сами параметры разделяются & и указываются в формате key=value. И вся эта каша находится именно в браузерной строке.

POST - чуть сложнее в понимании потому что не видно где параметры. А параметры передаются в теле запроса. Пост запросы обычно используют там где требуется скрыть передаваемые параметры. Так например, если ты авторизуешься на сайте и он шлет твои логин и пароль методом GET, то уже не важно использует ли сайт ssl, снифер поймает пакет и получит эти данные, так как они будут частью URL. А вот если отправить методом пост, то данные будут частью тела запроса и будут зашифрованы, также как и тело ответа.

Некоторые важные заголовки.

Host - заголовок указывающий адрес хоста для которого предназначается пакет. Указывается в запросе.

Location - указывает новый адрес запрашиваемого ресурса. Указывается в ответе.

Referer - указывает на url где ты был перед тем как кликнул по ссылке. Указывается в запросе.

Cookie - печеньки. Выставляются при ответе и всегда указываются при запросе. Имеют срок жизни.

User-Agent - строка содержащая информацию о браузере и системе. Указывается при запросе.

Content-Length - длина контента.

Content-Type - MIME тип отправляемого контента. Wiki

Основные коды ответа HTTP, остальные есть в вики.

200 - Этот код говорит, что запрос выполнился успешно.

301 - Значит, что указанный uri постоянно перемещен. Куда указывается в заголовке Location.

302 - Временное перенаправление на странице.

400 - Не верно составленный HTTP запрос.

403 - Требуется авторизация.

404 - uri не существует.

50x - Эти коды состояния означают внутреннюю ошибку сервера.

Давай я тебе покажу примеры HTTP запросов. Чтобы это узреть запусти утилиту netcat.

$ echo 'Hello!' | nc -l 8080

Таким образом мы попросим netcat слушать подключение на 127.0.0.1:8080. В браузере можно отправить запрос на этот IP. Просто введи в адресную строку http://127.0.0.1:8080/ и ты увидешь Hello. Теперь в терминале у тебя сырой HTTP запрос от браузера.

GET / HTTP/1.1

Host: 127.0.0.1:8080

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:56.0) Gecko/20100101 Firefox/56.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate

DNT: 1

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Это будет нашим запросом. Сохраняем в файл. Теперь пропрбуй использовать этот запрос чтобы обратиться к другому сайту из netcat

$ cat request.txt|nc trello.com 80

443 нам не ответит, позже разберем

Что в ответ? 400 Bad Request - это значит что наш HTTP запрос не правильный. А все потому что надо было указать правильный заголовок Host, сейчас он указывает на старый адрес. Меняем на "trello.com" и пробуем еще раз.

HTTP/1.1 301 Moved Permanently

Content-Length: 0

Location: https://trello.com/

Date: Thu, 31 May 2018 15:59:42 GMT

Connection: keep-alive

301 - понял, да? Сохраняем ответ в файл response.txt. Теперь можно заменить адрес в поле Location например на https://google.com/ и зпустить netcat

$ cat response.txt|nc -l 8080

И откроется google :-). Теперь ответим 200-м статусом)

```
HTTP/2.0 200 OK
content-encoding: br
content-type: text/html; charset=UTF-8
date: Thu, 31 May 2018 16:50:25 GMT
server: gws
domain=.google.com
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
<!DOCTYPE html>
<html>
<head>
   <title></title>
</head>
<body>
   <h1>Hello 2.0</h1>
</body>
</html>
```

Сохрани и запусти netcat

```
$ cat response.txt|nc -l 8080
```

В данном случае netcat выступал в качестве HTTP сервера, который принимал запрос и отдавал ответ. Вот только у нас нет никакой логики. В реальном сайте в качестве веб сервера обычно используется Apache2 или Nginx, они передают запрос в бекенд, а в качестве бекенда может использоваться: PHP, ASP, Python, etc. Именно бекенд отвечает за обработку данных отправленных в HTML запросе, генерирует ответ и отдает его вебсерверу, а веб сервер твоему браузеру. Данные которые отображаются на странице: содержимое сайта, данные пользователей и т.д. хранятся в базе данных, с которой работает бекенд. А вот HTML, JavaScript, CSS и все остальное, что выполняется в самом браузере, относится к фронтэнду.

Мы будем разбирать несколько уязвимостей связанных с неправильной работой бекенда - это SQL-Injection, Local File Include, Remote File Include. Подробнее о них позже. Тебе для начала нужно осознать суть, что эксплуатировать данные уязвимости можно, если кодер не подумал о фильтрации данных отправленных пользователем. Вообще весь user input является потенциально опасным и к данным пришедшим от пользователя нужно относиться очень серьезно. Возможно ты сейчас подумаешь: "Ну и что? Раз всем изместно, что данные от пользователя небезопасны, значит все их фильтруют и я могу сломать только что то самописное, где неопытный кодер мог ошибиться, а какой нибудь движек не получится сломать, потому что там трудится куча

народу!" А вот и нет! Во-первых: ты не раз слышал о дырках в винде и ошибках выпускаемых с обновлением винды, а ведь это крупная корпорация выпускающая платный продукт, над которым трудится команда профессиональных разрабочиков на протяжении N-лет; Во-вторых: drupalgeddon2; Так что наивно полагать, что если над проектом трудится много народу, то там все гайки закручены, у семи нянек дитя без глазу. Этих знаний хватит, чтобы начать. И помни - Стоячая вода превращается в болото!

Вернемся к запросам HTTP и netcat. Нам не ответит сервер по 443 порту потому что сначала надо установить сессию SSL. Чтобы пообщаться с сервером сырыми запросами, нужно использовать утилиту openssl. Например так:

```
$ openssl s_client -connect server:port
```

Да, бывают некоторые команды которые принимают параметры по своему