Fundamental Skills - Curl

Category	Experience Level
Tools	Novice

Contents

Introduction

`curl` is a versatile Command Line tool for making HTTP requests. Please see the Web 2 - Understanding HTTP Requests-lesson-for-more-details-before-reading-this-lesson-(unless-you-already-know-what-they-are).

You will also need a command line interface to run `curl`. We recommend using a Linux distribution such as Kali Linux, which you can setup by following our Misc - Setting up a Virtual Machine lesson.

Installing Curl

In the command line, check whether `curl` is installed by running the following command:

```
$ which curl
```

If you do not get an output, `curl` is not installed. Install it on Kali with:

```
$ sudo apt-get install curl
```

Or visit command-not-found.com to find the correct command.

Using Curl

You can send many kinds of request with `curl`, but we'll focus on GET and POST.

GET Requests

To make a HTTP GET request with `curl`, simply type `curl` followed by the URL:

```
$ curl http://example.com
```

Remember the `http://` or `https://` prefix, otherwise it will not work.

If the webserver is running on a non-standard port (80 for HTTP, and 443 for HTTPS), you can specify that port with a colon:

```
$ curl http://example.com:8080
```

Note: don't worry if you're not sure what ports are - we'll cover them in a later lesson. Think of them as a secondary part of the address that let you host multiple things on one server.

By default, `curl` will output the server's response to the screen. This might be the HTML code for a webpage, some JSON data, an error message, or any other data a server could respond with.

To output to a file instead, use the `-o` flag or the `>` redirection control character:

```
$ curl http://example.com -o index.html
$ curl http://example.com > index.html
```

To see more verbose input, including the headers sent, use `curl -v`:

```
$ curl -v http://example.com
*   Trying 93.184.216.34:80...
* Connected to example.com (93.184.216.34) port 80 (#0)
> GET / HTTP/1.1
> Host: example.com
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Age: 57629
< Cache-Control: max-age=604800
< Content-Type: text/html; charset=UTF-8
< Date: Fri, 01 Oct 2021 19:25:48 GMT
< Etag: "3147526947+ident"
< Expires: Fri, 08 Oct 2021 19:25:48 GMT
< Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
< Server: ECS (nyb/1D2C)
< Vary: Accept-Encoding
< X-Cache: HIT
< Content-Length: 1256
<
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

...[rest of response]...
```

The above shows the request prefixed by `>` characters, and the response prefixed by `<` characters.

As with a normal HTTP request in a browser, you can add parameters to the request with a `?param=value&param2=value2` string:

```
$ curl -v http://example.com?foo=bar
```

To just see the headers, use `curl -I` - this is useful for determining useful site information that may be in the headers, such as the `X-Powered-By` header which may indicate the technology used on the site:

```
$ curl -I http://example.com
HTTP/1.1 200 OK
Content-Encoding: gzip
Accept-Ranges: bytes
Age: 402849
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Fri, 01 Oct 2021 19:24:36 GMT
Etag: "3147526947+gzip"
Expires: Fri, 08 Oct 2021 19:24:36 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
```

```
Server: ECS (nyb/1D0B)
X-Cache: HIT
Content-Length: 648
```

To add a User Agent to your request, use the `-A` flag:

```
$ curl -v http://example.com -A "Fake User Agent"
*   Trying 93.184.216.34:80...
* Connected to example.com (93.184.216.34) port 80 (#0)
> GET / HTTP/1.1
> Host: example.com
> User-Agent: Fake User Agent
> Accept: */*
```

To set any other arbitrary header, use the `-H` flag and specify the header name followed by its value, comma-separated; you can specify any number of headers:

```
$ curl -v http://example.com -H 'Fake-Header: Fake-Contents' -H 'Fake-
Header-2: More Contents'
*   Trying 93.184.216.34:80...
* Connected to example.com (93.184.216.34) port 80 (#0)
> GET / HTTP/1.1
> Host: example.com
> User-Agent: curl/7.74.0
> Accept: */*
> Fake-Header: Fake-Contents
> Fake-Header-2: More Contents
```

If you want to use `curl` on an authenticated URL, such as an admin platform, there are a couple of ways of authenticating:

- if the site uses basic HTTP authentication, you can supply it with the `-u` flag; for example, this command would supply the username `admin` and the password `admin123`: `curl -u admin:admin123 http://example.com`
- if the site uses cookies, you can specify cookies in a header; for example, `curl -H 'Authorization: Bearer 88b7...98hb' http://example.com`

Finally, if you are testing a site running an old version of HTTPS, `curl` might not accept its SSL certificate. If you get any errors concerning SSL certificates being invalid, you can ignore them with the `-k` flag:

```
$ curl -k https://example.com
```

This is equivalent to 'accepting' the certificate in browser.

POST Requests

You can submit data in a `curl` command by specifying the `-d` flag (short for data). The presence of this flag will, by default, make `curl` send a POST request. For example, here we are submitting some JSON data (as indicated by the `Content-Type` header):

```
$ curl -v http://example.com -H "Content-type: application/json" -d
'{"param":"value"}'
*   Trying 93.184.216.34:80...
* Connected to example.com (93.184.216.34) port 80 (#0)
> POST / HTTP/1.1
> Host: example.com
> User-Agent: curl/7.74.0
> Accept: */*
> Content-type: application/json
> Content-Length: 17
>
* upload completely sent off: 17 out of 17 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Accept-Ranges: bytes
< Cache-Control: max-age=604800
< Content-Type: text/html; charset=UTF-8
< Date: Fri, 01 Oct 2021 20:01:02 GMT
< Etag: "3147526947"
< Expires: Fri, 08 Oct 2021 20:01:02 GMT
< Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
< Server: EOS (vny/044E)
< Content-Length: 1256
```

This submits the JSON in the request body, with the key-value pair `"param":"value"`.

If you wish to submit data in the body of a GET request, the syntax is similar - but you can use the `-X` flag to specify the `GET` method:

```
$ curl -v -X GET http://example.com -H "Content-type: application/json" -d
'{"param":"value"}'
*   Trying 93.184.216.34:80...
* Connected to example.com (93.184.216.34) port 80 (#0)
```

```
> GET / HTTP/1.1
> Host: example.com
> User-Agent: curl/7.74.0
> Accept: */*
> Content-type: application/json
> Content-Length: 17
>
* upload completely sent off: 17 out of 17 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 400 Bad Request
< Content-Type: text/html
< Content-Length: 349
< Date: Fri, 01 Oct 2021 20:04:30 GMT
< Server: ECSF (nyb/1D0A)
<
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
        <head>
                <title>400 - Bad Request</title>
        </head>
        <body>
                <h1>400 - Bad Request</h1>
        </body>
</html>
* Connection #0 to host example.com left intact
```

In fact, you can specify *any* valid HTTP verb with the `-X` flag.

Proxies

We will talk about proxies in more detail in a later lesson, but if you know what you're doing with them already you can use the `-x` flag. You must specify the protocol (usually HTTP), address, and port of the proxy:

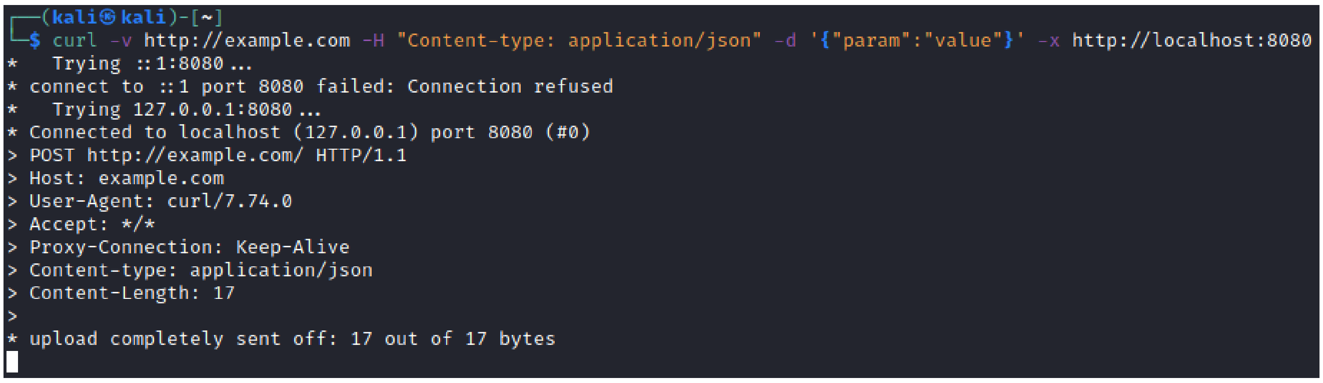
```
$ curl https://example.com -x https://192.168.119.10:8443
```

The above command will pass the request to a proxy listening at IP address 192.168.119.10 on port 8443, over HTTPS, instead of going straight to `example.com`.

An easy example of a proxy is Burp Suite - this will let you visualise your `curl` requests even better:

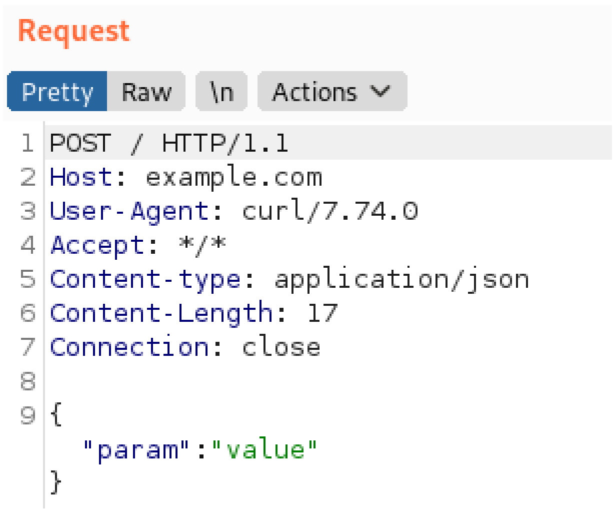
```
┌──(kali㉿kali)-[~]
└─$ curl -v http://example.com -H "Content-type: application/json" -d
'{"param":"value"}' -x http://localhost:8080
```

As you can see, the request 'hangs' while it waits for Burp Suite to pass it on:



```
┌──(kali㉿kali)-[~]
└─$ curl -v http://example.com -H "Content-type: application/json" -d '{"param":"value"}' -x http://localhost:8080
*   Trying ::1:8080...
* connect to ::1 port 8080 failed: Connection refused
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> POST http://example.com/ HTTP/1.1
> Host: example.com
> User-Agent: curl/7.74.0
> Accept: */*
> Proxy-Connection: Keep-Alive
> Content-type: application/json
> Content-Length: 17
>
* upload completely sent off: 17 out of 17 bytes
```

In Burp Suite, we can see the contents of the request body:



```
POST / HTTP/1.1
Host: example.com
User-Agent: curl/7.74.0
Accept: */*
Content-type: application/json
Content-Length: 17
Connection: close

{
  "param":"value"
}
```

Cheatsheet

Standard GET request:

```
$ curl [PROTOCOL]://[URL]:[PORT]
```

Standard POST request with data:

```
$ curl [PROTOCOL]://[URL]:[PORT] -d [DATA]
```

Request with proxy:

```
$ curl [PROTOCOL]://[URL]:[PORT] -x [PROXY_PROTOCOL]://[PROXY_URL]:
[PROXY_PORT]
```

Request with headers:

```
$ curl [PROTOCOL]://[URL]:[PORT] -H '[HEADER 1 NAME]: [HEADER 1 VALUE]' -H
...[any number of headers here]...
```

Worksheet

1. Get the index of shefesh.com using `curl`
2. Look at the headers of the request - can you figure out what service the website is hosted with?
3. Pass the request to Burp Suite, or another proxy, and view it in more detail