CURL CHEATSHEET

FOR BUG BOUNTY HUNTERS, RED TEAMERS & ETHICAL HACKERS

1. BASIC REQUEST METHODS

GET Request (Default):

```
curl https://example.com
```

→ Performs a simple GET request to retrieve content from the URL

GET with Verbose Output:

```
curl -v https://example.com
```

→ Shows detailed request/response headers, useful for debugging

HEAD Request (Headers Only):

```
curl -I https://example.com
```

→ Fetches only HTTP headers without the body, useful for checking status

Specific HTTP Method:

```
curl -X POST https://example.com
```

→ Specifies the HTTP method to use for the request

Follow Redirects:

```
curl -L https://example.com
```

→ Follows HTTP 3xx redirects automatically (important for finding hidden endpoints)

Silent Mode:

```
curl -s https://example.com
```

→ Silent mode, hides progress meter and error messages

Show Only HTTP Status Code:

```
curl -o /dev/null -s -w "%{http_code}\n" https://example.com
```

→ Returns only the HTTP status code, useful in scripts

2. AUTHENTICATION & HEADERS

Basic Authentication:

```
curl -u username:password https://example.com
```

→ Sends credentials using HTTP Basic Auth

Bearer Token:

```
curl -H "Authorization: Bearer TOKEN" https://example.com
```

→ Sends JWT or OAuth bearer token in Authorization header

API Key in Header:

```
curl -H "X-API-Key: your_api_key" https://example.com
```

→ Common pattern for API authentication

Custom User-Agent:

```
curl -A "Mozilla/5.0 (Windows NT 10.0; Win64; x64)" https://example.com
```

→ Changes User-Agent string to bypass basic filtering

Multiple Custom Headers:

```
curl -H "Header1: value1" -H "Header2: value2" https://example.com
```

→ Adds multiple custom headers to the request

Referer Header:

```
curl -H "Referer: https://trusted-site.com" https://example.com
```

→ Sets the Referer header, useful for bypassing basic referer checks

X-Forwarded-For (IP Spoofing):

```
curl -H "X-Forwarded-For: 127.0.0.1" https://example.com
```

→ Attempts to spoof IP address (effectiveness depends on server configuration)

Host Header Override:

```
curl -H "Host: internal.example.com" https://1.2.3.4
```

→ Useful for testing virtual host configurations and Host header attacks

Origin Header (CORS Testing):

```
curl -H "Origin: https://evil.com" https://example.com
```

→ Tests CORS policies by simulating cross-origin requests

3. DATA SENDING (POST/PUT)

POST with Form Data:

```
curl -d "param1=value1&param2=value2" https://example.com
```

→ Sends URL-encoded form data in POST request

POST JSON Data:

```
curl -X POST -H "Content-Type: application/json" -d '{"key":"value"}' https://example.com
```

→ Sends JSON payload in POST request

POST from File:

```
curl -d @data.json https://example.com
```

→ Reads POST data from a file

PUT Request with Data:

```
curl -X PUT -d "data=value" https://example.com
```

→ Sends data using PUT method

URL-Encoded POST:

```
curl --data-urlencode "param=value with spaces" https://example.com
```

→ Automatically URL-encodes the data before sending

Multipart Form Data:

```
curl -F "field1=value1" -F "field2=value2" https://example.com
```

→ Sends data as multipart/form-data (useful for file uploads)

Upload File:

```
curl -F "file=@/path/to/file.txt" https://example.com/upload
```

→ Uploads a file using multipart/form-data

GraphQL Query:

```
curl -X POST -H "Content-Type: application/json" -d '{"query":"{ user { name } }"}'
https://example.com/graphql
```

→ Executes a GraphQL query

4. FILE OPERATIONS

Download File:

```
curl -O https://example.com/file.txt
```

→ Downloads file and saves with original filename

Download with Custom Name:

```
curl -o custom_name.txt https://example.com/file.txt
```

→ Downloads file and saves with specified filename

Resume Interrupted Download:

```
curl -C - -O https://example.com/largefile.zip
```

→ Continues a previously interrupted download

Download Multiple Files:

```
curl -O https://example.com/file1.txt -O https://example.com/file2.txt
```

→ Downloads multiple files in one command

Limit Download Rate:

```
curl --limit-rate 100K -O https://example.com/file.zip
```

→ Limits download speed to 100 KB/s

Upload File via PUT:

```
curl -T file.txt https://example.com/upload
```

→ Uploads file using HTTP PUT method

5. COOKIE MANAGEMENT

Send Cookie:

```
curl -b "session=abc123" https://example.com
```

→ Sends a cookie with the request

Send Multiple Cookies:

```
curl -b "cookie1=value1; cookie2=value2" https://example.com
```

→ Sends multiple cookies in the request

Save Cookies to File:

```
curl -c cookies.txt https://example.com
```

→ Saves cookies received from server to a file

Load Cookies from File:

```
curl -b cookies.txt https://example.com
```

→ Sends cookies from a file with the request

Session Management:

```
curl -c cookies.txt -b cookies.txt https://example.com
```

→ Maintains session by saving and loading cookies

6. PROXY & TUNNELING

HTTP Proxy:

```
curl -x http://proxy.example.com:8080 https://example.com
```

→ Routes request through HTTP proxy server

SOCKS5 Proxy:

```
curl --socks5 127.0.0.1:9050 https://example.com
```

→ Routes request through SOCKS5 proxy (e.g., Tor)

Proxy with Authentication:

```
curl -x http://proxy.example.com:8080 -U username:password https://example.com
```

→ Uses authenticated proxy server

Burp Suite Proxy:

```
curl -x http://127.0.0.1:8080 -k https://example.com
```

→ Routes through Burp Suite for interception (-k ignores SSL warnings)

No Proxy for Specific Host:

```
curl --noproxy localhost,127.0.0.1 https://example.com
```

→ Bypasses proxy for specified hosts

7. SSL/TLS OPTIONS

Ignore SSL Certificate Errors:

```
curl -k https://self-signed.example.com
```

→ Bypasses SSL certificate verification (use cautiously!)

Specific SSL/TLS Version:

```
curl --tlsv1.2 https://example.com
```

→ Forces specific TLS version for the connection

Client Certificate Authentication:

```
curl --cert client.pem --key client.key https://example.com
```

→ Uses client certificate for mutual TLS authentication

Show SSL Certificate Details:

```
curl -vI https://example.com 2>>&1 | grep -A 20 "Server certificate"
```

→ Displays detailed SSL certificate information

SNI (Server Name Indication):

```
curl --resolve example.com:443:1.2.3.4 https://example.com
```

→ Overrides DNS resolution for testing specific IPs

8. RESPONSE HANDLING

Save Response Headers:

```
curl -D headers.txt https://example.com
```

→ Saves response headers to a separate file

Show Only Response Headers:

```
curl -I https://example.com
```

→ Fetches and displays only the response headers

Show Request and Response Headers:

```
curl -v https://example.com
```

→ Shows detailed request/response including all headers

Custom Output Format:

```
curl -w "HTTP Status: %{http_code}\nTime: %{time_total}s\n" -o /dev/null -s https://example.com
```

→ Displays custom formatted output with status and timing

Response Time Measurement:

```
curl -o /dev/null -s -w "Time: %{time_total}s\n" https://example.com
```

→ Measures total response time

JSON Response Pretty Print:

```
curl -s https://api.example.com/data | jq .
```

→ Formats JSON response for readability (requires jq)

9. ADVANCED TECHNIQUES

HTTP/2 Request:

```
curl --http2 https://example.com
```
→ Forces HTTP/2 protocol if server supports it

HTTP/3 (QUIC):

```
curl --http3 https://example.com
```
→ Uses HTTP/3 protocol (if curl is built with HTTP/3 support)

DNS-over-HTTPS:

```
curl --doh-url https://1.1.1.1/dns-query https://example.com
```
→ Uses DNS-over-HTTPS for domain resolution

Custom DNS Resolution:

```
curl --resolve example.com:443:1.2.3.4 https://example.com
```
→ Overrides DNS lookup for testing specific IPs

Connection Timeout:

```
curl --connect-timeout 10 https://example.com
```
→ Sets maximum time for connection phase (10 seconds)

Max Request Time:

```
curl --max-time 30 https://example.com
```
→ Sets maximum total time for the request (30 seconds)

Retry on Failure:

```
curl --retry 5 --retry-delay 3 https://example.com
```
→ Retries failed requests 5 times with 3-second delays

Range Request (Partial Content):

```
curl -r 0-999 https://example.com/file.txt
```
→ Requests only bytes 0-999 (useful for testing range support)

Compressed Response:

```
curl --compressed https://example.com
```
→ Requests compressed response (gzip, deflate)

10. RATE LIMITING & TIMING

Delay Between Requests:

```
for i in {1..10}; do curl https://example.com; sleep 2; done
```

→ Sends 10 requests with 2-second delays between each

Parallel Requests:

```
curl https://example.com & curl https://example.com/page2 & wait
```

→ Sends multiple requests in parallel

Rate-Limited Testing:

```
for i in {1..100}; do curl -w "Status: %{http_code}\n" https://example.com; sleep 0.1; done
```

→ Tests rate limiting with 100 requests at 10 req/sec

Measure Connection Time:

```
curl -w "DNS: %{time_namelookup}s\nConnect: %{time_connect}s\nTLS: %{time_appconnect}s\nTotal:
%{time_total}s\n" -o /dev/null -s https://example.com
```

→ Shows detailed timing breakdown of request phases

Speed Limit Testing:

```
curl --limit-rate 1K https://example.com/api
```

→ Simulates slow connection to test timeout handling

11. FUZZING & TESTING

Parameter Fuzzing:

```
for param in id user admin test; do curl "https://example.com/api?param=$param"; done
```

→ Tests different parameter values

Path Fuzzing:

```
for path in admin config backup test; do curl -o /dev/null -s -w "%{http_code} - $path\n"
"https://example.com/$path"; done
```

→ Discovers hidden directories by status codes

Header Fuzzing:

```
for header in "X-Original-URL: /admin" "X-Rewrite-URL: /admin"; do curl -H "$header"
https://example.com; done
```

→ Tests various header bypass techniques

Method Tampering:

```
for method in GET POST PUT DELETE PATCH OPTIONS TRACE; do curl -X $method -o /dev/null -s -w
"$method: %{http_code}\n" https://example.com/api; done
```

→ Tests which HTTP methods are allowed

User-Agent Fuzzing:

```
for ua in "Googlebot" "curl" "Mozilla/5.0" ""; do curl -A "$ua" -s -w "UA=$ua
Status=%{http_code}\n" -o /dev/null https://example.com; done
```

→ Tests behavior with different user agents

12. COMMON BUG BOUNTY SCENARIOS

CORS Misconfiguration Test:

```
curl -H "Origin: https://evil.com" -I https://example.com/api
```

→ Checks if server reflects arbitrary origins in CORS headers

Authentication Bypass Test:

```
curl -X POST https://example.com/api/admin -H "X-Original-URL: /api/user"
```

→ Tests path-based authentication bypass

JWT Token Test:

```
curl -H "Authorization: Bearer MODIFIED_TOKEN" https://example.com/api/protected
```

→ Tests JWT validation by sending modified tokens

IDOR Test:

```
for id in {1..100}; do curl -b "session=TOKEN" "https://example.com/api/user/$id"; done
```

→ Tests for Insecure Direct Object References

API Versioning Test:

```
for ver in v1 v2 v3 api old beta; do curl "https://example.com/$ver/endpoint"; done
```

→ Discovers different API versions

Mass Assignment Test:

```
curl -X POST -H "Content-Type: application/json" -d '{"email":"test@test.com","role":"admin"}'
https://example.com/api/register
```

→ Tests if additional fields can be injected

Rate Limit Bypass:

```
curl -H "X-Forwarded-For: 1.2.3.4" https://example.com/api
```

→ Attempts to bypass rate limiting with IP spoofing

Information Disclosure:

```
curl -I https://example.com | grep -i "server\|x-powered-by\|x-aspnet-version"
```

→ Checks for information leakage in headers

GraphQL Introspection:

```
curl -X POST -H "Content-Type: application/json" -d '{"query":"{ __schema { types { name } }
}"}' https://example.com/graphql
```

→ Attempts GraphQL introspection query

Open Redirect Test:

```
curl -I "https://example.com/redirect?url=https://evil.com"
```

→ Tests for open redirect vulnerabilities

USEFUL CURL OPTIONS REFERENCE

Option	Description
-A, --user-agent	Set User-Agent header
-b, --cookie	Send cookies from string/file
-c, --cookie-jar	Write cookies to file
-d, --data	Send POST data
-D, --dump-header	Write headers to file
-F, --form	Multipart form data
-H, --header	Custom header to include
-I, --head	Show document headers only
-k, --insecure	Allow insecure SSL connections
-L, --location	Follow redirects
-o, --output	Write output to file
-O, --remote-name	Write output to remote filename
-s, --silent	Silent mode
-u, --user	Server authentication
-v, --verbose	Make operation more talkative
-w, --write-out	Custom output format
-x, --proxy	Use proxy server
-X, --request	Specify request method
--compressed	Request compressed response
--connect-timeout	Maximum connection time
--http2	Use HTTP/2
--limit-rate	Limit transfer speed
--max-time	Maximum total time
--retry	Retry on transient errors

TIPS FOR BUG BOUNTY HUNTERS

1. Always use -v or -I to see full headers when testing

2. Save cookies with -c for session-based testing

3. Use -x with Burp Suite for detailed traffic analysis

4. Combine curl with tools like jq, grep, and awk for parsing

5. Script repetitive tests to save time

6. Use -k carefully - only on your test targets

7. Document all requests that produce interesting results

8. Test with different User-Agents to find hidden behavior

9. Always respect rate limits and scope

10. Use --resolve to test specific IPs without modifying /etc/hosts

LEGAL DISCLAIMER

This cheatsheet is intended for authorized security testing only. Always:

• Obtain proper authorization before testing
• Stay within the defined scope of your engagement
• Respect rate limits and system resources
• Follow responsible disclosure practices
• Comply with all applicable laws and regulations

Unauthorized access to computer systems is illegal. Use these techniques only on systems you own or have explicit permission to test.

Created for bug bounty hunters, red teamers, and ethical hackers.
Stay curious, stay legal, stay ethical.