2.9 curl – HTTP-, HTTPS-, API-, DNS-, Proxy- und Verbindungsdiagnose

Ziel dieser Seite

`curl` überträgt Daten anhand einer URL und eignet sich besonders zur Diagnose von Webdiensten und APIs. Administratoren können damit unter anderem prüfen:

- ob ein TCP-Dienst erreichbar ist;
- ob ein Webserver auf HTTP oder HTTPS antwortet;
- welchen HTTP-Statuscode ein Server zurückgibt;
- ob Weiterleitungen funktionieren;
- ob DNS-Auflösung und Zielserver zusammenpassen;
- ob TLS-Verbindungen und Zertifikatsprüfungen funktionieren;
- ob SNI und virtuelle Hosts korrekt verarbeitet werden;
- wie lange DNS, TCP-Verbindungsaufbau, TLS und Serverantwort dauern;
- ob ein Proxy Verbindungen erlaubt oder verändert;
- ob bestimmte HTTP-Header übertragen werden;
- ob eine API auf GET-, POST-, PUT-, PATCH- oder DELETE-Anfragen reagiert;
- ob IPv4 und IPv6 unterschiedliche Ergebnisse liefern.

> `curl` ist kein vollständiger Browser. JavaScript, grafische Darstellung, Browser-Erweiterungen und komplexe Browser-Sitzungen werden nicht wie in einem normalen Webbrowser ausgeführt.

Kennzeichnungen

Kennzeichnung	Bedeutung
`[RO]`	Lokale Informationsabfrage
`[TEST]`	Aktive Netzwerk- oder Anwendungsanfrage
`[PRIV]`	Erhöhte Berechtigungen können erforderlich sein
`[FILE]`	Befehl liest oder erstellt eine Datei
`[SENS]`	Anfrage oder Ausgabe kann sensible Informationen enthalten
`[CHANGE]`	Anfrage kann Daten oder einen Zustand verändern
`[DISRUPT]`	Anfrage kann einen Dienst oder produktive Daten beeinflussen

---

1. Wie wird curl unter Windows, Linux und macOS aufgerufen?

Betriebssystemübersicht anzeigen

Aufgabe	Windows	Linux	macOS
Programmpfad	`[RO] Get-Command curl.exe`	`[RO] command -v curl`	`[RO] command -v curl`
Version	`[RO] curl.exe --version`	`[RO] curl --version`	`[RO] curl --version`
Kurzhilfe	`[RO] curl.exe --help`	`[RO] curl --help`	`[RO] curl --help`
Gesamte Hilfe	`[RO] curl.exe --manual`	`[RO] curl --manual`	`[RO] curl --manual`
Lokales Handbuch	Nicht standardmäßig	`[RO] man curl`	`[RO] man curl`

Wichtig unter Windows PowerShell

In älteren Windows-PowerShell-Versionen kann `curl` als Alias für `Invoke-WebRequest` definiert sein. Dann verhält sich der Befehl nicht wie das echte curl-Programm.

Deshalb unter Windows eindeutig verwenden:

```
[RO] curl.exe --version
```

Alias prüfen:

```
[RO] Get-Command curl
```

Wenn als Befehlstyp `Alias` und als Ziel `Invoke-WebRequest` erscheint, muss für die in dieser Seite gezeigte Syntax ausdrücklich `curl.exe` verwendet werden.

Build-Funktionen prüfen

```
[RO] curl --version
```

Die Ausgabe zeigt unter anderem:

- curl-Version;
- verwendete TLS-Bibliothek;
- unterstützte Protokolle;
- unterstützte Funktionen;
- HTTP/2- oder HTTP/3-Unterstützung;
- IPv6-Unterstützung;

- Kompressionsunterstützung.

Nicht jede curl-Installation unterstützt alle Optionen und Protokolle. Entscheidend ist der lokal installierte Build.

---

2. Was passiert bei einem einfachen curl-Aufruf?

Grundlegende Anfrage anzeigen

Linux und macOS:

```
[TEST][SENS] curl https://example.com/
```

Windows:

```
[TEST][SENS] curl.exe https://example.com/
```

Ohne weitere Optionen schreibt curl den empfangenen Antwortinhalt auf die Standardausgabe.

Bei einer HTML-Seite erscheint daher der HTML-Quelltext:

```
<!doctype html>
<html>
...
</html>
```

Explizites URL-Schema verwenden

Empfohlen:

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

Nicht empfohlen:

```
example.com
```

Ohne Schema versucht curl, das Protokoll zu erraten. Für reproduzierbare Diagnosen sollte immer ausdrücklich `http://` oder `https://` angegeben werden.

URL in Anführungszeichen setzen

```
[TEST][SENS] curl 'https://example.com/search?q=test&lang=de'
```

Besonders Zeichen wie diese können von einer Shell interpretiert werden:

```
&
?
*
[
]
{
}
```

In PowerShell und klassischen Windows-Kommandozeilen können sich die Regeln für Anführungszeichen unterscheiden. Bei einfachen URLs funktionieren doppelte Anführungszeichen meist plattformübergreifend:

```
[TEST][SENS] curl "https://example.com/search?q=test&lang=de"
```

---

3. Wie werden HTTP-Header angezeigt?

Headeroptionen anzeigen

Nur Antwortheader anfordern

```
[TEST][SENS] curl -I https://example.com/
```

`-I` beziehungsweise `--head` veranlasst curl bei HTTP, eine `HEAD`-Anfrage zu senden.

Header und Antwortinhalt anzeigen

```
[TEST][SENS] curl -i https://example.com/
```

`-i` fügt die Antwortheader vor dem Inhalt ein.

Antwortheader separat auf die Standardausgabe schreiben

```
[TEST][SENS] curl -D - https://example.com/
```

Header in eine Datei schreiben

```
[TEST][FILE][SENS] curl -D response-headers.txt -o response-body.html https://example.com/
```

Wichtiger Unterschied

Option	Wirkung
`-I`	Sendet bei HTTP eine HEAD-Anfrage
`-i`	Zeigt Header und Antwortinhalt einer normalen Anfrage
`-D DATEI`	Schreibt empfangene Header separat
`-o DATEI`	Schreibt den Antwortinhalt in eine Datei

Nicht jeder Server behandelt `HEAD` und `GET` identisch. Ein erfolgreicher HEAD-Test beweist deshalb nicht sicher, dass ein GET-Aufruf denselben Status erhält.

GET ausführen, aber Antwortinhalt verwerfen

Linux und macOS:

```
[TEST][SENS] curl -sS -o /dev/null -D - https://example.com/
```

Windows:

```
[TEST][SENS] curl.exe -sS -o NUL -D - https://example.com/
```

Damit wird eine normale GET-Anfrage durchgeführt, ohne den vollständigen Inhalt im Terminal auszugeben.

---

4. Wie wird nur der HTTP-Statuscode ausgegeben?

Statuscode-Prüfung anzeigen

Linux und macOS:

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

Windows:

```
[TEST] curl.exe -sS -o NUL -w "%{http_code}\n" https://example.com/
```

Statuscode und Ziel-URL nach Weiterleitungen

Linux und macOS:

```
[TEST] curl -sS -L -o /dev/null -w "HTTP=%{http_code} URL=%{url_effective}\n" https://example.com/
```

Windows:

```
[TEST] curl.exe -sS -L -o NUL -w "HTTP=%{http_code} URL=%{url_effective}\n" https://example.com/
```

HTTP-Statusgruppen

Bereich	Bedeutung
100–199	Information
200–299	Erfolgreiche Verarbeitung
300–399	Weiterleitung
400–499	Anfrage- oder Clientfehler
500–599	Serverfehler

Häufige Statuscodes

Status	Bedeutung
200 OK	Anfrage erfolgreich
201 Created	Ressource erstellt
204 No Content	Erfolgreich, aber ohne Antwortinhalt
301 Moved Permanently	Dauerhafte Weiterleitung
302 Found	Temporäre Weiterleitung
304 Not Modified	Ressource seit Cacheprüfung nicht geändert

Status	Bedeutung
307 Temporary Redirect	Temporäre Weiterleitung unter Beibehaltung der Methode
308 Permanent Redirect	Dauerhafte Weiterleitung unter Beibehaltung der Methode
400 Bad Request	Anfrage syntaktisch oder fachlich ungültig
401 Unauthorized	Authentifizierung fehlt oder ist ungültig
403 Forbidden	Anfrage verstanden, Zugriff verweigert
404 Not Found	Ressource nicht gefunden
405 Method Not Allowed	HTTP-Methode nicht erlaubt
408 Request Timeout	Server wartete zu lange auf die Anfrage
409 Conflict	Konflikt mit aktuellem Ressourcenstatus
429 Too Many Requests	Rate Limit erreicht
500 Internal Server Error	Interner Serverfehler
502 Bad Gateway	Gateway oder Proxy erhielt ungültige Upstream-Antwort
503 Service Unavailable	Dienst momentan nicht verfügbar
504 Gateway Timeout	Gateway erhielt nicht rechtzeitig eine Upstream-Antwort

> Ein HTTP-Fehlercode bedeutet, dass die HTTP-Kommunikation grundsätzlich stattgefunden hat. DNS, TCP und normalerweise auch TLS waren bis zu diesem Punkt bereits erfolgreich.

---

5. Warum liefert curl bei einem HTTP-Fehler trotzdem Exitcode 0?

HTTP-Status und curl-Exitcode unterscheiden

Ohne `--fail` bewertet curl eine erfolgreich empfangene HTTP-Antwort als erfolgreiche Übertragung – auch wenn der Server beispielsweise `404` oder `500` zurückgibt.

```
HTTP-Statuscode 404
curl-Exitcode 0
```

Das bedeutet:

Übertragung technisch erfolgreich

Anwendung meldet HTTP-Fehler

Bei HTTP-Fehlern mit Exitcode ungleich 0 beenden

[TEST] curl --fail https://example.com/missing

Kurzform:

[TEST] curl -f https://example.com/missing

HTTP-Fehlercode liefern und Antwortinhalt behalten

[TEST][SENS] curl --fail-with-body https://example.com/missing

`--fail-with-body`:

- liefert bei HTTP-Status 400 oder höher einen curl-Fehler;
- behält den Server-Antwortinhalt bei;
- verwendet bei einem entsprechenden HTTP-Fehler normalerweise curl-Exitcode `22`.

Für Skripte häufig sinnvoll

[TEST] curl -sS --fail-with-body https://example.com/health

Option	Verhalten
`-s`	Fortschrittsanzeige unterdrücken
`-S`	Fehlermeldung trotz `-s` anzeigen
`--fail-with-body`	HTTP-Fehler als curl-Fehler behandeln und Inhalt behalten

---

6. Wie werden Weiterleitungen untersucht?

Redirect-Diagnose anzeigen

Nur erste Antwort anzeigen

```
[TEST][SENS] curl -I http://example.com/
```

Mögliche Ausgabe:

```
HTTP/1.1 301 Moved Permanently
Location: https://example.com/
```

Weiterleitungen automatisch verfolgen

```
[TEST][SENS] curl -L http://example.com/
```

Header aller Weiterleitungsstufen anzeigen

```
[TEST][SENS] curl -L -I http://example.com/
```

Maximale Anzahl Weiterleitungen begrenzen

```
[TEST][SENS] curl -L --max-redirs 5 https://example.com/
```

Effektive Ziel-URL ausgeben

Linux und macOS:

```
[TEST] curl -sS -L -o /dev/null -w "%{url_effective}\n" http://example.com/
```

Windows:

```
[TEST] curl.exe -sS -L -o NUL -w "%{url_effective}\n" http://example.com/
```

Typische Redirect-Probleme

- HTTP leitet nicht auf HTTPS weiter;
- Weiterleitung zeigt auf falschen Hostnamen;
- Endlosschleife zwischen zwei URLs;
- falscher Port in `Location`;
- interne Serveradresse wird offengelegt;
- Reverse Proxy erzeugt falsches Schema;

- Anwendung berücksichtigt `X-Forwarded-Proto` nicht;
- Authentifizierungscookie gilt nicht für das neue Ziel;
- Weiterleitung funktioniert im Browser wegen Cache, mit curl aber nicht.

> “ `--location-trusted` kann Zugangsdaten auch an andere Weiterleitungsziele weitergeben. Diese Option sollte aus Sicherheitsgründen nicht unüberlegt verwendet werden.

---

7. Wie wird eine ausführliche Verbindungsdiagnose durchgeführt?

Verbose-Ausgabe `-v` anzeigen

```
[TEST][SENS] curl -v https://example.com/
```

Die ausführliche Ausgabe enthält unter anderem:

- DNS-Ergebnis;
- ausgewählte IP-Adresse;
- TCP-Verbindungsaufbau;
- TLS-Verhandlung;
- Zertifikatsinformationen;
- ausgehandeltes Protokoll;
- gesendete HTTP-Header;
- empfangene HTTP-Header;
- Weiterleitungsinformationen;
- Wiederverwendung einer Verbindung.

Kennzeichnungen in der Ausgabe

Zeichen	Bedeutung
`*`	curl-interne Status- oder Verbindungsinformation
`>`	Von curl an den Server gesendete Daten beziehungsweise Header
`<`	Vom Server empfangene Daten beziehungsweise Header
`{` oder `}`	In bestimmten Trace-Ausgaben übertragene Daten

Beispiel:

```
> GET / HTTP/1.1
> Host: example.com
> User-Agent: curl/...
> Accept: */*
```

```
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 1256
```

Nur Header und Verbindungsinformationen, Inhalt verwerfen

Linux und macOS:

```
[TEST][SENS] curl -v -o /dev/null https://example.com/
```

Windows:

```
[TEST][SENS] curl.exe -v -o NUL https://example.com/
```

> Die Verbose-Ausgabe kann Zugangsdaten, Cookies, API-Token oder interne Header enthalten. Sie muss vor einer Weitergabe geprüft und bereinigt werden.

---

8. Wie wird eine detaillierte Ablaufverfolgung erstellt?

Trace-Optionen anzeigen

ASCII-Trace in eine Datei schreiben

```
[TEST][FILE][SENS] curl --trace-ascii curl-trace.txt https://example.com/
```

Trace mit Zeitstempeln

```
[TEST][FILE][SENS] curl --trace-time --trace-ascii curl-trace.txt https://example.com/
```

Binären Trace schreiben

```
[TEST][FILE][SENS] curl --trace curl-trace.bin https://example.com/
```

Ein Trace kann deutlich mehr Informationen als `-v` enthalten, darunter:

- vollständige Header;
- Nutzdaten;
- Cookies;
- Formulardaten;
- Authentifizierungsinformationen;
- API-Antworten;
- interne Adressen.

Trace-Dateien sind daher als sensible Diagnosedaten zu behandeln.

Nicht gleichzeitig unkontrolliert verwenden

```
-v
--trace
--trace-ascii
```

`--trace` beziehungsweise `--trace-ascii` überschreibt die frühere Auswahl der Trace-Ausgabe. Für eine reproduzierbare Diagnose sollte nur die tatsächlich benötigte Variante verwendet werden.

---

9. Wie werden DNS-, TCP-, TLS- und Serverzeiten gemessen?

Zeitmessung mit `--write-out` anzeigen

Linux und macOS:

```
[TEST] curl -sS -o /dev/null -w "DNS=%{time_namelookup}s TCP=%{time_connect}s
TLS=%{time_appconnect}s TTFB=%{time_starttransfer}s Gesamt=%{time_total}s
HTTP=%{http_code}\n" https://example.com/
```

Windows:

```
[TEST] curl.exe -sS -o NUL -w "DNS=%{time_namelookup}s TCP=%{time_connect}s
TLS=%{time_appconnect}s TTFB=%{time_starttransfer}s Gesamt=%{time_total}s
HTTP=%{http_code}\n" https://example.com/
```

Bedeutung der Zeitwerte

Variable	Bedeutung
`time_namelookup`	Zeit bis zum Abschluss der Namensauflösung
`time_connect`	Zeit vom Start bis zur hergestellten TCP-Verbindung
`time_appconnect`	Zeit bis zum Abschluss des TLS- oder anderen Anwendungs-Handshakes
`time_pretransfer`	Zeit bis unmittelbar vor Beginn der Übertragung
`time_starttransfer`	Zeit bis zum ersten empfangenen Antwortbyte
`time_redirect`	Gesamtdauer vorheriger Weiterleitungen
`time_total`	Gesamtdauer der Übertragung
`http_code`	Letzter empfangener HTTP-Statuscode
`remote_ip`	Tatsächlich verwendete Ziel-IP-Adresse
`remote_port`	Tatsächlich verwendeter Zielport
`local_ip`	Verwendete lokale IP-Adresse
`num_redirects`	Anzahl verfolgter Weiterleitungen
`url_effective`	Effektive URL nach Weiterleitungen

Erweiterte Messung

Linux und macOS:

```
[TEST] curl -sS -L -o /dev/null -w "Lokal=%{local_ip} Remote=%{remote_ip}:%{remote_port}
DNS=%{time_namelookup}s TCP=%{time_connect}s TLS=%{time_appconnect}s
TTFB=%{time_starttransfer}s Redirect=%{time_redirect}s Gesamt=%{time_total}s Status=%{http_code}
Redirects=%{num_redirects} URL=%{url_effective}\n" https://example.com/
```

Windows:

```
[TEST] curl.exe -sS -L -o NUL -w "Lokal=%{local_ip} Remote=%{remote_ip}:%{remote_port}
DNS=%{time_namelookup}s TCP=%{time_connect}s TLS=%{time_appconnect}s
```

```
TTFB=%{time_starttransfer}s Redirect=%{time_redirect}s Gesamt=%{time_total}s Status=%{http_code}
Redirects=%{num_redirects} URL=%{url_effective}\n" https://example.com/
```

---

10. Wie werden Zeitmessungen richtig interpretiert?

Zeitanteile einordnen

Die Zeitvariablen sind überwiegend kumulativ seit Beginn des curl-Aufrufs. Für einzelne Phasen müssen Differenzen gebildet werden.

DNS-Dauer

```
DNS = time_namelookup
```

TCP-Aufbau nach DNS

```
TCP-Phase = time_connect - time_namelookup
```

TLS-Handshake nach TCP-Aufbau

```
TLS-Phase = time_appconnect - time_connect
```

Serververarbeitung bis zum ersten Byte

Bei HTTPS vereinfacht:

```
Server-/Anwendungsphase = time_starttransfer - time_appconnect
```

Übertragung nach dem ersten Byte

```
Downloadphase = time_total - time_starttransfer
```

Beispiel

Messwert	Wert
`time_namelookup`	0,020 s

Messwert	Wert
time_connect	0,050 s
time_appconnect	0,120 s
time_starttransfer	0,420 s
time_total	0,500 s

Daraus folgt:

```
DNS = 0,020 s
TCP = 0,050 - 0,020 = 0,030 s
TLS = 0,120 - 0,050 = 0,070 s
Server bis erstes Byte = 0,420 - 0,120 = 0,300 s
Restliche Übertragung = 0,500 - 0,420 = 0,080 s
```

Diese Zuordnung ist eine Diagnosehilfe. Proxys, wiederverwendete Verbindungen, Weiterleitungen und unterschiedliche Protokolle können die Interpretation verändern.

---

11. Wie werden Verbindungs- und Gesamtzeit begrenzt?

Timeouts anzeigen

Maximal fünf Sekunden für den Verbindungsaufbau

```
[TEST] curl --connect-timeout 5 https://example.com/
```

Maximal 15 Sekunden für den gesamten Vorgang

```
[TEST] curl --max-time 15 https://example.com/
```

Kurzform:

```
[TEST] curl -m 15 https://example.com/
```

Kombination

```
[TEST] curl --connect-timeout 5 --max-time 15 https://example.com/
```

Option	Begrenzter Bereich
`--connect-timeout`	Verbindungsphase einschließlich notwendiger DNS-, TCP- und gegebenenfalls Proxy-/TLS-Vorgänge bis zur Verbindung
`--max-time`	Gesamter curl-Vorgang

Langsame Übertragung abbrechen

```
[TEST] curl --speed-limit 1000 --speed-time 10 https://example.com/large-file
```

Der Vorgang wird abgebrochen, wenn die Übertragungsrate während des festgelegten Zeitraums unter dem Grenzwert liegt.

> Zu kurze Timeouts können langsame, aber funktionierende Verbindungen fälschlich als Fehler erscheinen lassen.

---

12. Wie werden IPv4 und IPv6 getrennt geprüft?

IPv4-/IPv6-Vergleich anzeigen

IPv4 erzwingen

```
[TEST] curl -4 -v https://example.com/
```

IPv6 erzwingen

```
[TEST] curl -6 -v https://example.com/
```

Nur Status und Ziel-IP vergleichen

Linux und macOS:

```
[TEST] curl -4 -sS -o /dev/null -w "IPv4=%{remote_ip} HTTP=%{http_code} Zeit=%{time_total}s\n" https://example.com/
```

```
[TEST] curl -6 -sS -o /dev/null -w "IPv6=%{remote_ip} HTTP=%{http_code} Zeit=%{time_total}s\n" https://example.com/
```

Windows:

```
[TEST] curl.exe -4 -sS -o NUL -w "IPv4=%{remote_ip} HTTP=%{http_code} Zeit=%{time_total}s\n" https://example.com/
```

```
[TEST] curl.exe -6 -sS -o NUL -w "IPv6=%{remote_ip} HTTP=%{http_code} Zeit=%{time_total}s\n" https://example.com/
```

Interpretation

IPv4	IPv6	Mögliche Ursache
Funktioniert	Funktioniert nicht	IPv6-DNS, Routing, Firewall, Neighbor Discovery oder MTU
Langsam	Schnell	Unterschiedlicher Netzwerkpfad oder unterschiedliche Gegenstelle
Schnell	Langsam	IPv6-Pfad, Tunnel oder Zielserver prüfen
Unterschiedliche Inhalte	DNS-/CDN-/Proxy-Zuordnung untersuchen	

---

13. Wie wird ein Hostname gezielt gegen eine bestimmte IP-Adresse getestet?

DNS umgehen, Hostname und TLS-SNI erhalten

Für HTTPS muss der Hostname normalerweise gleichzeitig für diese Funktionen erhalten bleiben:

- HTTP-`Host`-Header;
- TLS-SNI;
- Zertifikatsprüfung;

- virtuelle Hostauswahl.

Dafür eignet sich `--resolve`.

```
[TEST][SENS] curl --resolve example.com:443:192.0.2.20 https://example.com/
```

Dieser Befehl bedeutet:

```
Hostname in URL: example.com
Zielport: 443
Tatsächliche Ziel-IP: 192.0.2.20
HTTP-Host: example.com
TLS-SNI: example.com
Zertifikatsname: example.com
```

Ausführlicher Test

```
[TEST][SENS] curl -v --resolve example.com:443:192.0.2.20 https://example.com/
```

HTTP ohne TLS

```
[TEST][SENS] curl --resolve example.com:80:192.0.2.20 http://example.com/
```

Warum nicht nur die IP-Adresse aufrufen?

```
curl https://192.0.2.20/
```

Dabei verwendet curl die IP-Adresse als Zielnamen. Das kann verursachen:

- falschen virtuellen Host;
- falsches TLS-SNI;
- Zertifikatsnamensfehler;
- andere Serverantwort;
- Standardseite des Reverse Proxys.

Nur einen Host-Header setzen

```
curl -H "Host: example.com" https://192.0.2.20/
```

setzt zwar den HTTP-Host-Header, aber nicht automatisch das passende TLS-SNI für `example.com`. Für HTTPS ist `--resolve` deshalb in der Regel die richtige Diagnoseoption.

---

14. Was ist der Unterschied zwischen `--resolve` und `--connect-to`?

Zielumleitung erklären

`--resolve`

```
[TEST][SENS] curl --resolve example.com:443:192.0.2.20 https://example.com/
```

`--resolve` fügt für die angegebene Host-/Portkombination eine temporäre Namenszuordnung hinzu.

`--connect-to`

```
[TEST][SENS] curl --connect-to example.com:443:192.0.2.20:8443 https://example.com/
```

Damit verbindet sich curl tatsächlich mit:

```
192.0.2.20:8443
```

Die ursprüngliche URL bleibt:

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

Dadurch bleiben insbesondere:

- URL-Hostname;
- TLS-SNI;
- Zertifikatsprüfung;
- HTTP-Host

auf `example.com` bezogen.

Typischer Einsatz

- neuen Reverse Proxy vor DNS-Umschaltung testen;
- Backend auf einem anderen Port prüfen;

- Load-Balancer-Knoten gezielt untersuchen;
- Blue-Green-Deployment vergleichen;
- fehlerhaften DNS-Eintrag umgehen, ohne Hostname und SNI zu verlieren.

Beide Optionen gelten nur für den jeweiligen curl-Aufruf und verändern nicht dauerhaft die lokale DNS-Konfiguration.

---

15. Wie werden TLS und Zertifikate geprüft?

TLS-Diagnose anzeigen

Normale Zertifikatsprüfung

```
[TEST][SENS] curl -v https://example.com/
```

curl prüft dabei abhängig vom verwendeten TLS-Backend unter anderem:

- Vertrauenskette;
- Gültigkeitszeitraum;
- Hostname;
- unterstützte TLS-Versionen;
- Zertifizierungsstelle;
- TLS-Handshake.

Bestimmte CA-Datei verwenden

```
[TEST][FILE][SENS] curl --cacert company-ca.pem https://internal.example.com/
```

TLS 1.2 oder höher anfordern

```
[TEST][SENS] curl --tlsv1.2 https://example.com/
```

`--tlsv1.2` legt die minimale TLS-Version auf 1.2 fest. Neuere Versionen können weiterhin ausgehandelt werden.

Maximal TLS 1.2

```
[TEST][SENS] curl --tlsv1.2 --tls-max 1.2 https://example.com/
```

TLS 1.3 oder höher anfordern

```
[TEST][SENS] curl --tlsv1.3 https://example.com/
```

Die Unterstützung hängt von curl-Version und TLS-Bibliothek ab.

Clientzertifikat

```
[TEST][FILE][SENS] curl --cert client.pem --key client.key https://example.com/
```

Format und Optionen können vom verwendeten TLS-Backend abhängen.

Zertifikatsprüfung deaktivieren

```
[TEST][SENS] curl -k https://example.com/
```

`-k` beziehungsweise `--insecure` deaktiviert die normale Echtheitsprüfung. Dadurch kann curl keine vertrauenswürdige Identität des Servers garantieren.

> `-k` darf höchstens als klar gekennzeichneter Vergleichstest verwendet werden. Es ist keine Lösung für ein Zertifikatsproblem.

Sinnvolle Interpretation

Normaler Test	Mit `-k`	Mögliche Ursache
Fehler	Erfolgreich	Zertifikatsvertrauen, Hostname oder Zertifikatskette prüfen
Fehler	Fehler	Problem wahrscheinlich nicht nur Zertifikatsprüfung
Erfolgreich	Erfolgreich	Kein Grund, `-k` dauerhaft zu verwenden

---

16. Wie werden HTTP-Versionen geprüft?

HTTP/1.1, HTTP/2 und HTTP/3 anzeigen

HTTP/1.1 anfordern

```
[TEST][SENS] curl --http1.1 -v https://example.com/
```

HTTP/2 anfordern

```
[TEST][SENS] curl --http2 -v https://example.com/
```

HTTP/3 versuchen

```
[TEST][SENS] curl --http3 -v https://example.com/
```

Ausschließlich HTTP/3

```
[TEST][SENS] curl --http3-only -v https://example.com/
```

Die Optionen funktionieren nur, wenn der installierte curl-Build die jeweilige Protokollversion unterstützt:

```
[RO] curl --version
```

Interpretation

Beobachtung	Mögliche Ursache
HTTP/1.1 funktioniert, HTTP/2 nicht	Proxy, TLS-ALPN, Server- oder curl-Buildproblem
HTTP/2 funktioniert, HTTP/3 nicht	QUIC/UDP, Firewall, Serverunterstützung oder curl-Build
HTTP/3 funktioniert nur ohne VPN	VPN oder Firewall blockiert beziehungsweise beeinträchtigt UDP
Unterschiedliche Antworten	Reverse Proxy oder Anwendung behandelt Protokolle unterschiedlich

HTTP/3 verwendet QUIC über UDP. Ein erfolgreicher HTTPS-Test über TCP beweist deshalb nicht, dass HTTP/3 erreichbar ist.

---

17. Wie werden eigene Request-Header gesetzt?

Header mit `-H` anzeigen

Accept-Header setzen

```
[TEST][SENS] curl -H "Accept: application/json" https://api.example.com/status
```

Benutzerdefinierten Header setzen

```
[TEST][SENS] curl -H "X-Diagnostic-ID: test-001" https://example.com/
```

Mehrere Header

```
[TEST][SENS] curl -H "Accept: application/json" -H "X-Diagnostic-ID: test-001"
https://api.example.com/status
```

User-Agent setzen

```
[TEST][SENS] curl -A "IT-Diagnose/1.0" https://example.com/
```

Alternativ:

```
[TEST][SENS] curl -H "User-Agent: IT-Diagnose/1.0" https://example.com/
```

Header entfernen

```
[TEST][SENS] curl -H "User-Agent:" https://example.com/
```

Ein Header mit leerem Wert nach dem Doppelpunkt wird entfernt.

Vorsicht

Benutzerdefinierte Header können:

- Routing im Reverse Proxy verändern;
- Authentifizierung beeinflussen;
- Caches umgehen;
- Sicherheitsregeln auslösen;
- sensible Informationen enthalten.

Header aus Browser-Entwicklerwerkzeugen dürfen nicht ungeprüft übernommen werden. Insbesondere Cookies, Tokens und Sitzungskennungen müssen geschützt werden.

18. Wie werden GET-Parameter korrekt übertragen?

Query-Parameter anzeigen

Parameter direkt in der URL

```
[TEST][SENS] curl "https://api.example.com/search?q=server&limit=10"
```

Parameter URL-codieren

```
[TEST][SENS] curl -G --data-urlencode "q=Server Fehleranalyse" --data-urlencode "limit=10"
https://api.example.com/search
```

`-G` sorgt dafür, dass die mit `--data-urlencode` angegebenen Daten als URL-Query und nicht als POST-Body verwendet werden.

Ergebnis sinngemäß:

```
https://api.example.com/search?q=Server%20Fehleranalyse&limit=10
```

Warum URL-Encoding wichtig ist

Zeichen wie diese besitzen in URLs eine besondere Bedeutung:

```
Leerzeichen
&
=
?
#
+
%
```

`--data-urlencode` verhindert viele Fehler durch nicht korrekt codierte Werte.

---

19. Wie werden POST-, PUT-, PATCH- und DELETE-Anfragen getestet?

HTTP-Methoden und API-Beispiele anzeigen

> Ändernde API-Anfragen dürfen nur gegen ausdrücklich freigegebene Testressourcen ausgeführt werden.

POST mit Formulardaten

```
[TEST][CHANGE][SENS] curl -X POST -d "name=Max Mustermann" -d "active=true" https://api.example.com/users
```

Bei Verwendung von `-d` wählt curl bei HTTP automatisch POST, sofern keine andere Methode angegeben wird. `-X POST` ist deshalb häufig nicht erforderlich:

```
[TEST][CHANGE][SENS] curl -d "name=Max Mustermann" -d "active=true" https://api.example.com/users
```

POST mit JSON

```
[TEST][CHANGE][SENS] curl -H "Content-Type: application/json" -d '{"name":"Max Mustermann","active":true}' https://api.example.com/users
```

Neuere curl-Versionen unterstützen:

```
[TEST][CHANGE][SENS] curl --json '{"name":"Max Mustermann","active":true}' https://api.example.com/users
```

`--json` setzt geeignete JSON-Header und verwendet die Daten als Request-Body. Die Verfügbarkeit hängt von der curl-Version ab.

JSON aus Datei

```
[TEST][FILE][CHANGE][SENS] curl -H "Content-Type: application/json" --data-binary @request.json https://api.example.com/users
```

PUT

[TEST][CHANGE][SENS] curl -X PUT -H "Content-Type: application/json" --data-binary @request.json https://api.example.com/users/123

PATCH

[TEST][CHANGE][SENS] curl -X PATCH -H "Content-Type: application/json" -d '{"active":false}' https://api.example.com/users/123

DELETE

[TEST][CHANGE][DISRUPT][SENS] curl -X DELETE https://api.example.com/users/123

OPTIONS

[TEST][SENS] curl -i -X OPTIONS https://api.example.com/users

`OPTIONS` kann Hinweise auf erlaubte Methoden oder CORS-Header liefern. Server müssen jedoch nicht alle Fähigkeiten vollständig darüber offenlegen.

---

20. Was ist der Unterschied zwischen `-d` und `--data-binary`?

Übertragung von Request-Daten erklären

Option	Verhalten
`-d` beziehungsweise `--data`	Sendet HTTP-Daten; Zeilenenden und Dateieinlesung folgen den Regeln dieser Option
`--data-raw`	Wie `--data` aber `@` wird nicht als Dateiverweis behandelt
`--data-binary`	Überträgt Daten weitgehend unverändert
`--data-urlencode`	URL-codiert die Daten
`--json`	Sendet JSON und setzt passende Header

Textdaten

[TEST][CHANGE][SENS] curl -d "name=Max Mustermann" https://api.example.com/users

Datei binär beziehungsweise unverändert senden

```
[TEST][FILE][CHANGE][SENS] curl --data-binary @request.json https://api.example.com/import
```

Wörtliches `@` senden

```
[TEST][CHANGE][SENS] curl --data-raw "@example" https://api.example.com/test
```

Bei `-d @datei` interpretiert curl den Wert als Dateipfad. Bei `--data-raw` wird ein führendes `@` dagegen wörtlich übertragen.

---

21. Wie werden Dateien hoch- und heruntergeladen?

Dateitransfer anzeigen

Antwort in benannte Datei schreiben

```
[TEST][FILE][SENS] curl -o download.bin https://example.com/file.bin
```

Remote-Dateinamen verwenden

```
[TEST][FILE][SENS] curl -O https://example.com/file.bin
```

Weiterleitung verfolgen und Remote-Dateinamen verwenden

```
[TEST][FILE][SENS] curl -L -O https://example.com/file.bin
```

Teilweise Übertragung fortsetzen

```
[TEST][FILE][SENS] curl -C - -O https://example.com/file.bin
```

Download bei Fehler entfernen

```
[TEST][FILE][SENS] curl --fail --remove-on-error -o download.bin https://example.com/file.bin
```

Datei mit PUT hochladen

```
[TEST][FILE][CHANGE][SENS] curl -T upload.bin https://example.com/upload.bin
```

Multipart-Formularupload

```
[TEST][FILE][CHANGE][SENS] curl -F "file=@upload.bin" https://example.com/upload
```

Vor einem Upload müssen Ziel, Methode, Überschreibverhalten und Berechtigung geprüft werden.

---

22. Wie wird eine HTTP-Authentifizierung getestet?

Basic, Digest, Bearer und Negotiate anzeigen

Benutzername angeben und Passwort interaktiv abfragen

```
[TEST][SENS] curl -u "max.mustermann" https://example.com/protected
```

curl fordert das Passwort interaktiv an.

Benutzername und Passwort direkt angeben

```
[TEST][SENS] curl -u "max.mustermann:BEISPIELPASSWORT" https://example.com/protected
```

Diese Schreibweise ist für reale Zugangsdaten nicht empfohlen, weil das Passwort:

- in der Shell-Historie;
- in Prozessinformationen;
- in Bildschirmaufzeichnungen;
- in Supportprotokollen

sichtbar werden kann.

Basic Authentication ausdrücklich verwenden

```
[TEST][SENS] curl --basic -u "max.mustermann" https://example.com/protected
```

Digest Authentication

```
[TEST][SENS] curl --digest -u "max.mustermann" https://example.com/protected
```

Unterstützte Methode automatisch auswählen

```
[TEST][SENS] curl --anyauth -u "max.mustermann" https://example.com/protected
```

`--anyauth` kann eine zusätzliche Anfrage erzeugen, um die angebotenen Methoden zu erkennen.

Bearer-Token

```
[TEST][SENS] curl -H "Authorization: Bearer TOKEN_NICHT_HIER_EINTRAGEN"
https://api.example.com/status
```

Negotiate/Kerberos, sofern Build und Umgebung es unterstützen

```
[TEST][SENS] curl --negotiate -u : https://example.com/protected
```

> Zugangsdaten, Tokens und Cookies dürfen nicht in BookStack-Seiten, Tickets oder allgemein lesbaren Skripten gespeichert werden.

---

23. Wie werden Cookies untersucht?

Cookie-Befehle anzeigen

Cookie direkt senden

```
[TEST][SENS] curl -b "session=BEISPIELWERT" https://example.com/
```

Cookies aus Datei lesen

```
[TEST][FILE][SENS] curl -b cookies.txt https://example.com/
```

Empfangene Cookies speichern

```
[TEST][FILE][SENS] curl -c cookies.txt https://example.com/login
```

Cookies lesen und aktualisiert wieder speichern

```
[TEST][FILE][SENS] curl -b cookies.txt -c cookies.txt https://example.com/
```

Option	Bedeutung
`-b` beziehungsweise `--cookie`	Cookies senden oder aus Datei lesen
`-c` beziehungsweise `--cookie-jar`	Empfangene Cookies speichern

Cookie-Dateien können aktive Sitzungen ermöglichen und sind daher sensible Zugangsdaten.

Typische Cookie-Probleme

- falsche Domain;
- falscher Pfad;
- `Secure`-Cookie über HTTP;
- abgelaufenes Cookie;
- SameSite-Verhalten;
- Weiterleitung zu anderem Host;
- mehrere Cookies mit gleichem Namen;
- Anwendung erwartet zusätzliche CSRF-Token.

curl bildet nicht jedes Browser-Sicherheitsverhalten vollständig identisch ab. Ein mit curl funktionierender Cookie-Test beweist daher nicht automatisch, dass eine Browseranwendung fehlerfrei arbeitet.

---

24. Wie wird ein HTTP-Proxy getestet?

Proxy-Diagnose anzeigen

HTTP-Proxy verwenden

```
[TEST][SENS] curl -x http://proxy.example.com:8080 https://example.com/
```

Langform:

```
[TEST][SENS] curl --proxy http://proxy.example.com:8080 https://example.com/
```

Proxy-Benutzername interaktiv verwenden

```
[TEST][SENS] curl -x http://proxy.example.com:8080 -U "max.mustermann" https://example.com/
```

Proxy umgehen

```
[TEST][SENS] curl --noproxy example.com https://example.com/
```

Proxy für alle Ziele umgehen

```
[TEST][SENS] curl --noproxy "*" https://example.com/
```

Umgebungsvariablen prüfen

Linux und macOS:

```
[RO][SENS] env | grep -i proxy
```

PowerShell:

```
[RO][SENS] Get-ChildItem Env: | Where-Object Name -Match 'proxy'
```

Häufig verwendete Variablen:

```
http_proxy
https_proxy
all_proxy
NO_PROXY
no_proxy
```

Wichtiger Sicherheitshinweis

Die Variable `http_proxy` wird von curl aus Sicherheitsgründen nur in Kleinschreibung akzeptiert. Andere Proxyvariablen können je nach Protokoll und Umgebung unterschiedliche Groß-/Kleinschreibungen unterstützen.

Direkt- und Proxytest vergleichen

Direkt:

```
[TEST][SENS] curl --noproxy "*" -v https://example.com/
```

Über Proxy:

```
[TEST][SENS] curl -x http://proxy.example.com:8080 -v https://example.com/
```

---

25. Wie wird ein SOCKS-Proxy getestet?

SOCKS4- und SOCKS5-Optionen anzeigen

SOCKS5-Proxy, DNS-Auflösung durch den Proxy

```
[TEST][SENS] curl --socks5-hostname 192.0.2.50:1080 https://example.com/
```

SOCKS5-Proxy, lokale DNS-Auflösung

```
[TEST][SENS] curl --socks5 192.0.2.50:1080 https://example.com/
```

SOCKS4a

```
[TEST][SENS] curl --socks4a 192.0.2.50:1080 https://example.com/
```

Wichtiger Unterschied

Option	DNS-Auflösung
`--socks5`	Lokal durch curl
`--socks5-hostname`	Durch den SOCKS5-Proxy
`--socks4`	Lokal
`--socks4a`	Durch den Proxy

Der Unterschied ist wichtig, wenn:

- interne DNS-Namen nur am Proxy auflösbar sind;
- DNS-Leaks vermieden werden sollen;
- lokale und entfernte DNS-Ergebnisse voneinander abweichen.

---

26. Wie werden komprimierte Antworten geprüft?

Kompression anzeigen

```
[TEST][SENS] curl --compressed -v https://example.com/
```

Mit `--compressed` fordert curl eine unterstützte komprimierte Antwort an und dekomprimiert sie anschließend.

In der Anfrage kann beispielsweise erscheinen:

```
Accept-Encoding: deflate, gzip, br, zstd
```

Die tatsächlich angebotenen Verfahren hängen vom curl-Build ab.

Header prüfen

```
[TEST][SENS] curl --compressed -I https://example.com/
```

Mögliche Antwort:

```
Content-Encoding: gzip
```

Typische Probleme

- Proxy entfernt `Accept-Encoding`;
- Server sendet falschen `Content-Length`;
- Inhalt wird doppelt komprimiert;
- Reverse Proxy und Anwendung komprimieren gleichzeitig;
- bestimmte Kompressionsverfahren werden vom curl-Build nicht unterstützt.

---

27. Wie werden lokale Unix-Sockets getestet?

Unix-Domain-Socket-Diagnose anzeigen

Auf Linux und anderen Unix-Systemen können HTTP-Dienste über einen Unix-Domain-Socket erreichbar sein.

```
[TEST][SENS] curl --unix-socket /run/example/app.sock http://localhost/health
```

curl verbindet sich dabei mit dem lokalen Socket. Der Hostname in der URL wird weiterhin für die HTTP-Anfrage verwendet.

Typische Einsatzbereiche

- Docker Engine API;
- lokale Reverse-Proxy-Backends;
- PHP-FPM-nahe Testdienste;
- lokale Verwaltungs-APIs;
- systemd-aktivierte Dienste.

Beispiel Docker-Socket

```
[TEST][PRIV][SENS] curl --unix-socket /var/run/docker.sock http://localhost/_ping
```

> Zugriff auf `/var/run/docker.sock` entspricht auf vielen Systemen weitreichenden administrativen Rechten. Dieser Zugriff darf nicht leichtfertig vergeben oder in Container durchgereicht werden.

---

28. Wie werden Wiederholungsversuche verwendet?

Retry-Optionen anzeigen

Bis zu drei Wiederholungsversuche

```
[TEST][SENS] curl --retry 3 https://example.com/
```

Maximale Gesamtzeit für Wiederholungen

```
[TEST][SENS] curl --retry 3 --retry-max-time 30 https://example.com/
```

Feste Verzögerung zwischen Versuchen

```
[TEST][SENS] curl --retry 3 --retry-delay 2 https://example.com/
```

Auch abgelehnte TCP-Verbindungen erneut versuchen

```
[TEST][SENS] curl --retry 3 --retry-connrefused https://example.com/
```

Alle Fehler wiederholen

```
[TEST][CHANGE][DISRUPT][SENS] curl --retry 3 --retry-all-errors https://example.com/
```

`--retry-all-errors` darf bei ändernden Anfragen nicht unüberlegt verwendet werden. Ein POST, PUT oder DELETE könnte auf dem Server bereits verarbeitet worden sein, obwohl curl die Antwort nicht erhalten hat.

Dadurch könnte eine Wiederholung:

- eine Ressource doppelt anlegen;
- eine Buchung doppelt ausführen;
- eine Nachricht mehrfach senden;
- einen Löschvorgang wiederholen.

Wiederholungsversuche sind bei idempotenten GET- oder HEAD-Anfragen meist leichter zu bewerten als bei ändernden API-Aufrufen.

---

29. Welche curl-Exitcodes sind für die Diagnose wichtig?

Exitcode-Tabelle anzeigen

Exitcode	Bedeutung
0	Vorgang aus Sicht von curl erfolgreich
3	URL fehlerhaft
5	Proxyname konnte nicht aufgelöst werden

Exitcode	Bedeutung
6	Hostname konnte nicht aufgelöst werden
7	Verbindung zum Ziel konnte nicht hergestellt werden
22	HTTP-Fehler bei Verwendung von `--fail` oder `--fail-with-body`
23	Fehler beim Schreiben empfangener Daten
26	Fehler beim Lesen lokaler Daten
28	Zeitüberschreitung
35	TLS-/SSL-Verbindungsfehler
47	Zu viele Weiterleitungen
52	Leere Serverantwort
55	Fehler beim Senden von Netzwerkdaten
56	Fehler beim Empfangen von Netzwerkdaten
60	Zertifikat konnte nicht verifiziert werden
77	Problem beim Lesen der CA-Zertifikatsdatei
92	HTTP/2-Protokollfehler

Die vollständige Liste hängt von der curl-Version ab:

```
[RO] curl --manual
```

Exitcode unter Linux und macOS anzeigen

```
curl -sS --fail-with-body https://example.com/health
echo $?
```

Exitcode unter PowerShell anzeigen

```
curl.exe -sS --fail-with-body https://example.com/health
$LASTEXITCODE
```

Wichtige Unterscheidung

HTTP-Statuscode

≠

curl-Exitcode

Beispiel:

HTTP 404 ohne --fail → curl-Exitcode kann 0 sein

HTTP 404 mit --fail → curl-Exitcode 22

---

30. Wie werden Fehler nach Diagnoseebene eingeordnet?

Fehlerkette anzeigen

curl-Beobachtung	Wahrscheinliche Ebene
Exitcode 6	DNS-Auflösung
Exitcode 7	TCP-Verbindung, Routing, Firewall oder Listener
Exitcode 28	DNS, Verbindung, Server oder Übertragung zu langsam
Exitcode 35	TLS-Handshake
Exitcode 60	Zertifikatsvertrauen oder Hostname
HTTP 301/302	Weiterleitung
HTTP 401	Authentifizierung
HTTP 403	Autorisierung, WAF oder Richtlinie
HTTP 404	URL, Routing oder Ressource
HTTP 429	Rate Limit
HTTP 500	Anwendung oder Backend
HTTP 502	Proxy/Gateway und Upstream
HTTP 503	Dienst nicht verfügbar oder Wartungszustand
HTTP 504	Gateway wartet vergeblich auf Upstream
Verbindung funktioniert, Antwort langsam	Anwendung, Datenbank, Upstream oder Serverlast
IP funktioniert, Name nicht	DNS
`--resolve` funktioniert, normaler Aufruf nicht	DNS-Zuordnung oder Load-Balancer-Ziel
`-k` funktioniert, normaler Test nicht	Zertifikatsprüfung

curl-Beobachtung	Wahrscheinliche Ebene
IPv4 funktioniert, IPv6 nicht	IPv6-DNS, Routing, Firewall oder MTU
Direkt funktioniert, Proxy nicht	Proxy, Authentifizierung oder Proxy-Richtlinie

---

31. Welche typischen Fehlinterpretationen gibt es?

Praxisfallen anzeigen

Fehlinterpretation	Richtige Einordnung
`curl` ohne Fehler bedeutet HTTP 200	Ohne `--fail` kann auch HTTP 404 oder 500 Exitcode 0 ergeben
`-I` prüft exakt dasselbe wie GET	`-I` sendet bei HTTP eine HEAD-Anfrage
`-k` behebt TLS	Es deaktiviert nur die Zertifikatsprüfung
Aufruf über IP testet denselben virtuellen Host	Host-Header, SNI und Zertifikatsprüfung können abweichen
Eigener `Host`-Header setzt automatisch SNI	TLS-SNI wird dadurch nicht zuverlässig angepasst
`--resolve` verändert lokales DNS dauerhaft	Gilt nur für den jeweiligen curl-Aufruf
`time_connect` ist nur die TCP-Dauer	Wert ist kumulativ seit Beginn des Aufrufs
Hohe TTFB beweist Netzwerkproblem	Server, Anwendung oder Backend können langsam sein
HTTP 403 bedeutet falsches Passwort	Authentifizierung kann korrekt sein, aber Zugriff ist verboten
HTTP 502 ist immer der Webserver	Häufig meldet ein Gateway ein Upstream-Problem
Bearer-Token in der Kommandozeile ist sicher	Token kann in Historie und Prozessinformationen erscheinen
Verbose-Ausgabe kann bedenkenlos geteilt werden	Header, Cookies und Tokens können enthalten sein
curl verhält sich wie ein Browser	JavaScript und Browserkontext fehlen
Retry ist bei POST immer sicher	Änderung kann bereits verarbeitet worden sein
Proxyvariable ist ausgeschlossen, weil `-x` fehlt	curl kann Proxyvariablen aus der Umgebung verwenden

---

32. Wie sieht ein systematischer curl-Diagnoseablauf aus?

Empfohlene Schrittfolge anzeigen

Vorbereitung

1. Erwartete URL, Methode und Antwort dokumentieren.
2. Hostname, IP-Adresse, Port und Protokoll bestimmen.
3. Prüfen, ob Proxy, VPN oder Load Balancer beteiligt sind.
4. Authentifizierung und Schutzbedarf der Daten klären.
5. Ändernde Anfragen nur gegen freigegebene Testressourcen richten.

Grundtest

6. curl-Version und unterstützte Funktionen prüfen.
7. URL mit explizitem `https://` oder `http://` verwenden.
8. Verbindung mit `-v` untersuchen.
9. HTTP-Statuscode und effektive URL ausgeben.
10. Exitcode kontrollieren.

Eingrenzung

11. IPv4 mit `-4` und IPv6 mit `-6` vergleichen.
12. DNS mit `--resolve` kontrolliert umgehen.
13. Direktverbindung und Proxyverbindung vergleichen.
14. Redirect-Kette mit `-L -I` prüfen.
15. TLS normal und höchstens vergleichsweise mit `-k` testen.
16. HTTP/1.1 und HTTP/2 bei Bedarf vergleichen.
17. DNS-, TCP-, TLS-, TTFB- und Gesamtzeit messen.

Anwendung

18. Header und Content-Type prüfen.
19. Erwartete HTTP-Methode verwenden.
20. Request-Body und Zeichenkodierung kontrollieren.
21. Authentifizierung ohne Offenlegung der Zugangsdaten testen.
22. API-Antwort und Serverlogs zeitlich vergleichen.

Validierung

23. Ergebnis aus einem zweiten Netzwerksegment vergleichen.
24. Reverse-Proxy-, Firewall- und Anwendungslogs prüfen.
25. Bei Netzwerkverdacht Paketmitschnitt erstellen.
26. Nach einer Änderung denselben curl-Befehl erneut ausführen.
27. Befehl, Zeitpunkt, Exitcode, HTTP-Status und Zeiten dokumentieren.
28. Diagnoseausgaben vor Weitergabe von Secrets bereinigen.

---

33. Kurzreferenz – häufige curl-Befehle

Befehlstabelle anzeigen

Aufgabe	Linux/macOS
Version	[RO] curl --version
Seite abrufen	[TEST][SENS] curl https://example.com/
Nur HEAD-Anfrage	[TEST][SENS] curl -I https://example.com/
Header und Inhalt	[TEST][SENS] curl -i https://example.com/
Ausführliche Diagnose	[TEST][SENS] curl -v https://example.com/
Redirects verfolgen	[TEST][SENS] curl -L https://example.com/
Statuscode	[TEST] curl -sS -o /dev/null -w "%{http_code}\n" https://example.com/
HTTP-Fehler als Fehler behandeln	[TEST] curl -sS --fail-with-body https://example.com/
Connect-Timeout	[TEST] curl --connect-timeout 5 https://example.com/
Gesamttimeout	[TEST] curl --max-time 15 https://example.com/
IPv4	[TEST] curl -4 https://example.com/
IPv6	[TEST] curl -6 https://example.com/
DNS temporär überschreiben	[TEST][SENS] curl --resolve example.com:443:192.0.2.20 https://example.com/
Bestimmte CA-Datei	[TEST][FILE][SENS] curl --cacert company-ca.pem https://example.com/
HTTP/1.1	[TEST] curl --http1.1 https://example.com/
HTTP/2	[TEST] curl --http2 https://example.com/
JSON senden	[TEST][CHANGE][SENS] curl -H "Content-Type: application/json" -d '{"active":true}' https://api.example.com/item
Datei herunterladen	[TEST][FILE][SENS] curl -o file.bin https://example.com/file.bin
Proxy verwenden	[TEST][SENS] curl -x http://proxy.example.com:8080 https://example.com/
Proxy umgehen	[TEST][SENS] curl --noproxy "*" https://example.com/
Trace erstellen	[TEST][FILE][SENS] curl --trace-time --trace-ascii curl-trace.txt https://example.com/

Windows verwendet dieselbe Optionssyntax, aber eindeutig `curl.exe` und `NUL` statt `/dev/null`:

```
[TEST] curl.exe -sS -o NUL -w "HTTP=%{http_code} Zeit=%{time_total}s\n" https://example.com/
```

34. Kurzreferenz – wichtigste Optionen

Optionstabelle anzeigen

Option	Bedeutung
`-V`, `--version`	Version und Build-Funktionen anzeigen
`-v`, `--verbose`	Ausführliche Verbindungsinformationen
`-I`, `--head`	Bei HTTP HEAD-Anfrage senden
`-i`, `--show-headers`	Antwortheader zusammen mit Inhalt anzeigen
`-D`, `--dump-header`	Antwortheader separat schreiben
`-o`, `--output`	Antwortinhalt in Datei schreiben
`-O`, `--remote-name`	Remote-Dateinamen verwenden
`-s`, `--silent`	Fortschritts- und normale Fehlermeldungen unterdrücken
`-S`, `--show-error`	Fehler trotz `--silent` anzeigen
`-f`, `--fail`	HTTP-Fehler als curl-Fehler behandeln
`--fail-with-body`	HTTP-Fehler melden und Inhalt behalten
`-L`, `--location`	Weiterleitungen verfolgen
`--max-redirs`	Anzahl der Weiterleitungen begrenzen
`-w`, `--write-out`	Messwerte und Metadaten ausgeben
`--connect-timeout`	Verbindungszeit begrenzen
`-m`, `--max-time`	Gesamtdauer begrenzen
`-4`	IPv4 erzwingen
`-6`	IPv6 erzwingen
`--resolve`	Temporäre Host-/Port-/IP-Zuordnung
`--connect-to`	Tatsächliches Verbindungsziel ändern
`-k`, `--insecure`	Zertifikatsprüfung deaktivieren
`--cacert`	Bestimmte CA-Datei verwenden
`--cert`	Clientzertifikat verwenden
`--key`	Privaten Schlüssel verwenden
`--tlsv1.2`	Mindestens TLS 1.2
`--tls-max`	Maximale TLS-Version
`--http1.1`	HTTP/1.1 verwenden

Option	Bedeutung
`--http2`	HTTP/2 anfordern
`--http3`	HTTP/3 versuchen
`-H`, `--header`	Request-Header setzen
`-A`, `--user-agent`	User-Agent setzen
`-d`, `--data`	Request-Daten senden
`--data-binary`	Daten weitgehend unverändert senden
`--data-urlencode`	Daten URL-codieren
`--json`	JSON-Daten senden
`-X`, `--request`	HTTP-Methode ausdrücklich festlegen
`-u`, `--user`	Serverauthentifizierung
`-U`, `--proxy-user`	Proxyauthentifizierung
`-b`, `--cookie`	Cookies senden beziehungsweise lesen
`-c`, `--cookie-jar`	Cookies speichern
`-x`, `--proxy`	Proxy verwenden
`--noproxy`	Proxy für Ziele umgehen
`--compressed`	Komprimierte Antwort anfordern
`--retry`	Wiederholungsversuche
`--trace-ascii`	Lesbaren Trace schreiben
`--trace-time`	Trace mit Zeitstempeln versehen

Merksätze

- Unter Windows für echte curl-Syntax ausdrücklich `curl.exe` verwenden.
- Eine empfangene HTTP-Fehlerseite kann ohne `--fail` trotzdem curl-Exitcode 0 ergeben.
- HTTP-Statuscode und curl-Exitcode beantworten unterschiedliche Fragen.
- `-I` sendet bei HTTP eine HEAD-Anfrage und ist nicht identisch mit einem GET-Test.
- `-v` zeigt DNS, TCP, TLS und HTTP, kann aber sensible Header enthalten.
- `--resolve` umgeht DNS für den Test und erhält Hostname, HTTP-Host, TLS-SNI und Zertifikatsprüfung.
- Ein Aufruf über die reine IP-Adresse kann einen anderen virtuellen Host erreichen.
- `-k` deaktiviert die Zertifikatsprüfung und behebt kein TLS-Problem.
- `time_connect`, `time_appconnect` und `time_starttransfer` sind kumulative Zeitwerte.
- Eine hohe TTFB kann durch Anwendung, Datenbank, Proxy oder Backend entstehen.
- Bei UDP-basiertem HTTP/3 können andere Fehler als bei HTTP/1.1 oder HTTP/2 auftreten.

- Proxyvariablen können einen curl-Aufruf beeinflussen, auch wenn kein `-x` angegeben wurde.
- Zugangsdaten, Cookies und Tokens gehören nicht in Dokumentationen oder gemeinsam lesbare Skripte.
- POST-, PUT-, PATCH- und DELETE-Anfragen können Daten verändern und benötigen besondere Vorsicht.
- curl ist ein präzises Protokollwerkzeug, aber kein vollständiger Webbrowser.

Quellen

- Offizielle curl-Manpage
- Offizielle curl-Dokumentationsübersicht
- Everything curl – Command Line Transfers
- Everything curl – Verbose Operations
- Everything curl – Name Resolve Tricks
- Everything curl – HTTP Responses
- Everything curl – Proxies
- curl – Exit Codes
- curl – SSL Certificate Verification
- curl – Supported Protocols
