zurück

Traefik – Catch-all, Errorpage und Security-Middlewares

Zweck

Diese Dokumentation beschreibt die Schutz- und Fehlerbehandlung des Traefik-Reverse-Proxys für unbekannte Hosts, verdächtige URL-Pfade und automatisierte Exploit-Scans.

Die Konfiguration ist historisch gewachsen und wurde insbesondere im Zusammenhang mit Nextcloud eingerichtet. Mehrere Middleware-Regeln entstanden nach umfangreicher Fehlersuche, um Probleme mit Redirects, Security-Headern, WebDAV/CalDAV und anderen Nextcloud-Funktionen zu vermeiden.

Die bestehenden Middleware-Ketten nicht ohne vorherige Tests vereinfachen oder entfernen.

Insbesondere Änderungen an `block-paths`, `php-endings`, `global-chain` und `global-secure-chain` können Auswirkungen auf bestehende Anwendungen haben.

Grundsatz: **Funktionierende Konfiguration zuerst dokumentieren, dann nur kontrolliert und einzeln verändern.**

Architektur

Der öffentliche HTTP-/HTTPS-Verkehr erreicht zunächst Traefik.

Vereinfachter Ablauf:

```
Internet
   |
   v
FRITZ!Box
   |
   v
Traefik
   |
   +---- bekannter Host ------> definierter Router ------> Anwendung
   |
   +---- unbekannter Host ----> Catch-all
                                   |
                                   v
                               Middleware
                                   |
                                   v
```

```
                                errorpage
                                    |
                                    v
                                HTTP 418
```

Der Errorpage-Container dient dabei als kontrolliertes Ziel für nicht zuordenbare bzw. fehlerhafte Requests.

Traefik EntryPoints

Wichtige EntryPoints:

```
entryPoints:
  web:
    address: ":80"

  websecure:
    address: ":443"

  imap:
    address: ":993"

  smtp:
    address: ":587"

  smtp25:
    address: ":25"

  ssh:
    address: ":58222"

  matrix:
    address: ":8448"

  metrics:
    address: ":8082"
```

Der Metrics-EntryPoint ist für Prometheus vorgesehen und wird nicht nach außen veröffentlicht.

Docker Provider

Traefik verwendet den Docker Provider:

```
providers:
  docker:
```

```
    exposedByDefault: false
    network: docker_backend

  file:
    directory: /etc/traefik/dynamic/
    watch: true
```

Durch

```
exposedByDefault: false
```

werden Docker-Container nicht automatisch über Traefik veröffentlicht.

Ein Container muss explizit über entsprechende Traefik-Labels aktiviert werden.

Dynamische Konfiguration

Die dynamische Konfiguration befindet sich unter:

```
/etc/traefik/dynamic/
```

Zum Zeitpunkt der Dokumentation existieren unter anderem:

```
crowdsec.yml
dokuzusatz.yml
errorpage.yml
security.yml
adynamic.yml
tcp.yml
```

Zusätzlich existieren einige `.bak`-Dateien.

Bei der Fehlersuche immer zuerst prüfen, welche Dateien tatsächlich aktiv geladen werden und welche lediglich Backups sind.

Catch-all Router

In `adynamic.yml` existiert ein Catch-all:

```
catchall:
  rule: "HostRegexp(`{host:.+}`)"
  entryPoints:
    - web
```

```
    - websecure
  service: noop@internal
  middlewares:
    - custom-404
    - global-secure-chain
```

Dieser Router kann Requests auffangen, deren Hostname keinem spezifischeren Router zugeordnet wird.

Dadurch landen beispielsweise automatisierte Internet-Scans gegen die öffentliche IP-Adresse nicht direkt bei einer internen Anwendung.

Zusätzliche Catch-all Router

In `errorpage.yml` existieren zusätzlich Router mit niedriger Priorität:

```
catchall-http:
  rule: "PathPrefix(`/`)"
  entryPoints: [web]
  priority: -10
  service: error-svc
  middlewares:
    - custom-404
    - global-chain
```

Für HTTPS:

```
catchall-https:
  rule: "PathPrefix(`/`)"
  entryPoints: [websecure]
  priority: -10
  tls: {}
  service: error-svc
  middlewares:
    - custom-404
    - global-chain
```

Die niedrige Priorität sorgt dafür, dass spezifische Router bevorzugt werden.

Custom Error Middleware

Die Fehlerbehandlung erfolgt über:

```
custom-404:
```

```
  errors:
    status:
      - "404-599"
    service: error-svc
    query: "/?s={status}"
```

Als Error-Service dient:

```
error-svc:
  loadBalancer:
    servers:
      - url: "http://errorpage:8080"
```

Damit werden HTTP-Fehler im Bereich 404 bis 599 an den eigenen Errorpage-Container weitergereicht.

Errorpage-Container

Der Container basiert auf Nginx.

Mounts:

```
/opt/stacks/traefik/errorpage/nginx.conf
    -> /etc/nginx/nginx.conf

/opt/stacks/traefik/errorpage
    -> /usr/share/nginx/html
```

Die relevante Nginx-Konfiguration lautet:

```
events {}

http {
    server {
        listen 8080;
        listen [::]:8080;

        server_name localhost;

        location / {
            return 418;
        }

        error_page 418 /index.html;

        location = /index.html {
```

```
            root /usr/share/nginx/html;
            internal;
        }

        location = /sprueche.json {
            root /usr/share/nginx/html;
            internal;
        }
    }
}
```

Damit beantwortet der Container beliebige Requests kontrolliert mit:

```
HTTP/1.1 418
```

also:

```
418 I'm a teapot
```

Test des Errorpage-Containers

Direkter Test innerhalb des Docker-Netzes:

```
docker run --rm \
  --network docker_backend \
  curlimages/curl \
  -sv \
  -H 'Host: 79.254.196.8' \
  'http://errorpage:8080/irgendwas-vollkommen-unbekannt.php' \
  -o /dev/null
```

Erwartete Antwort:

```
HTTP/1.1 418
Server: nginx
Content-Type: text/html
```

Bei der Analyse vom 21.08.2026 betrug die Größe der Fehlerseite:

```
Content-Length: 38337
```

Dies ist wichtig für die Analyse von Suricata-Ereignissen:

Wenn dort bei einem verdächtigen Request eine Antwortgröße von ungefähr 38337 Byte zusammen

mit HTTP 418 auftaucht, handelt es sich sehr wahrscheinlich um die eigene Errorpage und nicht um die Ausgabe einer angegriffenen Anwendung.

Security Middleware

Die zentrale Security-Konfiguration befindet sich in:

```
/etc/traefik/dynamic/security.yml
```

CrowdSec

CrowdSec wird über ForwardAuth eingebunden:

```
crowdsec-forwardauth:
  forwardAuth:
    address: "http://crowdsec-bouncer:8080/api/v1/forwardAuth"
    trustForwardHeader: true
    authResponseHeaders:
      - X-Forwarded-For
      - X-Real-IP
```

Blockierte Pfade

Typische Scanner- und Angriffspfade werden durch `block-paths` behandelt:

```
block-paths:
  redirectRegex:
    regex:
^/(?:\.env(?:$|/)|\.git(?:$|/)|\.svn(?:$|/)|\.hg(?:$|/)|\.idea(?:$|/)|backup
(?:$|/)|vendor(?:$|/)|cgi-bin(?:$|/)).*
    replacement: /
    permanent: false
```

Erfasst werden insbesondere:

- /.env
- /.git
- /.svn
- /.hg
- /.idea
- /backup
- /vendor
- /cgi-bin

`permanent: false` erzeugt bei einem Treffer einen temporären Redirect.

Dadurch können bei Suricata-Ereignissen HTTP-302-Antworten sichtbar werden.

Ein HTTP 302 auf einen Exploit-Pfad bedeutet daher nicht automatisch, dass die angegriffene Anwendung diesen Redirect erzeugt hat.

Der Redirect kann vollständig von der Traefik-Security-Middleware stammen.

PHP-Endungen

Zusätzlich existiert:

```
php-endings:
  redirectRegex:
    regex: \.php(?:$|\?)
    replacement: /
    permanent: false
```

Diese Middleware ist dafür vorgesehen, direkte Requests auf PHP-Dateien abzufangen.

Sie ist nicht in allen Middleware-Ketten aktiv.

Middleware Chains

global-chain

Ohne CrowdSec:

```
global-chain:
  chain:
    middlewares:
      - block-paths
      - php-endings
      - sec-headers
      - ratelimit
      - compress
```

global-secure-chain

Mit CrowdSec:

```
global-secure-chain:
  chain:
    middlewares:
      - crowdsec-forwardauth
      - block-paths
      # - php-endings
      - sec-headers
      - ratelimit
      - compress
```

Wichtig:

`php-endings` ist hier bewusst nicht aktiv.

admin-chain

Für administrative Oberflächen existiert eine strengere Chain:

```
admin-chain:
  chain:
    middlewares:
      - crowdsec-forwardauth
      - block-paths
      - php-endings
      - admin-ips
      - admin-rl
      - sec-headers
      - compress
      - admin-basic
```

Security Header

Die allgemeinen Security Header umfassen unter anderem:

```
sec-headers:
  headers:
    frameDeny: true
    contentTypeNosniff: true
    referrerPolicy: no-referrer-when-downgrade
    permissionsPolicy: geolocation=(), microphone=(), camera=()
    customResponseHeaders:
      Server: ""
```

Rate Limiting

Allgemeines Rate Limit:

```
ratelimit:
  rateLimit:
    average: 50
    burst: 100
    period: 1s
```

Für administrative Oberflächen:

```
admin-rl:
  rateLimit:
    average: 10
    burst: 20
    period: 1s
```

Besonderheit Nextcloud

Für Nextcloud existieren gesonderte Header-Einstellungen.

Beispielsweise:

```
security-headers:
  headers:
    frameDeny: false
    customFrameOptionsValue: "SAMEORIGIN"
    contentTypeNosniff: true
    browserXssFilter: true
```

Die Konfiguration wurde nach umfangreicher Fehlersuche eingeführt, da allgemeinere bzw. strengere Middleware-Regeln wiederholt Probleme mit Nextcloud und einzelnen Funktionen verursachten.

Nextcloud reagiert empfindlich auf Änderungen an Reverse-Proxy-, Header- und Redirect-Konfigurationen.

Änderungen deshalb niemals gleichzeitig an mehreren Middleware-Komponenten durchführen.

Analyse eines automatisierten Exploit-Scans – 21.08.2026

Am 21.08.2026 wurde über Suricata ein automatisierter Exploitversuch gegen den öffentlich

erreichbaren Traefik erkannt.

Unter anderem wurden typische PHP-/PHPUnit- und RCE-Pfade angesprochen.

Beispiel:

```
/vendor/phpunit/...
```

Solche Requests treffen auf die vorhandene Middleware:

```
Request
   |
   v
Traefik
   |
   v
global-secure-chain
   |
   +--> CrowdSec
   |
   +--> block-paths
             |
             +--> /vendor/... erkannt
                       |
                       v
                HTTP Redirect
```

Andere nicht zuordenbare Requests können dagegen beim Catch-all bzw. Errorpage-Service enden:

```
Internet Scanner
       |
       v
    Traefik
       |
       v
   Catch-all
       |
       v
 custom-404
       |
       v
 errorpage:8080
       |
       v
 HTTP 418
```

Bewertung des Vorfalls

Die Untersuchung ergab:

- Der eingehende Request war ein echter automatisierter Exploitversuch.
- Es handelte sich sehr wahrscheinlich um allgemeines Internet-Scanning.
- Der Scanner verwendete unter anderem bekannte PHP-/PHPUnit-Angriffspfade.
- Traefik nahm den Request entgegen.
- Verdächtige Pfade wurden von vorhandenen Middleware-Regeln behandelt.
- Unbekannte Requests können auf der eigenen Errorpage landen.
- Der Errorpage-Container führt kein PHP aus.
- Der Errorpage-Container liefert grundsätzlich HTTP 418.
- Die beobachteten 302-Antworten können durch `redirectRegex` der Security-Middlewares entstehen.
- Die beobachteten ca. 38337 Byte entsprechen der eigenen HTML-Fehlerseite.
- Es wurde kein Hinweis auf eine erfolgreiche PHP-Code-Ausführung festgestellt.
- Es wurde kein Hinweis darauf gefunden, dass PHPUnit tatsächlich erreicht wurde.
- Es wurde kein Hinweis auf erfolgreiche Remote Code Execution festgestellt.

Bewertung: automatisierter Exploit-Scan, erfolgreich durch vorhandene Proxy-/Security-Struktur abgefangen.

Wichtige Hinweise für zukünftige Suricata-Alarme

Bei einem neuen Web-Attack-Alarm nicht allein anhand des HTTP-Status entscheiden, ob ein Backend tatsächlich angesprochen wurde.

Folgende Punkte prüfen:

1. Suricata `flow_id`
2. Host-Header
3. Request-URI
4. HTTP-Methode
5. HTTP-Statuscode
6. Response-Größe
7. Traefik Access Log
8. zuständiger Traefik-Router
9. angewendete Middleware
10. Backend-Logs
11. CrowdSec-Ereignisse

Besonders wichtig:

```
HTTP 302
```

kann durch `redirectRegex` entstehen.

```
HTTP 418
Content-Length: ~38337
```

ist ein starker Hinweis auf die eigene Errorpage.

Suricata Flow untersuchen

Einen kompletten Flow aus `eve.json` ausgeben:

```
sudo jq -c '
select(.flow_id == FLOW_ID)
' /opt/stacks/suricata/data/eve.json
```

Beispiel:

```
sudo jq -c '
select(.flow_id == 2246373244831953)
' /opt/stacks/suricata/data/eve.json
```

Catch-all manuell testen

Test gegen Traefik mit unbekanntem Host:

```
curl -sv \
  -H 'Host: 79.254.196.8' \
  'http://192.168.178.96/irgendwas-vollkommen-unbekannt.php' \
  -o /dev/null
```

Damit lässt sich prüfen, wie Traefik einen Request behandelt, der keinem normalen Host-Router entspricht.

Errorpage direkt testen

Ohne Traefik:

```
docker exec traefik sh -c '
wget -S -O /dev/null \
  --header="Host: test.invalid" \
  "http://errorpage:8080/test.php"
'
```

Erwartet:

```
HTTP/1.1 418
```

Dieser Test ist besonders hilfreich, um zwischen einer Traefik-Antwort und einer Antwort des Errorpage-Containers zu unterscheiden.

Konfiguration prüfen

Geladene dynamische Dateien:

```
docker exec traefik sh -c '
find /etc/traefik/dynamic -maxdepth 2 -type f -print
'
```

Nach Redirects und Catch-all-Regeln suchen:

```
docker exec traefik sh -c '
grep -RniE
"HostRegexp|redirect|replacePath|replacePathRegex|stripPrefix|addPrefix|418|
catch|fallback" \
/etc/traefik/dynamic 2>/dev/null
'
```

Errorpage-Nginx prüfen:

```
docker exec errorpage nginx -T
```

Traefik-Konfiguration anzeigen:

```
docker exec traefik cat /etc/traefik/traefik.yml
```

Wartungsregel

Never change a running system ohne Rückfallplan.

Vor Änderungen an der Traefik-Security-Konfiguration:

1. Konfigurationsdateien sichern
2. aktuellen Zustand dokumentieren
3. nur eine Regel gleichzeitig verändern
4. Traefik-Logs kontrollieren
5. Nextcloud testen
6. WebDAV/CalDAV testen
7. administrative Dienste testen

8. Catch-all testen
9. Errorpage testen
10. erst danach die nächste Änderung durchführen

Falls nach einer Änderung unerklärliche Nextcloud-Fehler auftreten, zuerst die zuletzt geänderte Middleware zurücksetzen.

Fazit

Die Kombination aus Traefik, CrowdSec, Security-Middlewares, Catch-all-Routern und eigener HTTP-418-Errorpage bildet eine zusätzliche Schutzschicht vor den eigentlichen Anwendungen.

Automatisierte Scanner können dadurch Antworten wie HTTP 302 oder HTTP 418 erhalten, ohne dass der angesprochene Exploit-Pfad auf einem Backend tatsächlich existiert.

Die vorhandene Konfiguration ist historisch gewachsen und teilweise das Ergebnis umfangreicher Fehlersuche, insbesondere mit Nextcloud.

Sie sollte daher nicht allein aus Gründen einer vermeintlich saubereren oder einfacheren Konfiguration verändert werden.

Stabilität und nachvollziehbares Verhalten haben Vorrang vor kosmetischer Vereinfachung.