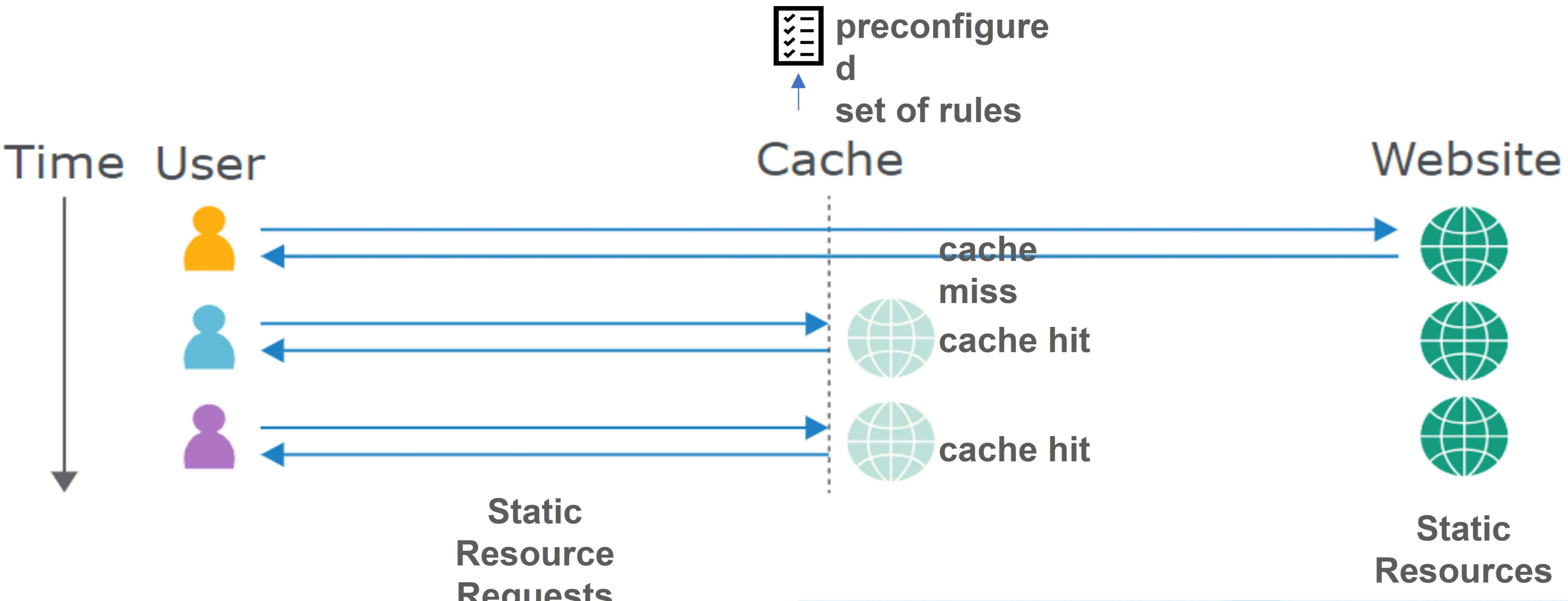
TURKISH TECHNOLOGY

Web Cache Deception

02.10.2024

Web Caches



preconfigured set of rules

Time

User

Cache

Website

cache miss

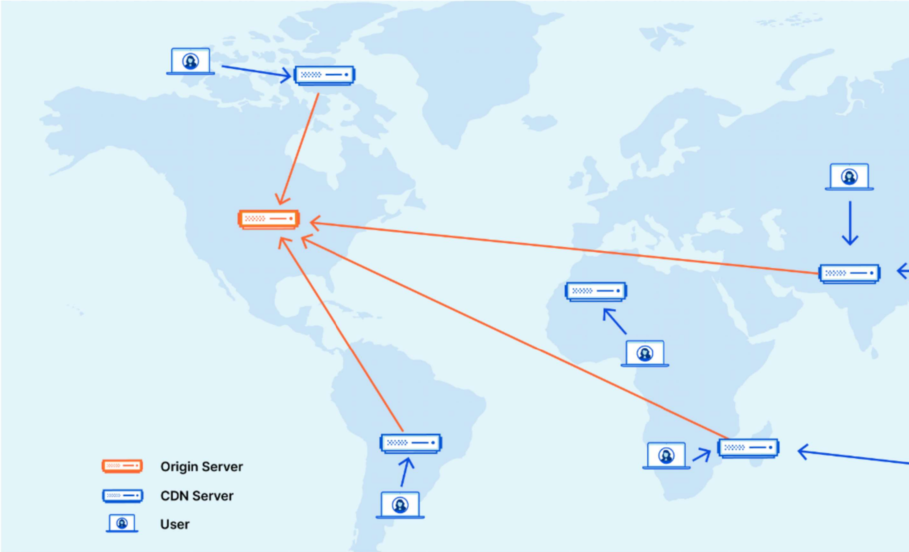
cache hit

cache hit

Static Resource Requests

Static Resources

Content Delivery Networks (CDNs)



https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

Caches Keys & Caches Rules

Cache Key oluşturma genellikle **URL path, query parameters** gibi HTTP isteği elementlerinde oluşturulur. Ayrıca **headers** ve **content type** gibi farklı elementler de kullanılabilir.

Gelen istekler eğer önceki isteklerdeki **Cache Key** ile eşleşirse, cache sunusu aynı içerik olduğ karar vererek **önbelleğe** aldığı bir önceki cevabı gönderir.

Cache Rule ise genellikle uzun süre değişmeyen ve bir çok farklı istekte tekrarlı kullanılan stat içerikleri **önbelleğe** almak için kullanılır.

Neyin ve **Ne Kadar Süre** ile önbellekte duracağına karar verir.

Static file extension rules -> **.css** veya **.js**

Static directory rules -> **/static** veya **/assets**

File name rules -> **robots.txt** veya **favicon.ico**

Custom rules -> **URL parametreleri** veya **dinamik analiz** gibi farklı kriterlere göre

Cached Responses

X-Cache başlığı ile önbelleklenmiş cevapları tespit ediyoruz

```
X-Cache: hit
X-Cache: miss
X-Cache: dynamic
X-Cache: refresh
```

Cache-Control başlığı ile önbelleğe alınmış olabilecek içerikler için ip ucu yakalayabiliriz. Mes aşağıdaki gibi max-age 0'dan büyük ve public olanlar

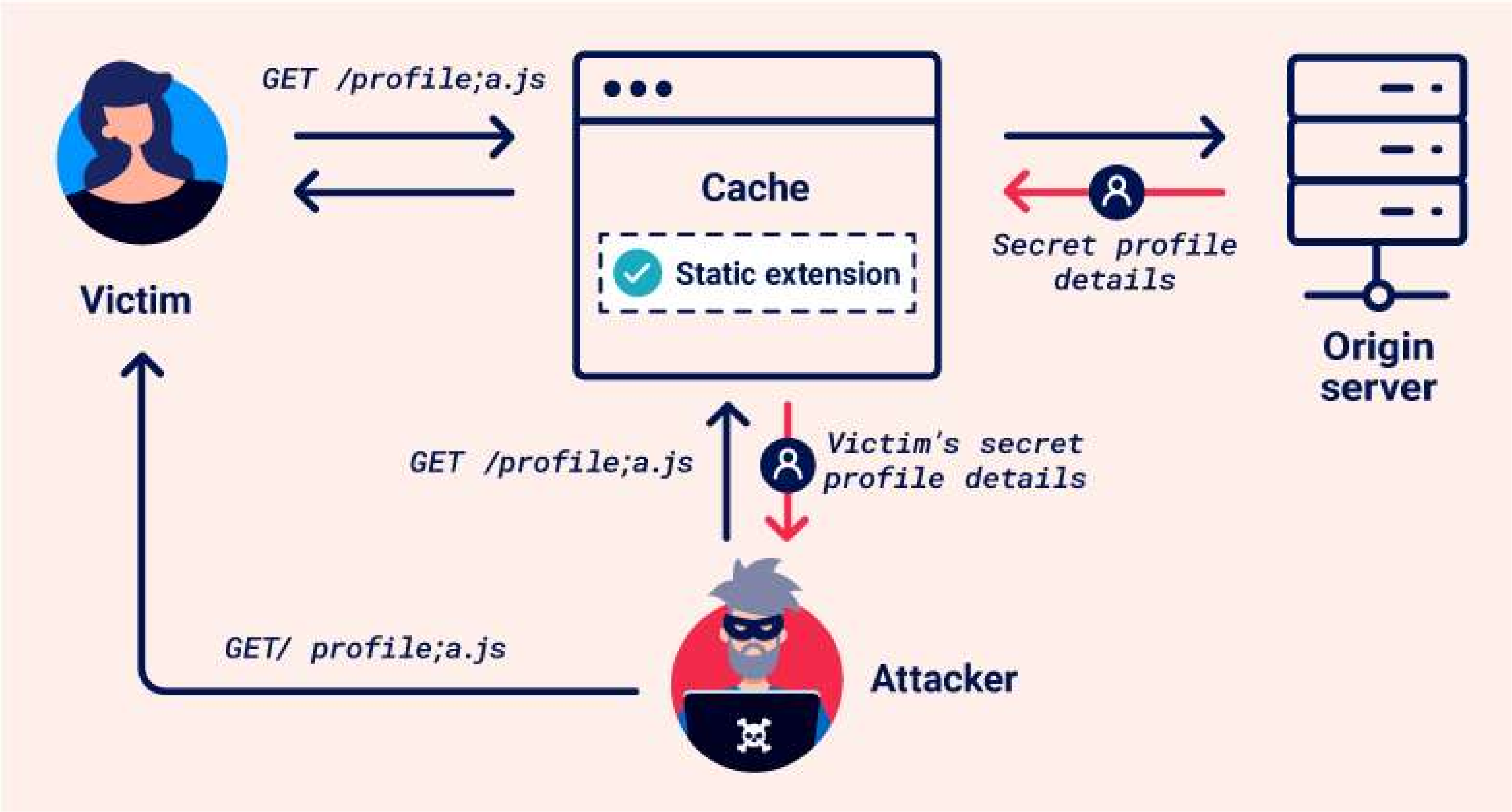
```
Cache-control: max-age=180, public
```

Bazen cache sunucusu bu başlığın üzerine yazabilir. Kesin olarak önbelleğe alındığını ifade etr sadece önbelleğe kaydedilebileceğini belirtir.

Ek olarak aynı iki istek arasındaki cevap sürelerinde çok büyük farklar varsa hızlı olan cache'd gelmiş olabilir.

Web Cache Deception

Cache sunucusu ile origin sunucusunun istekleri işlemesindeki farklılıkdan kaynaklanır

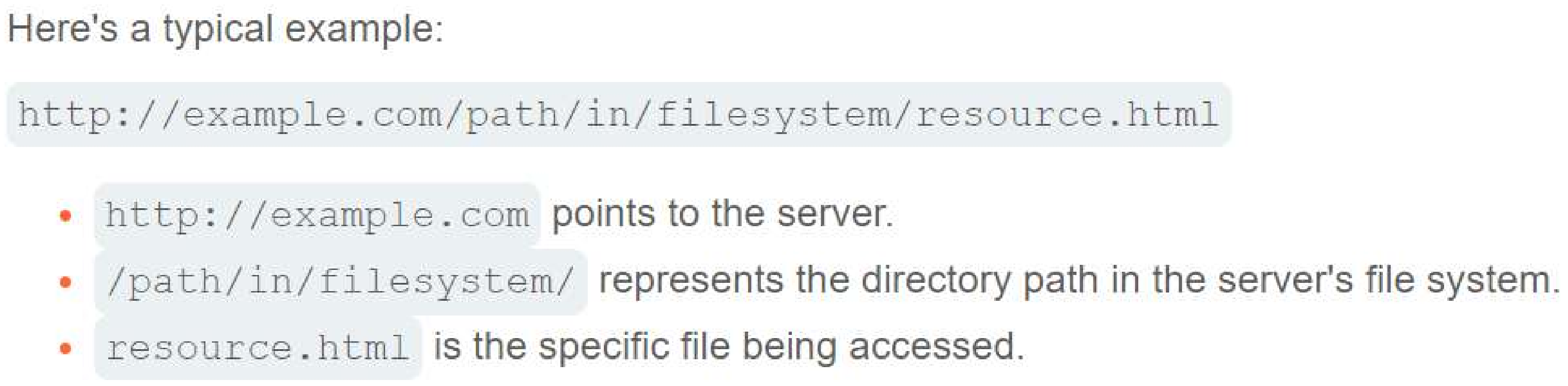


Web Cache Deception vs Web Cache Poisoning

	Web Cache Deception	Web Cache Poisonin
Önbellekleme mekanizmasını sömürür	✓	✓
Önbelleklenmiş cevaba zararlı içerik enjekte etmek için önbellek anahtarlarını manipüle eder	✗	✓
Hassas ve gizli içerikleri önbelleğe alıp kaydetmesi için önbellek kurallarını sömürür	✓	✗

Exploiting Static Extension Cache Rules

Traditional URL mapping

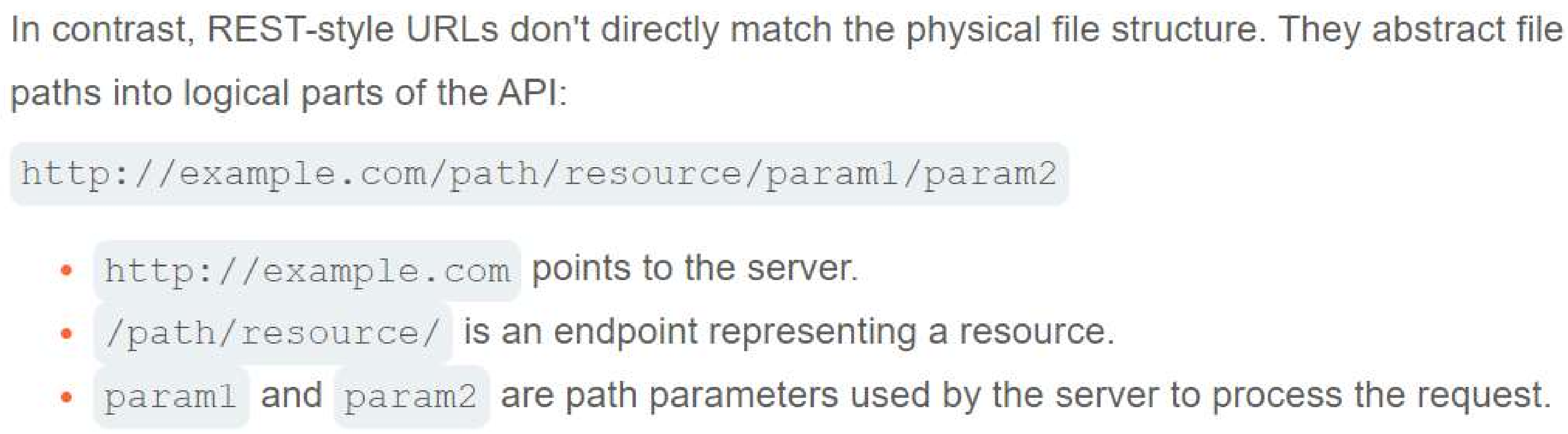


Here's a typical example:

```
http://example.com/path/in/filesystem/resource.html
```

- `http://example.com` points to the server.
- `/path/in/filesystem/` represents the directory path in the server's file system.
- `resource.html` is the specific file being accessed.

RESTful URL mapping

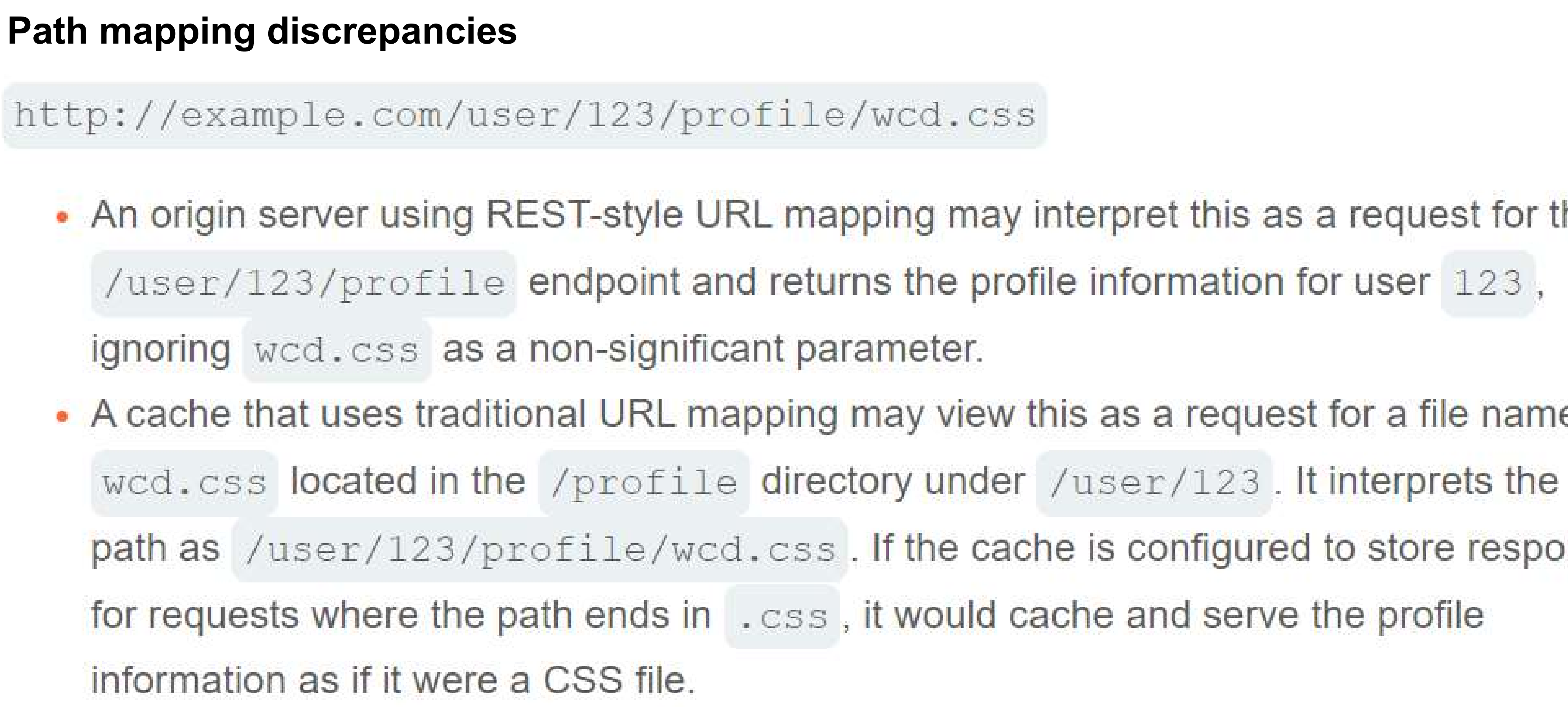


In contrast, REST-style URLs don't directly match the physical file structure. They abstract file paths into logical parts of the API:

```
http://example.com/path/resource/param1/param2
```

- `http://example.com` points to the server.
- `/path/resource/` is an endpoint representing a resource.
- `param1` and `param2` are path parameters used by the server to process the request.

Exploiting Static Extension Cache Rules



Path mapping discrepancies

```
http://example.com/user/123/profile/wcd.css
```

- An origin server using REST-style URL mapping may interpret this as a request for th `/user/123/profile` endpoint and returns the profile information for user `123`, ignoring `wcd.css` as a non-significant parameter.
- A cache that uses traditional URL mapping may view this as a request for a file name `wcd.css` located in the `/profile` directory under `/user/123`. It interprets the path as `/user/123/profile/wcd.css`. If the cache is configured to store respo for requests where the path ends in `.css`, it would cache and serve the profile information as if it were a CSS file.

Exploiting Static Extension Cache Rules - Uygulama 1

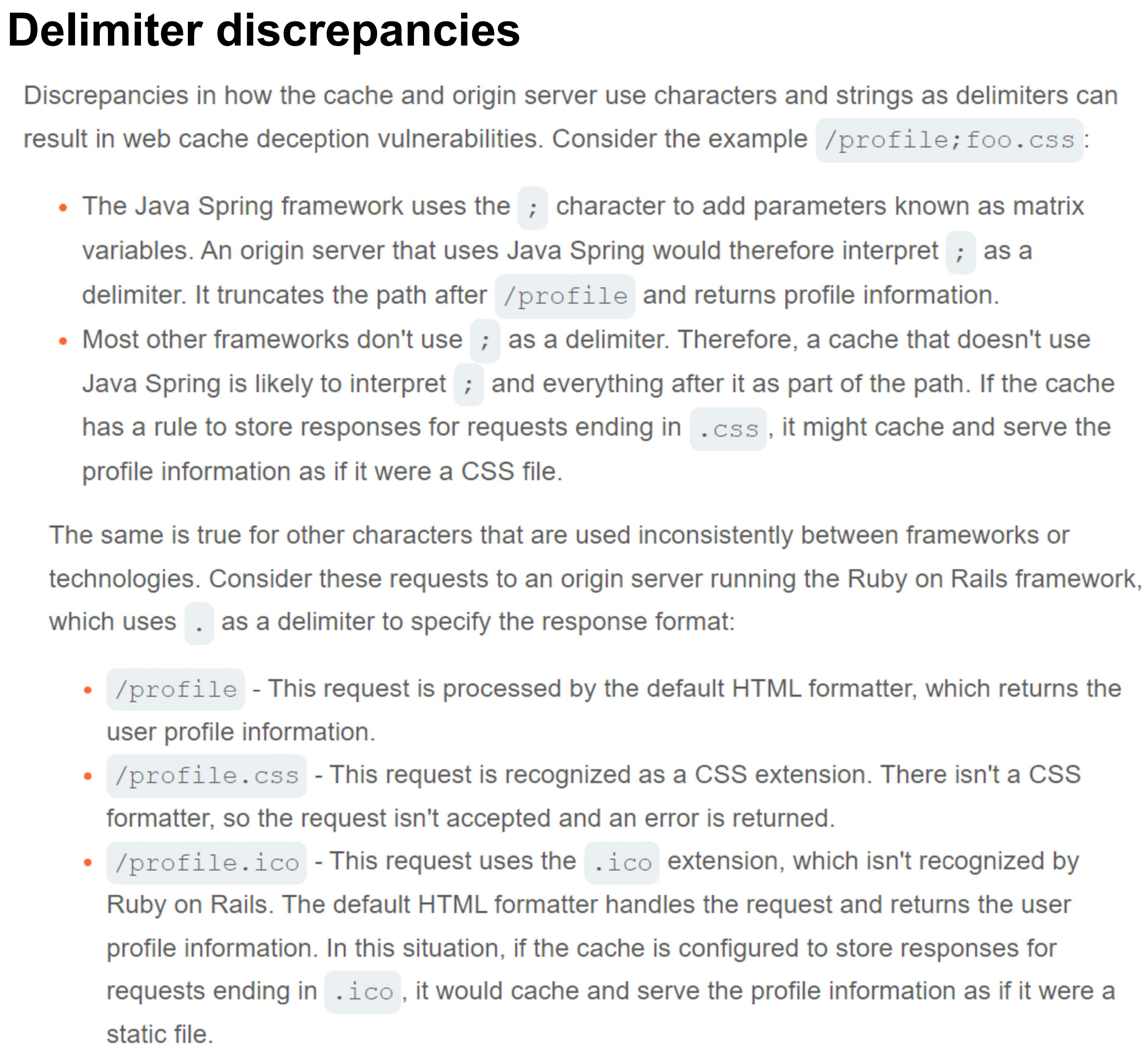
Web Security Academy

https://portswigger.net/web-security/all-labs

Lab: Exploiting path mapping for web cac

https://portswigger.net/web-security/web-cache
lab-wcd-exploiting-path-mapping

Exploiting Static Extension Cache Rules



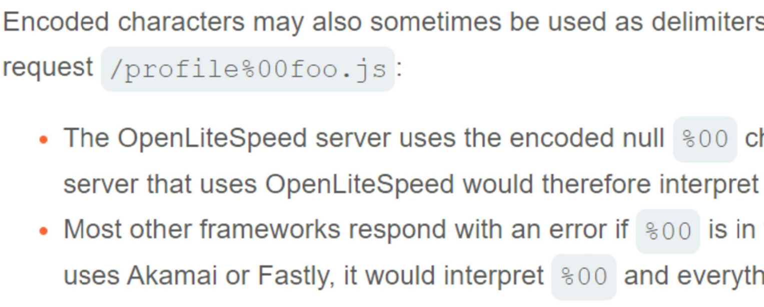
Delimiter discrepancies

Discrepancies in how the cache and origin server use characters and strings as delimiters can result in web cache deception vulnerabilities. Consider the example `/profile;foo.css`:

- The Java Spring framework uses the `;` character to add parameters known as matrix variables. An origin server that uses Java Spring would therefore interpret `;` as a delimiter. It truncates the path after `/profile` and returns profile information.
- Most other frameworks don't use `;` as a delimiter. Therefore, a cache that doesn't use Java Spring is likely to interpret `;` and everything after it as part of the path. If the cache has a rule to store responses for requests ending in `.css`, it might cache and serve the profile information as if it were a CSS file.

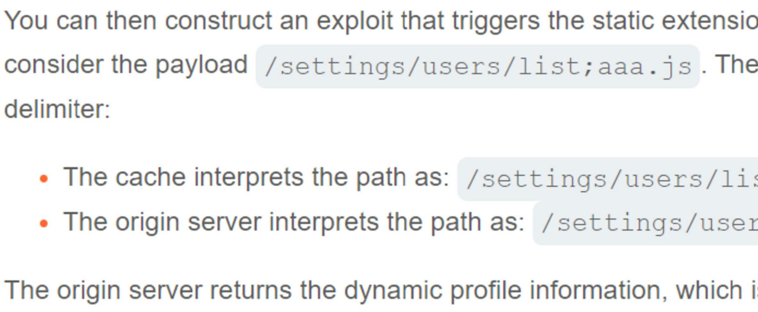
The same is true for other characters that are used inconsistently between frameworks or technologies. Consider these requests to an origin server running the Ruby on Rails framework, which uses `.` as a delimiter to specify the response format:

- `/profile` - This request is processed by the default HTML formatter, which returns the user profile information.
- `/profile.css` - This request is recognized as a CSS extension. There isn't a CSS formatter, so the request isn't accepted and an error is returned.
- `/profile.ico` - This request uses the `.ico` extension, which isn't recognized by Ruby on Rails. The default HTML formatter handles the request and returns the user profile information. In this situation, if the cache is configured to store responses for requests ending in `.ico`, it would cache and serve the profile information as if it were a static file.



Encoded characters may also sometimes be used as delimiters request `/profile%00foo.js`:

- The OpenLiteSpeed server uses the encoded null `%00` ch server that uses OpenLiteSpeed would therefore interpret
- Most other frameworks respond with an error if `%00` is in uses Akamai or Fastly, it would interpret `%00` and everyth



You can then construct an exploit that triggers the static extensio consider the payload `/settings/users/list;aaa.js`. The delimiter:

- The cache interprets the path as: `/settings/users/lis`
- The origin server interprets the path as: `/settings/user`

The origin server returns the dynamic profile information, which i

Web cache deception lab delimiter list.

https://portswigger.net/web-security/web-c lab-delimiter-list

Exploiting Static Extension Cache Rules - Uygulama 2

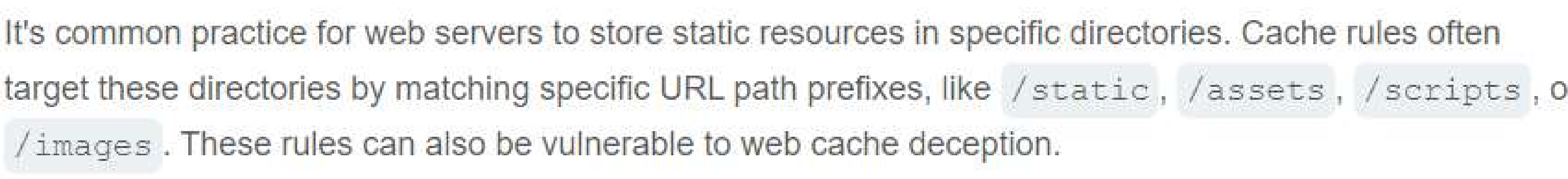
Web Security Academy

https://portswigger.net/web-security/all-labs

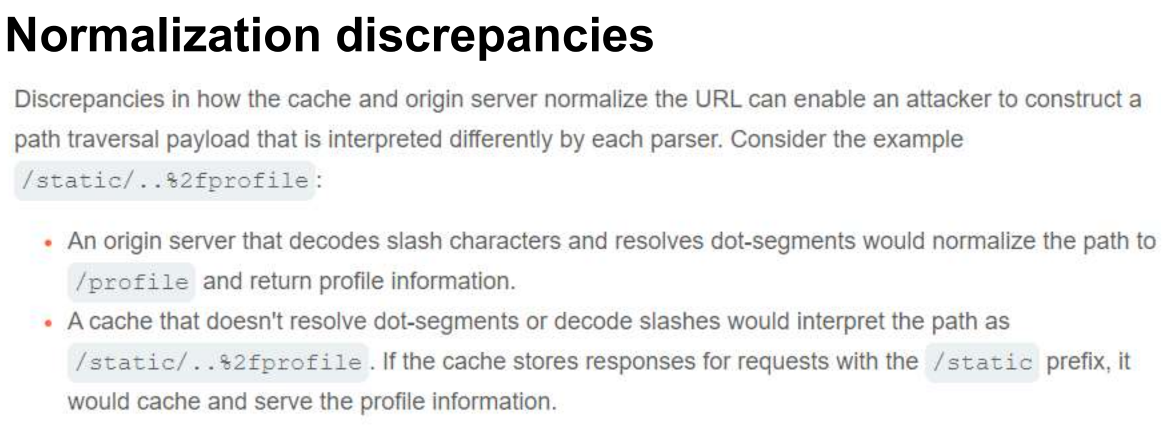
Lab: Exploiting path delimiters for web c deception

https://portswigger.net/web-security/web-cach lab-wcd-exploiting-path-delimiters

Exploiting Static Directory Cache Rules



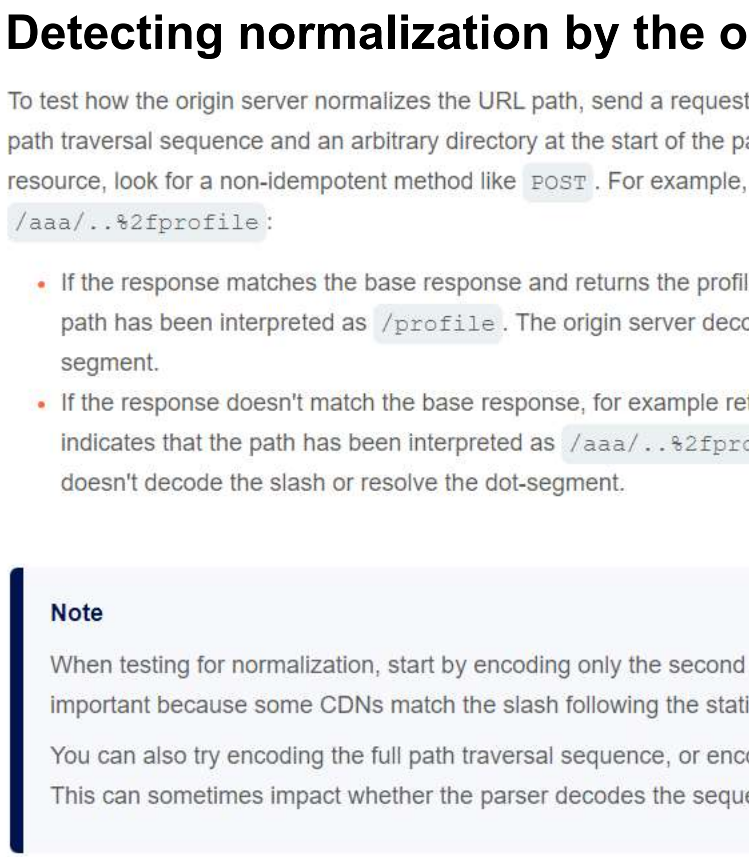
It's common practice for web servers to store static resources in specific directories. Cache rules often target these directories by matching specific URL path prefixes, like `/static`, `/assets`, `/scripts`, o `/images`. These rules can also be vulnerable to web cache deception.



Normalization discrepancies

Discrepancies in how the cache and origin server normalize the URL can enable an attacker to construct a path traversal payload that is interpreted differently by each parser. Consider the example `/static/..%2fprofile`:

- An origin server that decodes slash characters and resolves dot-segments would normalize the path to `/profile` and return profile information.
- A cache that doesn't resolve dot-segments or decode slashes would interpret the path as `/static/..%2fprofile`. If the cache stores responses for requests with the `/static` prefix, it would cache and serve the profile information.



Detecting normalization by the o

To test how the origin server normalizes the URL path, send a request path traversal sequence and an arbitrary directory at the start of the pa resource, look for a non-idempotent method like `POST`. For example, `/aaa/..%2fprofile`:

- If the response matches the base response and returns the profil path has been interpreted as `/profile`. The origin server dec segment.
- If the response doesn't match the base response, for example re indicates that the path has been interpreted as `/aaa/..%2fpro` doesn't decode the slash or resolve the dot-segment.

Note

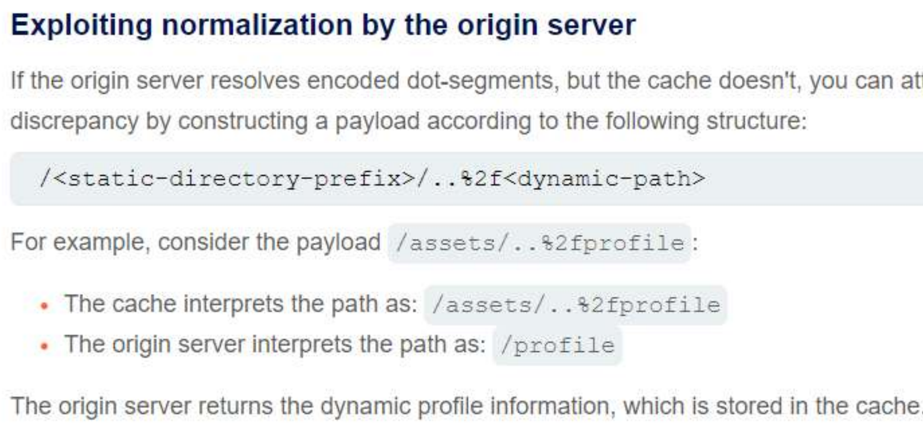
When testing for normalization, start by encoding only the second important because some CDNs match the slash following the stati

You can also try encoding the full path traversal sequence, or enc This can sometimes impact whether the parser decodes the seque

Exploiting Static Directory Cache Rules - Uygulama 3

Web Security Academy

https://portswigger.net/web-security/all-labs



Exploiting normalization by the origin server

If the origin server resolves encoded dot-segments, but the cache doesn't, you can at
discrepancy by constructing a payload according to the following structure:

```
/<static-directory-prefix>/..%2f<dynamic-path>
```

For example, consider the payload `/assets/..%2fprofile`:

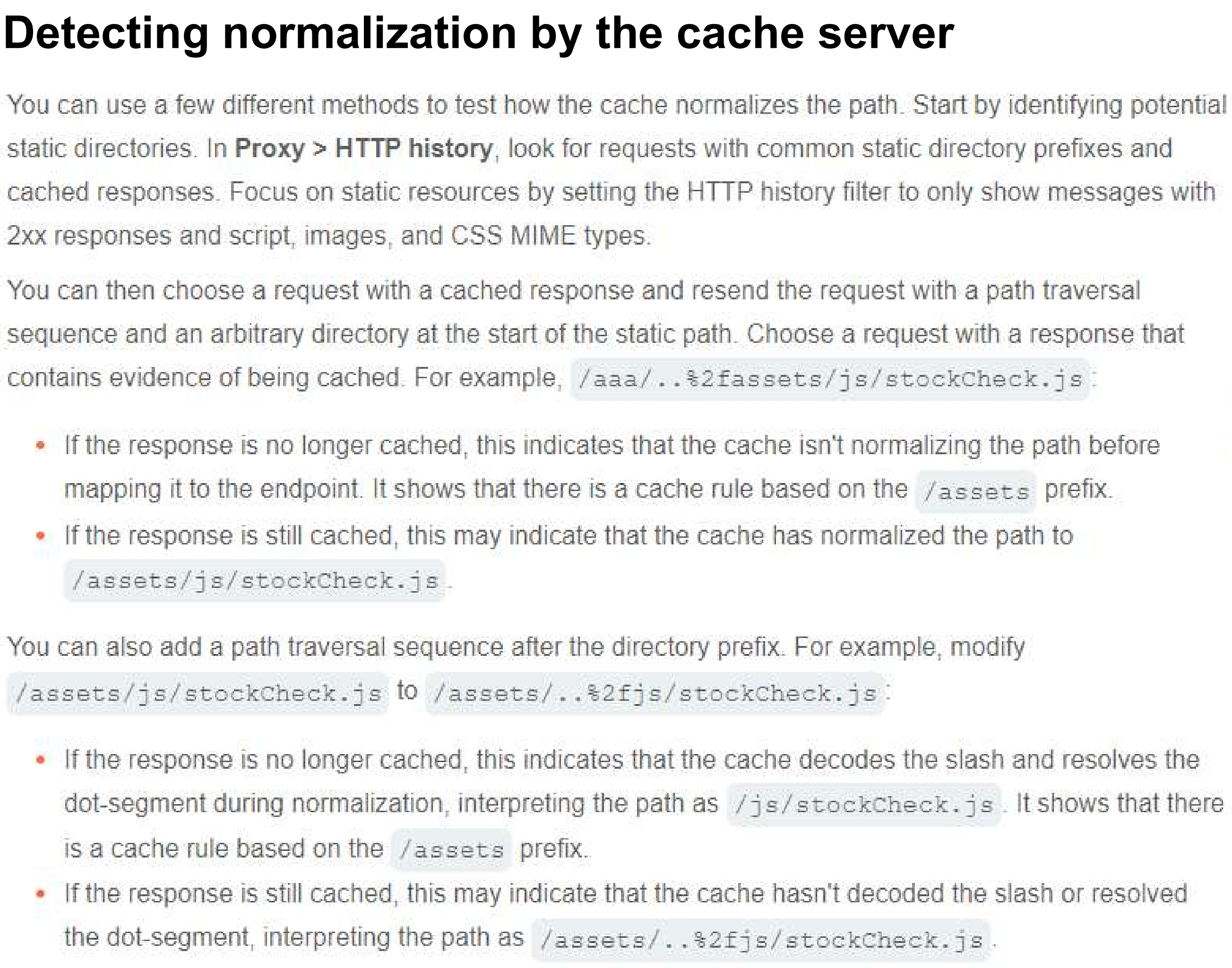
- The cache interprets the path as: `/assets/..%2fprofile`
- The origin server interprets the path as: `/profile`

The origin server returns the dynamic profile information, which is stored in the cache.

Lab: Exploiting origin server normalizati cache deception

https://portswigger.net/web-security/web-cach lab-wcd-exploiting-origin-server-normalization

Exploiting Static Directory Cache Rules



Detecting normalization by the cache server

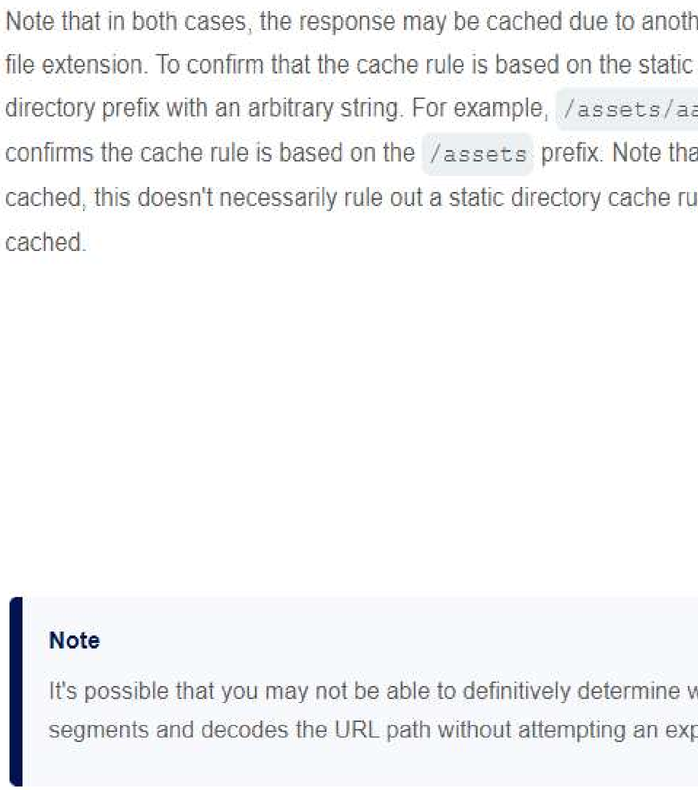
You can use a few different methods to test how the cache normalizes the path. Start by identifying potential static directories. In **Proxy > HTTP history**, look for requests with common static directory prefixes and cached responses. Focus on static resources by setting the HTTP history filter to only show messages with 2xx responses and script, images, and CSS MIME types.

You can then choose a request with a cached response and resend the request with a path traversal sequence and an arbitrary directory at the start of the static path. Choose a request with a response that contains evidence of being cached. For example, `/aaa/..%2fassets/js/stockCheck.js`:

- If the response is no longer cached, this indicates that the cache isn't normalizing the path before mapping it to the endpoint. It shows that there is a cache rule based on the `/assets` prefix.
- If the response is still cached, this may indicate that the cache has normalized the path to `/assets/js/stockCheck.js`.

You can also add a path traversal sequence after the directory prefix. For example, modify `/assets/js/stockCheck.js` to `/assets/..%2fjs/stockCheck.js`:

- If the response is no longer cached, this indicates that the cache decodes the slash and resolves the dot-segment during normalization, interpreting the path as `/js/stockCheck.js`. It shows that there is a cache rule based on the `/assets` prefix.
- If the response is still cached, this may indicate that the cache hasn't decoded the slash or resolved the dot-segment, interpreting the path as `/assets/..%2fjs/stockCheck.js`.



Note that in both cases, the response may be cached due to anoth
file extension. To confirm that the cache rule is based on the static
directory prefix with an arbitrary string. For example, `/assets/aa`
confirms the cache rule is based on the `/assets` prefix. Note tha
cached, this doesn't necessarily rule out a static directory cache ru
cached.

Note

It's possible that you may not be able to definitively determine w
segments and decodes the URL path without attempting an exp

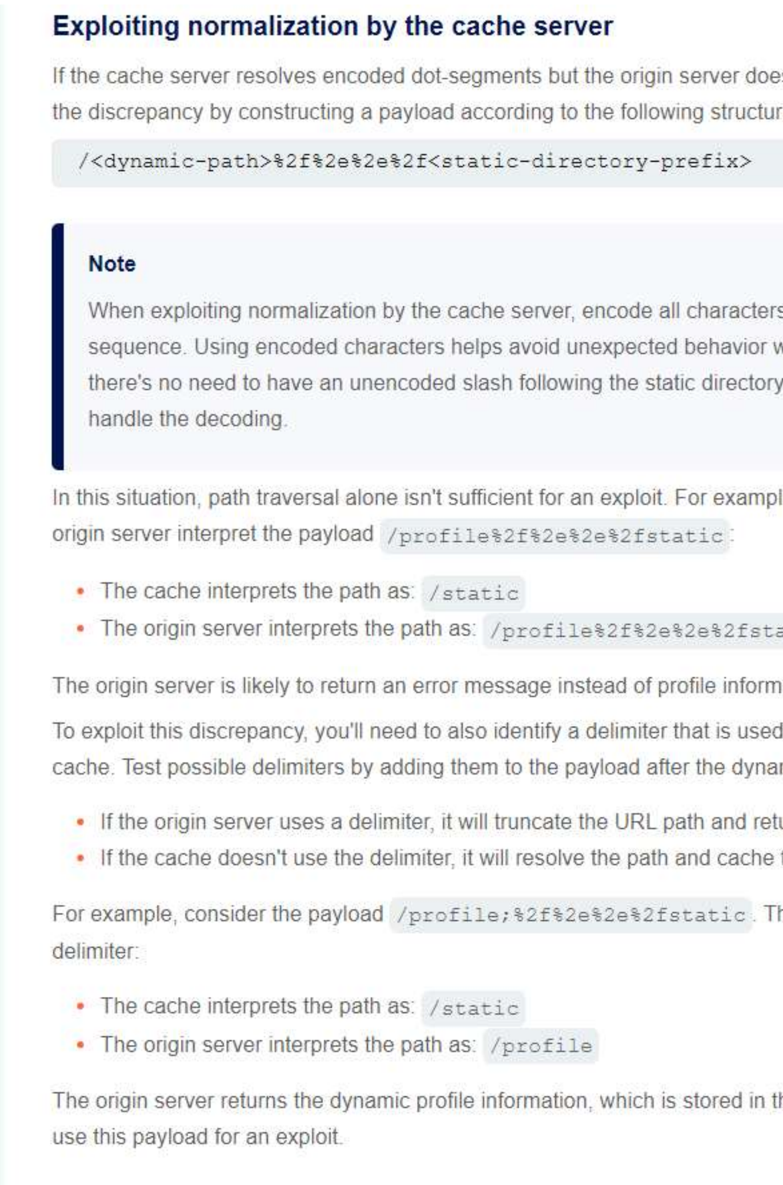
Exploiting Static Directory Cache Rules - Uygulama 4

Web Security Academy

https://portswigger.net/web-security/all-labs

Lab: Exploiting cache server normalization for web cache deception

https://portswigger.net/web-security/web-cache-deception/lab-wcd-exploiting-cache-server-normalization



Exploiting normalization by the cache server

If the cache server resolves encoded dot-segments but the origin server doe
the discrepancy by constructing a payload according to the following structur

```
/<dynamic-path>%2f%2e%2e%2f<static-directory-prefix>
```

Note

When exploiting normalization by the cache server, encode all characters
sequence. Using encoded characters helps avoid unexpected behavior w
there's no need to have an unencoded slash following the static directory
handle the decoding.

In this situation, path traversal alone isn't sufficient for an exploit. For exampl
origin server interpret the payload `/profile%2f%2e%2e%2fstatic`:

- The cache interprets the path as: `/static`
- The origin server interprets the path as: `/profile%2f%2e%2e%2fsta`

The origin server is likely to return an error message instead of profile inform

To exploit this discrepancy, you'll need to also identify a delimiter that is used
cache. Test possible delimiters by adding them to the payload after the dynar

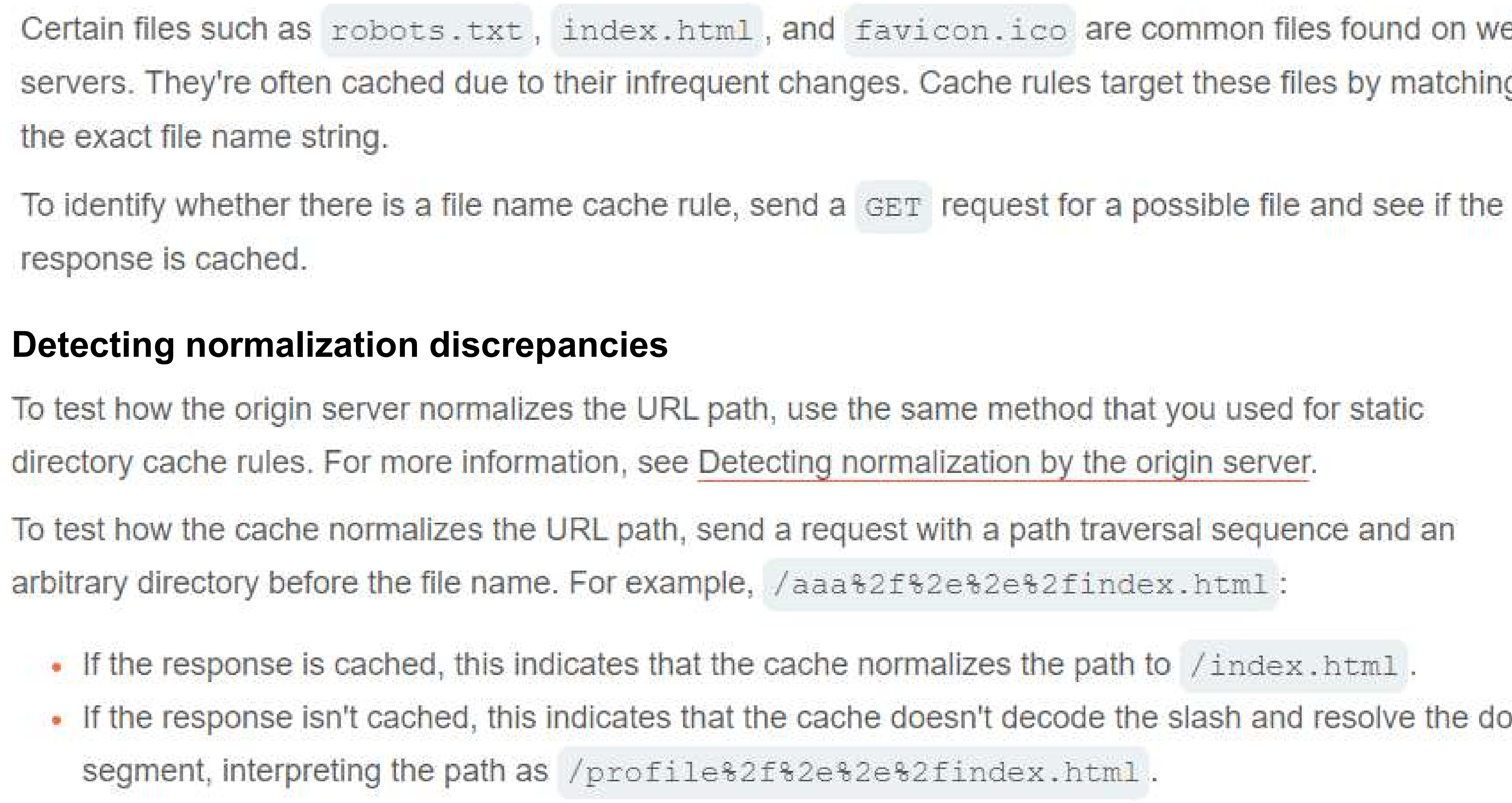
- If the origin server uses a delimiter, it will truncate the URL path and retu
- If the cache doesn't use the delimiter, it will resolve the path and cache t

For example, consider the payload `/profile;%2f%2e%2e%2fstatic`. Th
delimiter:

- The cache interprets the path as: `/static`
- The origin server interprets the path as: `/profile`

The origin server returns the dynamic profile information, which is stored in th
use this payload for an exploit.

Exploiting File Name Cache Rules



Certain files such as `robots.txt`, `index.html`, and `favicon.ico` are common files found on we servers. They're often cached due to their infrequent changes. Cache rules target these files by matching the exact file name string.

To identify whether there is a file name cache rule, send a `GET` request for a possible file and see if the response is cached.

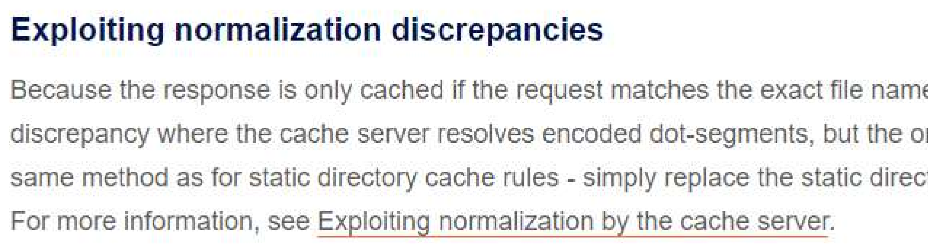
Detecting normalization discrepancies

To test how the origin server normalizes the URL path, use the same method that you used for static directory cache rules. For more information, see Detecting normalization by the origin server.

To test how the cache normalizes the URL path, send a request with a path traversal sequence and an arbitrary directory before the file name. For example, `/aaa%2f%2e%2e%2findex.html`:

- If the response is cached, this indicates that the cache normalizes the path to `/index.html`.
- If the response isn't cached, this indicates that the cache doesn't decode the slash and resolve the do segment, interpreting the path as `/profile%2f%2e%2e%2findex.html`.

Exploiting File Name Cache Rules - Uygulama 5



Exploiting normalization discrepancies

Because the response is only cached if the request matches the exact file name
discrepancy where the cache server resolves encoded dot-segments, but the o
same method as for static directory cache rules - simply replace the static direc
For more information, see Exploiting normalization by the cache server.

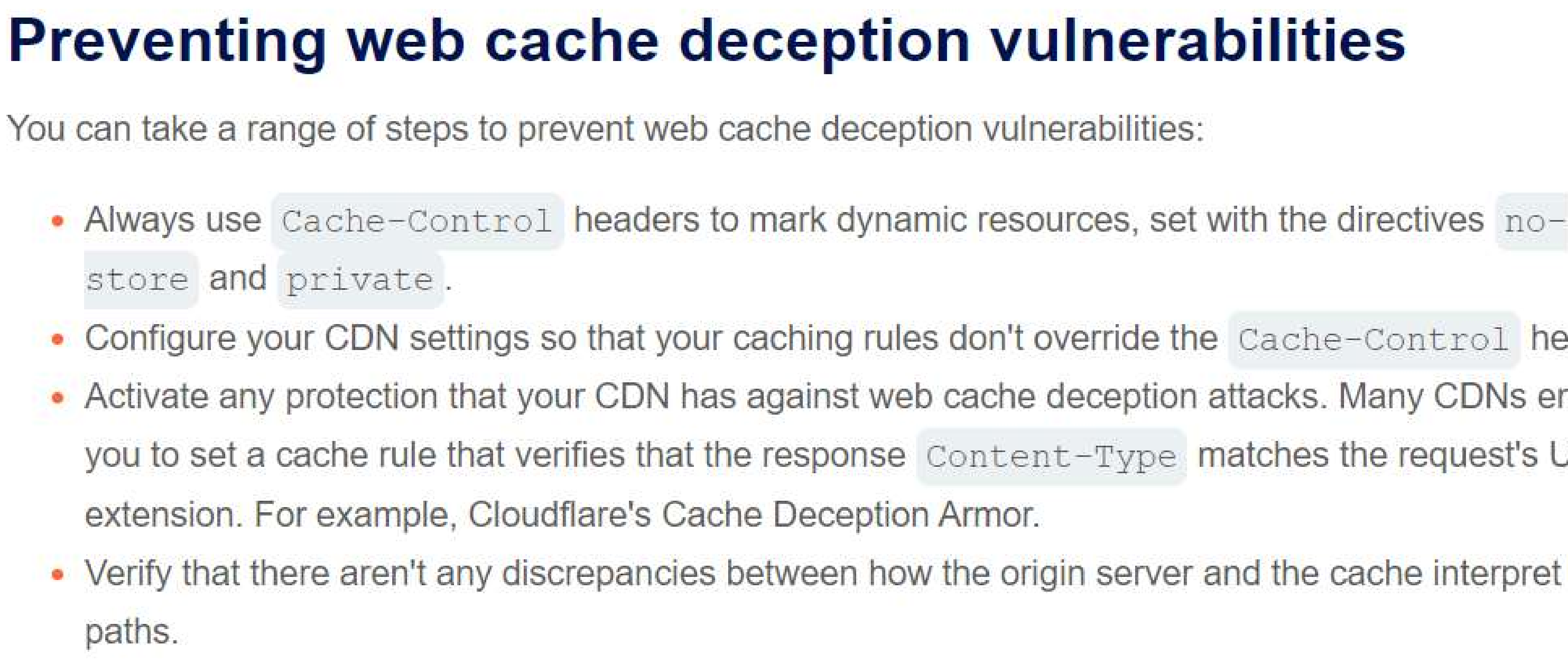
Web Security Academy

https://portswigger.net/web-security/all-labs

**Lab: Exploiting exact-match cache rules
cache deception**

https://portswigger.net/web-security/web-cach
lab-wcd-exploiting-exact-match-cache-rules

Preventing web cache deception vulnerabilities

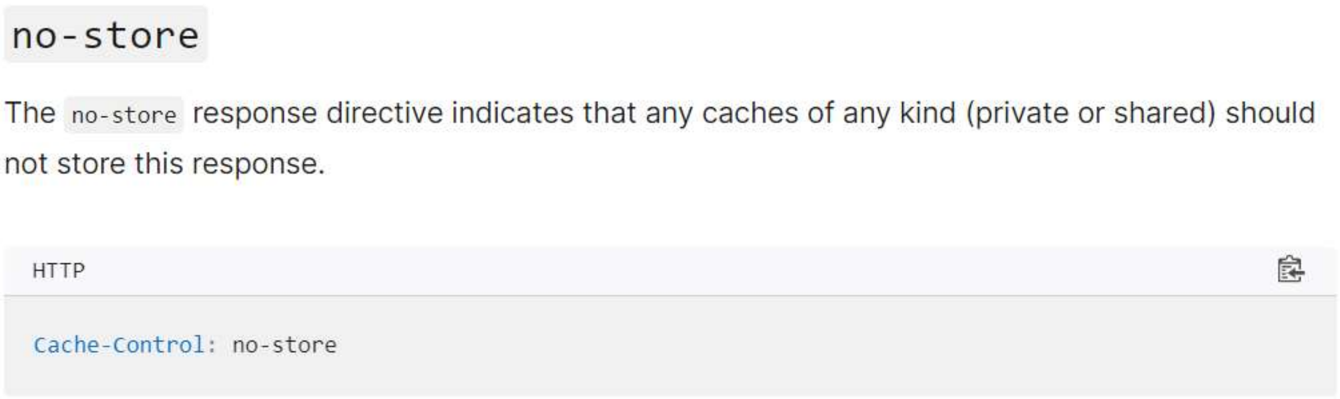


Preventing web cache deception vulnerabilities

You can take a range of steps to prevent web cache deception vulnerabilities:

- Always use `Cache-Control` headers to mark dynamic resources, set with the directives `no-store` and `private`.
- Configure your CDN settings so that your caching rules don't override the `Cache-Control` he
- Activate any protection that your CDN has against web cache deception attacks. Many CDNs er you to set a cache rule that verifies that the response `Content-Type` matches the request's U extension. For example, Cloudflare's Cache Deception Armor.
- Verify that there aren't any discrepancies between how the origin server and the cache interpret paths.

Preventing web cache deception vulnerabilities



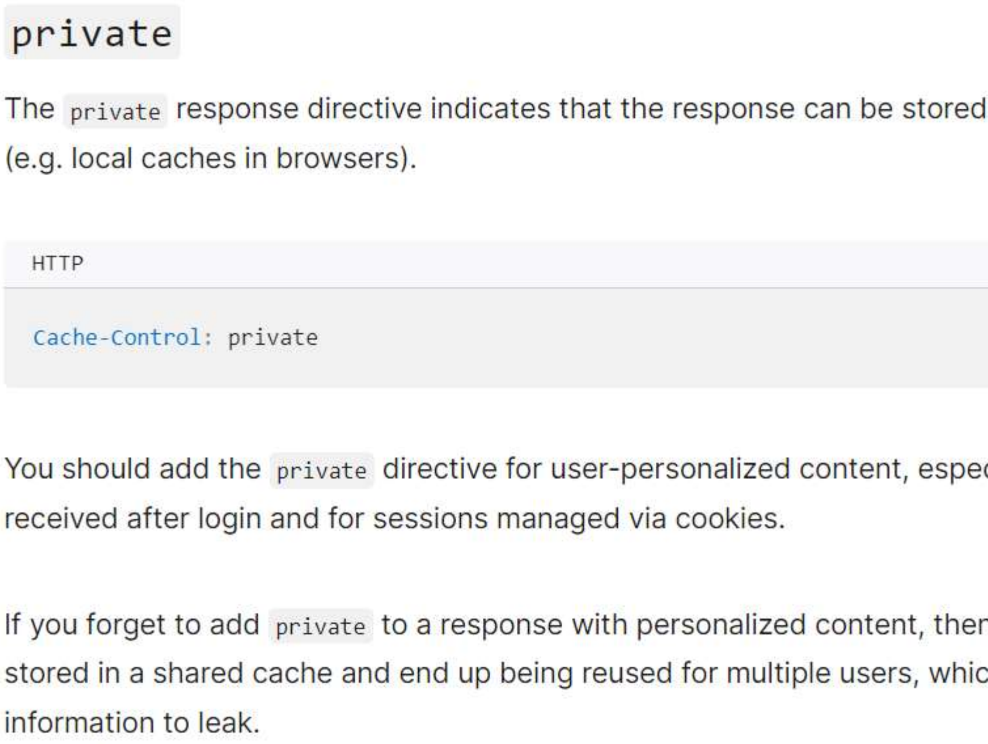
no-store

The `no-store` response directive indicates that any caches of any kind (private or shared) should not store this response.

HTTP

```
Cache-Control: no-store
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control



private

The `private` response directive indicates that the response can be stored (e.g. local caches in browsers).

HTTP

```
Cache-Control: private
```

You should add the `private` directive for user-personalized content, espe received after login and for sessions managed via cookies.

If you forget to add `private` to a response with personalized content, then stored in a shared cache and end up being reused for multiple users, whic information to leak.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control

WIDEN YOUR WORLD

TURKISH AIRLINES