Exposing Private Information from Side-Channel Leaks in your Browser

by Tom Van Goethem

PhD researcher @ imec - DistriNet, University of Leuven

Compression-based Attacks

“*Commonly-used lossless compression algorithms leak information about the data being compressed, in the size of the compressor output.*”

J. Kelsey - "Compression and Information Leakage of Plaintext" (2002)

Compression-based Attacks

- To reduce bandwidth, most websites use gzip compression
- As a direct result, they may become susceptible to compression-based attacks
    - Requirement 1: attacker input is on same page as secret
    - Requirement 2: attacker can determine exact (compressed) response size

```
GET /search?q=value
```

```
<title>Results for: value</title>
....
<input name="csrftoken" value="s3cr3t" type="hidden">
```

gzip

```
<title>Results for: value</title>
....
<input name="csrftoken" @(5,218)="s3cr3t" type="hidden">
```

23,844 bytes

```
GET /search?q=value="a
```

```
<title>Results for: value="a</title>
....
<input name="csrftoken" value="s3cr3t" type="hidden">
```

gzip

```
<title>Results for: value="a</title>
....
<input name="csrftoken" @(7,221)s3cr3t" type="hidden">
```

23,845 bytes

```
GET /search?q=value="s
```

```
<title>Results for: value="s</title>
....
<input name="csrftoken" value="s3cr3t" type="hidden">
```

gzip

```
<title>Results for: value="s</title>
....
<input name="csrftoken" @(8,221)3cr3t" type="hidden">
```

23,844 bytes

```
GET /search?q=value="sa
```

```
<title>Results for: value="sa</title>
....
<input name="csrftoken" value="s3cr3t" type="hidden">
```

gzip

```
<title>Results for: value="sa</title>
....
<input name="csrftoken" @(8,222)3cr3t" type="hidden">
```

23,845 bytes

```
GET /search?q=value="s3
```

```
<title>Results for: value="s3</title>
....
<input name="csrftoken" value="s3cr3t" type="hidden">
```

gzip

```
<title>Results for: value="s3</title>
....
<input name="csrftoken" @(9,222)cr3t" type="hidden">
```

23,844 bytes

Compression-based attacks

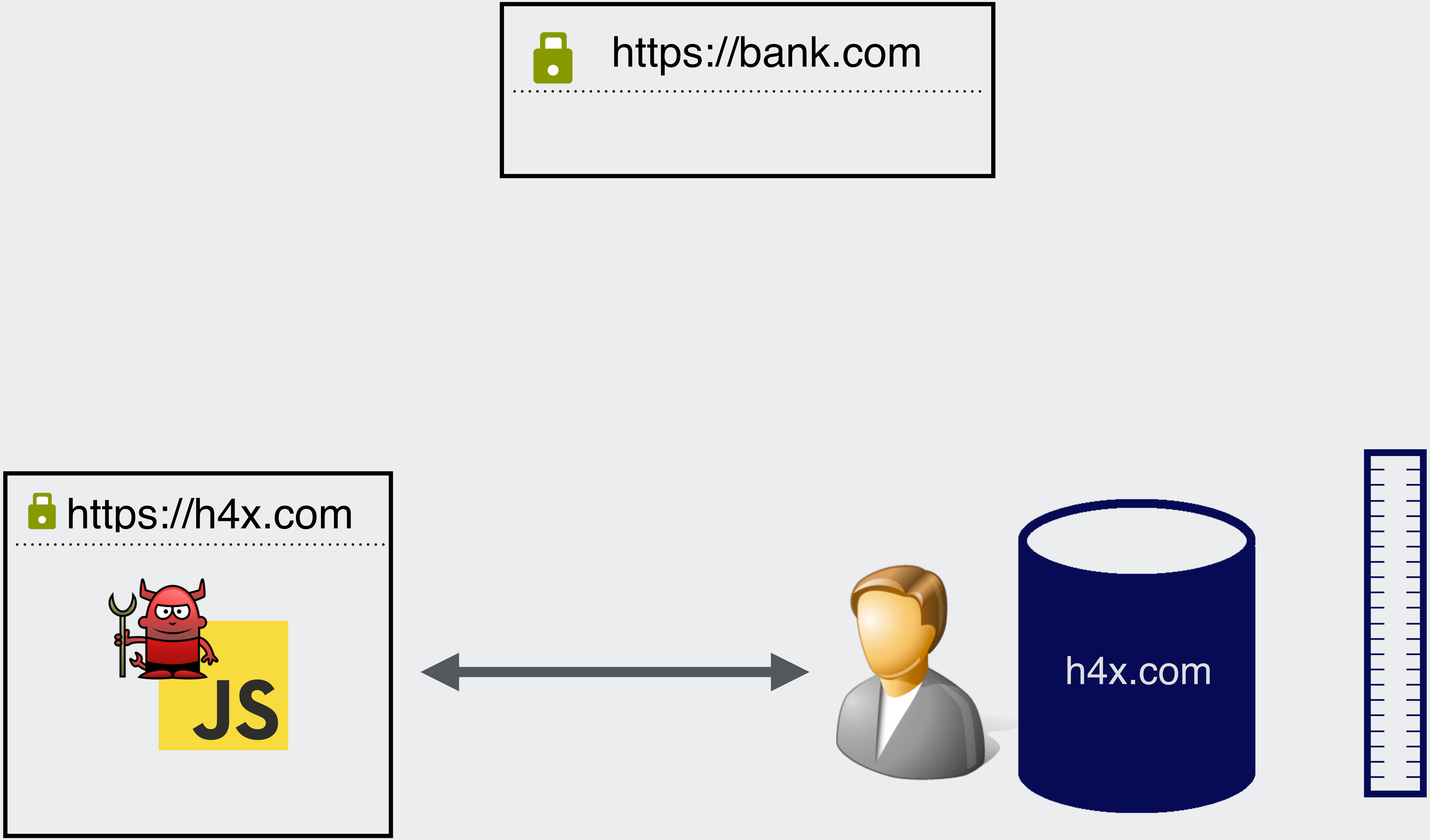
- Requirements for an attacker to extract secret information from a web page
  - gzip compression enabled
    - present on most websites
  - attacker-controlled input on the same page as the secret
    - application-specific
    - attacker only needs a single page that meets this requirement
  - determine exact response size (compressed)

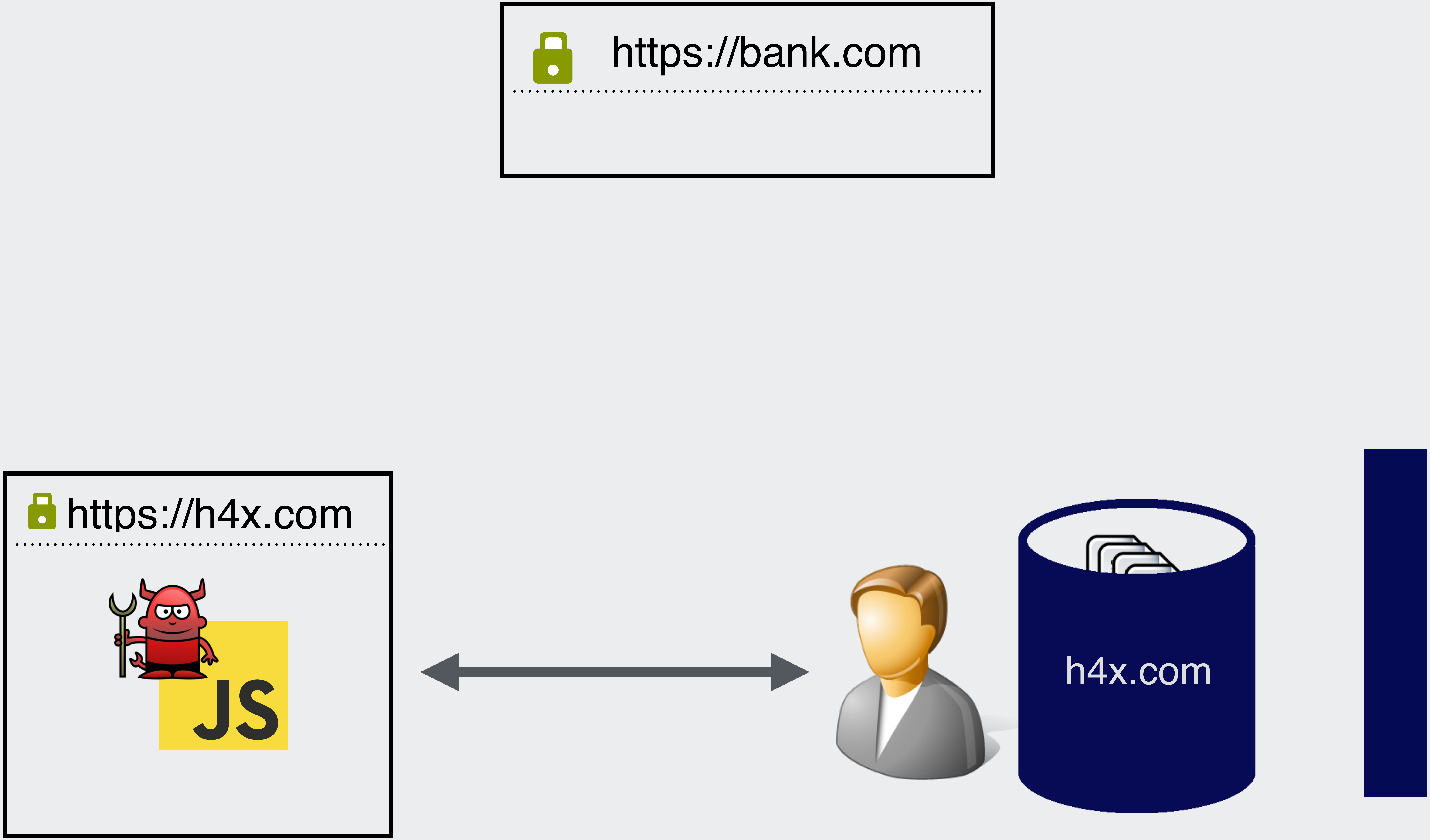
Determine exact response size

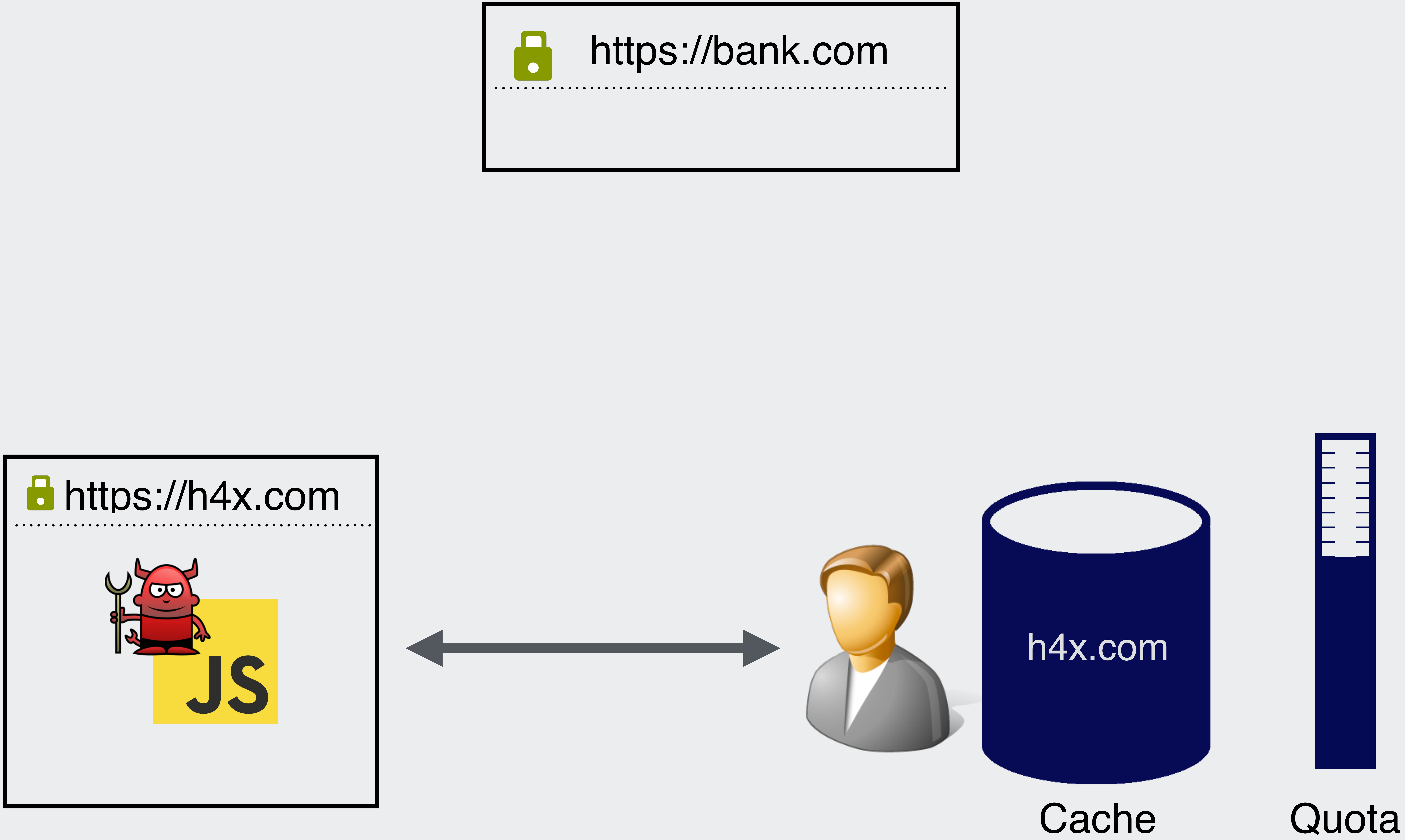
- Man-in-the-middle
  - Trivial
- Sniff (encrypted) Wi-Fi packets
  - Channel-based man-in-the-middle attack
- Browser-based side channel attack
  - Browser storage
  - TCP windows + browser timing APIs

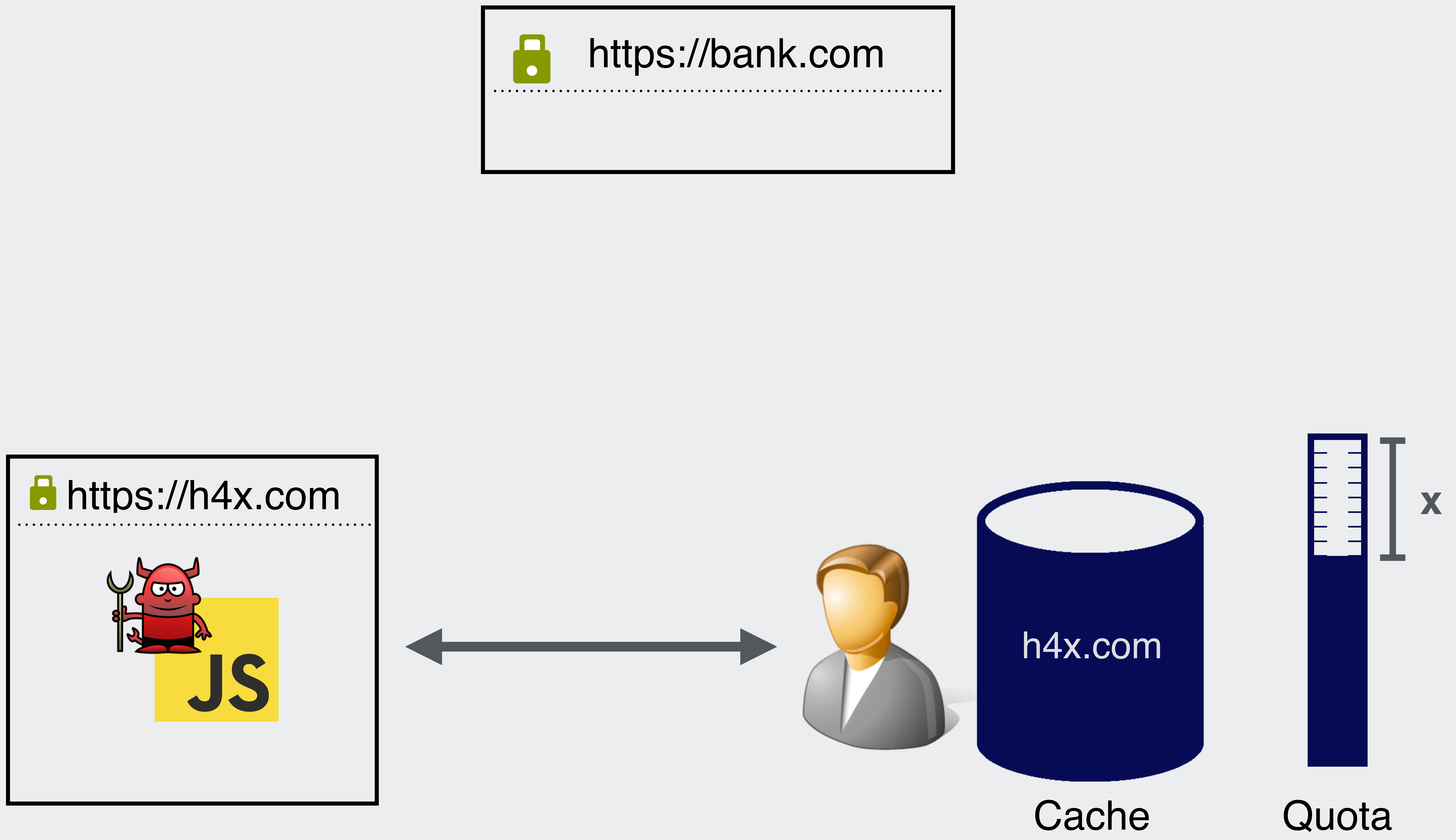
Browser storage side-channel

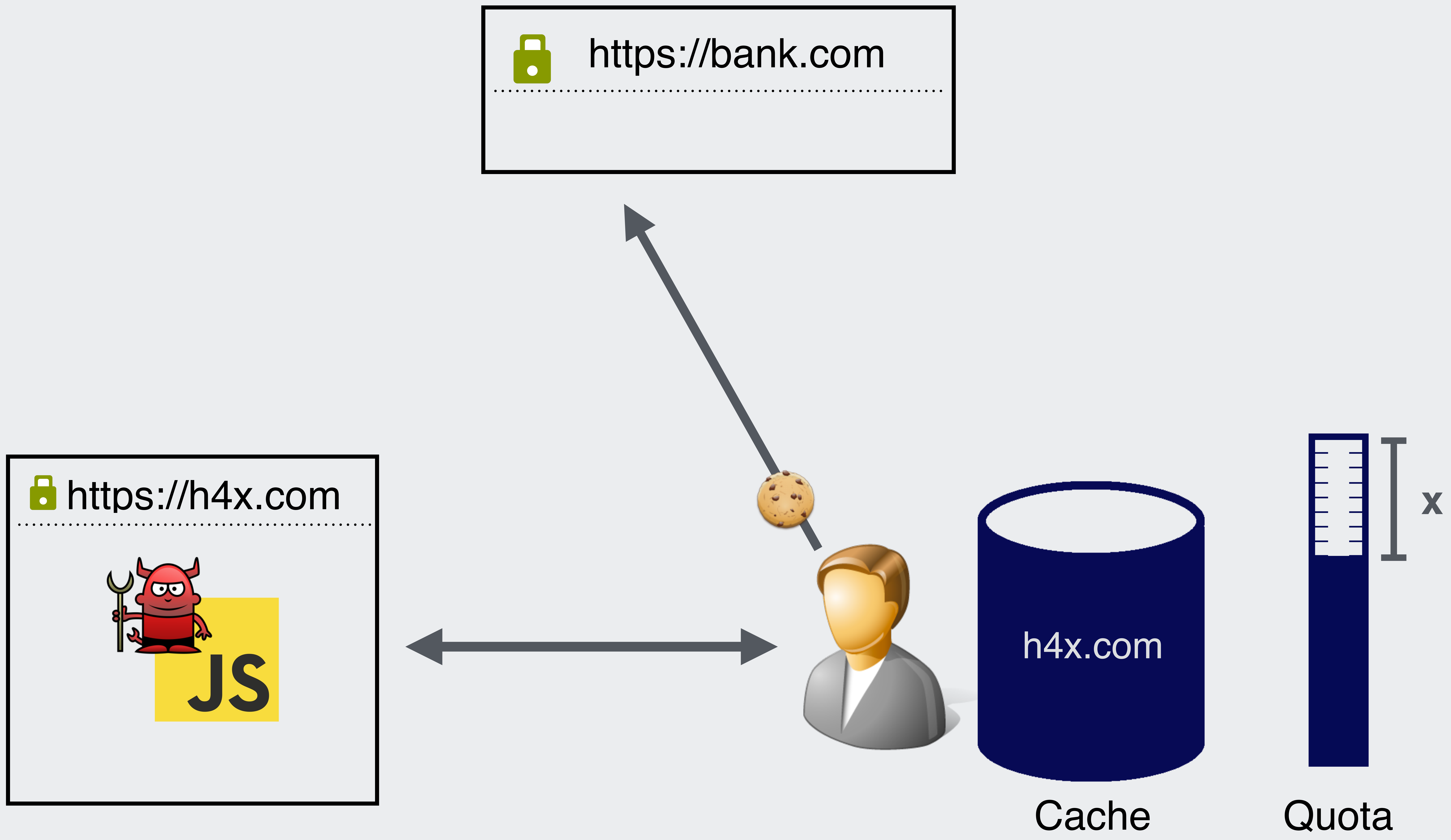
- Cache API introduces programmable cache
  - Part of service worker API
  - Allows web developers to place *any* resource in website's cache
    - Including authenticated cross-origin responses
- To prevent one party to take up all available space, the Cache API is subject to quota restrictions
  - Main cause of the side-channel leak

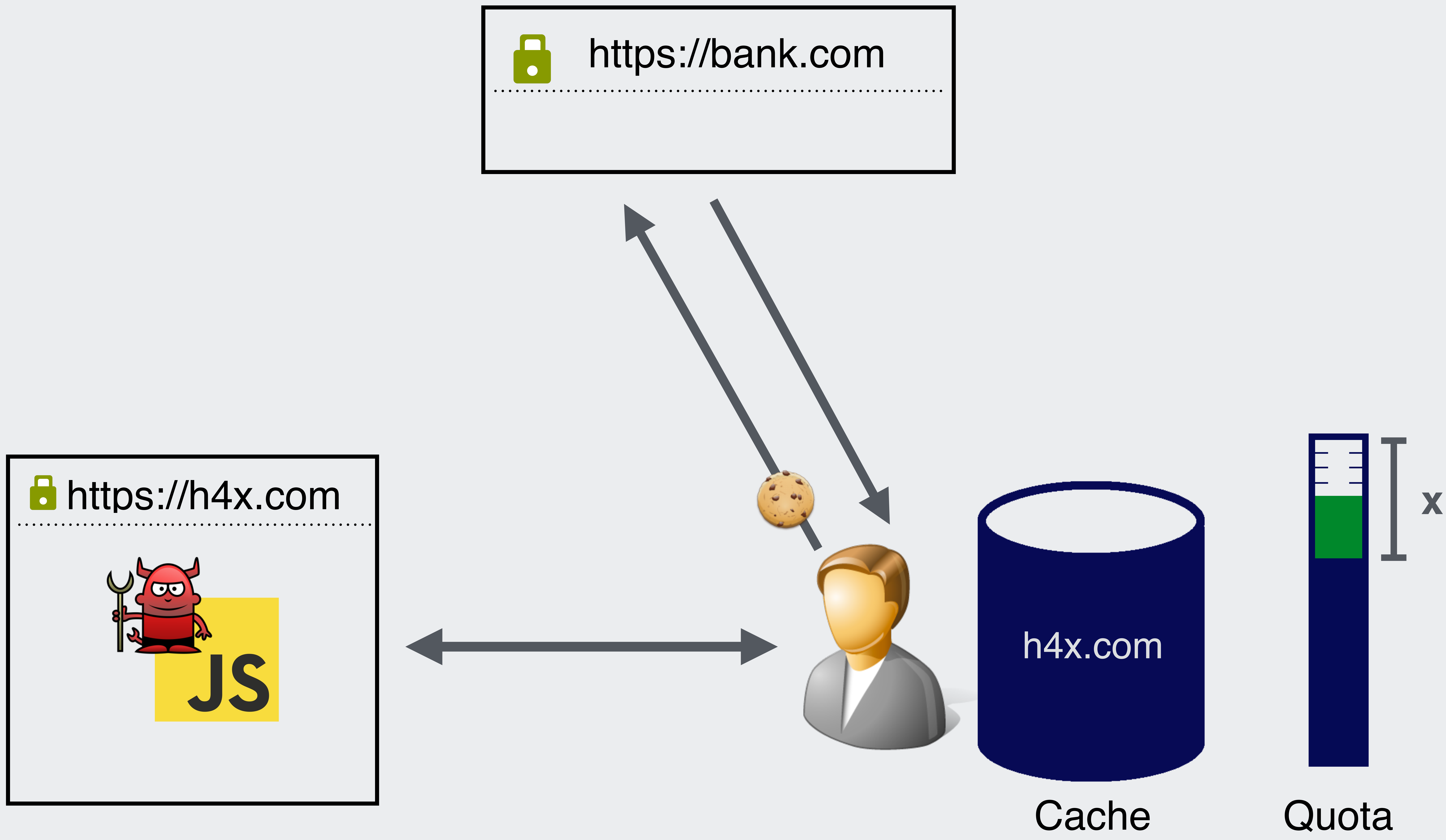

https://bank.com
https://h4x.com
JS
h4x.com

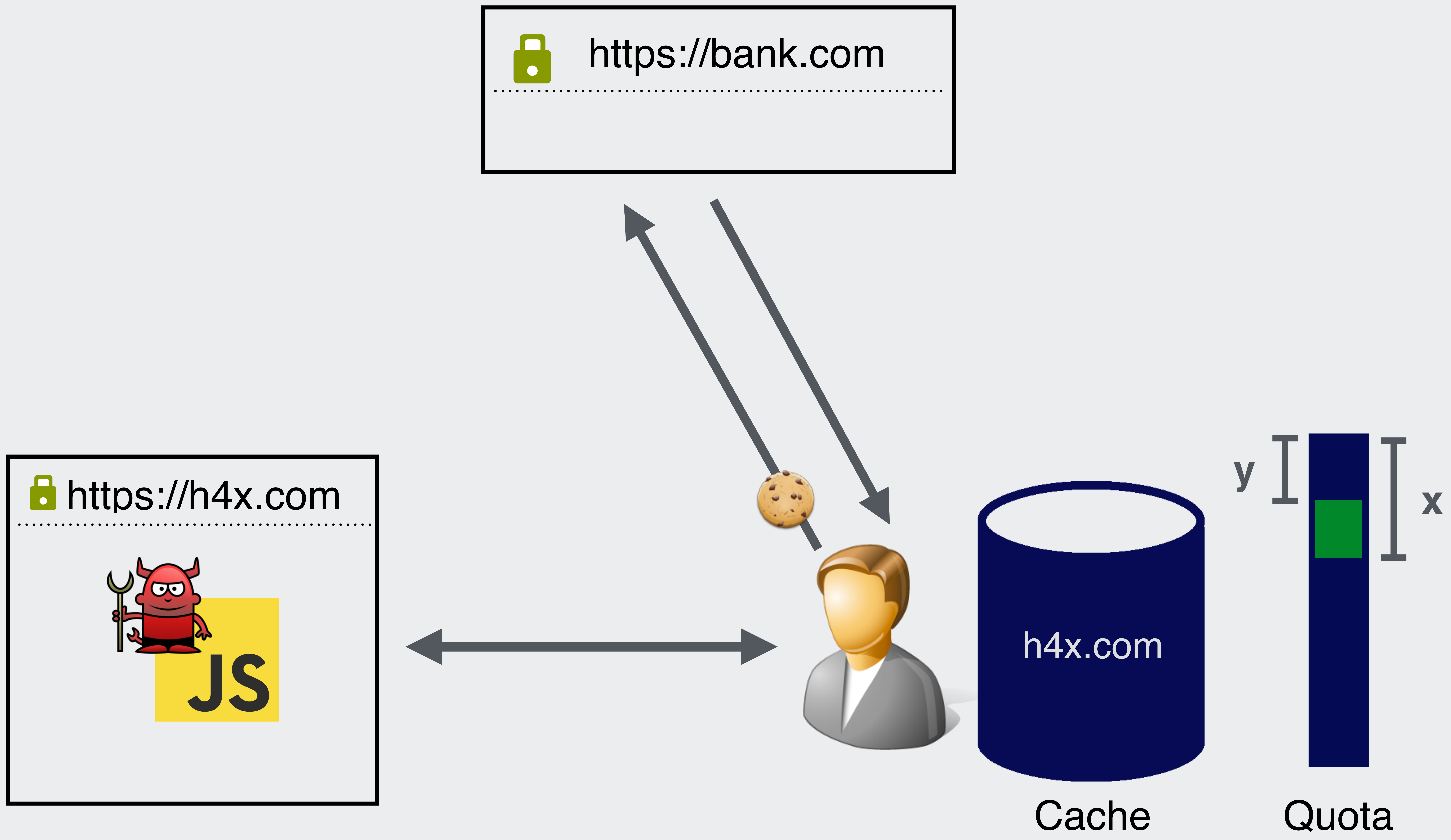

https://bank.com
https://h4x.com
JS
h4x.com

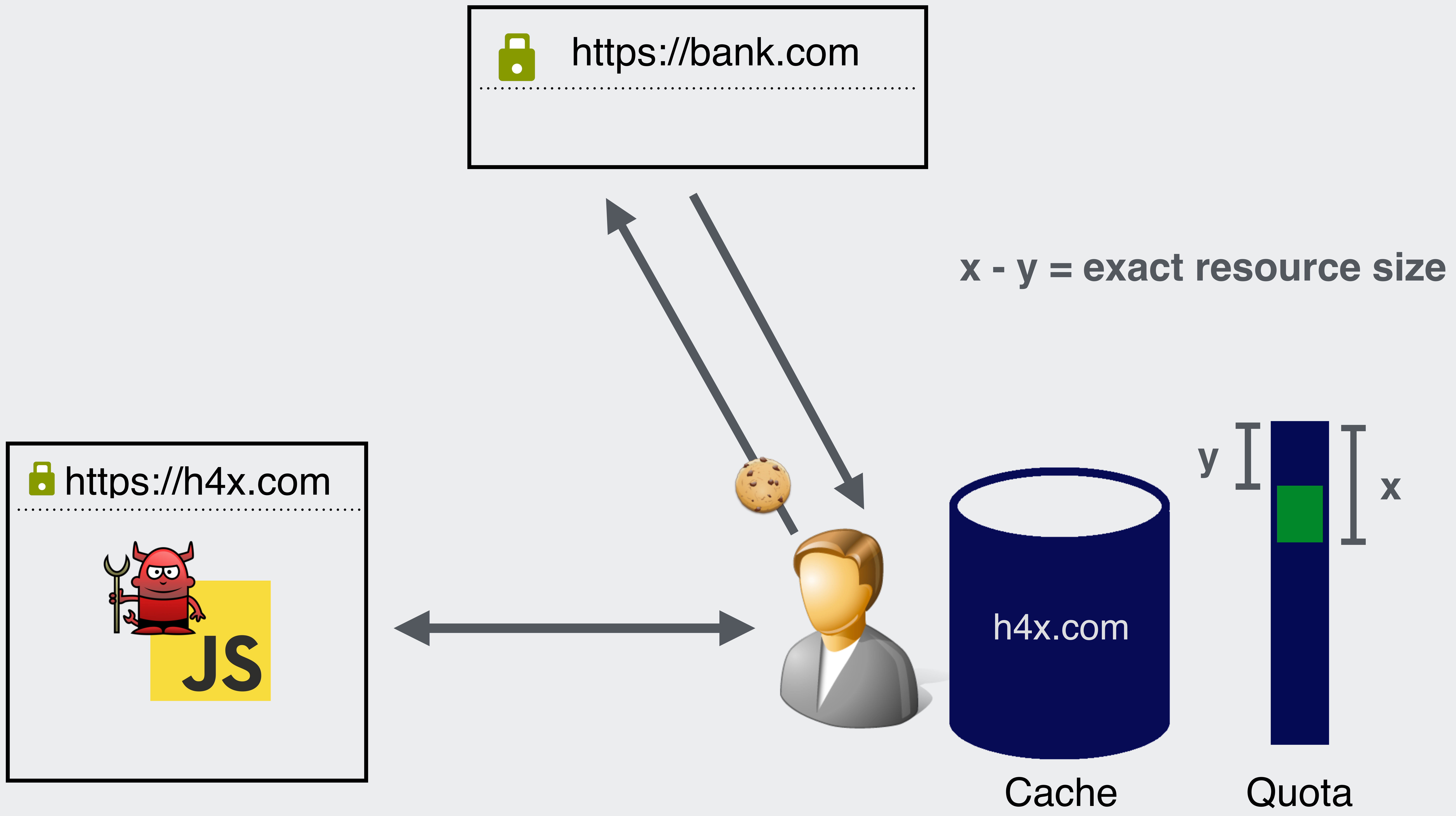

https://bank.com
https://h4x.com
JS
h4x.com
Cache
Quota


https://bank.com
https://h4x.com
JS
h4x.com
x
Cache
Quota

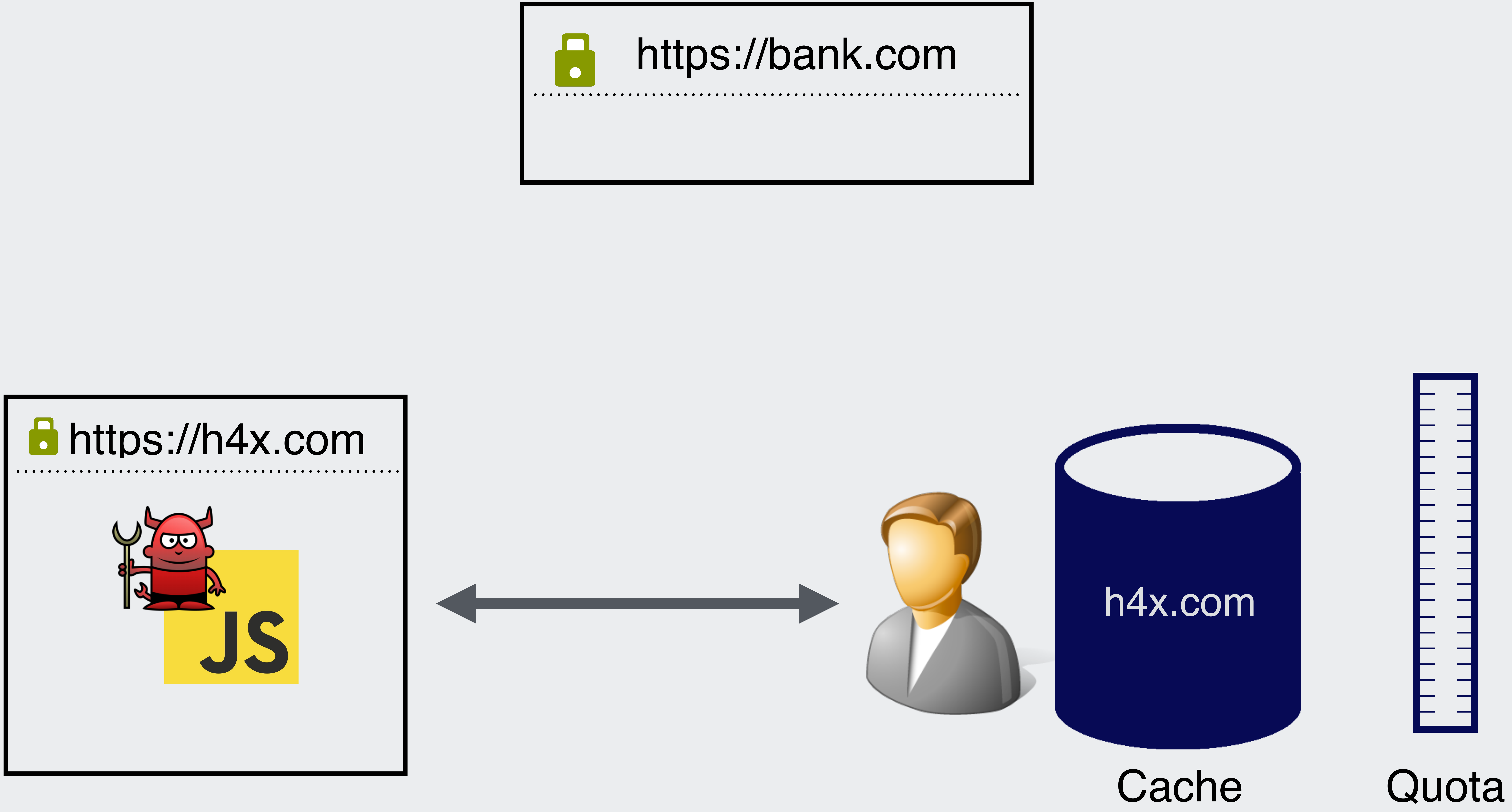

https://bank.com
https://h4x.com
JS
h4x.com
x
Cache
Quota


https://bank.com
https://h4x.com
JS
h4x.com
x
Cache
Quota

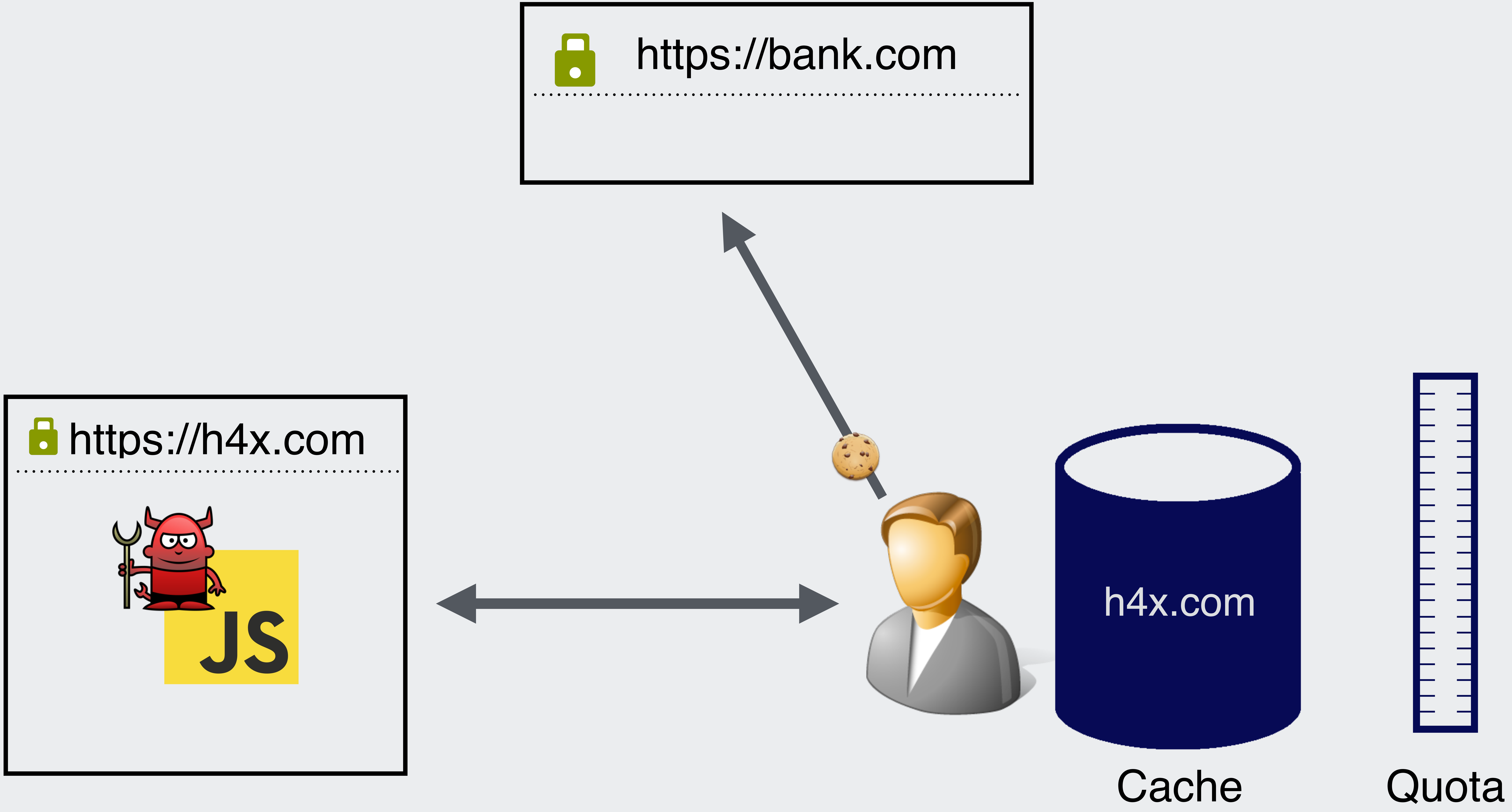

https://bank.com
https://h4x.com
JS
h4x.com
y
x
Cache
Quota


https://bank.com
x - y = exact resource size
https://h4x.com
JS
y
x
h4x.com
Cache
Quota

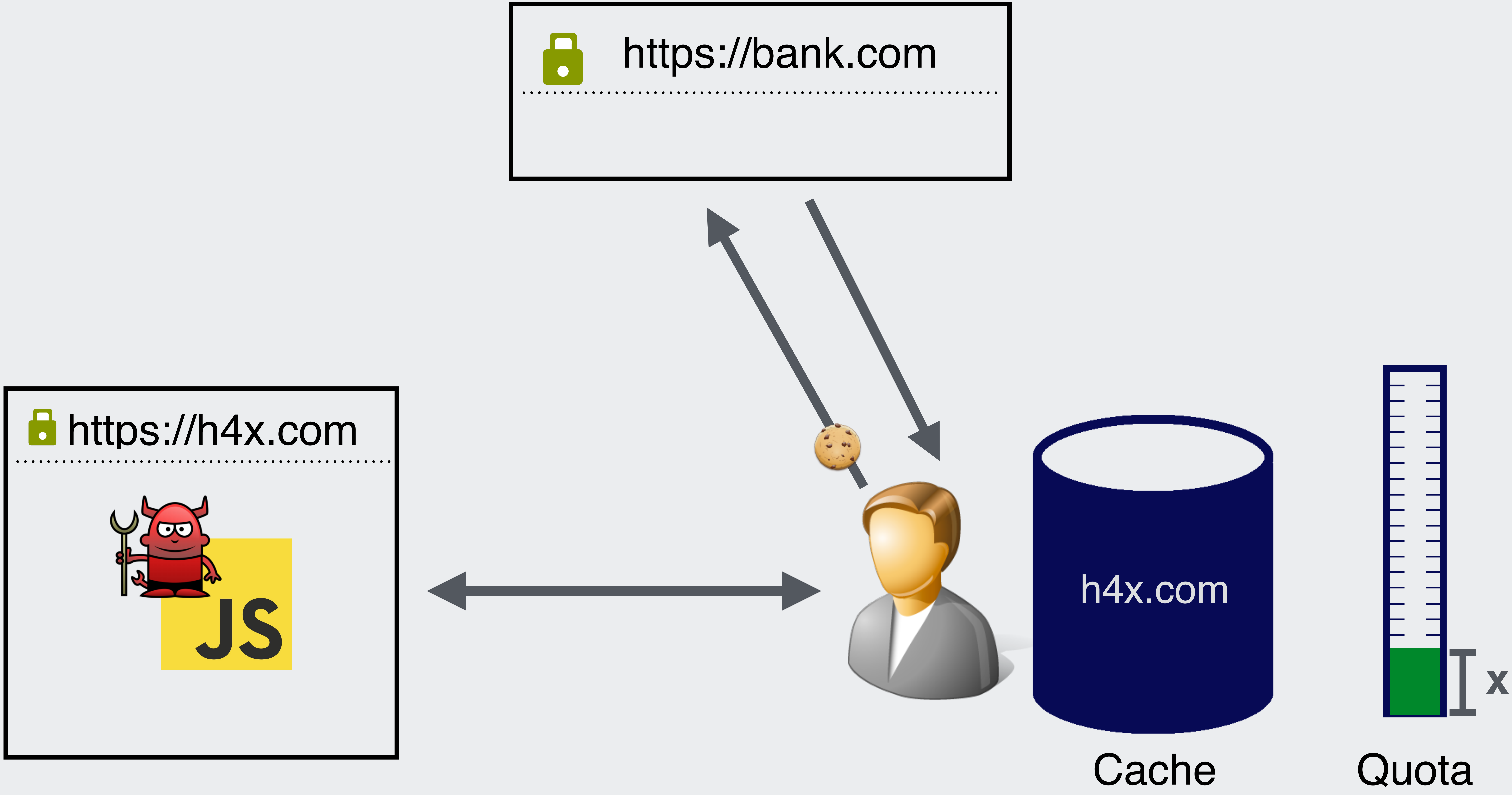
Quota Management/Storage API



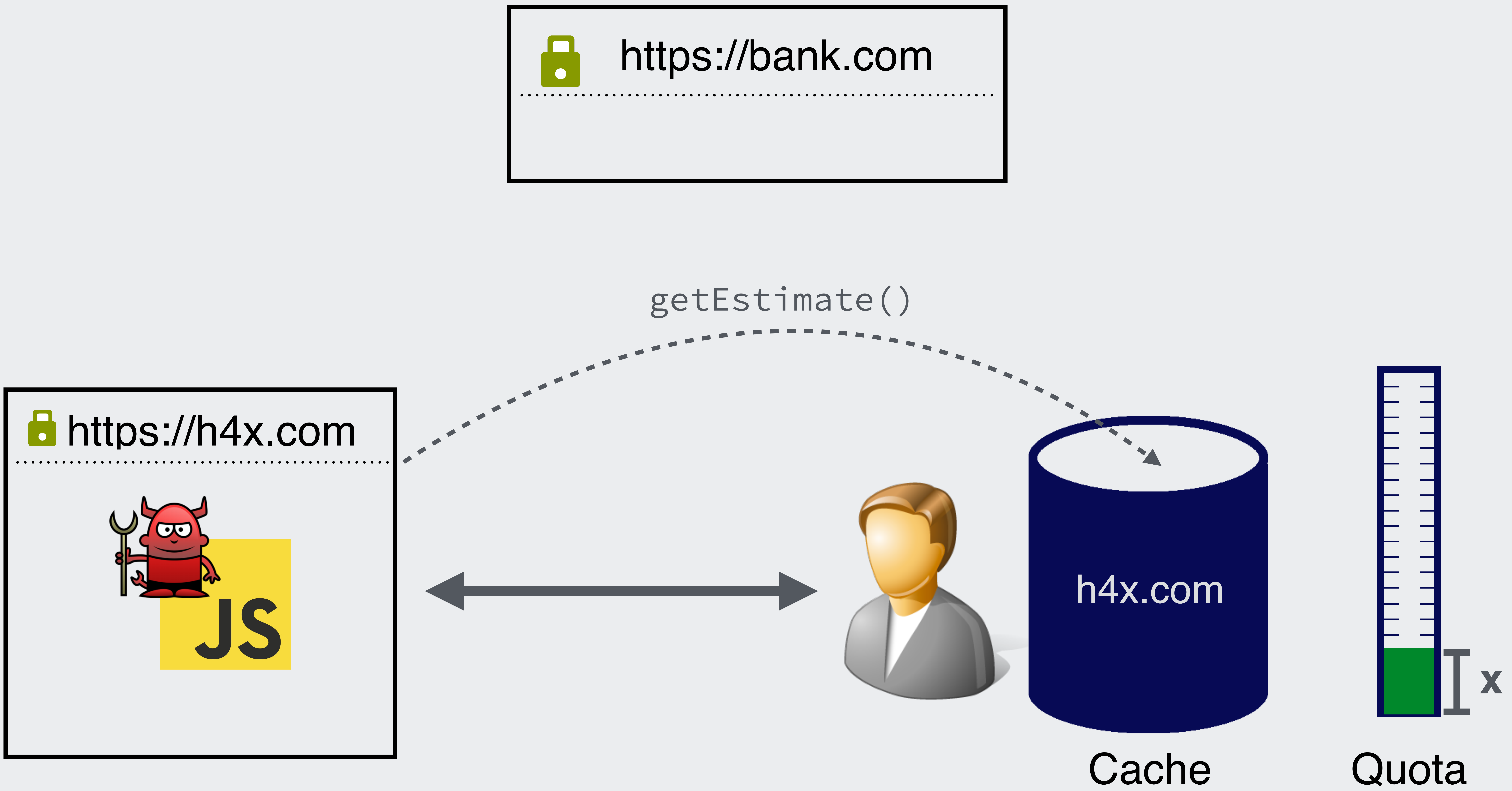
Quota Management/Storage API



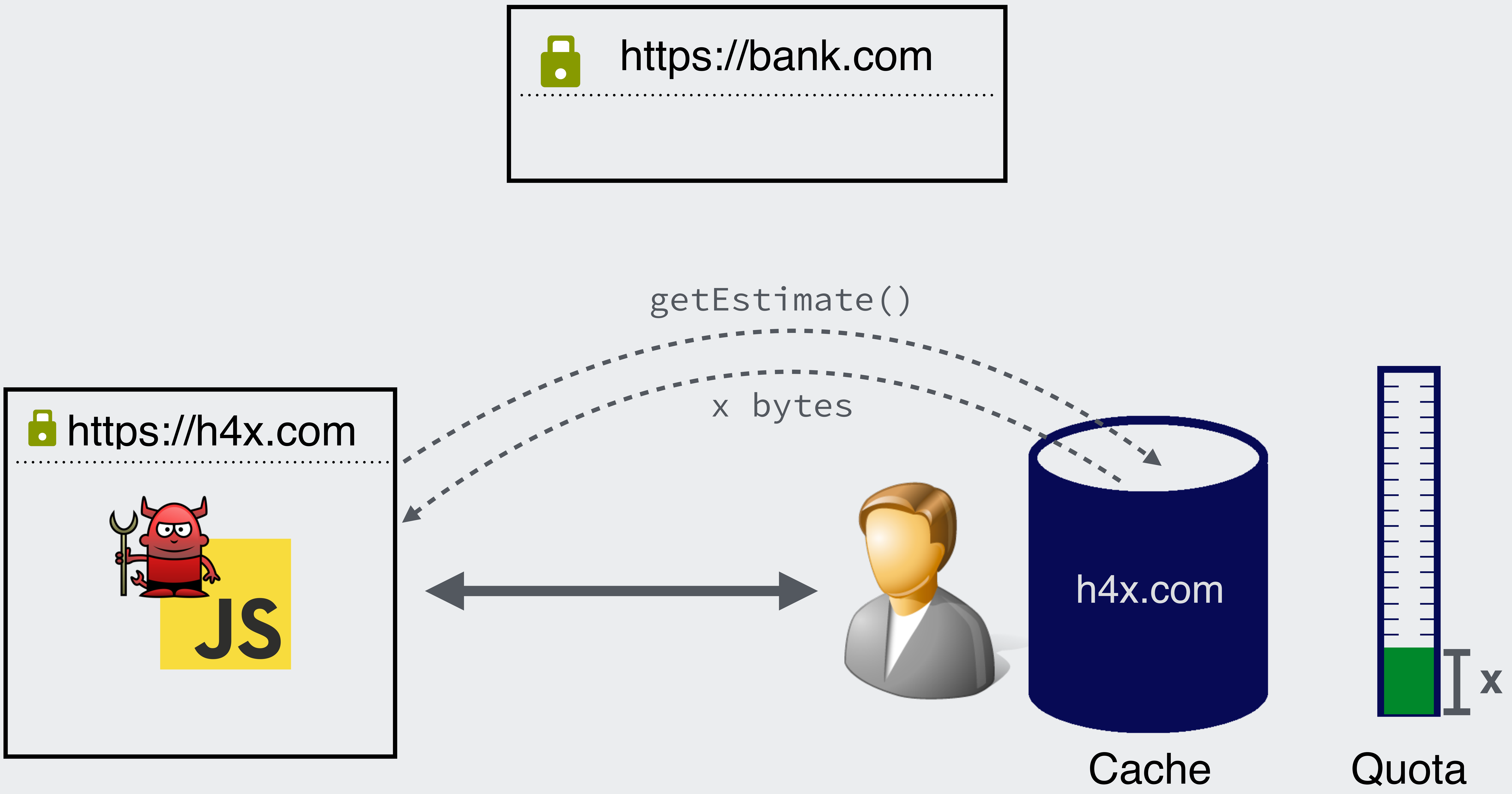
Quota Management/Storage API



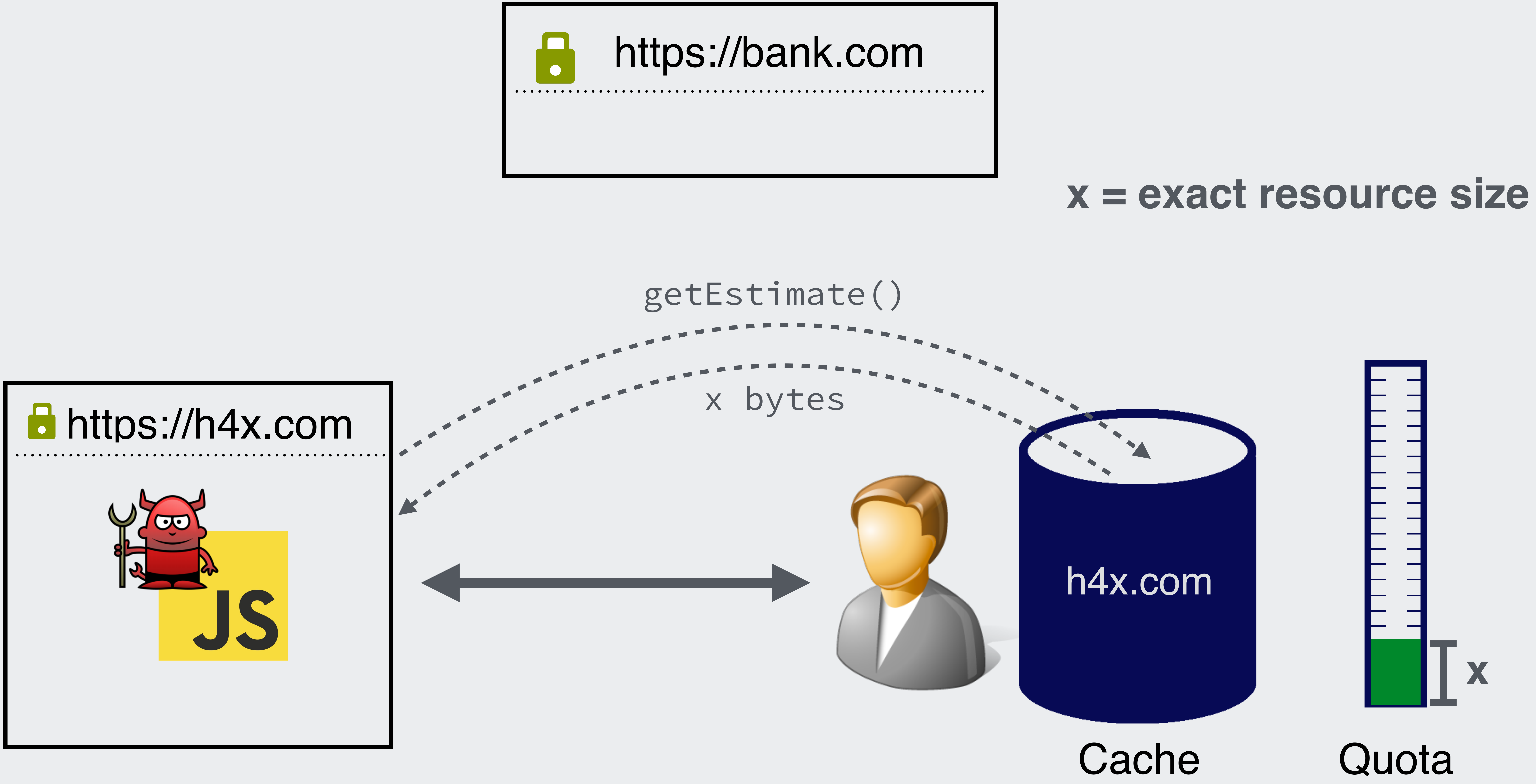
Quota Management/Storage API



Quota Management/Storage API



Quota Management/Storage API



- Exploiting Cache API/Quota Mgmt, we can find exact response size
  - But... *after* decompression → insufficient to launch compression-based attacks
- Cache API stores uncompressed response + headers
  - Perhaps we can abuse something there?

```
GET /search?q=foobar HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 9102
...

<!DOCTYPE html><html>
<title>Results for: foobar</title>
...
```

```
GET /search?q=foobar HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

response + headers + meta = 10,703 bytes

```
Content-Length: 9102
...

<!DOCTYPE html><html>
<title>Results for: foobar</title>
...
```

```
GET /search?q=foobars HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 9103
...

<!DOCTYPE html><html>
<title>Results for: foobars</title>
...
```

```
GET /search?q=foobars HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

response + headers + meta = 10,704 bytes

```
Content-Length: 9103
...

<!DOCTYPE html><html>
<title>Results for: foobars</title>
...
```

```
GET /search?q={897 bytes}foobar HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 9999
...

<!DOCTYPE html><html>
<title>Results for: {897 bytes}foobar</title>
...
```

```
GET /search?q={897 bytes}foobar HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

response + headers + meta = 11,600 bytes

```
Content-Length: 9999
...

<!DOCTYPE html><html>
<title>Results for: {897 bytes}foobar</title>
...
```

```
GET /search?q={897 bytes}foobars HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 10000
...

<!DOCTYPE html><html>
<title>Results for: {897 bytes}foobars</title>
...
```

```
GET /search?q={897 bytes}foobars HTTP/1.1
Host: example.com
User-Agent: Web Browser
```

response + headers + meta = 11,602 bytes

```
Content-Length: 10000
...

<!DOCTYPE html><html>
<title>Results for: {897 bytes}foobars</title>
...
```

GET param length	Content-Length	Cache size
6	9,102	10,703
7	9,103	10,704
903	9,999	11,600
904	10,000	11,602

- Exploiting Cache API/Quota Mgmt, we can find exact response size
  - But... *after* decompression → insufficient to launch compression-based attacks
- Cache API stores uncompressed response + headers
  - Perhaps we can abuse something there?
  - → YES! The Content-Length header

TCP windows + browser timing APIs

- HTTP responses are sent TCP windows
  - At most, 10 unacknowledged TCP packets can be sent from server to client
- Resources that do not fit in one TCP window require an additional round trip
  - If we can measure this, we can determine if response fits in a single TCP window, or required multiple
- ... at a certain tipping point for the response size, an additional round trip is needed

TCP windows + browser timing APIs

- We can use Fetch API to make authenticated requests

```
fetch('https://bank.com/resource',
        {mode: "no-cors", credentials:"include"})
```

- The returned Promise resolves when the browser receives the first byte of the response

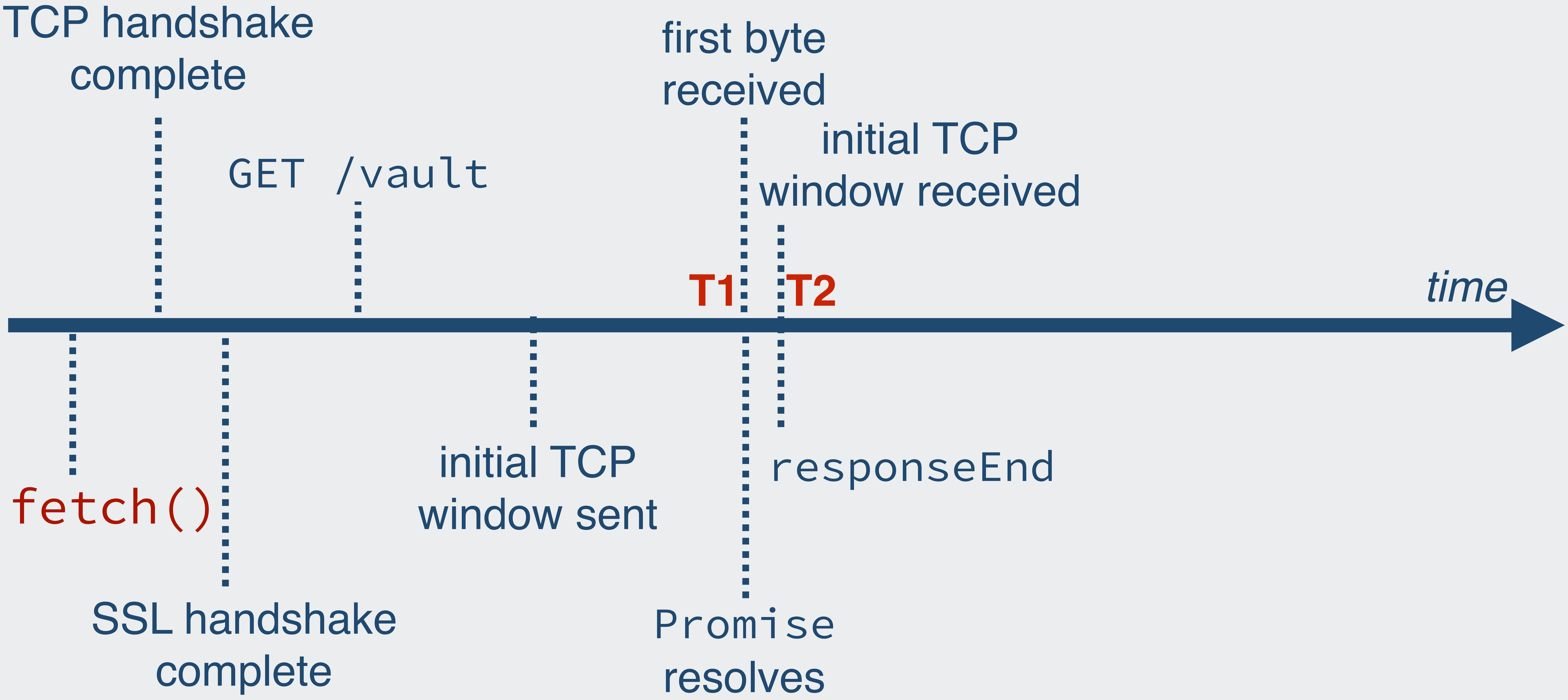
TCP windows + browser timing APIs

- Using the Performance API, we can measure when the response was completely downloaded

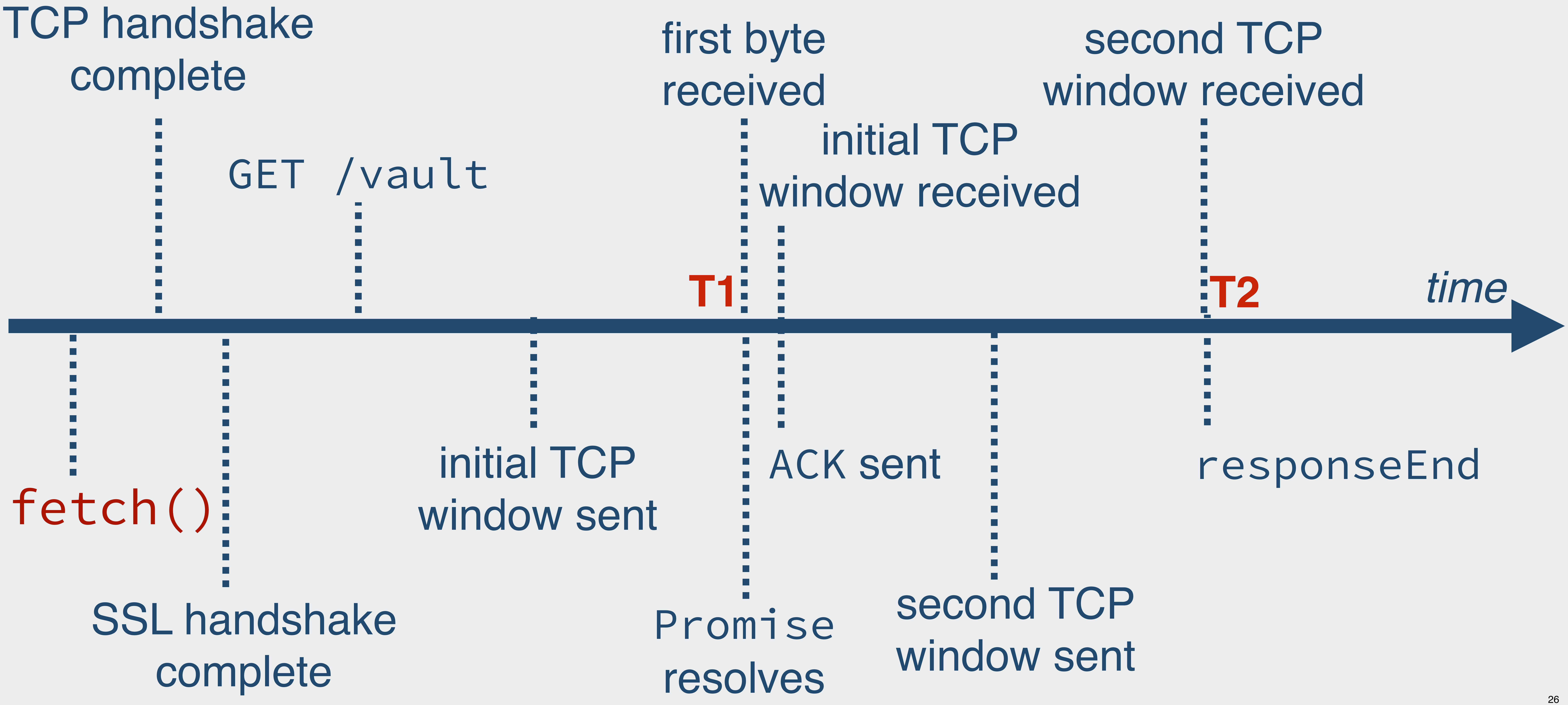
```
performance.getEntries()[-1].responseEnd
```

- This allows us to create a timeline of the response

Fetching small resource: T2 - T1 is very small



Fetching large resource: T2 - T1 is round-trip time



- Measuring time difference between resolution of Promise and responseEnd leaks information on the number of round trips
  - 1 round trip (= 1 TCP window): < 5ms
  - 2+ round trips (= multiple TCP windows): < 5ms + RTT

- Finding the exact size with similar technique as Cache/Quota side-channel attack
    - Add reflected content until tipping point is reached
- For larger resources: arbitrarily increase TCP window by first sending a request to another resource
    - For each received ACK, TCP window is increased by 1

Defence methods

- Disable compression
  - Bandwidth usage +++
- Do not compress secrets
  - How to determine what is secret information?
  - Work in progress
- SameSite cookies
  - Cookies are not attached to third-party requests
- Disable third-party cookies
  - Affects UX on some websites

Conclusion

- Compression-based attacks allow attackers to extract sensitive information (e.g. CSRF tokens)
- Information leaked by browser allows determining exact response size
  - Cache API + Quota
  - Response timing + TCP windows
- Very few websites defend against these attacks

Questions?

@tomvangoethem

tom.vangoethem@cs.kuleuven.be