HTTP
Encrypted
Information can be
Stolen through
TCP-windows

by

Mathy Vanhoef & Tom Van Goethem

Agenda

- Technical background
  - Same-Origin Policy
  - Compression-based attacks
  - SSL/TLS & TCP
- Nitty gritty HEIST details
- Demo
- Countermeasures

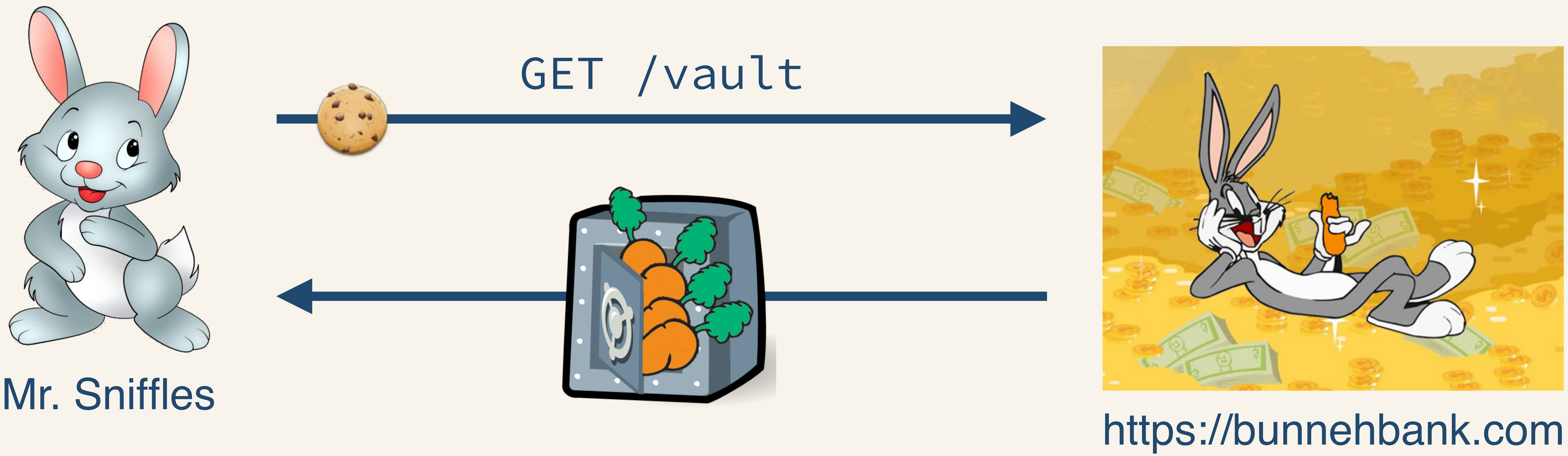
Same-Origin Policy

Mr. Sniffles

`GET /vault`

https://bunnehbank.com

Same-Origin Policy



the World Wide Web

Mr. Sniffles

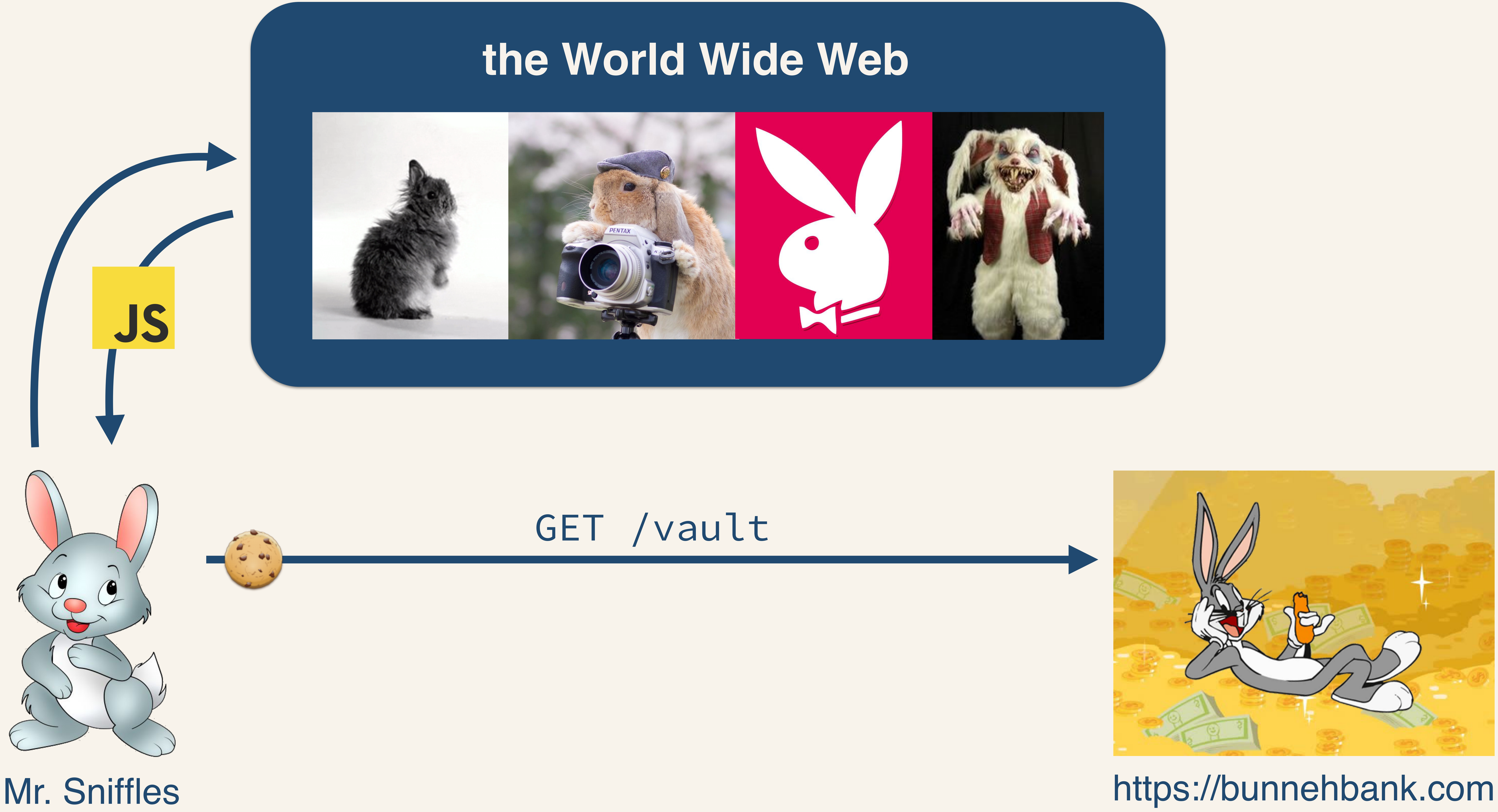
https://bunnehbank.com

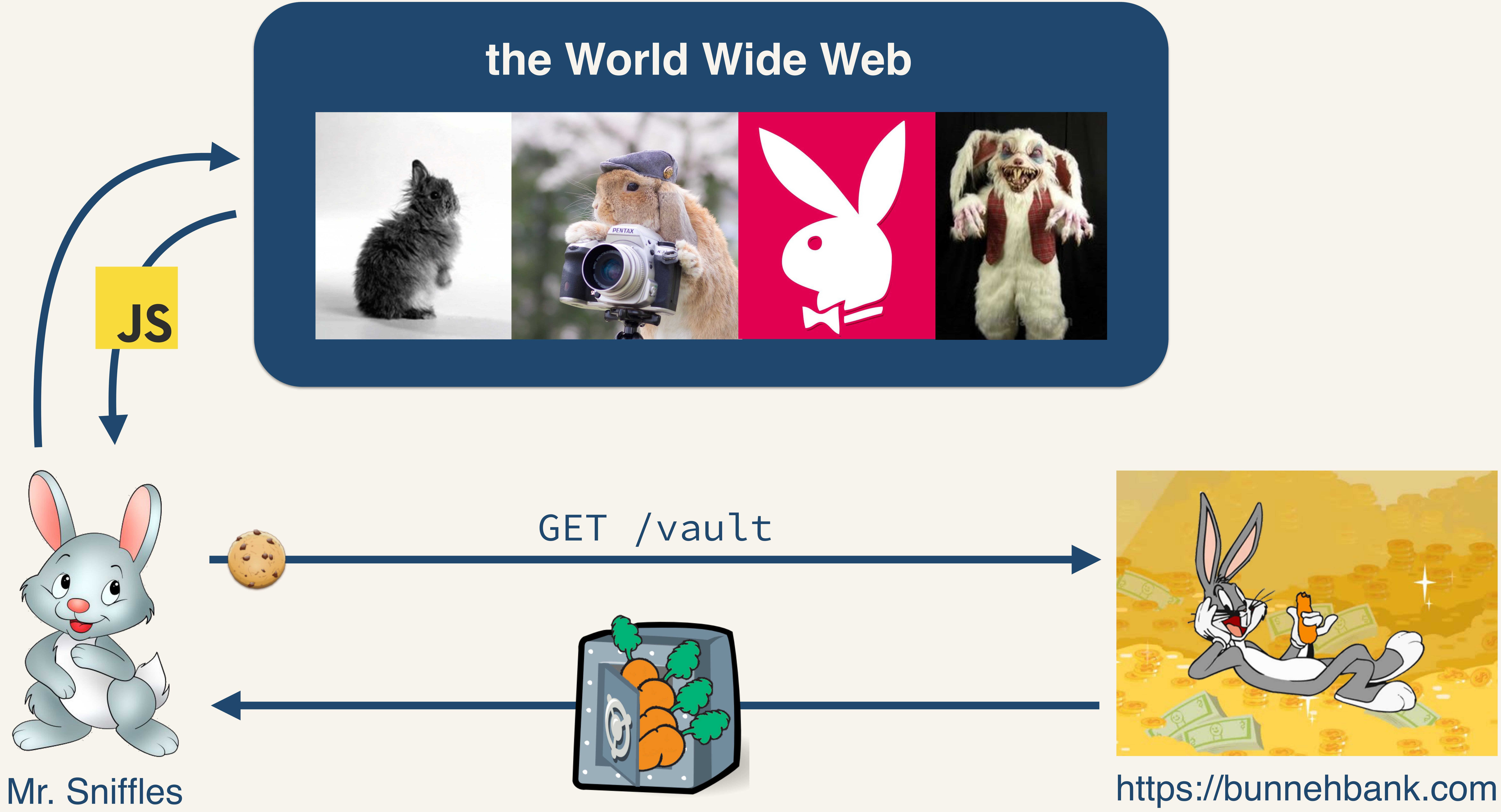
the World Wide Web

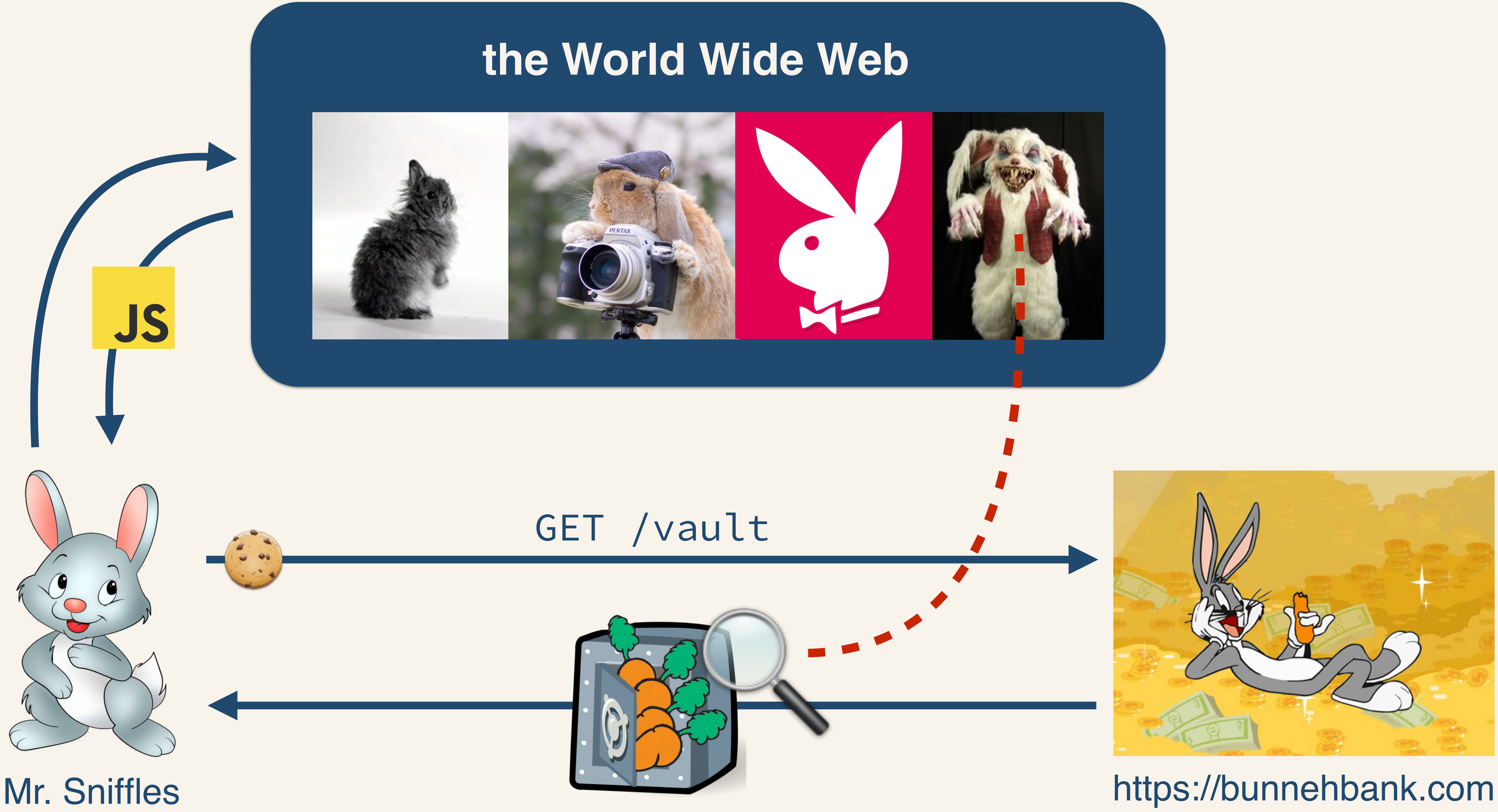
JS

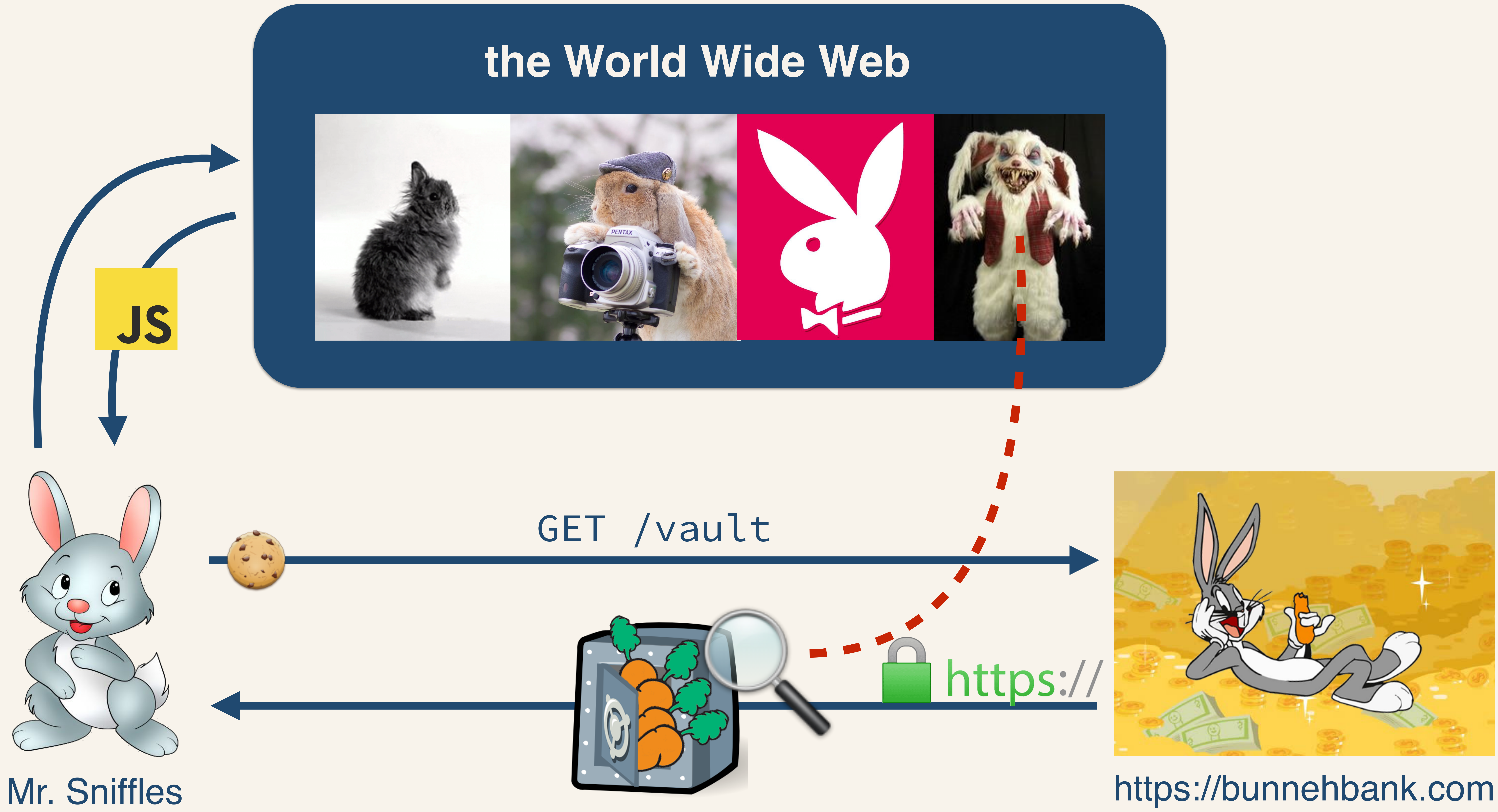
Mr. Sniffles

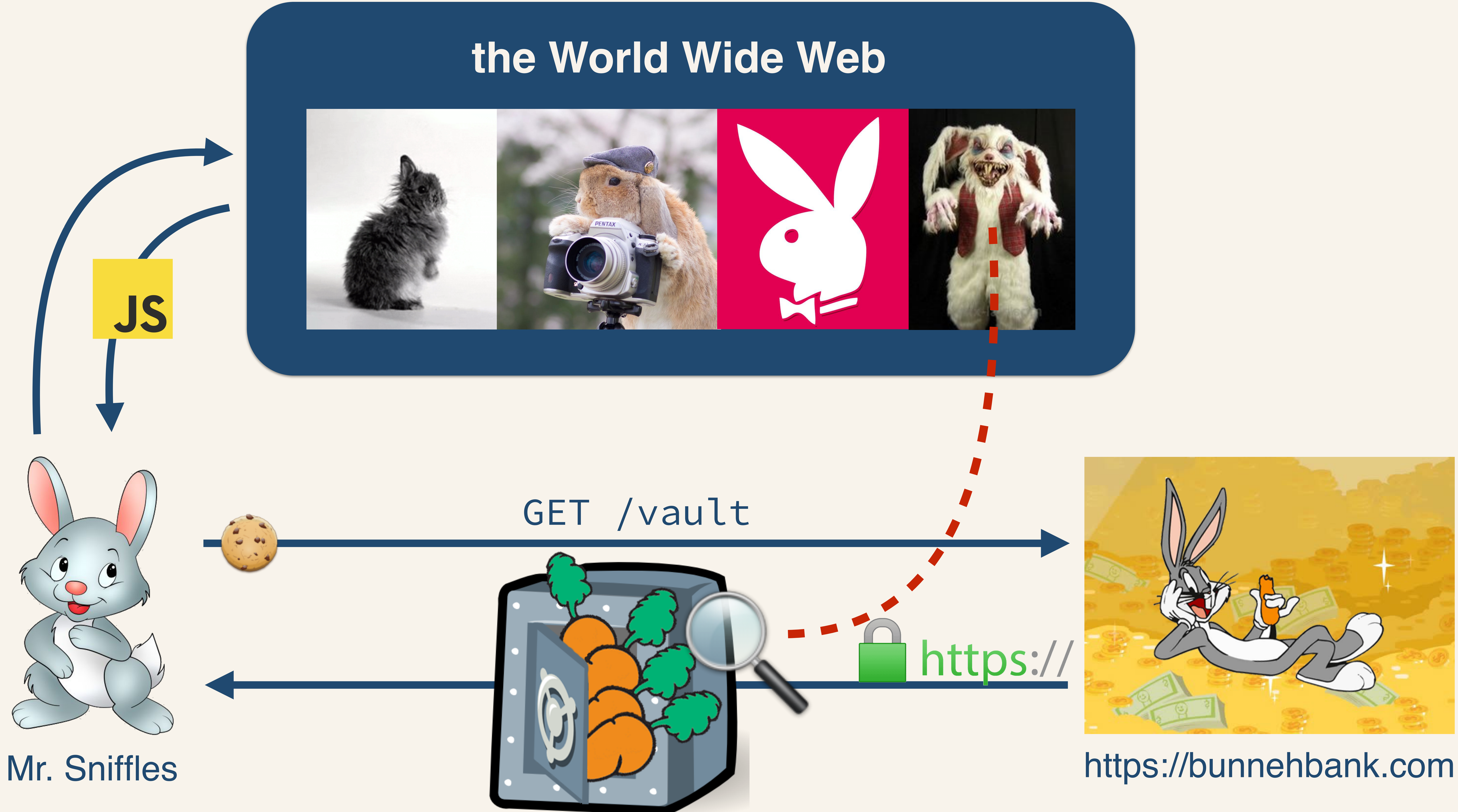
https://bunnehbank.com

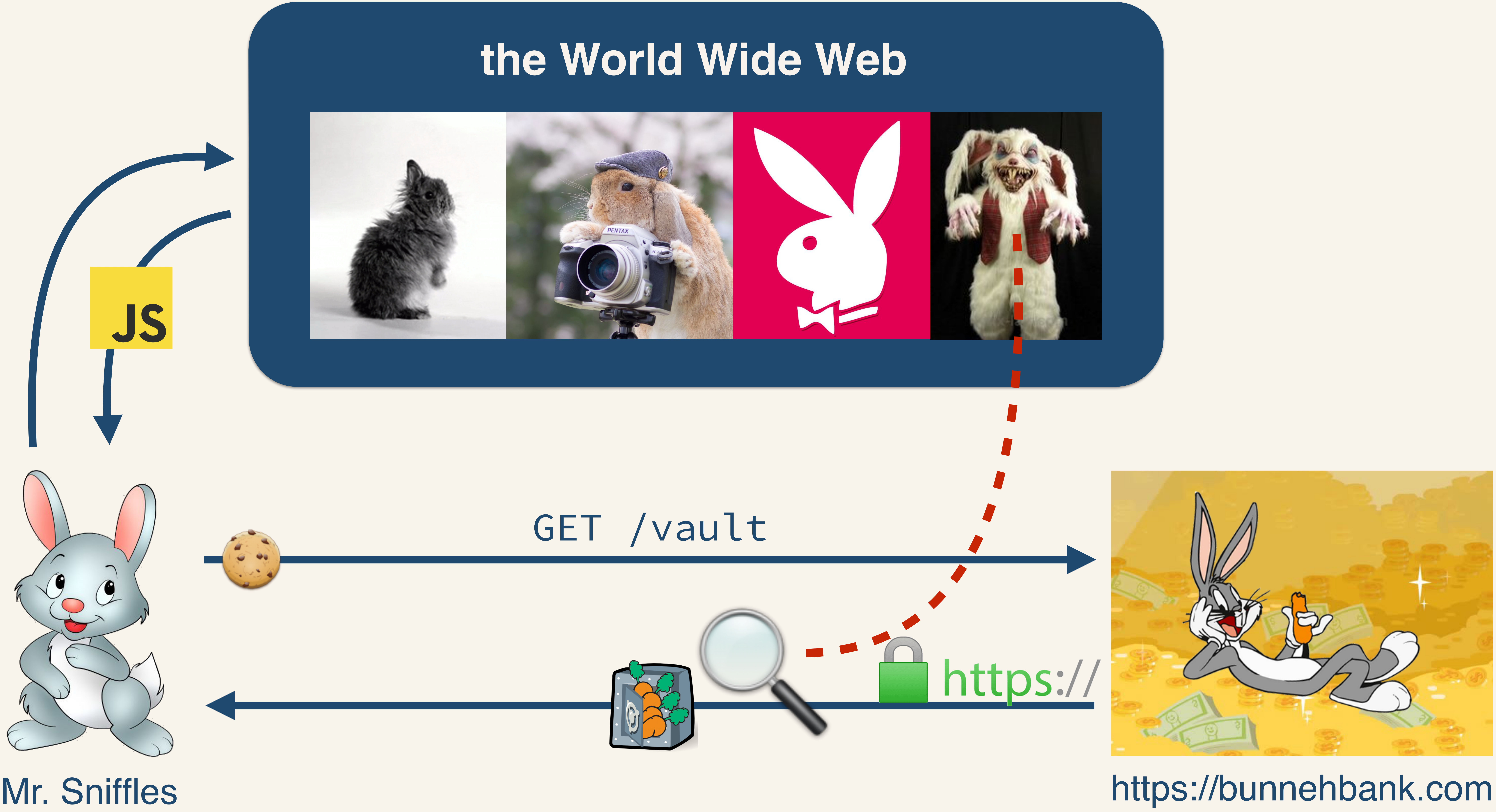

the World Wide Web
JS
GET /vault
Mr. Sniffles
https://bunnehbank.com


the World Wide Web
JS
GET /vault
Mr. Sniffles
https://bunnehbank.com


the World Wide Web
JS
GET /vault
Mr. Sniffles
https://bunnehbank.com


the World Wide Web
JS
GET /vault
https://
Mr. Sniffles
https://bunnehbank.com


the World Wide Web
JS
GET /vault
https://
Mr. Sniffles
https://bunnehbank.com


the World Wide Web
JS
GET /vault
https://
Mr. Sniffles
https://bunnehbank.com

Agenda

- Technical background
  - Same-Origin Policy
  - **Compression-based attacks**
  - SSL/TLS & TCP
- Nitty gritty HEIST details
- Demo
- Countermeasures

/vault

Uncompressed	Compressed

You requested:
  /vault
 	
vault_secret=carrots4life
	You requested:
	  /vault
	
	     _secret=carrots4life

→ 51 bytes	→ 47 bytes

```
/vault?secret=a
```

```
You requested:
  /vault?secret=a
```

```
_
carrots4life
```

→ 50 bytes

```
/vault?secret=c
```

```
You requested:
  /vault?secret=c
```

```
_
arrots4life
```

→ 49 bytes

`/vault?secret=a`

`/vault?secret=c`

49 bytes < 50 bytes → 'c' is a correct guess

→ 50 bytes

→ 49 bytes

`/vault?secret=ca`

`/vault?secret=cb`

```
You requested:
  /vault?secret=ca
_              rrots4life
```

→ 49 bytes

```
You requested:
  /vault?secret=cb
_              arrots4life
```

→ 50 bytes

`/vault?secret=ca`

`/vault?secret=cb`

49 bytes < 50 bytes → 'ca' is a correct guess

→ 49 bytes

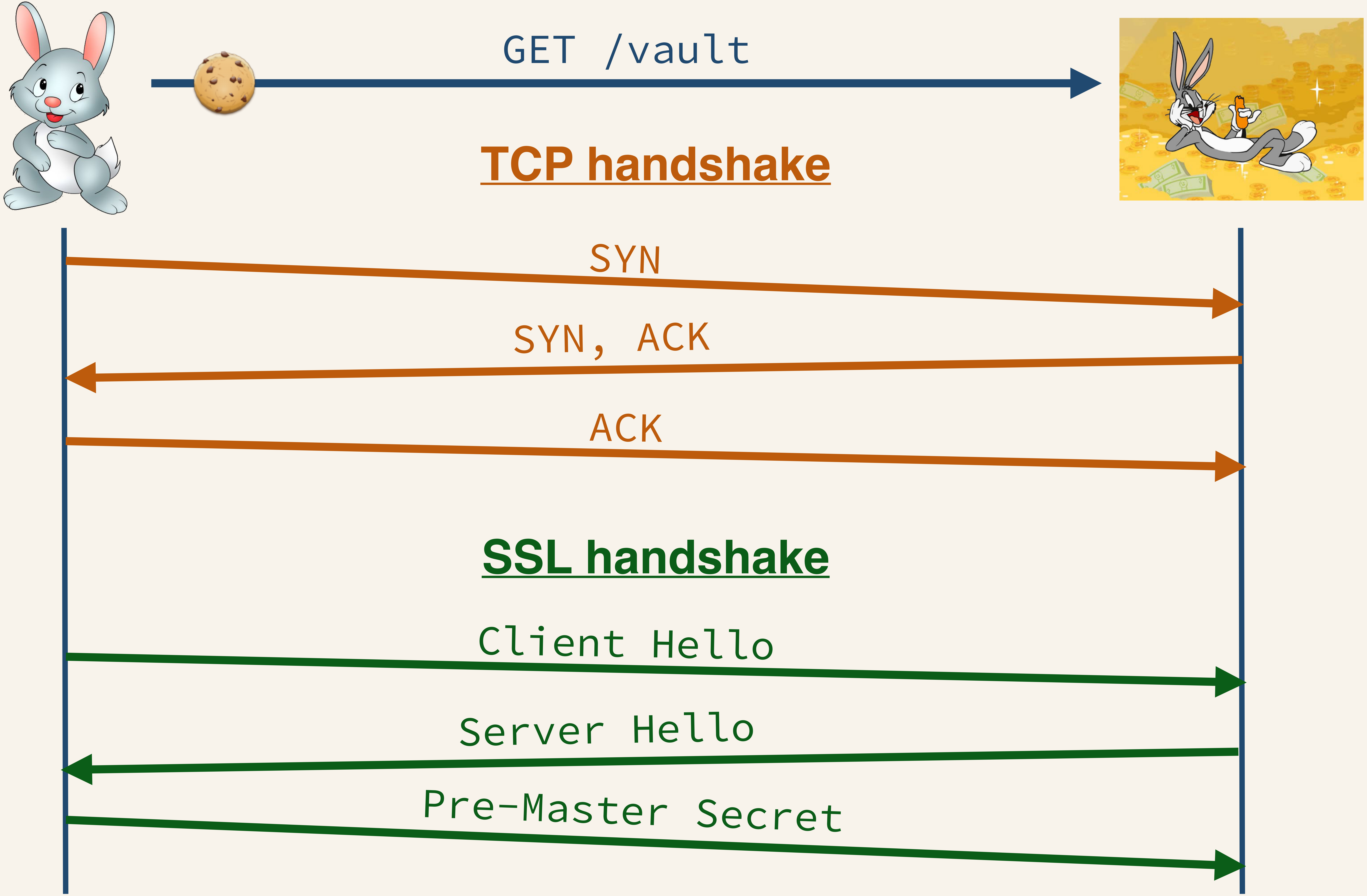
→ 50 bytes

Compression-based Attacks

- Compression and Information Leakage of Plaintext [FSE'02]
  - Chosen plaintext + compression = plaintext leakage
- Phonotactic Reconstruction of Encrypted VoIP Conversations [S&P'11]
  - Packet length + bitrate encoding
- CRIME [ekoparty'12]
  - Exploits SSL compression
- BREACH [Black Hat USA'13]
  - Exploits HTTP compression

Agenda

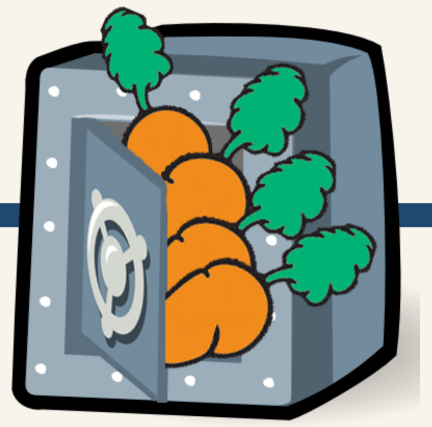
- Technical background
  - Same-Origin Policy
  - Compression-based attacks
  - **SSL/TLS & TCP**
- Nitty gritty HEIST details
- Demo
- Countermeasures

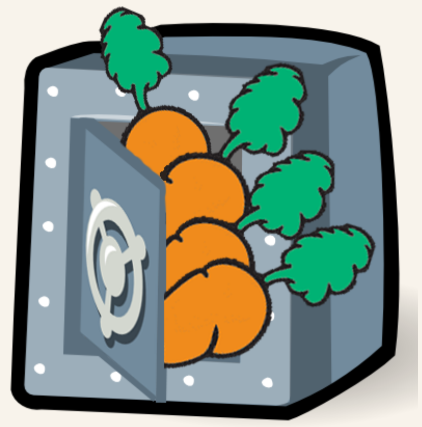

GET /vault
TCP handshake
SYN
SYN, ACK
ACK
SSL handshake
Client Hello
Server Hello
Pre-Master Secret

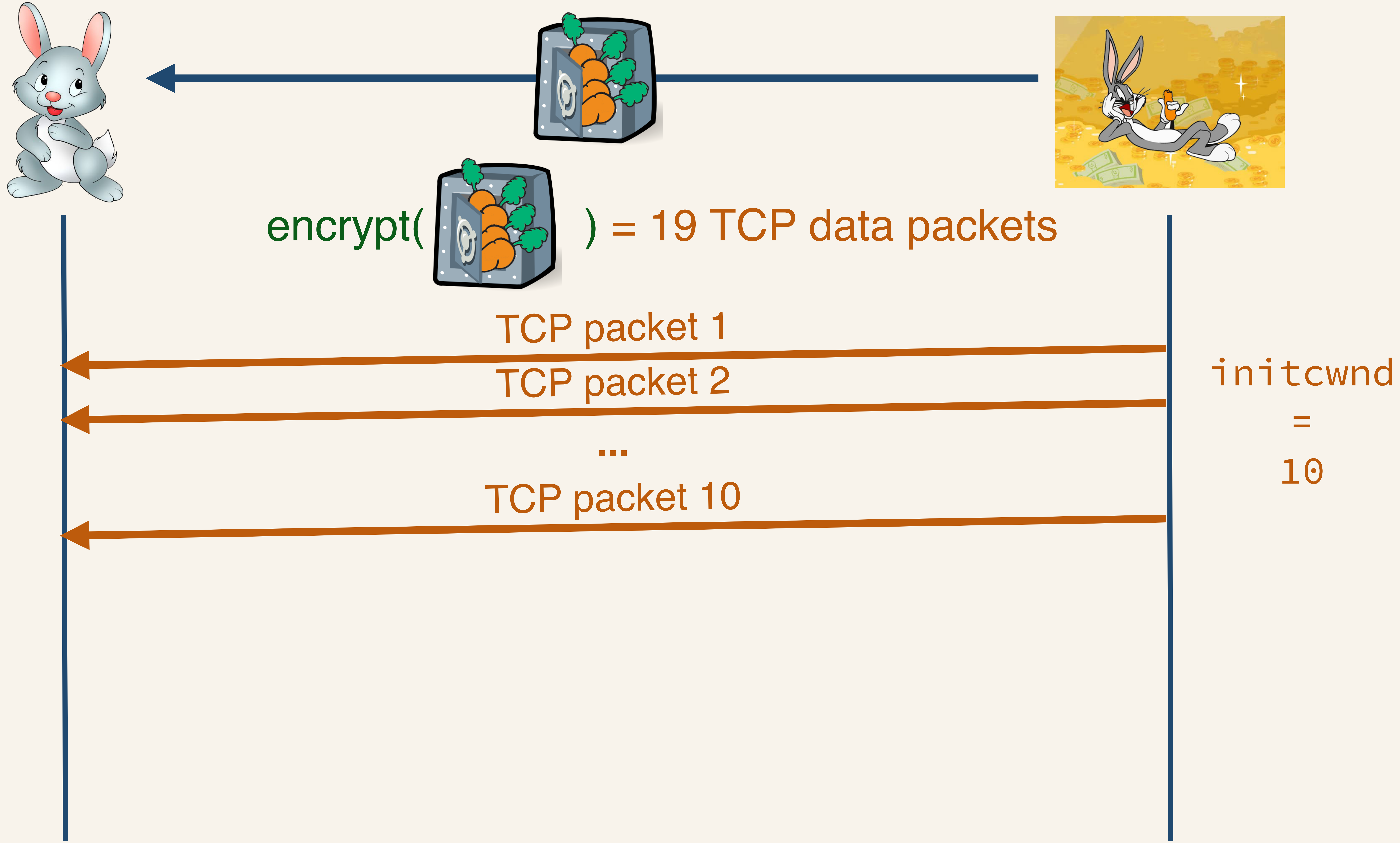
GET /vault

encrypt(
GET /vault HTTP/1.1
Cookie: user=mr.sniffles
Host: bunnehbank.com
....
)

1 TCP data packet



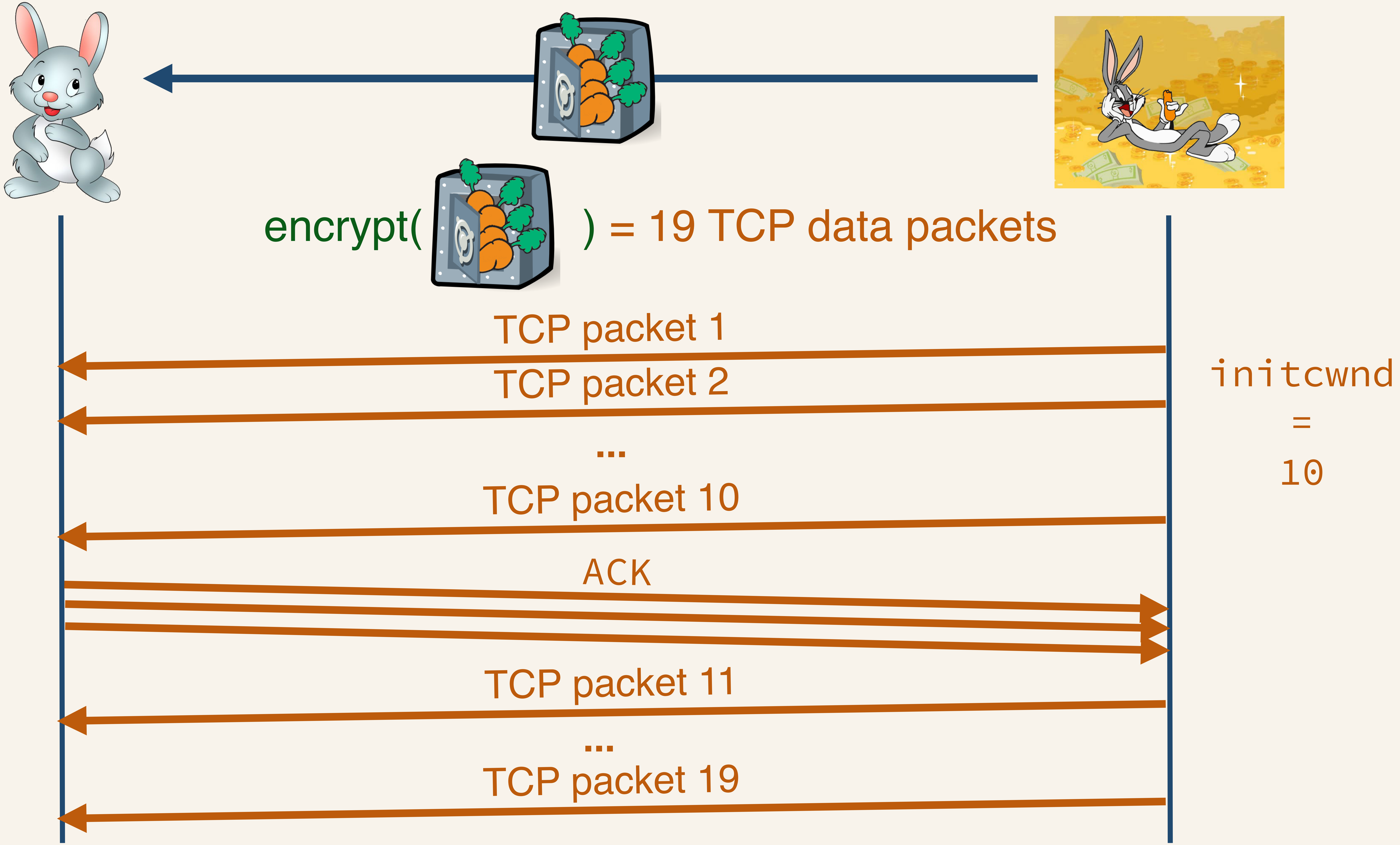
encrypt() = 19 TCP data packets



encrypt( ) = 19 TCP data packets
TCP packet 1
TCP packet 2
...
TCP packet 10
initcwnd
=
10

TCP Slow-start

- Not all TCP packets are sent at once
- TCP packets are sent in congestion windows
  - Congestion windows determine the amount of TCP packets that can be sent
  - Starts with the initial congestion window, `initcwnd`, typically set to 10
- When the packets of the first congestion window are `ACK`'d, the next congestion window is sent
  - Size of the next congestion window is doubled


encrypt( ) = 19 TCP data packets
TCP packet 1
TCP packet 2
...
TCP packet 10
ACK
TCP packet 11
...
TCP packet 19
initcwnd
=
10

HEIST

- A set of techniques that allow attacker to determine the exact size of a network response
- ... **purely in the browser**
- Leverages browser side-channels
- Can be used to perform compression-based attacks, such as CRIME and BREACH, in the browser

Browser Side-channels

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

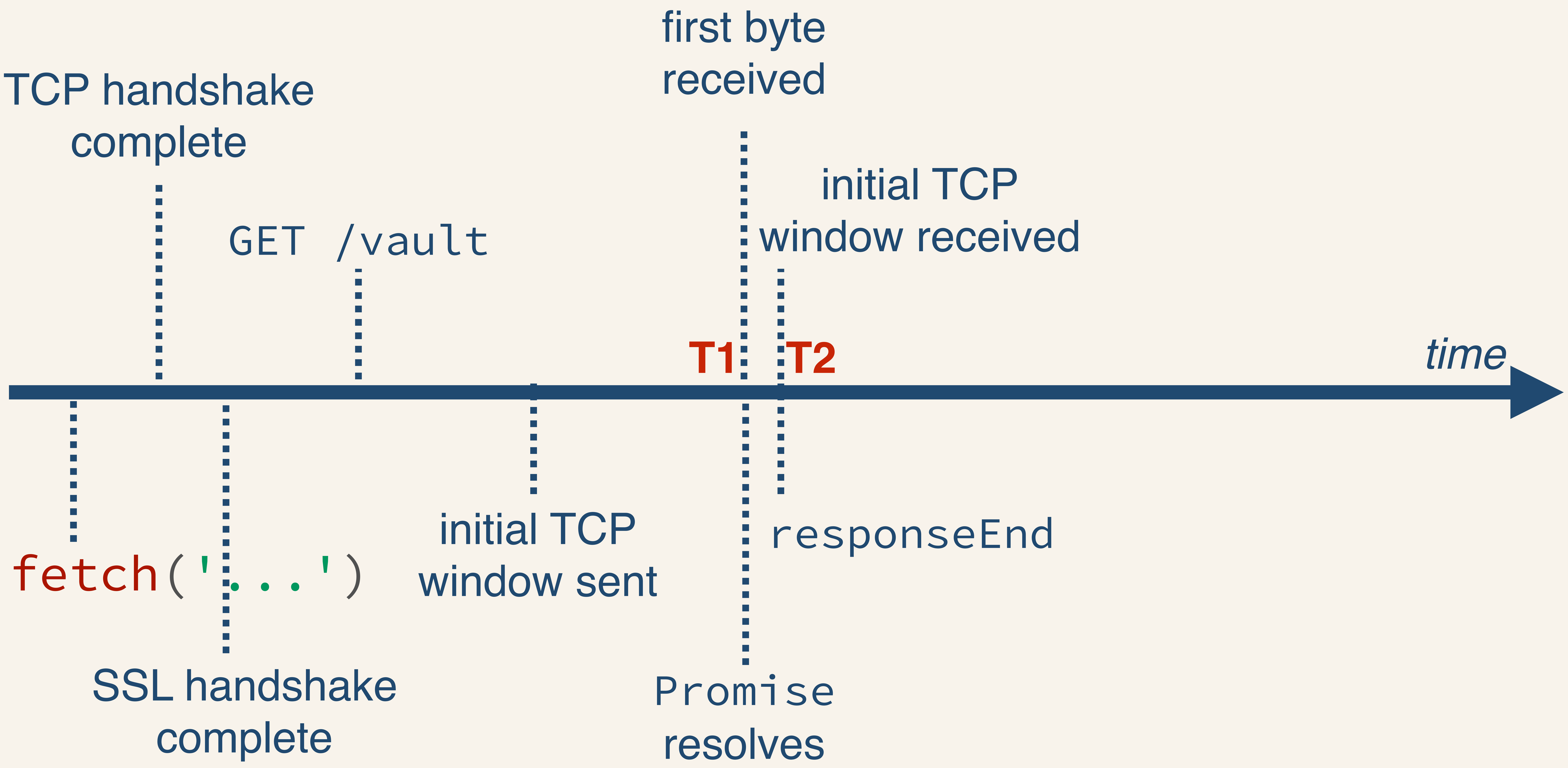
- Send authenticated request to `/vault` resource
- Returns a `Promise`, which resolves as soon as browser receives the first byte of the response

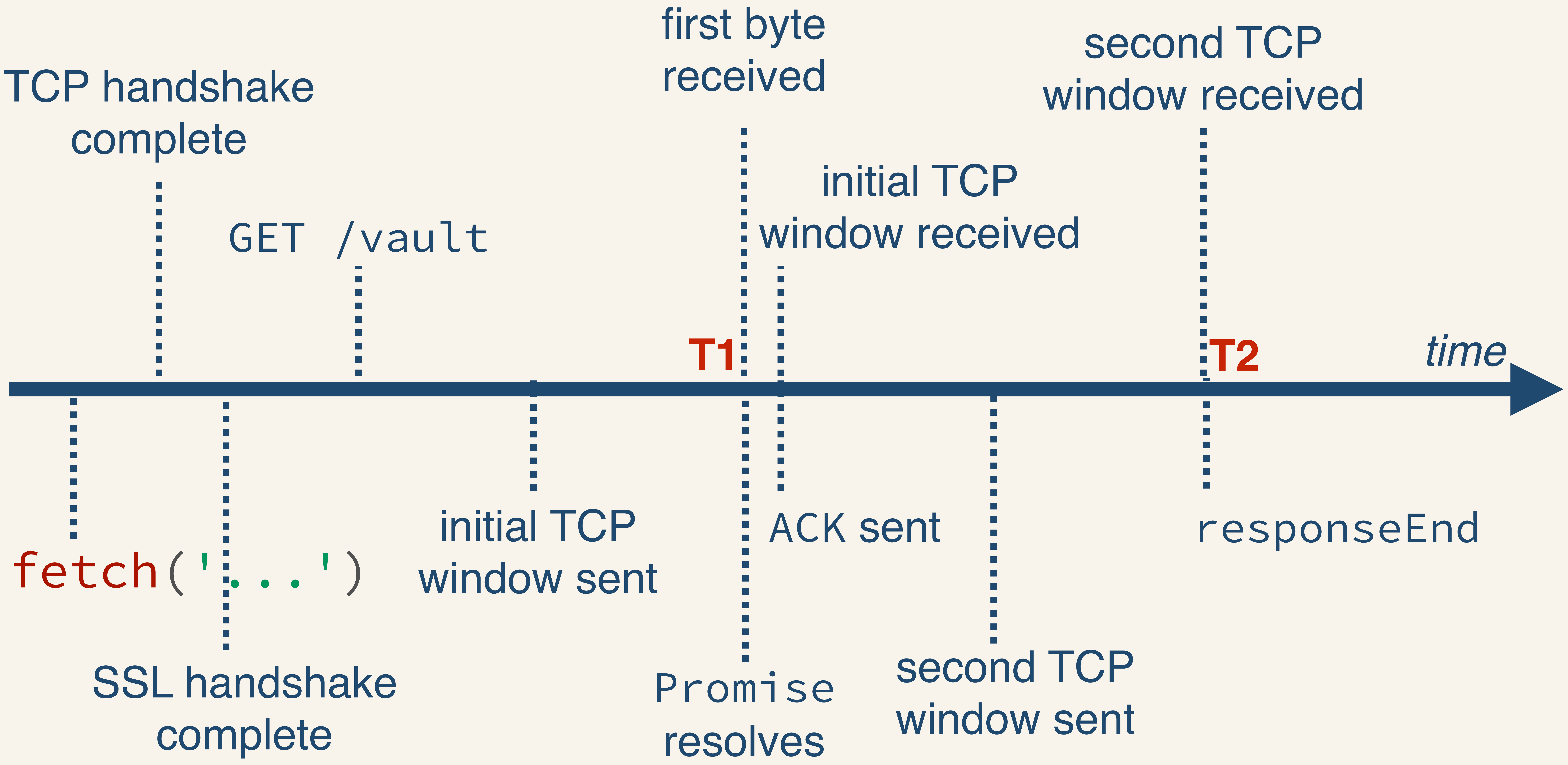
```
performance.getEntries()[-1].responseEnd
```

- Returns time when response was completely downloaded

HEIST

- Step 1: find out if response fits in a single TCP window

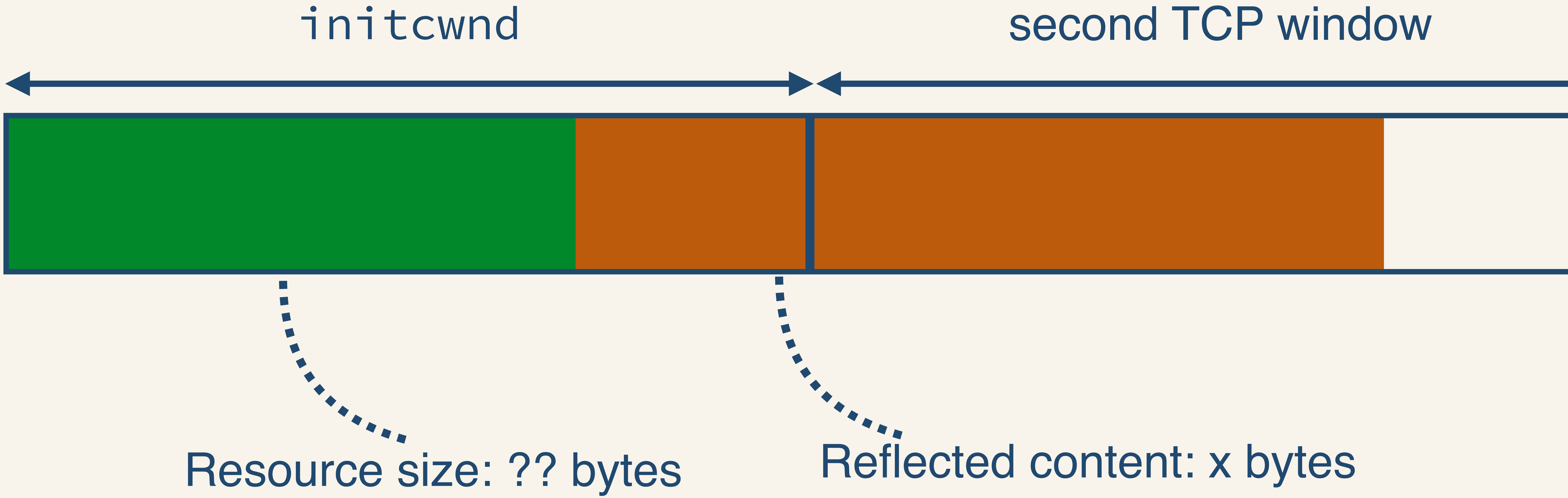

first byte
received
TCP handshake
complete
initial TCP
window received
GET /vault
T1
T2
time
fetch('...')
initial TCP
window sent
responseEnd
SSL handshake
complete
Promise
resolves


first byte received
second TCP window received
TCP handshake complete
initial TCP window received
GET /vault
T1
T2
time
fetch('...')
initial TCP window sent
ACK sent
responseEnd
SSL handshake complete
Promise resolves
second TCP window sent

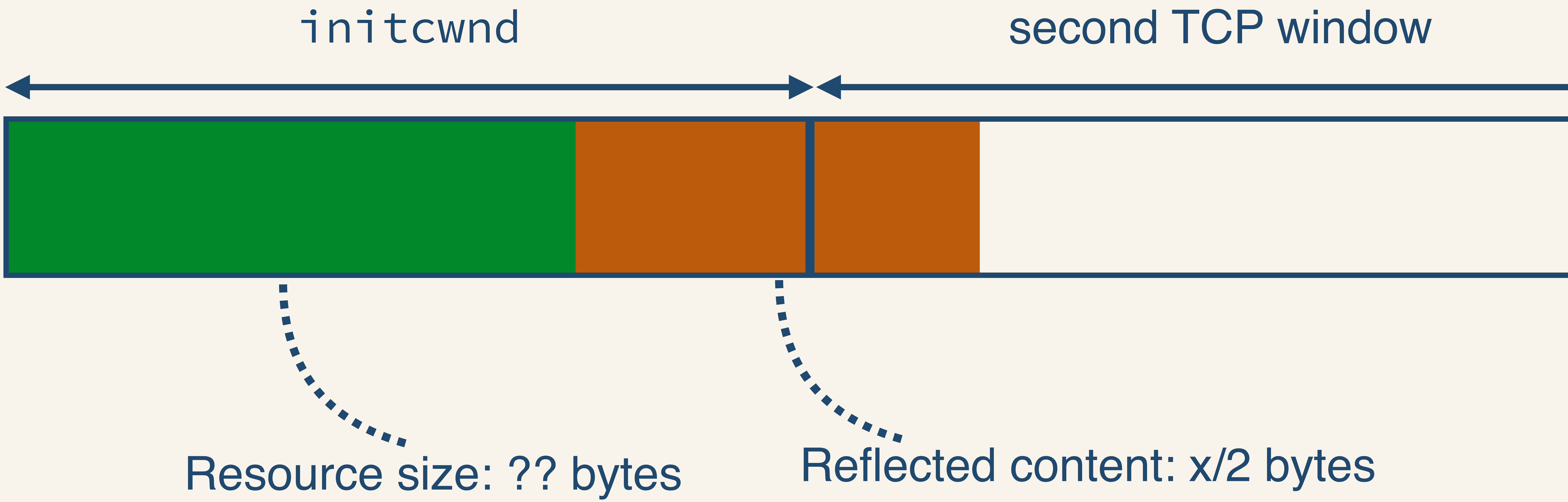
HEIST

- Step 1: find out if response fits in a single TCP window
- Step 2: discover exact response size

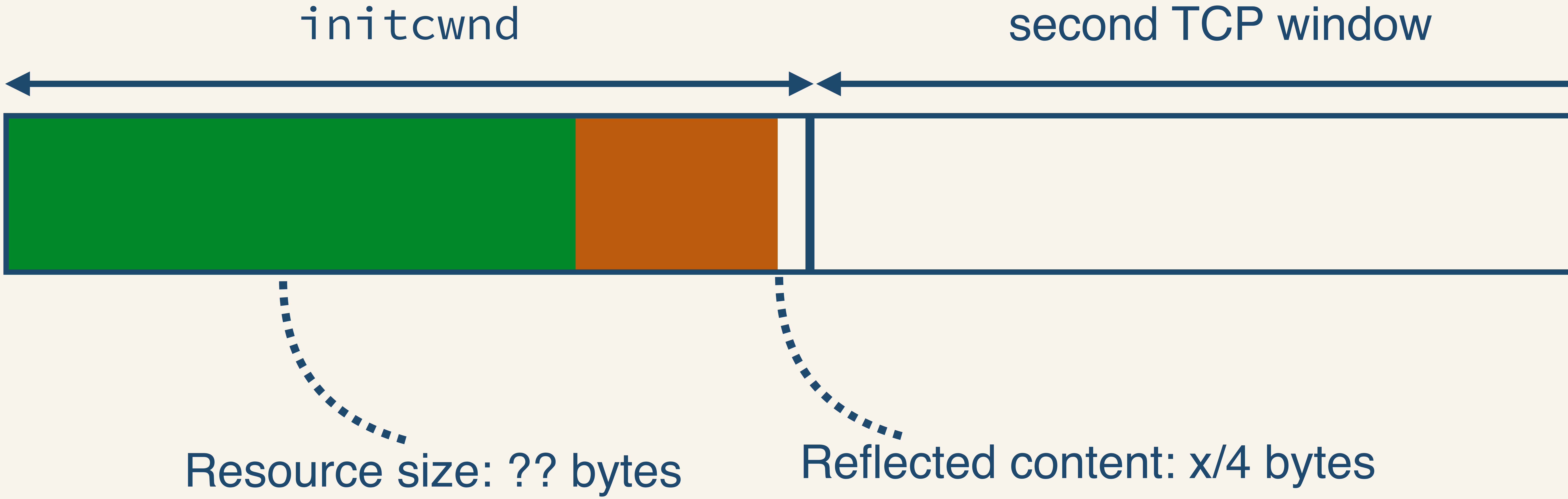
Discover Exact Response Size



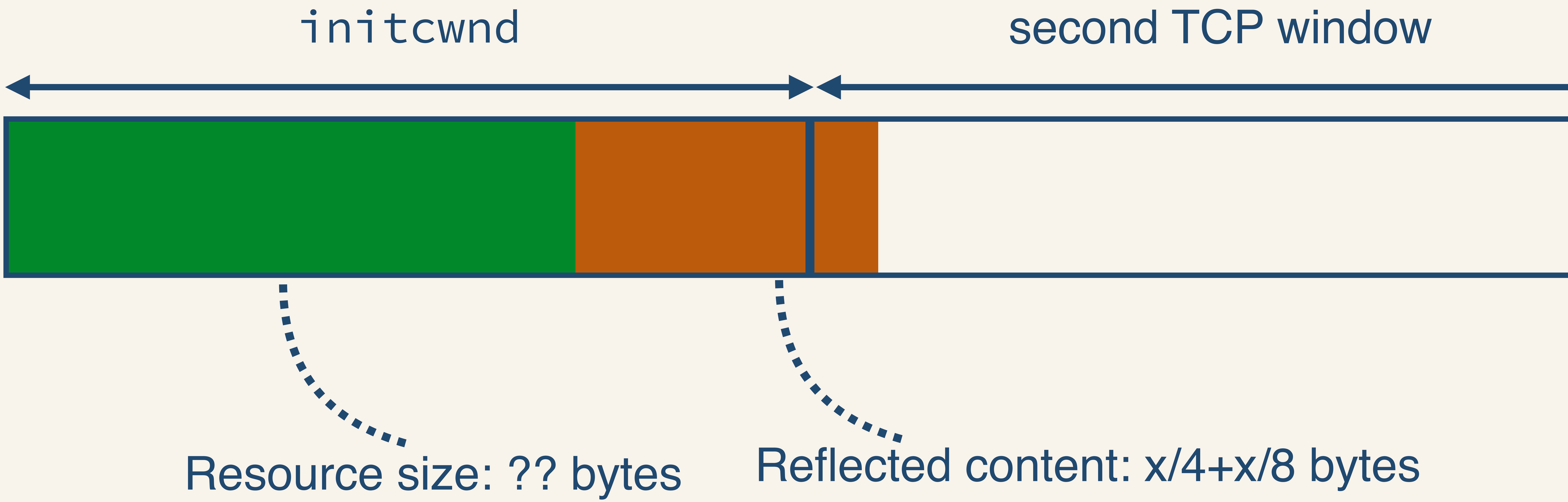
Discover Exact Response Size



Discover Exact Response Size



Discover Exact Response Size

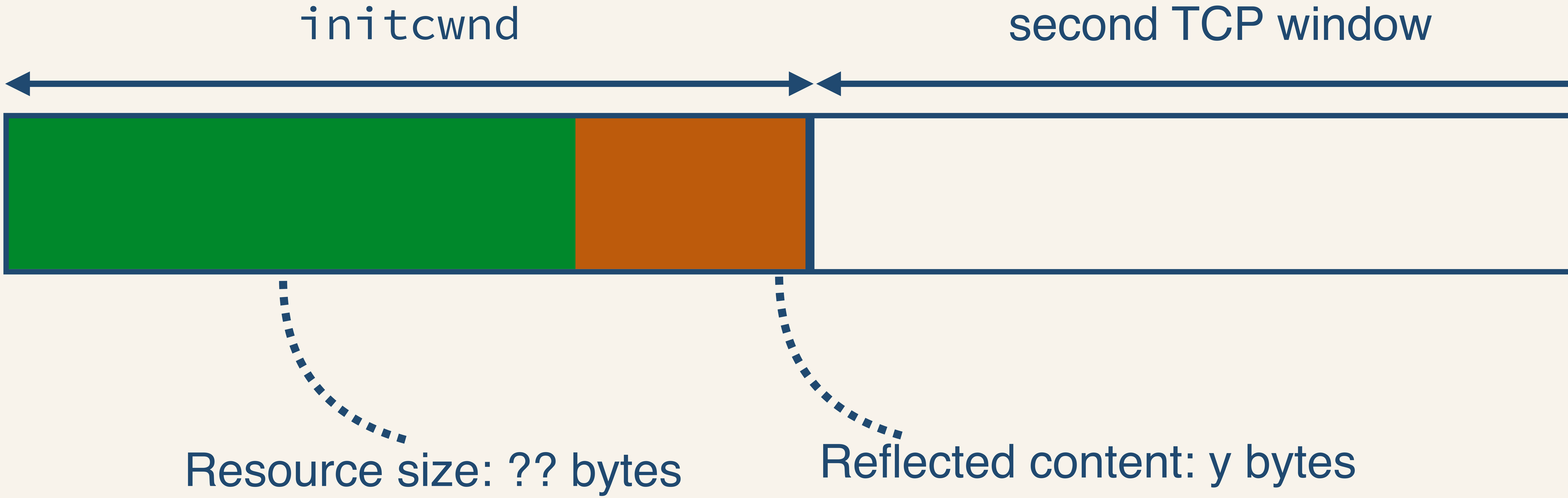


After *log(n)* checks, we find:

y bytes of reflected content = 1 TCP window

y+1 bytes of reflected content = 2 TCP windows

→ resource size = `initcwnd` - y bytes

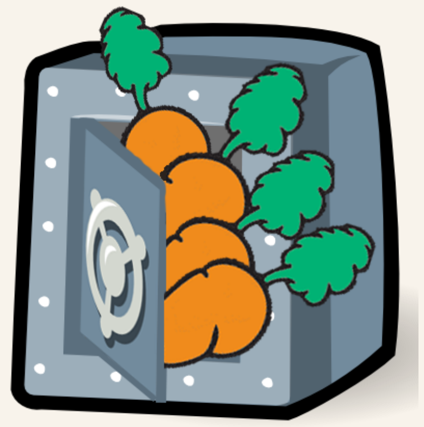


HEIST

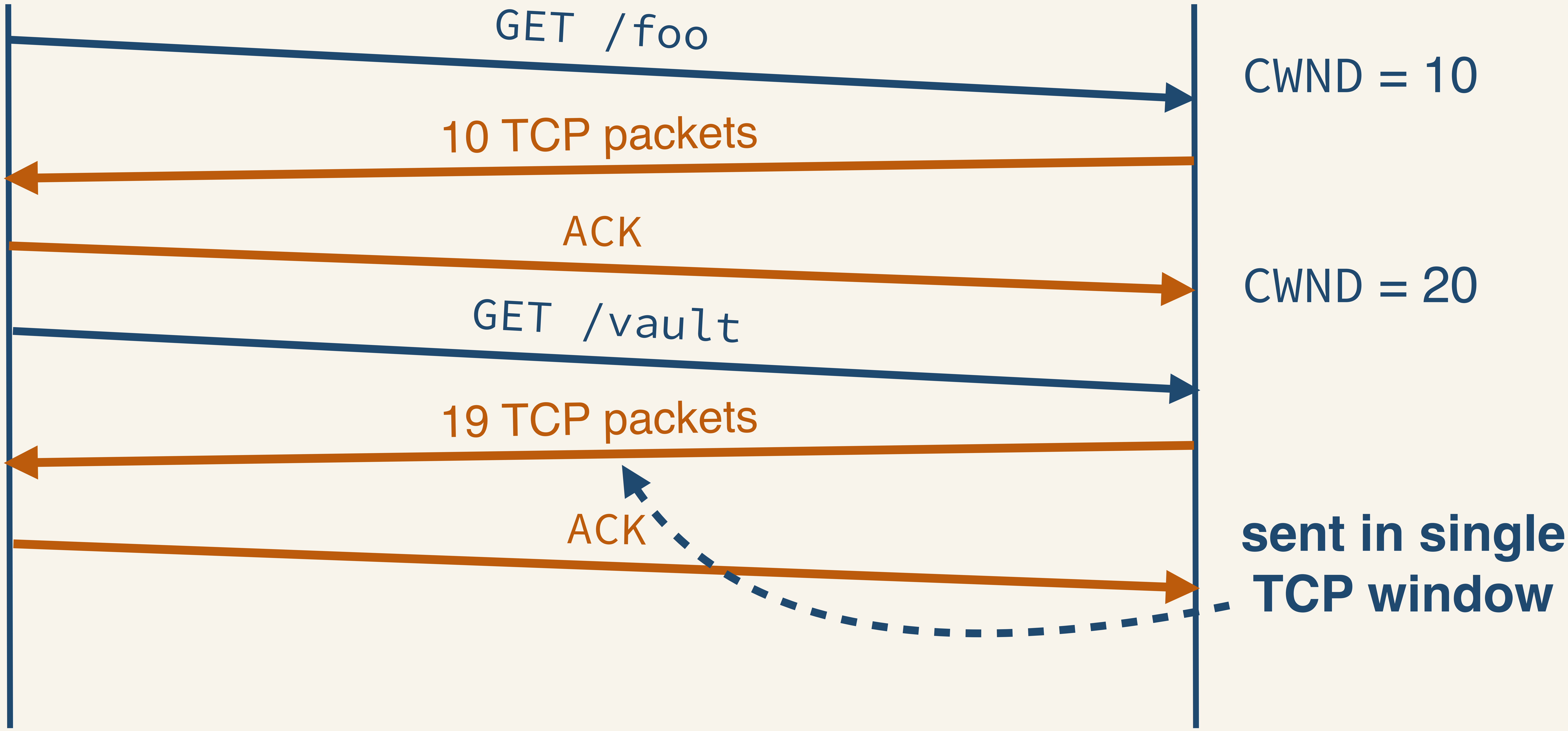
- Step 1: find out if response fits in a single TCP window
- Step 2: discover exact response size
- Step 3: do the same for large responses (> `initcwnd`)

Determine size of large responses

- `initcwnd` is typically set to 10 TCP packets
  - ~14kB
- TCP windows grow as packets are acknowledged
  - Second TCP window is 20 TCP packets, third is 40, ...
- We can arbitrarily increase window size
  - Send request to resource of known size
  - After response is in, send request to target resource, repeat step 2



= 19 TCP data packets



HEIST

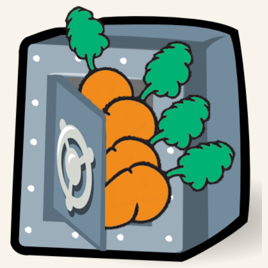
- Step 1: find out if response fits in a single TCP window
- Step 2: discover exact response size
- Step 3: do the same for large responses (> `initcwnd`)
- Step 4: if available, leverage HTTP/2

Leveraging HTTP/2

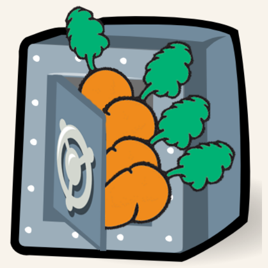
- HTTP/2 is the new HTTP version
  - Preserves the semantics of HTTP
- Main changes are on the network level
  - Only a single TCP connection is used for parallel requests
  - Headers are compressed using HPACK
    - Client and server build same lookup table
    - Header is now just a reference to an entry in the table
    - Mitigates CRIME

Leveraging HTTP/2

- HTTP/2 allows us to determine exact response size *without* needing reflected content in the same response
  - Only a single TCP connection is used for parallel requests
- Use (reflected) content in other responses on the same server
  - Note that BREACH still requires reflective content in the same resource
  - Response size can still be used to leak sensitive data (see examples later)

 = 6 TCP packets

/reflect = 2 TCP packets + reflected

 = 6 TCP packets

`/reflect` = 2 TCP packets + reflected

HEIST

- Step 1: find out if response fits in a single TCP window
- Step 2: discover exact response size
- Step 3: do the same for large responses (> `initcwnd`)
- Step 4: if available, leverage HTTP/2
- Step 5: exploit & profit


40 50 60 70

Exploit & profit

- Use HEIST to exploit BREACH/CRIME
  - Extract CSRF tokens, private message content, ...
  - Only 2 requirements: gzip/SSL compression + reflected content
- Obtain sensitive content from web services
  - Response size is related to user (victim) state

DEMO

Other targets

- Compression-based attacks
  - gzip compression is used by virtually every website
- Size-exposing attacks
  - Uncover victim's demographics from popular social networks
  - Reveal victim's health conditions from online health websites
  - Disclose victim's financial information
- Hard to find sites that are not vulnerable

Countermeasures

- Browser layer
  - Prevent side-channel leak *(infeasible)*
  - Disable third-party cookies *(complete)*
- HTTP layer
  - Block illicit requests *(inadequate)*
  - Disable compression *(incomplete)*
- Network layer
  - Randomize TCP congestion window *(inadequate)*
  - Apply random padding *(inadequate)*

Conclusion

- Collection of techniques to discover network response size **in the browser**, for all authenticated cross-origin resources
- Exploits the subtle interplay of browser and network layer
- HTTP/2 makes exploitation easier
- Allows for compression-based and size-exposing attacks
- Many countermeasures, few that actually work

HEIST

Questions?

Mathy Vanhoef
@vanhoefm
mathy.vanhoef@cs.kuleuven.be

Tom Van Goethem
@tomvangoethem
tom.vangoethem@cs.kuleuven.be