CS3235 Week 9 Tutorial

Authentication, SOP, CSRF

About Your TA: Jason Zhijingcheng Yu

Just call me Jason

PhD student with Prof Prateek Saxena

Came after undergrad in China

Second half of the semester?

Building blocks you have learned so far: cryptography, network protocols

Coming up: “higher-level” security

- Higher-level components
- Higher-level notion of security

More hands-on ~~and fun~~!

This week:

- Authentication: prove who you are and establish trust
- Authorisation: control what you can do

Example: SSH

```
ssh server.domain -l alice
ECDSA key fingerprint is
SHA256:9atnCFKuG4aOy4oBJO8rWGMZuqHAQeYlLOAgylaUShE.
Are you sure you want to continue connecting
(yes/no/[fingerprint])?
Warning: Permanently added … to the list of known hosts.
```

Trust on first use (TOFU)

Alice's laptop

server.domain

```
/home/alice/.ssh/known_hosts
```

```
/etc/ssh/ssh_host_rsa_key
```

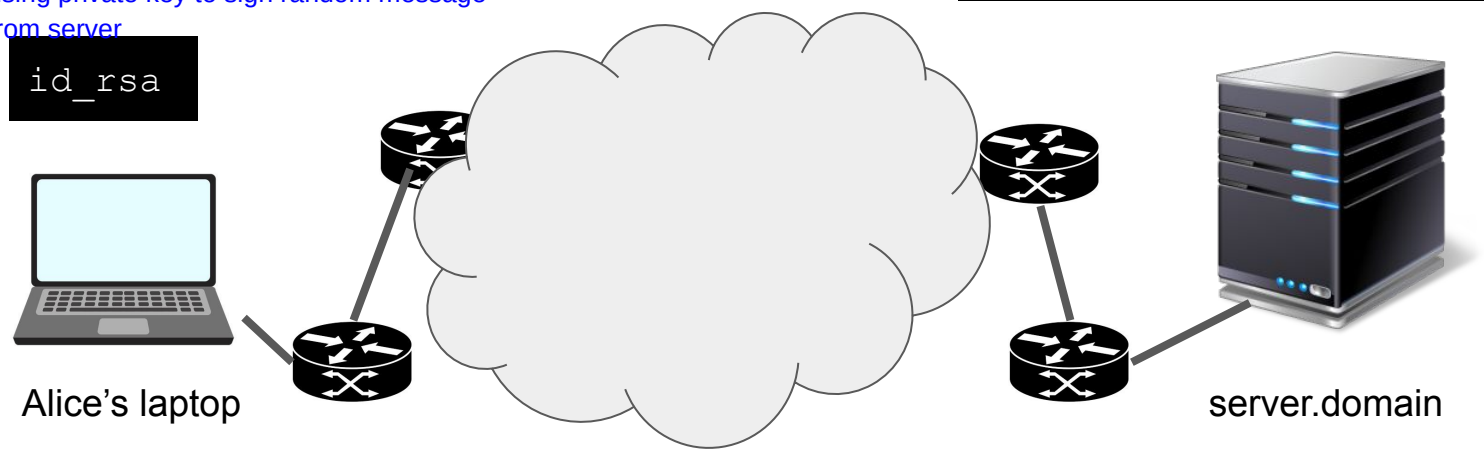
Pubkey authentication

alice pre computes public key for server to store

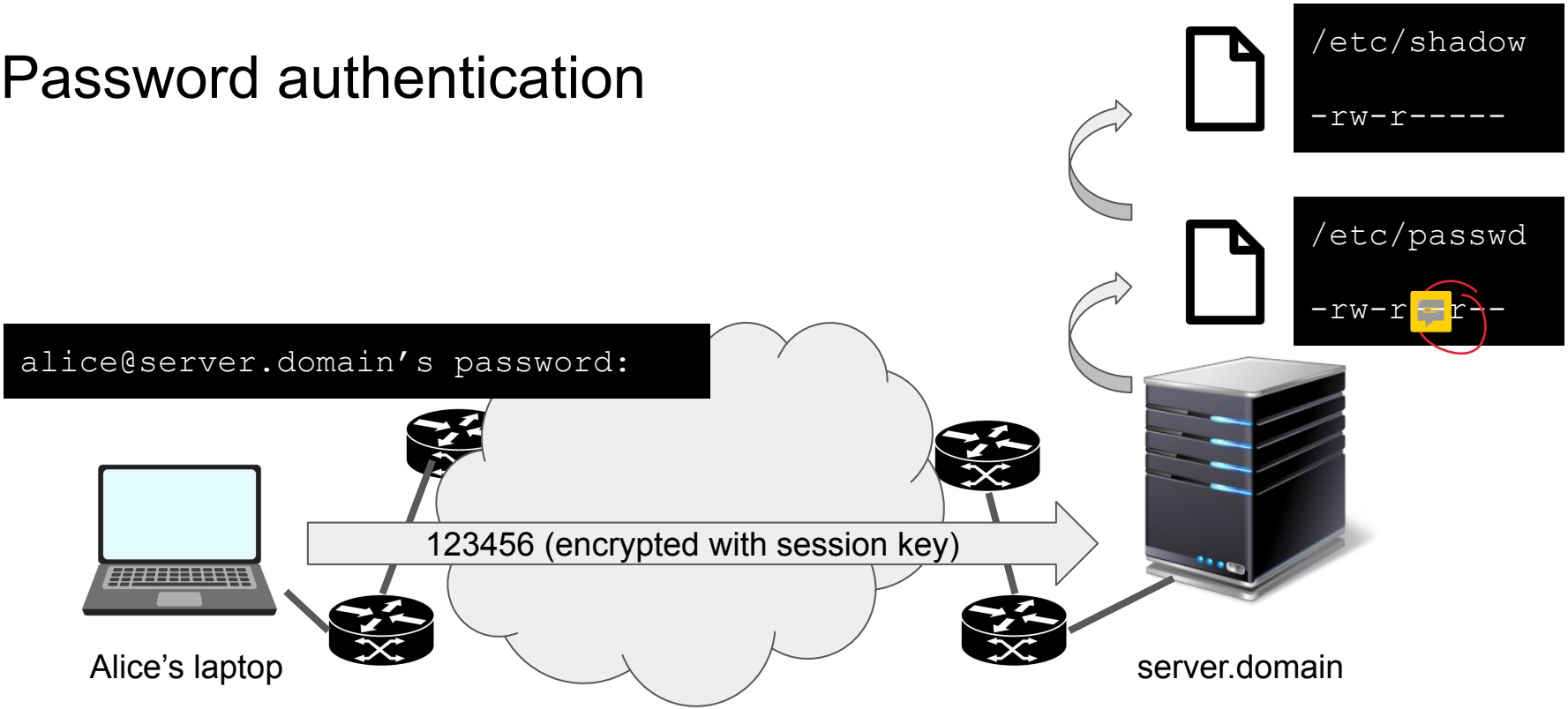
```
/home/alice/.ssh/authorized_keys
```

using private key to sign random message from server

```
id_rsa
```



Password authentication



Salted Hash

H(r || pwd) instead of H(pwd)

Bob wants to be authenticated as Alice on server1

Alice is suffering from password fatigue and uses the same password on server2.

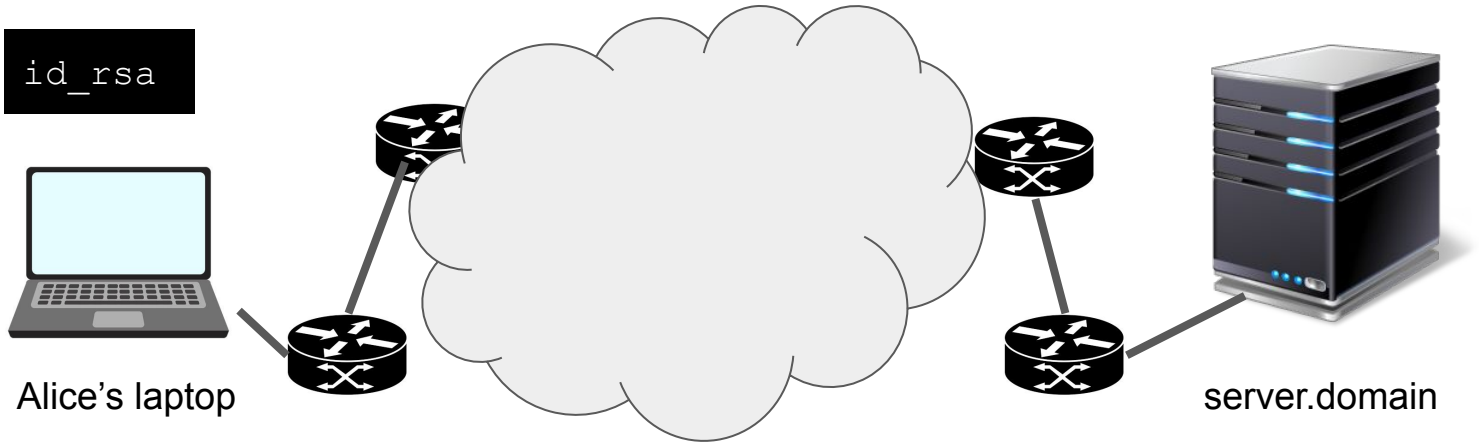
Compare Bob's costs in the following scenarios

- Bob knows nothing. Salted vs unsalted?
- Bob knows the password hash. Salted vs unsalted?
- Bob knows the password hash (unsalted) vs Bob knows both the password hash and salt (salted)?
- Bob knows the password hash on server2 (unsalted) vs Bob knows the password hash on server2 and salt on server1 (salted)?

Authorisation x authentication

```
/home/alice/.ssh/authorized_keys
```

```
id_rsa
```

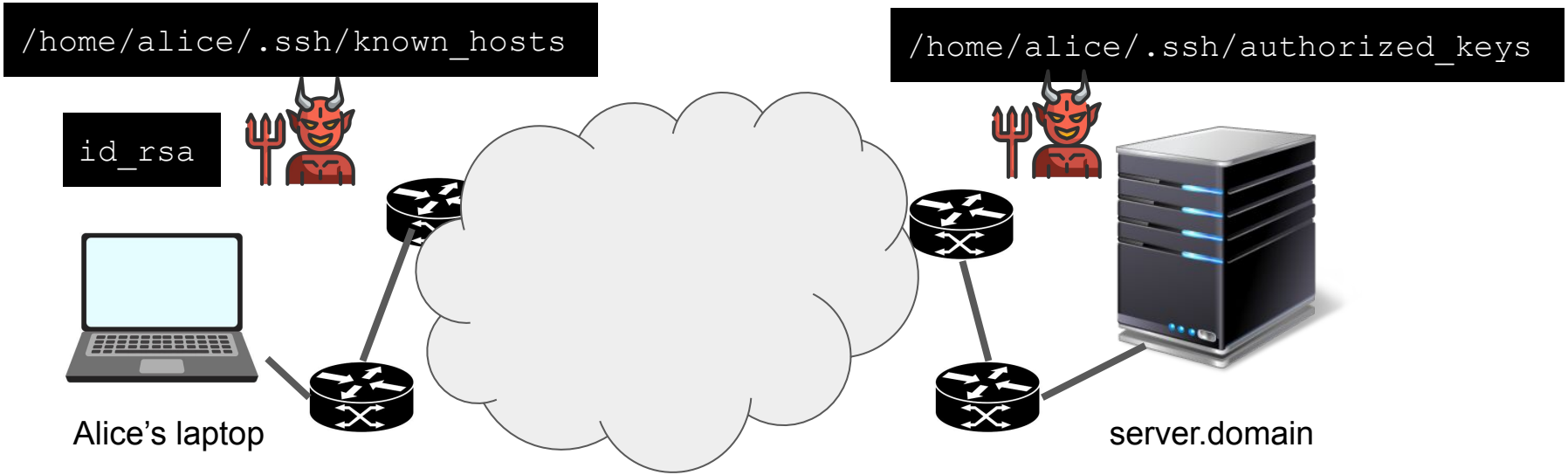


Alice’s laptop

server.domain

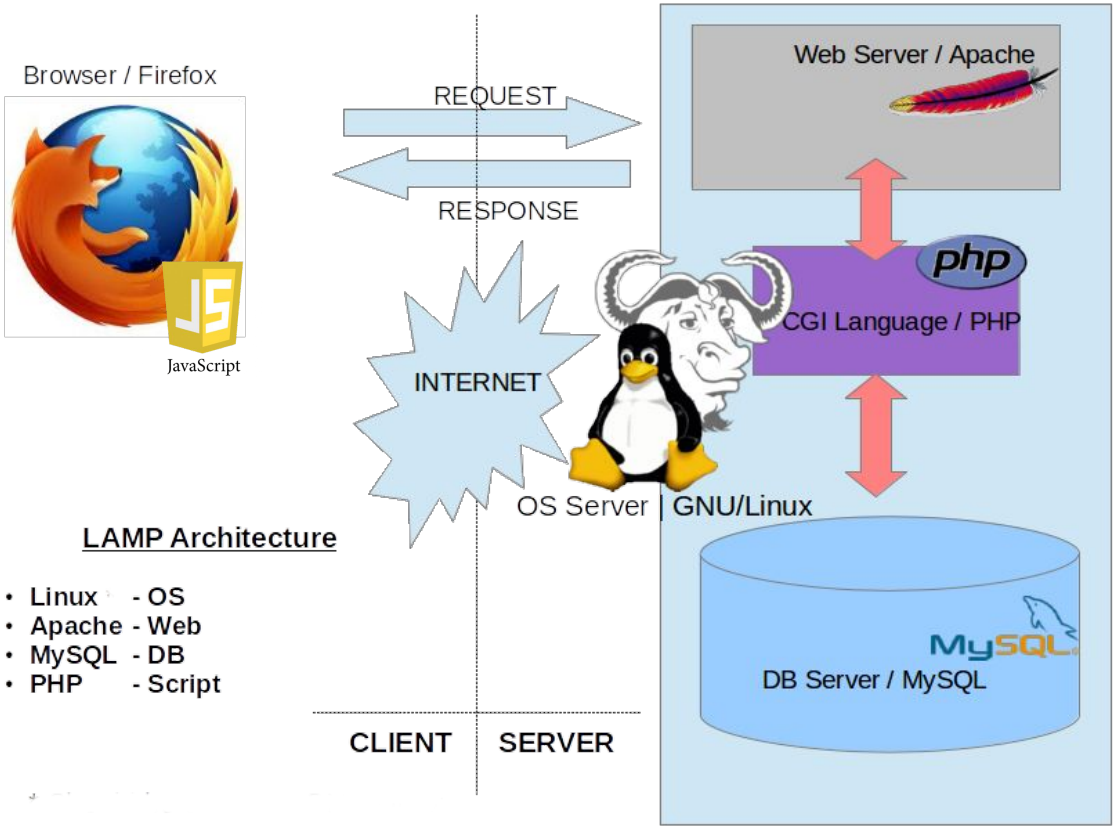
```
alice@server.domain$ chmod o+rwx
/home/alice
```

Authorisation x authentication



Web Primer

Lamp:



Web Browsers Developer tools

- (Ctrl+Shift+I or F12 on Chrome/Firefox)
- Console:
  - Browser warnings/errors...
  - Javascript
- Network:
  - Requests
  - Responses
- Storage:
  - Cookies

HTTP

- Headers
- Request methods
  - **GET**
  - **POST**
  - PUT
  - DELETE
  - OPTIONS
- Status code
  - 2xx: good
  - 3xx: further actions
  - 4xx: client errors
  - 5xx: server errors

Cookies

Set:

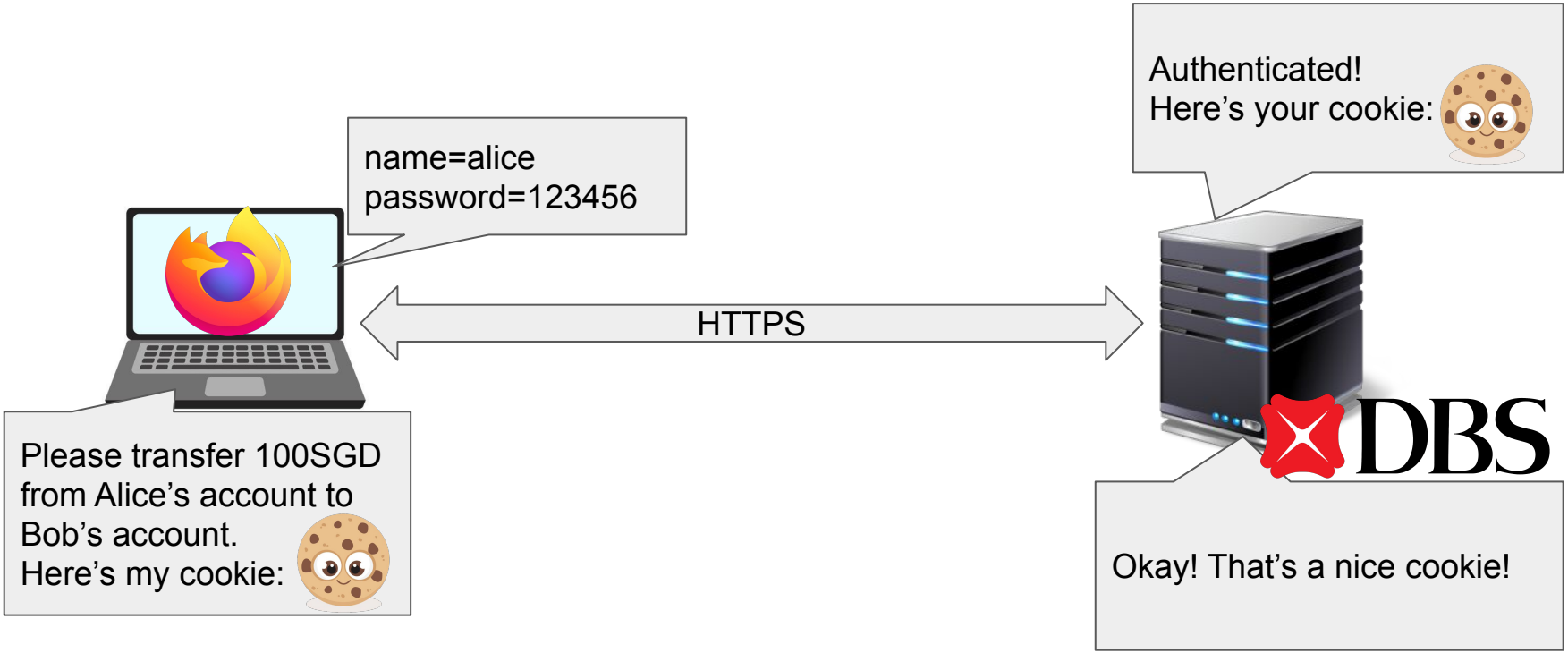
- Response header: Set-Cookie
- PHP:
  - setcookie(...)
- Javascript:
  - document.cookie = “...”

Get:

- Request header: Cookie

Normally, every request sends all cookies related to that domain (and path)

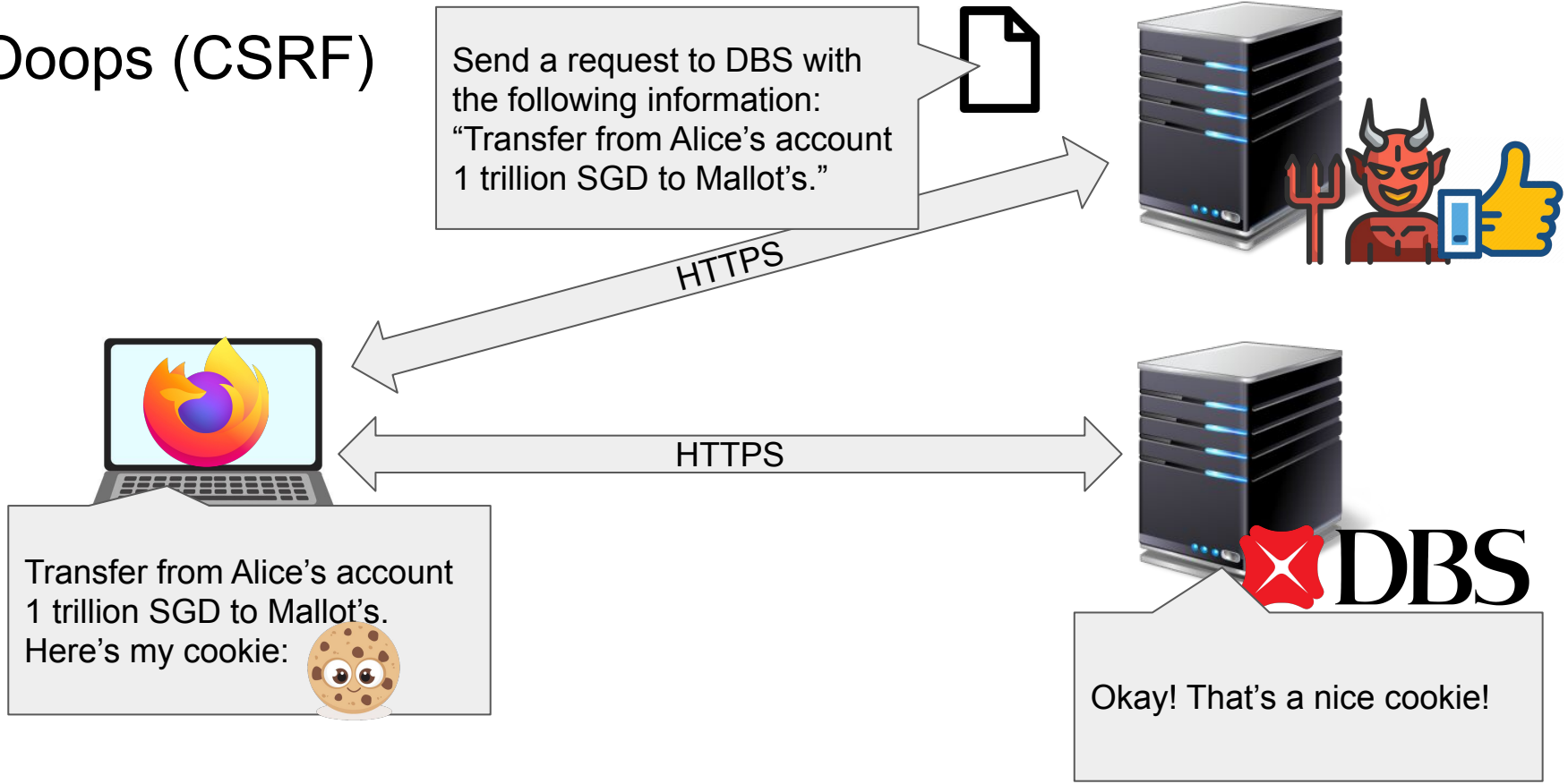
Authentication



Your browser talks to a lot of websites

- Links
- Subresources (images, css, javascript)
- iframes that load parts of other webpages
- XMLHttpRequest (XHR)
- Forms

Ooops (CSRF)


Send a request to DBS with the following information: “Transfer from Alice’s account 1 trillion SGD to Mallot’s.”
HTTPS
HTTPS
Transfer from Alice’s account 1 trillion SGD to Mallot’s. Here’s my cookie:
DBS
Okay! That’s a nice cookie!

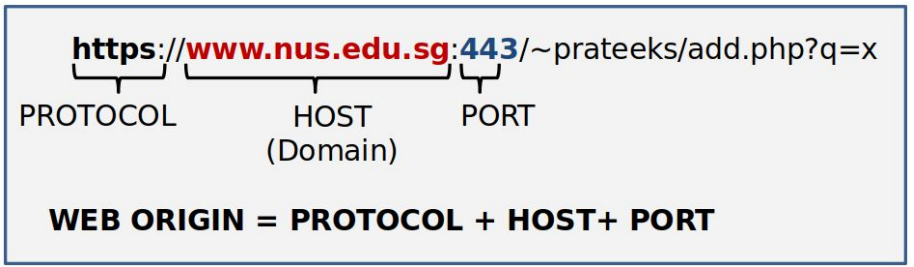
Defence I: CSRF Tokens

- Embed a secret token in a page (e.g., as hidden form field)
- Token sent along with requests
- Unlike cookies, such tokens are
  - Associated with a webpage instead of a website
  - Not automatically sent by the browser on every request to a website
- “Hacky” way to make sure the request is from the same

Assumption: other websites cannot obtain the token

Web Origin: What does a “website” even mean?

- Protocol:
  - Also known as Scheme (http, https, etc.)
- Host
  - Also known as Domain (google.com, etc.)
- Port
  - 80, 443, 8080, etc.



*

* Web Security: User Authentication & Authorization, Lecture slides, lecture 8 ~Prateek Saxena

Same-Origin Policy (SOP)

- Isolate websites in a browser
  - Security:
    - One website sends request to another website
    - Stealing data from other web apps (e.g., cookies)
  - Privacy:
    - Third-party tracking cookies
- Enforces
  - When requests can be made
  - How cookies can be accessed
  - …

Your browser's **default behaviour** is to restrict some cross-site resource usage!

Server can request for different behaviours!

- Through response headers
- iframe: X-Frame-Options
  - DENY
  - SAMEORIGIN
- XHR (CORS, Cross-Origin Resource Sharing):
  - https://api.spotify.com/v1

access-control-allow-credentials	true
access-control-allow-headers	Accept, App-Platform, Authorization, Content-Type, Origin, Retry-After, Spotify-App-Version, X-Cloud-Trace-Context, client-token, content-access-token
access-control-allow-methods	GET, POST, OPTIONS, PUT, DELETE, PATCH
access-control-allow-origin	*
access-control-max-age	604800

Defence II: Samesite Cookies

Idea: don’t always send cookies

- Samesite=None
- Samesite=Lax
- Samesite=Strict

Defence III: Referer (sic)

Correct spelling: referrer

Include the origin of a request in the request header

Web API?

- CORS opens it to cross-site XHR
  - Any website can send requests to it!
- CSRF?
  - No cookie, no CSRF!
  - RESTful APIs are stateless

How to defeat even those defences?

Idea: trigger a malicious request from the same origin

since CSRF protection is against malicious requests from different origins

Supplementary Reading

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

https://owasp.org/www-community/attacks/csrf