

# Authentication and Session Management

Basic, Digest, Password Storage, Brute Force, MFA, Federation, Hijacking,  
Fixation, CSRF, JSONP, XSS, SOP, CORS

# Rough Overview

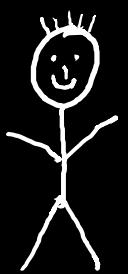
1. Introduction
2. Basic Principles and Resources
3. Architecture & Basic Web Procedure
4. >> Authentication and Session Management <<
5. Authorization
6. Server and Backend Attacks
7. Remaining Client Attacks
8. General Topics
9. Conclusions

Internet

DMZ

Application

Intranet



Log Server

Business Logic

Input Val.

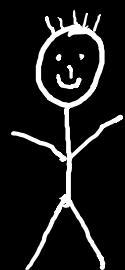
Auth Z

Output San.

Auth N

App Server

Web Server



Ext.  
Service

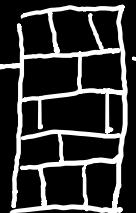


FW



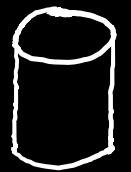
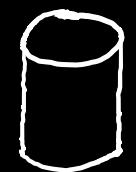
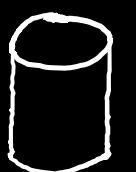
WAF

LB  
Proxy



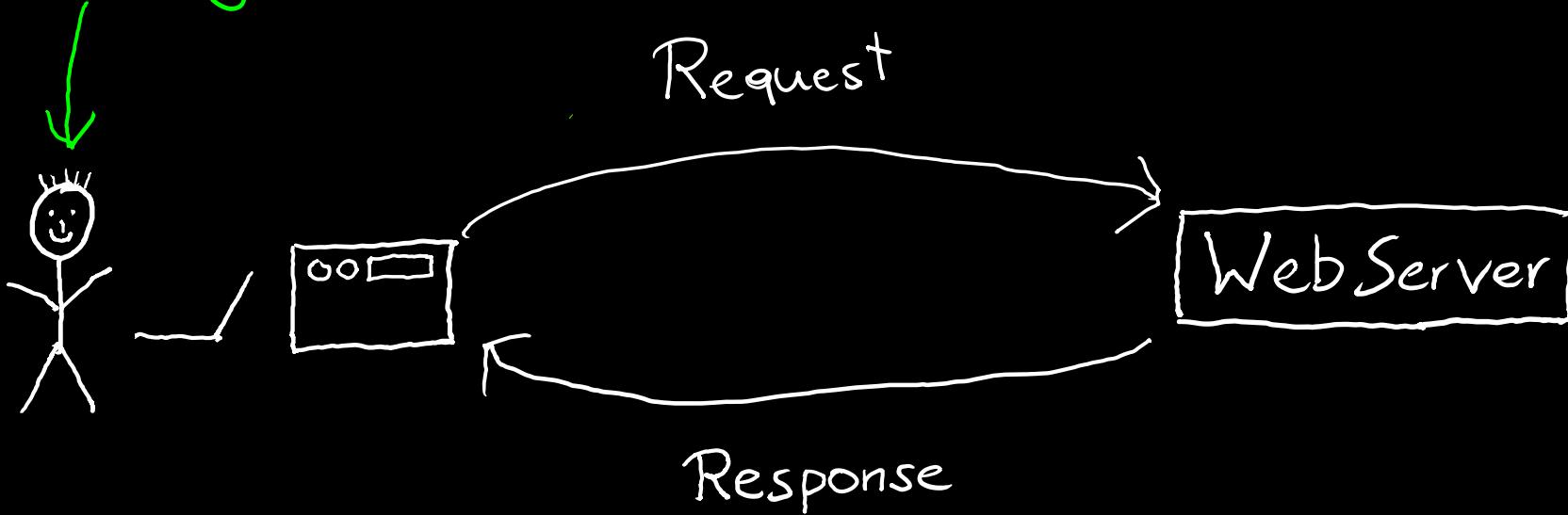
FW

Int.  
Service

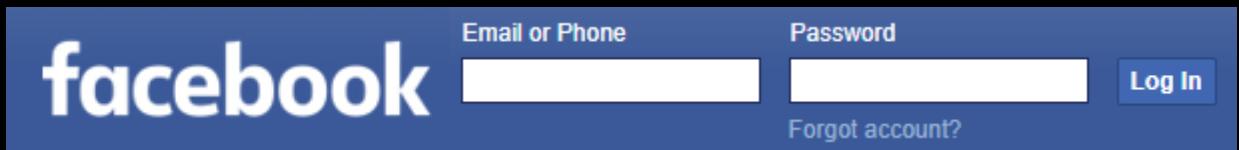


DBs LDAP File Share

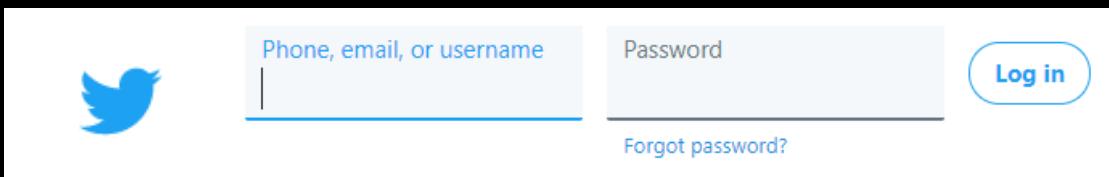
Who is this guy?



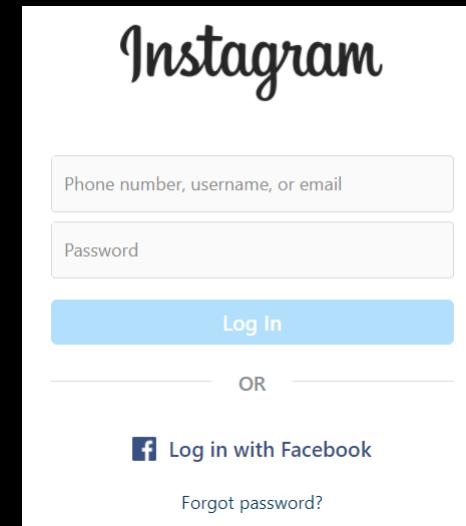
How do real  
webapps usually do  
authentication?



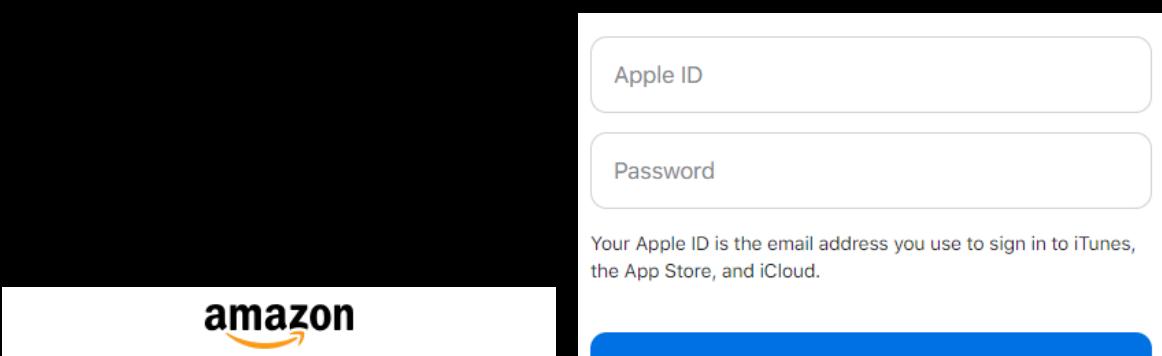
The Facebook login screen features a blue header bar with the word "facebook" in white. Below it, there are two input fields: "Email or Phone" and "Password", both with placeholder text. A "Log In" button is to the right. Below the fields is a "Forgot account?" link.



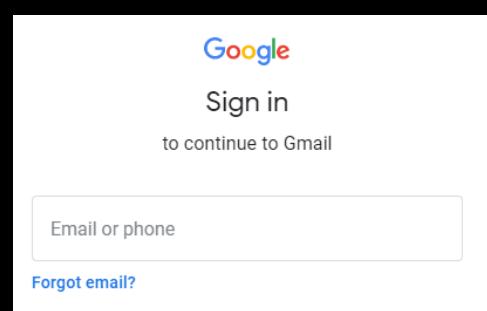
The Twitter login screen has a dark background. It includes a blue Twitter logo icon. There are two input fields: "Phone, email, or username" and "Password", followed by a "Log in" button. Below the fields is a "Forgot password?" link.



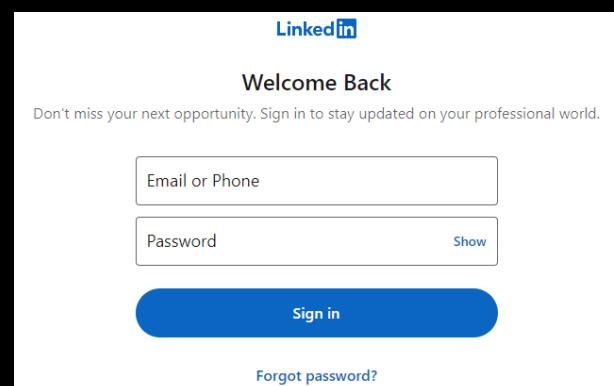
The Instagram login screen has a white background with a large "Instagram" logo at the top. It features two input fields: "Phone number, username, or email" and "Password", followed by a "Log In" button. Below the fields is a "Forgot password?" link. There is also a "Log in with Facebook" option with a Facebook icon.



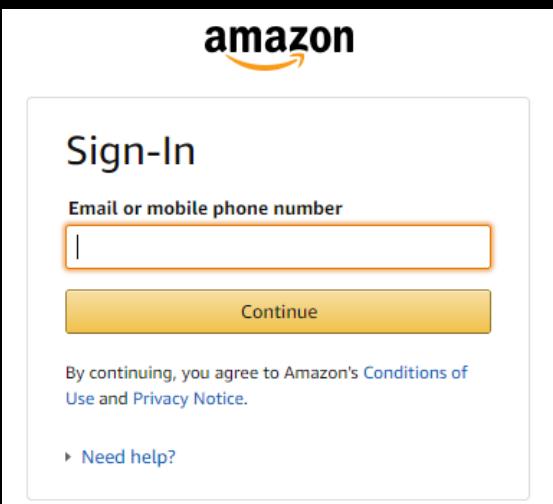
The Apple ID sign-in screen has a white background. It contains two input fields: "Apple ID" and "Password". Below the fields is a note: "Your Apple ID is the email address you use to sign in to iTunes, the App Store, and iCloud." At the bottom is a large blue "Sign In" button.



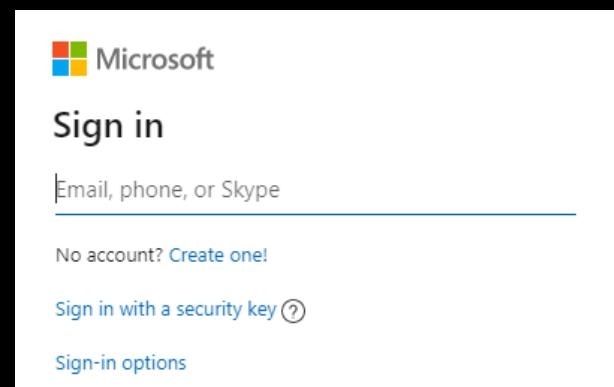
The Google sign-in screen has a white background. It features the Google logo and the word "Sign in" above a "to continue to Gmail" link. There is an "Email or phone" input field, a "Forgot email?" link, and a "Sign in" button.



The LinkedIn sign-in screen has a white background. It starts with the LinkedIn logo and the words "Welcome Back". Below that is a note: "Don't miss your next opportunity. Sign in to stay updated on your professional world." It has two input fields: "Email or Phone" and "Password", with a "Show" link next to the password field. At the bottom is a blue "Sign in" button and a "Forgot password?" link.



The Amazon sign-in screen has a white background. It features the Amazon logo and the word "Sign-In". There is an input field for "Email or mobile phone number" with an orange border, a yellow "Continue" button, and a note below: "By continuing, you agree to Amazon's [Conditions of Use](#) and [Privacy Notice](#)". At the bottom is a "Need help?" link.



The Microsoft sign-in screen has a white background. It features the Microsoft logo and the word "Sign in". There is an input field for "Email, phone, or Skype". Below it is a note: "No account? [Create one!](#)". At the bottom are links for "Sign in with a security key" with a help icon, and "Sign-in options".

# Form-based username/password auth

POST /shop/login.php HTTP/1.1

Host: lightside.me

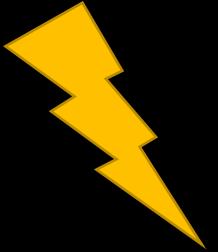
...

username=luke&password=whoismyfather%3F



do you see any problems?

# Plaintext Transmission (again)



Goal	Read credentials in transit
How	MitM between client and server <ul style="list-style-type: none"><li>- ARP Spoofing in your network (WIFI) or the servers' network</li><li>- Malicious ISP (employee)</li><li>- Gateways to foreign countries</li><li>- Spoofed Server (DNS Hijacking)</li><li>- ...</li></ul>
Solution	HTTPS (HTTP + SSL/TLS) HTTP/3 uses TLS 1.3 by default
OWASP Top 10	A02:2021-Cryptographic Failures
(Primary) Violated Principle	“Earn or give, but never assume, trust.”

# Form-based username/password auth

POST /shop/login.php HTTP/1.1

Host: lightside.me

...

username=luke&password=whoismyfather%3F



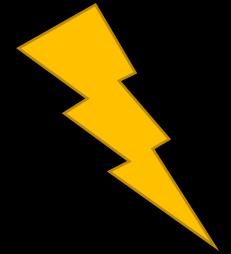
Let's say the username is invalid...

Username incorrect.

Username and/or password incorrect.

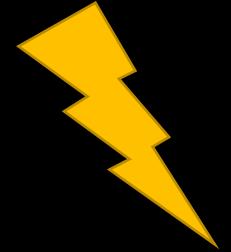
Which one is the better error message?

# Authentication Automation Attacks



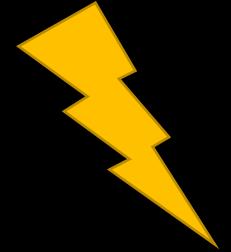
Goal	Guess/Crack the user's credentials (e.g. username / password)
How	
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Authentication Automation Attacks

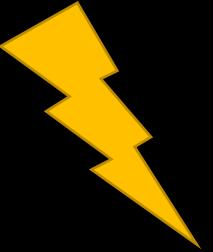


Goal	Guess/Crack the user's credentials (e.g. username / password)
How	Brute Force Attacks Dictionary Attacks Credential Stuffing ...
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Authentication Automation Attacks

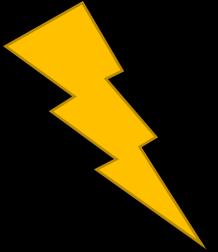


Goal	Guess/Crack the user's credentials (e.g. username / password)
How	Brute Force Attacks Dictionary Attacks Credential Stuffing ...
Solution	Generic Error Messages - Also consider this for the registration
OWASP Top 10	
(Primary) Violated Principle	



# Authentication Automation Attacks

Goal	Guess/Crack the user's credentials (e.g. username / password)
How	Brute Force Attacks Dictionary Attacks Credential Stuffing ...
Solution	Generic Error Messages <ul style="list-style-type: none"><li>- Also consider this for the registration</li></ul> Anti-Automation Techniques <ul style="list-style-type: none"><li>- Captcha, tmp. Lockout, Tarpit, etc.</li></ul>
OWASP Top 10	
(Primary) Violated Principle	



# Authentication Automation Attacks

Goal	Guess/Crack the user's credentials (e.g. username / password)
How	Brute Force Attacks Dictionary Attacks Credential Stuffing ...
Solution	Generic Error Messages <ul style="list-style-type: none"><li>- Also consider this for the registration</li></ul> Anti-Automation Techniques <ul style="list-style-type: none"><li>- Captcha, tmp. Lockout, Tarpit, etc.</li></ul> Password Complexity Requirements <ul style="list-style-type: none"><li>- Current recommendation: length &gt; complexity</li></ul>
OWASP Top 10	
(Primary) Violated Principle	

Type	Password	Time (HSIMP)	Time (PA)	Security Level
<b>8 character common word</b>	required	52 seconds	<1 day	Useless
<b>8 random characters</b>	qkcrmztd	52 seconds	<1 day	Useless
<b>8 random chars w/numbers</b>	kqwvbv832	11 minutes	<1 day	Useless
<b>8 random chars w/mixed case, symbols, &amp; numbers</b>	J5bZ>9p!	20 days	<1 day	Risky

Type	Password	Time (HSIMP)	Time (PA)	Security Level
<b>2 common word password</b>	orange tea	98 days	<1 day	Risky
<b>3 common word password</b>	this is cool	546 years	<1 day	Risky
<b>5 uncommon word password</b>	du-bi-du-bi-doo	12 million years	<1 day	Risky

Type	Password	Time (HSIMP)	Time (PA)	Security Level
<b>Passphrase 1</b>	i own 2 dogs and 1 cat	1 sextillion years	330130 centuries	Secure forever
<b>Passphrase 2</b>	I own 2 dogs and 1 cat!	30 octillion years	8594846 centuries	Secure forever
<b>Passphrase 3</b>	#I own 2 dogs and 1 cat!?	285 nonillion years	1220882818 centuries	Secure forever

# Passphrase Tips and Examples

Here is a list of tips to consider when creating a new passphrase:

- **Don't** use common dictionary words – Ex. orange, car, password
- **Don't** use sequential letters or numbers – Ex. 12345, abcde
- **Don't** use repeated letters/numbers or keyboard patterns – Ex. 111, aaa, qwerty, asdfgh
- **Don't** use the same passphrase for every site if you can help it. Something you could do would be to make a passphrase saying "I am accessing Facebook!" for Facebook, or "I am accessing YouTube!" for YouTube, etc.
- **Consider** using spaces – Ex. Instead of making it "iown2dogs", make it "i own 2 dogs"
- **Consider** using misspellings – Ex. Instead of making it "why would you do that", make it "why wuld u do tht". If the word is not in a dictionary, it helps make it more secure.
- **Consider** using a long **and** complex passphrase – Ex. Instead of making it "i own 2 dogs and 1 cat", add some punctuation and capitalization to make it something like "I own 2 dogs, and 1 cat!"

# Top 500 Passwords

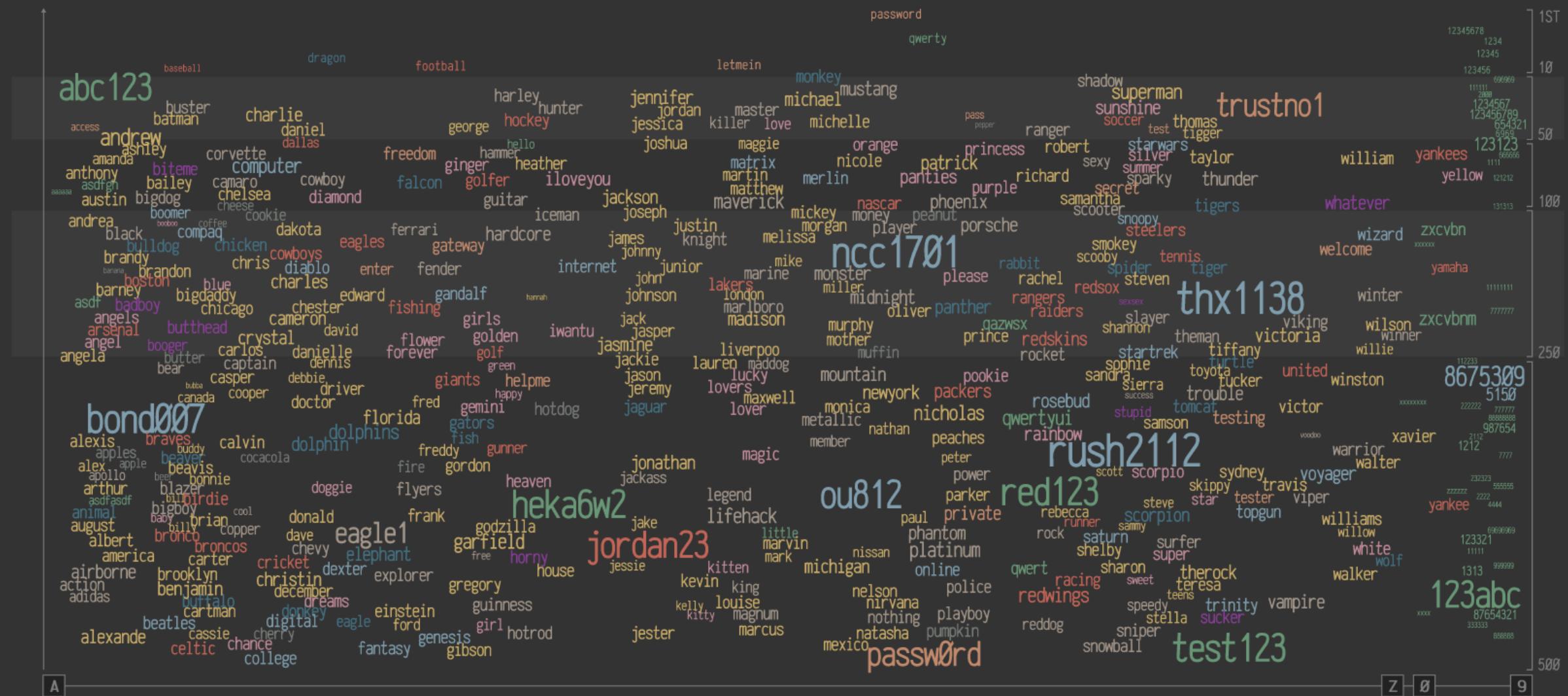
Is Yours Here?

Size = password strength

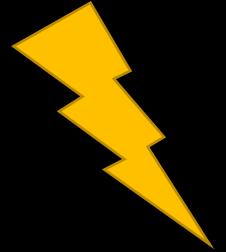
POPULARITY

Filter: Alphanumeric Animals Fluffy Food Macho Names Nerdy Rebellious Security Sport

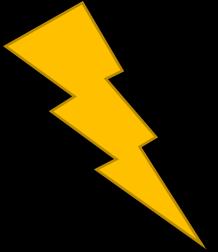
RANK



# Authentication Automation Attacks



Goal	Guess/Crack the user's credentials (e.g. username / password)
How	Brute Force Attacks Dictionary Attacks Credential Stuffing ...
Solution	Generic Error Messages <ul style="list-style-type: none"><li>- Also consider this for the registration</li></ul> Anti-Automation Techniques <ul style="list-style-type: none"><li>- Captcha, tmp. Lockout, Tarpit, etc.</li></ul> Password Complexity Requirements <ul style="list-style-type: none"><li>- Current recommendation: length &gt; complexity</li></ul> MFA (maybe even passwordless)
OWASP Top 10	
(Primary) Violated Principle	



# Authentication Automation Attacks

Goal	Guess/Crack the user's credentials (e.g. username / password)
How	Brute Force Attacks Dictionary Attacks Credential Stuffing ...
Solution	Generic Error Messages <ul style="list-style-type: none"><li>- Also consider this for the registration</li></ul> Anti-Automation Techniques <ul style="list-style-type: none"><li>- Captcha, tmp. Lockout, Tarpit, etc.</li></ul> Password Complexity Requirements <ul style="list-style-type: none"><li>- Current recommendation: length &gt; complexity</li></ul> MFA (maybe even passwordless) Compromised Credential Checking
OWASP Top 10	
(Primary) Violated Principle	

# ';--have i been pwned?

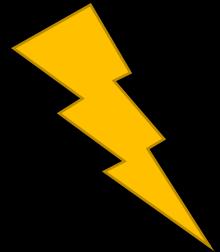
Check if you have an account that has been compromised in a data breach

pwned?

Also available as a service for your application!

<https://haveibeenpwned.com/>

# Authentication Automation Attacks



Goal	Guess/Crack the user's credentials (e.g. username / password)
How	Brute Force Attacks Dictionary Attacks Credential Stuffing ...
Solution	Generic Error Messages <ul style="list-style-type: none"><li>- Also consider this for the registration</li></ul> Anti-Automation Techniques <ul style="list-style-type: none"><li>- Captcha, tmp. Lockout, Tarpit, etc.</li></ul> Password Complexity Requirements <ul style="list-style-type: none"><li>- Current recommendation: length &gt; complexity</li></ul> MFA (maybe even passwordless) Compromised Credential Checking
OWASP Top 10	A07:2021-Identification and Authentication Failures
(Primary) Violated Principle	„Use an authentication mechanism that cannot be bypassed or tampered with.“

Talking about credential  
stuffing...

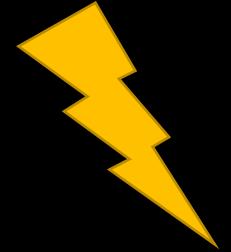
how come those passwords  
got leaked in the first place?

MD5(<password>)

SHA1(<password>)

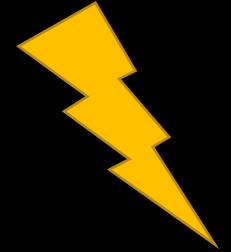
secure?

# Insecure Password Storage



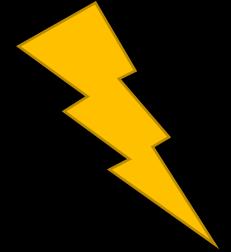
Goal	Steal user credentials
How	
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Insecure Password Storage



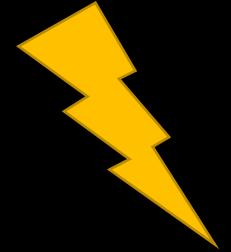
Goal	Steal user credentials
How	SQLi Server misconfigurations Malicious insider ...
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Insecure Password Storage



Goal	Steal user credentials
How	SQLi Server misconfigurations Malicious insider ...
Solution	Use proper mechanisms to store credentials  Special Password Hash Functions (PHF) or Password-based Key Derivation Functions (KDFs) – slow on purpose  they use cryptographic hashes with large salt and high iteration count e.g. <b>bcrypt</b> , <b>scrypt</b> , <b>PBKDF2</b> , <b>HKDF</b> , Lyra2, [Argon2] etc.
OWASP Top 10  (Primary) Violated Principle	

# Insecure Password Storage

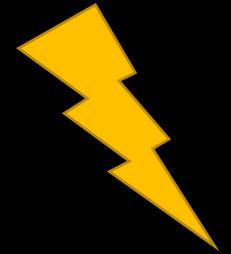


Goal	Steal user credentials
How	SQLi Server misconfigurations Malicious insider ...
Solution	Use proper mechanisms to store credentials  Special Password Hash Functions (PHF) or Password-based Key Derivation Functions (KDFs) – slow on purpose  they use cryptographic hashes with large salt and high iteration count e.g. <b>bcrypt</b> , <b>scrypt</b> , <b>PBKDF2</b> , <b>HKDF</b> , Lyra2, [Argon2] etc.
OWASP Top 10	A02:2021-Cryptographic Failures
(Primary) Violated Principle	„Use cryptography correctly.“

And what's about  
those passwords we  
need in our code?

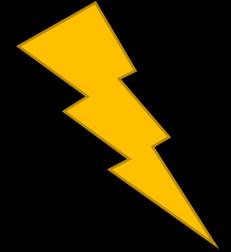
e.g. DB password, service accounts, etc.

# Passwords in Source Code



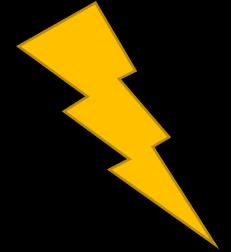
Goal	Steal passwords of technical users
How	
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Passwords in Source Code



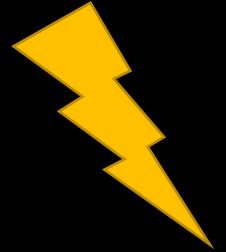
Goal	Steal passwords of technical users
How	Error messages Logs Version control system (GIT, SVN, ...)
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Passwords in Source Code



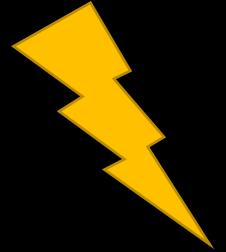
Goal	Steal passwords of technical users
How	Error messages Logs Version control system (GIT, SVN, ...)
Solution	bare minimum: put them in a config file  Recommended solution heavily depends on platform, e.g. secret stores encrypted config files  The „master key“ can be set e.g. via environment variable
OWASP Top 10	
(Primary) Violated Principle	

# Passwords in Source Code



Goal	Steal passwords of technical users
How	Error messages Logs Version control system (GIT, SVN, ...)
Solution	bare minimum: put them in a config file  Recommended solution heavily depends on platform, e.g. secret stores encrypted config files  The „master key“ can be set e.g. via environment variable
OWASP Top 10	A02:2021-Cryptographic Failures
(Primary) Violated Principle	

# Passwords in Source Code



Goal	Steal passwords of technical users
How	Error messages Logs Version control system (GIT, SVN, ...)
Solution	bare minimum: put them in a config file  Recommended solution heavily depends on platform, e.g. secret stores encrypted config files  The „master key“ can be set e.g. via environment variable
OWASP Top 10	A02:2021-Cryptographic Failures
(Primary) Violated Principle	„Identify sensitive data and how they should be handled.“

# Avoid common pitfalls

- Make sure users can change their passwords
  - always explicitly ask for the old one before changing it
- Consider the registration dialog
  - e.g. user enumeration can also happen there pretty easily
    - Possible solution: e-mail address as username
- Be really careful with „Forgot Password“ features
  - always give positive feedback
  - never give the user his old password
    - you shouldn't even be able to anyway!
  - Security-Questions are pretty weak
    - very easy for targeted attacks
    - especially with this social media stuff...

## YOUR SCOTTISH INSULTS

YOUR BIRTH MONTH	YOUR FAVOURITE COLOR
Jan : Hackit	White : Reekin'
Feb : Doaty	Black : Bushy-faced
Mar : Naffy	Grey : Blooming
Apr : Jakey	Red : Weapons-grade
May : Witless	Blue : Cock-juggling
Jun : Gommy	Purple : Ferret-wearing
Jul : Hackit	Pink : Knobdobbin'
Aug : Absolute	Yellow : Howlin'
Sep : Scabby	Orange : Lavvy-heidled
Oct : Fucking	Brown : Fuddy
Nov : Complete	Grey : Skelpie
Dec : Utter	Green : Weasel-headed
	Others : Dilly-daw

### THE LAST LETTER OF YOUR NAME

A: Arse	J: Jizztrumpet	S: Spoon
B: Dunderhead	K: Tattie	T: Thundercunt
C: Cockwomble	L: Ladle	U: Shitgibbon
D: Doughnut	M: Muppet	V: Scunner
E: Clawbaw	N: Numpty	W: Smout
F: Cocksplat	O: Old tit	X: Twonk
G: Gowk	P: Plum	Y: Puddock
H: Hellbeast	Q: Rocket	Z: Tube
I: Clawbraw	R: Roaster	

VIA 9GAG.COM

<https://9gag.com/gag/a5ojZ0O>

## SWEAR LIKE A PIRATE

### YOUR FAVOURITE SEASON

Spring : Arrr!	Summer : Ahoy!
Autumn : Yo-ho-ho!	Winter : Bite me will ye?

### YE + THE FIRST INITIAL OF YOUR NAME

A: Pig Faced	J: Pantywaist	S: Spider-kissin'
B: Butt Scratchin'	K: Nauseatin'	T: Toad Lickin'
C: Chimp Faced	L: Lard Brained	U: Snuff Snortin'
D: Dung Diggin'	M: Puppy Killin'	V: Scabby Arsed
E: Bug Brained	N: Nose Pickin'	W: Worm Livered
F: Fork Faced	O: Grog Faced	X: Screw Eyed
G: Greasy	P: Penny Lickin'	Y: Bug Chewin'
H: Gold Stealin'	Q: Peesoaked	Z: Pants Soolin'
I: Ladder Suckin'	R: Back Stabbin'	

### YOUR MONTH OF BIRTH

Jan : Cockroach	Jul : Gutless Kidneywife
Feb : Buttwipe	Aug : Son Of A Biscuit Eater
Mar : Gobshyte	Sep : Spineless Bastitch
Apr : Wiggly Maggot	Oct : Scurvy Baboon
May : Gaggin' Reptile	Nov : Power Monkey
Jun : Spit Weasel	Dec : Oyster Sucker

VIA 9GAG.COM

<https://9gag.com/gag/a7dnR4e>

Does HTTP itself  
support  
authentication?

sure does, but...

# HTTP Basic Auth

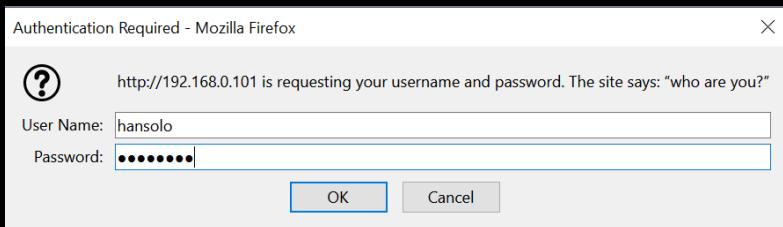
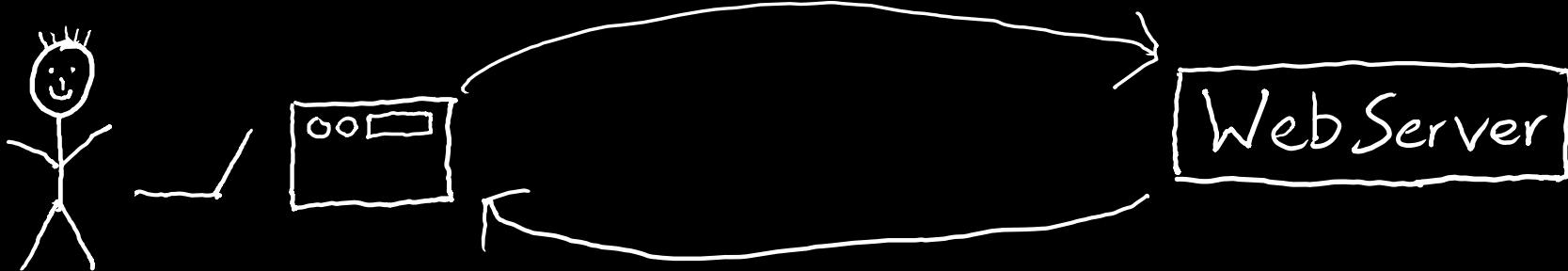


# HTTP Basic Auth

GET /secret.html HTTP/1.1

...

Authorization: Basic aGFuc29sbzpsb3ZlLWx1YQ==



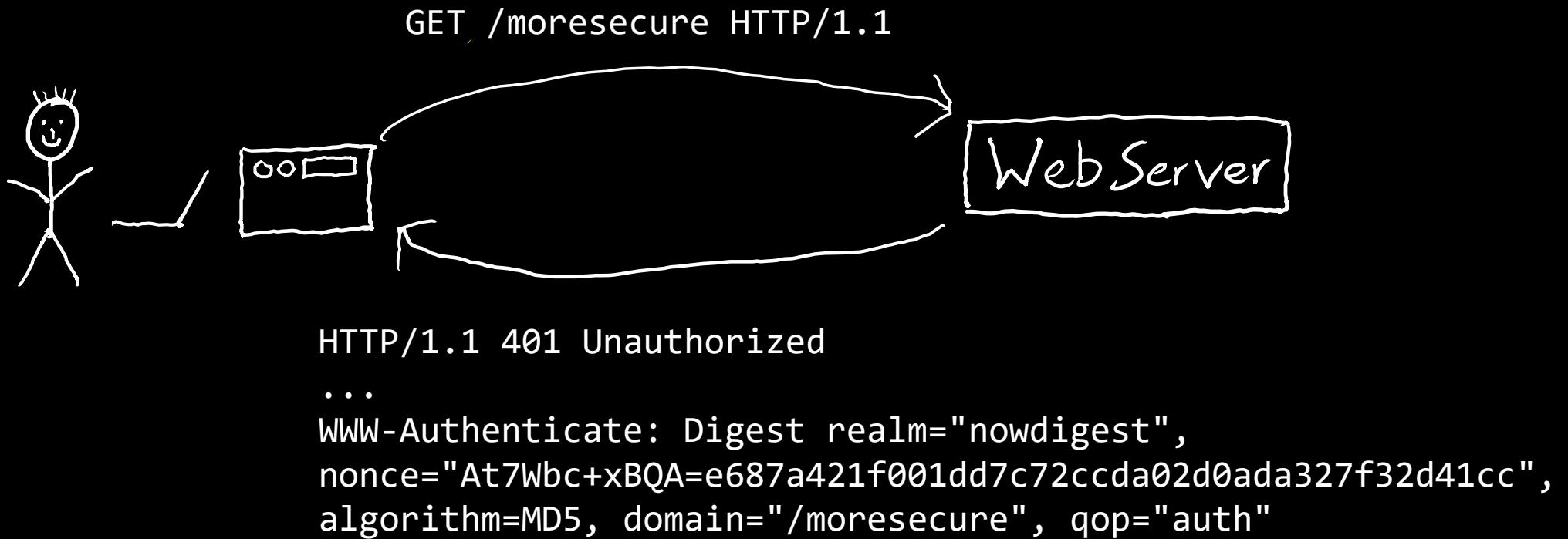
Base64(<username>:<password>)

do you see any problems?

# HTTP Basic Auth

- Similar to form-based
  - Doesn't protect credential's confidentiality and integrity
    - Everybody can decode base64
  - Doesn't protect against server-spoofing
  - Needs HTTPS
- User cannot manually logout
- Application cannot force user to logout
- Limited influence on password storage
  - depends on webserver
  - secure options available for most servers
    - e.g. Apache supports bcrypt

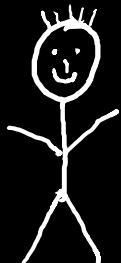
# HTTP Digest Auth



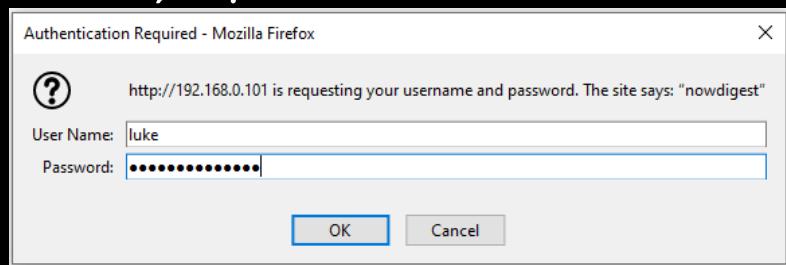
```
GET /moresecure HTTP/1.1
```

```
...
```

```
Authorization: Digest username="luke", realm="nowdigest",  
nonce="At7Wbc+xBQA=e687a421f001dd7c72ccda02d0ada327f32d41cc",  
uri="/moresecure/", algorithm=MD5,  
response="bbd3e5713f5f560e788ca0d34b2229d3",  
qop=auth, nc=00000001, cnonce="2c875dc2ca74015"
```



Web Server



$HA1 = \text{MD5}(\text{username}:\text{realm}:\text{password})$

$HA2 = \text{MD5}(\text{method}:\text{digestURI})$

$\text{response} = \text{MD5}(\text{HA1}:\text{nonce}:\text{nonceCount}:\text{cnonce}:\text{qop}:\text{HA2})$

do you see any problems?

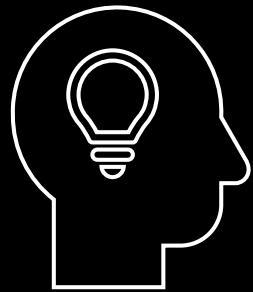
# HTTP Digest Auth

- Still doesn't really prevent MitM
- Still doesn't really prevent server spoofing
- Most security options are optional
- Password is stored either in plaintext or MD5 hashed
  - both terrible
- Just don't use it
- If you really want to use HTTP auth:
  - Basic + HTTPS + Strong password storage mechanism

How can we make  
authentication  
stronger?

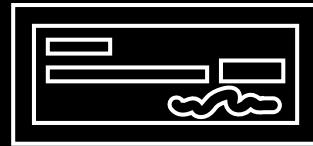
# Which Auth-Factors do you know?

something...



you know

username/password,  
PIN, etc.



you have

certificate, smartcard,  
token, etc.

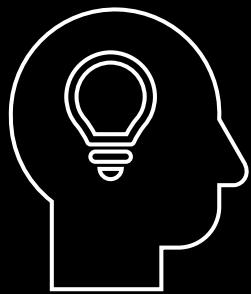


you are

fingerprint, iris,  
venes, etc.

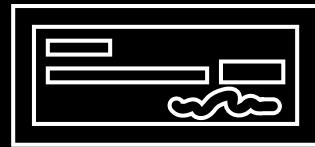
# Which Auth-Factors do you know?

increase security by combining different factors  
2FA / MFA



you know

username/password,  
PIN, etc.



you have

certificate, smartcard,  
token, etc.



you are

fingerprint, iris,  
venes, etc.

# Options I personally really like...

TOTP

Time-based  
One-time Password



FIDO 2

WebAuthN/CTAP  
U2F



Bürgerkarte

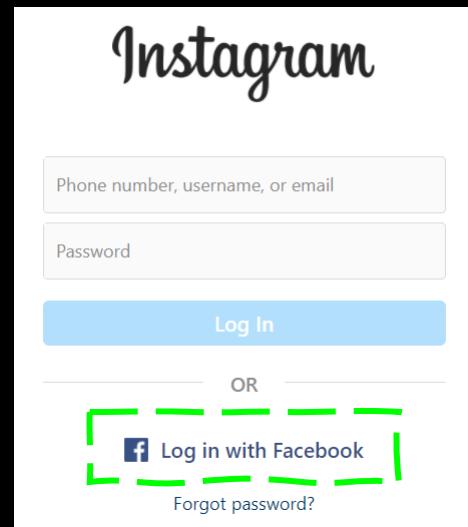
Personal Smartcard  
in Austria



account compromise is  
99.9%  
less likely with enabled MFA

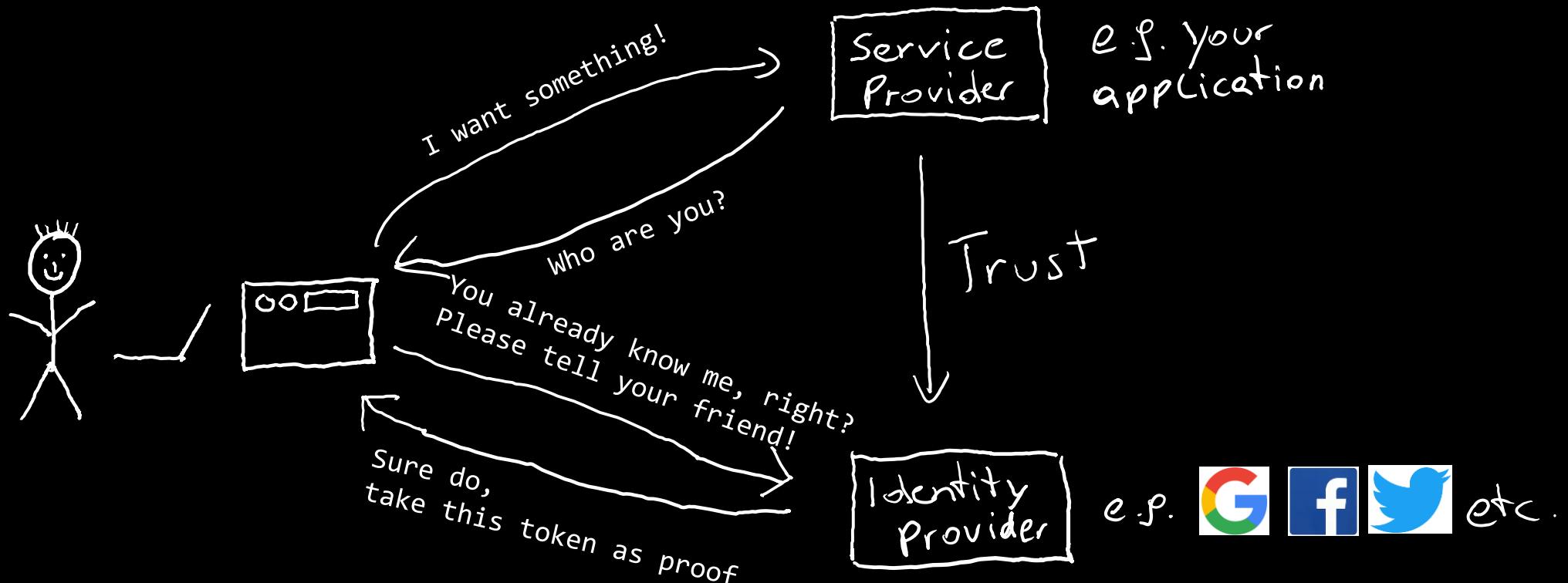
<https://techcommunity.microsoft.com/t5/azure-active-directory-identity/your-pa-word-doesn-t-matter/ba-p/731984>

# one more thing regarding authentication...

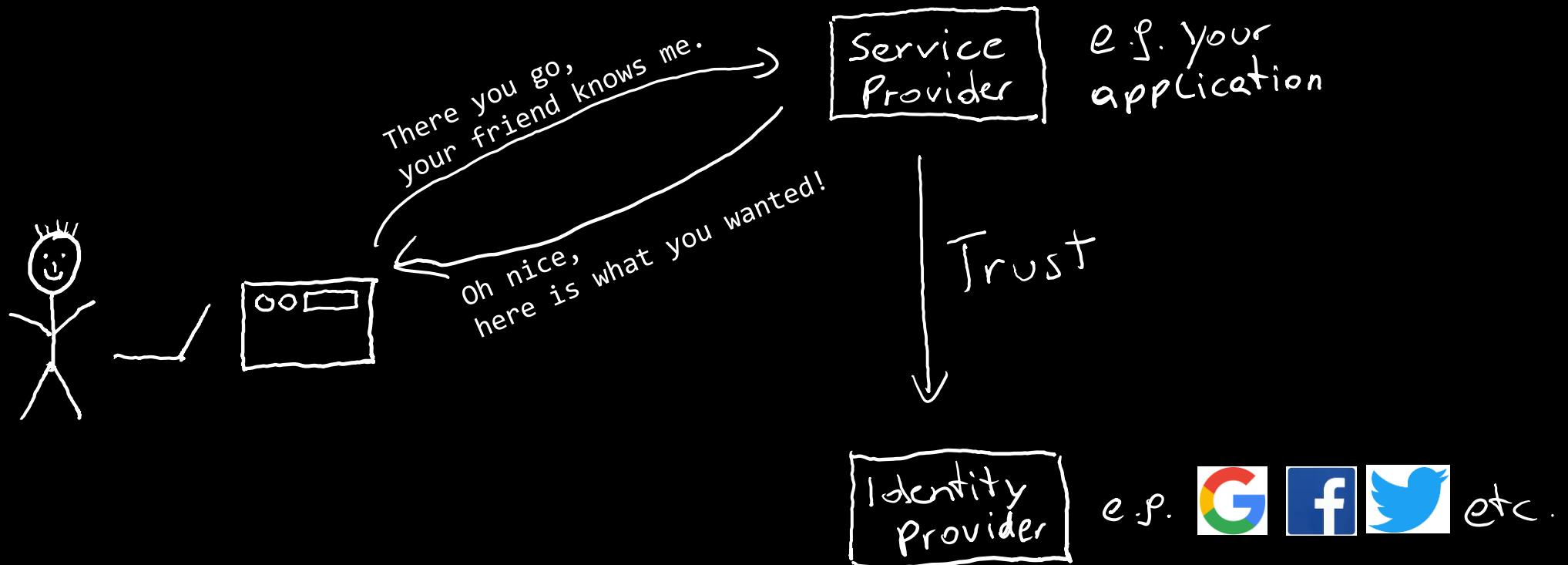


what's that?

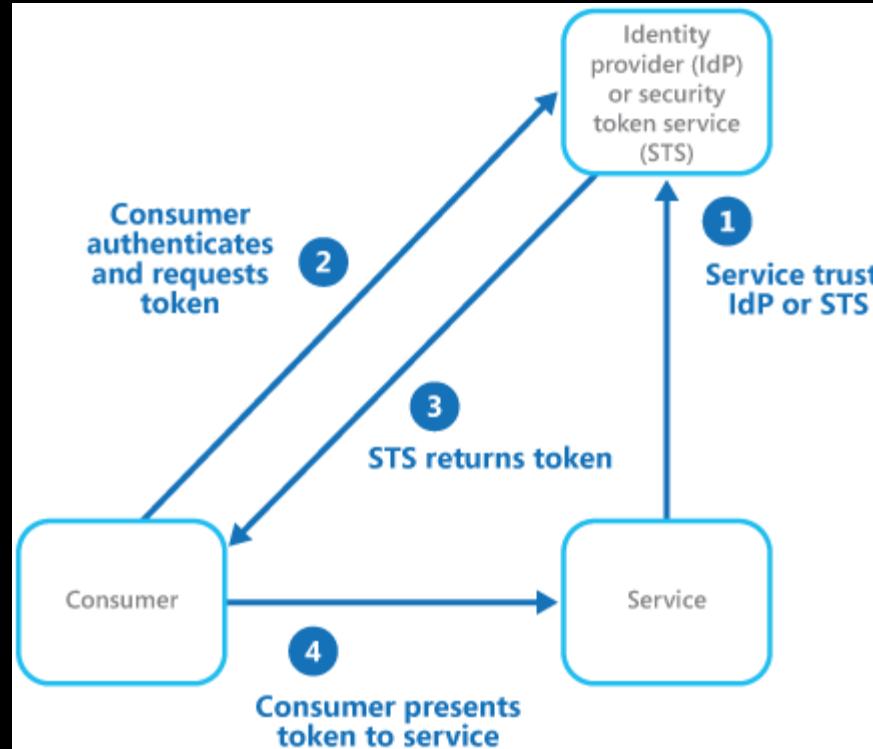
# Identity Federation simplified...



# Identity Federation simplified...

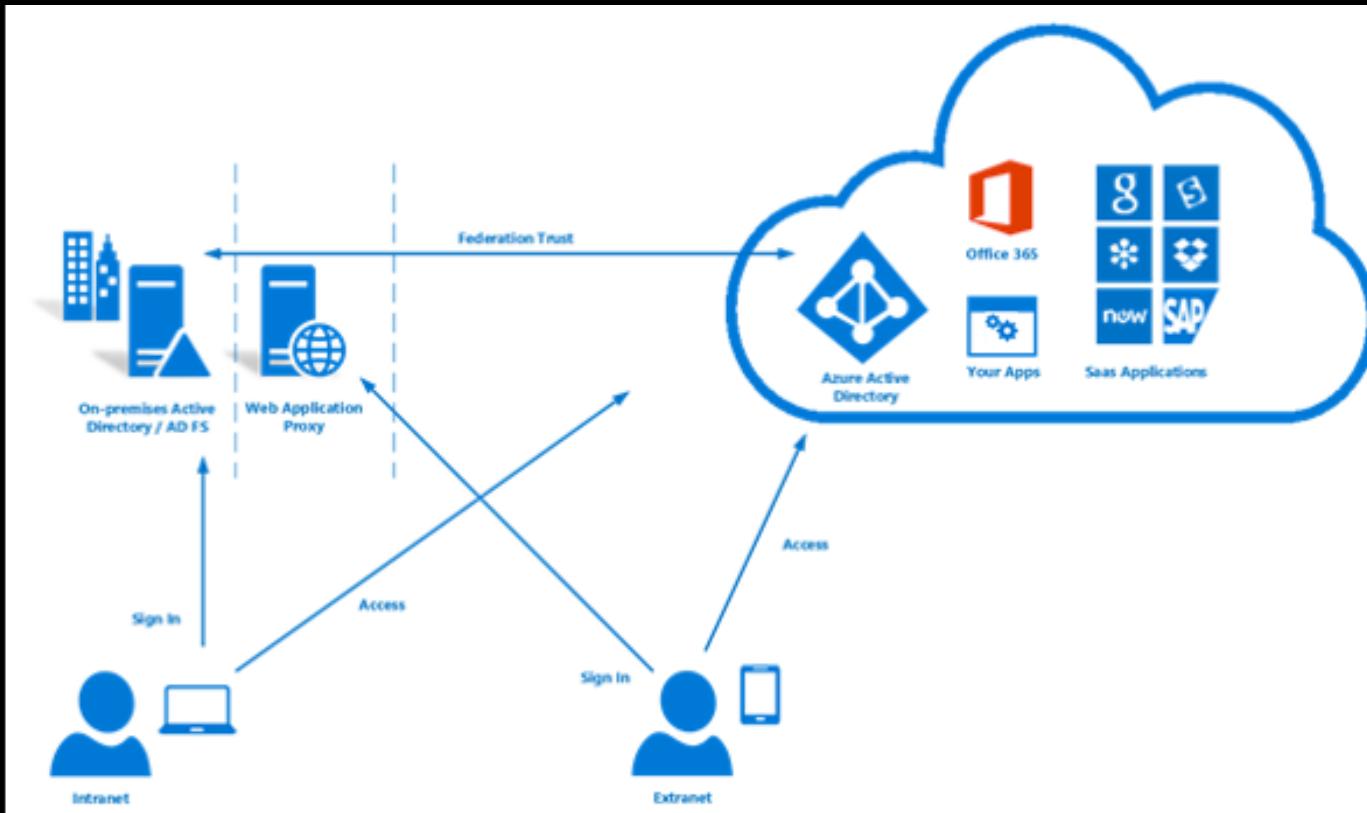


# Identity Federation or as microsoft describes it...



<https://docs.microsoft.com/en-us/azure/architecture/patterns/federated-identity>

# Identity Federation also used in enterprise env...



<https://docs.microsoft.com/en-us/azure/active-directory/hybrid/whatis-fed>

# Identity Federation

- Complex enough for an own course
- Several technologies available for implementation
  - e.g. SAML (Assertions), JWT, OAuth (2), OpenID Connect, etc.
- A lot of possible pitfalls...
  - e.g. Authentication vs. Authorization
- Recommended reading:
  - <https://oauth.net/articles/authentication/>
  - <https://openid.net/connect/>
  - <https://docs.microsoft.com/en-us/azure/architecture/patterns/federated-identity>
- Recommended reading for JWT Security:
  - <https://pragmaticwebsecurity.com/files/cheatsheets/jwt.pdf>
  - <https://datatracker.ietf.org/doc/html/rfc8725>

ok, one really last  
thing regarding  
authentication...

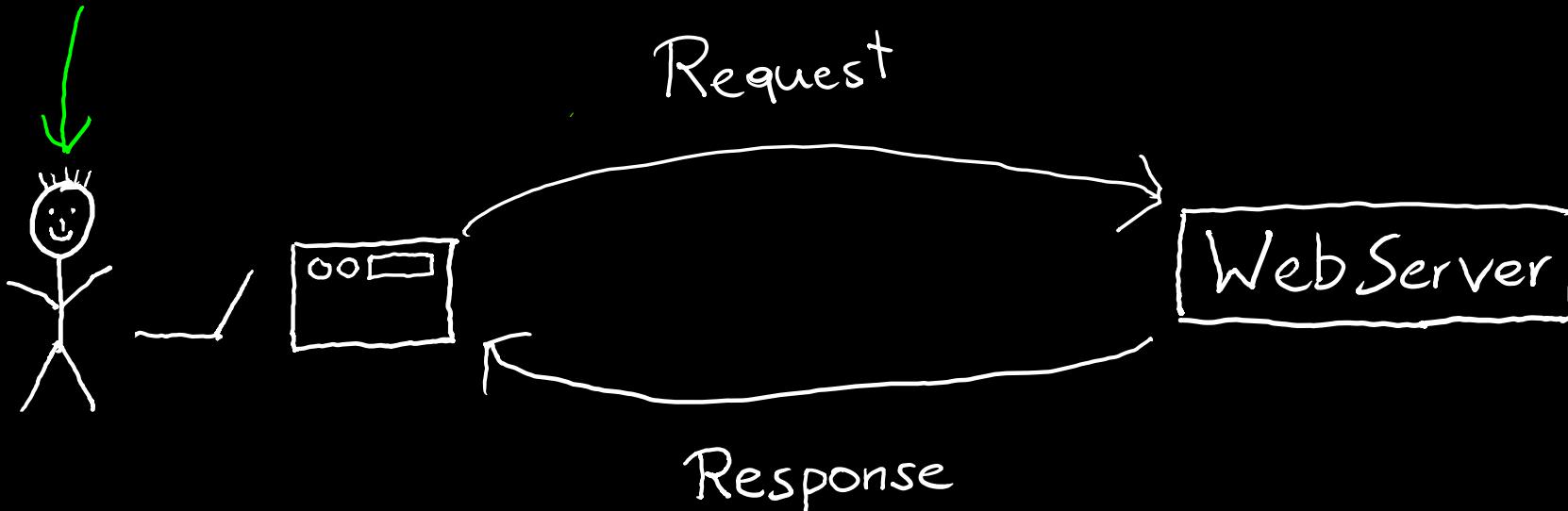
pretty powerful to include some  
„environmental checks“

e.g. login attempt from foreign IP,  
login attempt from new/unknown device,  
multiple failed login attempts, etc.

notify the user if something is suspicious

a.k.a „conditional access“

ok, we now know who this guy is...



... but HTTP itself is a stateless protocol  
how can we remember him between requests?

# Sessions - Cookie based client side

POST /testlogin HTTP/1.1

Host: 192.168.0.101

...

username=luke&password=whoismyfather%3F



**Login**

Username:  
luke

Password:  
\*\*\*\*\*

Login

HTTP/1.1 200 OK

Set-Cookie: username=luke

Set-Cookie: role=jedi

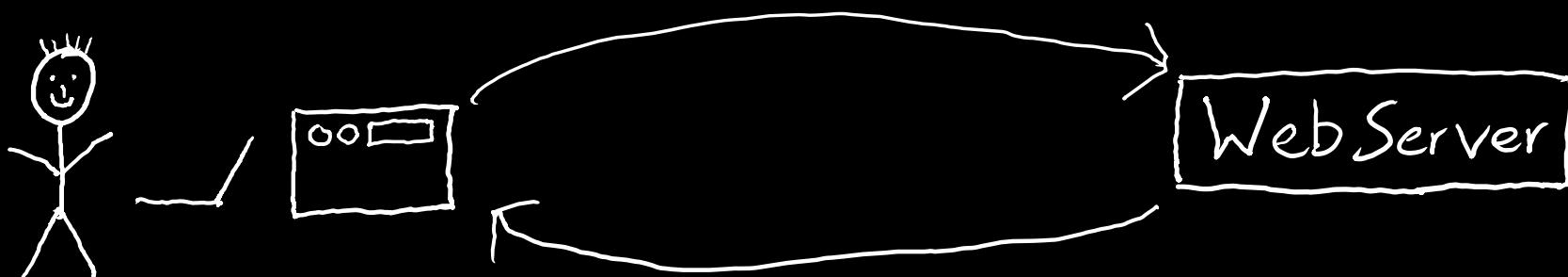
...

<some html site>

# Sessions - Cookie based client side

Cookie Store	
192.168.0.101	username=luke role=jedi

```
POST /somerequest HTTP/1.1
Host: 192.168.0.101
Cookie: username=luke; role=jedi
...
...
```



do you see  
any problems?

Cookie Quick Manager

Extension... Manager | moz-extension://522e0ba0-f091-48bc-b5ab-54...

Search a domain...  Sub-domains Context(s): All Auto-refresh

Domains (1)  
192.168.0.101 (2)

Cookies  
**username:luke**  
**role:jedi**

Details

Domain: 192.168.0.101

First-Party:

Name: role

Value: jedi  
 URL  Base64

Path: /

Context: Default

httpOnly  sameSite: No restriction

isSecure

Save the current cookie

Cookie Quick Manager

Extension... Manager | moz-extension://522e0ba0-f091-48bc-b5ab-54...

Search a domain...  Sub-domains Context(s): All Auto-refresh

Domains (1)  
192.168.0.101 (2)

Cookies  
username:luke  
role:jedi

Details

Domain: 192.168.0.101

First-Party

Name: role

Value: admin  
 URL  Base64

Path: /

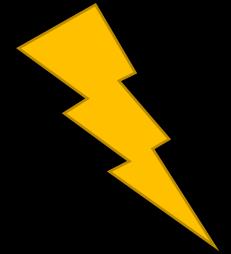
Context: Default

httpOnly  sameSite: No restriction

isSecure

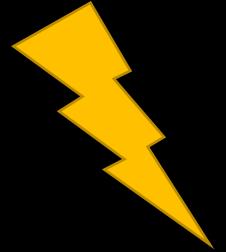
Save the current cookie

# Client-side Manipulation (of cookies)



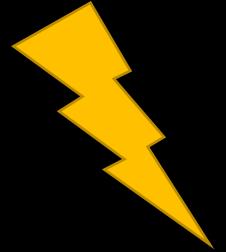
Goal	Manipulate data security checks depend on
How	
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Client-side Manipulation (of cookies)



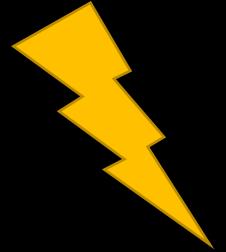
Goal	Manipulate data security checks depend on
How	Misplaced trust User has full control over his client and can manipulate everything on it!
Solution	
OWASP Top 10	
(Primary) Violated Principle	

# Client-side Manipulation (of cookies)



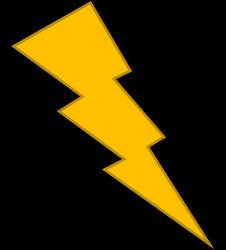
Goal	Manipulate data security checks depend on
How	Misplaced trust User has full control over his client and can manipulate everything on it!
Solution	Encrypt and sign cookie values
OWASP Top 10	
(Primary) Violated Principle	

# Client-side Manipulation (of cookies)



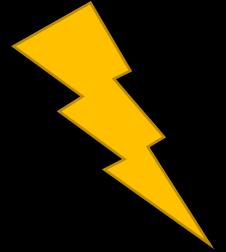
Goal	Manipulate data security checks depend on
How	Misplaced trust User has full control over his client and can manipulate everything on it!
Solution	Encrypt and sign cookie values Remaining Problem: Application cannot close all sessions of a user (e.g in case of compromise)
OWASP Top 10	
(Primary) Violated Principle	

# Client-side Manipulation (of cookies)



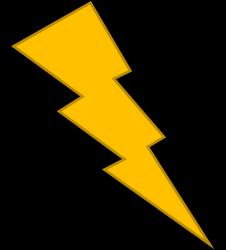
Goal	Manipulate data security checks depend on
How	Misplaced trust User has full control over his client and can manipulate everything on it!
Solution	Encrypt and sign cookie values Remaining Problem: Application cannot close all sessions of a user (e.g in case of compromise) ASP.NET ViewState works similar (hidden form field) JWT (JSON Web Tokens) also work similar
OWASP Top 10	
(Primary) Violated Principle	

# Client-side Manipulation (of cookies)



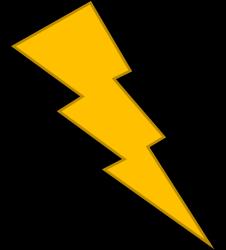
Goal	Manipulate data security checks depend on
How	Misplaced trust User has full control over his client and can manipulate everything on it!
Solution	Encrypt and sign cookie values Remaining Problem: Application cannot close all sessions of a user (e.g in case of compromise) ASP.NET ViewState works similar (hidden form field) JWT (JSON Web Tokens) also work similar
OWASP Top 10	Use Server-side Session Management
(Primary) Violated Principle	

# Client-side Manipulation (of cookies)



Goal	Manipulate data security checks depend on
How	Misplaced trust User has full control over his client and can manipulate everything on it!
Solution	Encrypt and sign cookie values Remaining Problem: Application cannot close all sessions of a user (e.g in case of compromise) ASP.NET ViewState works similar (hidden form field) JWT (JSON Web Tokens) also work similar
OWASP Top 10	A01:2021-Broken Access Control
(Primary) Violated Principle	

# Client-side Manipulation (of cookies)



Goal	Manipulate data security checks depend on
How	Misplaced trust User has full control over his client and can manipulate everything on it!
Solution	Encrypt and sign cookie values Remaining Problem: Application cannot close all sessions of a user (e.g in case of compromise) ASP.NET ViewState works similar (hidden form field) JWT (JSON Web Tokens) also work similar
OWASP Top 10	A01:2021-Broken Access Control
(Primary) Violated Principle	„Earn or give, but never assume, trust.“

# Avoid common pitfalls

- We just said, we cannot trust the client
- How about signed rich-clients?
  - e.g. it's possible to sign a Java client and prevent the JVM from executing unsigned Java applications via GPO in your company
- Still not a good idea!
  - Code-Signing protects the user from getting a compromised client!
  - Code-Signing will not protect your server-side application!
    - If an attacker wants to start a manipulated client, he will succeed eventually
      - alternative JVM
      - Virtual Machine
      - Foreign Device
      - etc.

**THE CLIENT,  
YOU MUST NEVER TRUST**



**MY YOUNG PADAWAN**

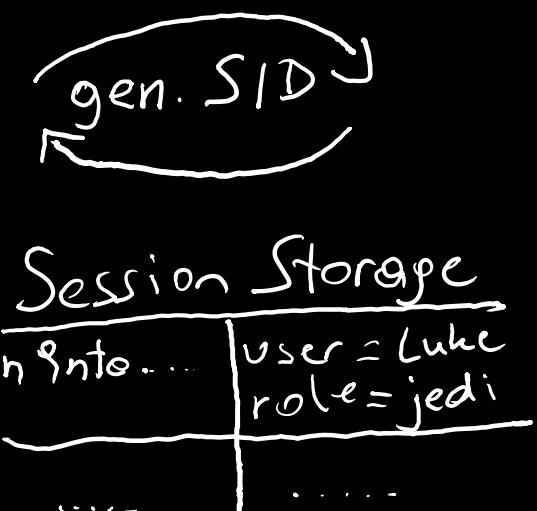
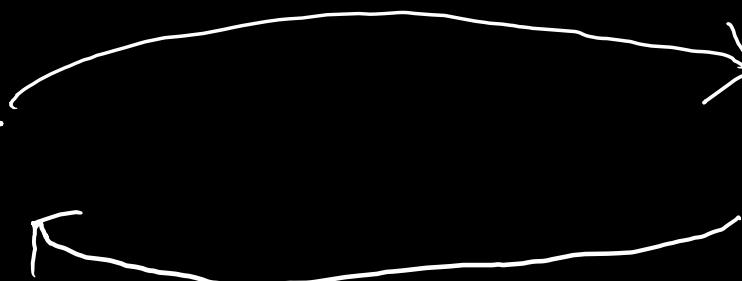
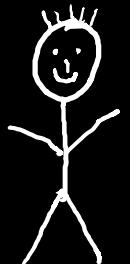
# Sessions - Cookie based server side

POST /testlogin HTTP/1.1

Host: 192.168.0.101

...

username=luke&password=whoismyfather%3F



**Login**

Username:

Password:

HTTP/1.1 200 OK

Set-Cookie: SESSID=n9ntoqopeu

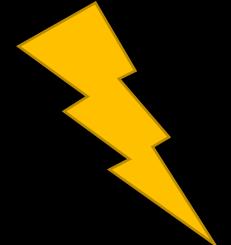
...

<some html site>

# Sessions - Cookie based server side

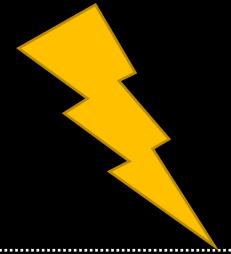


# Session Hijacking



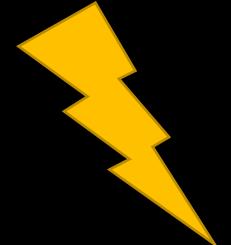
Goal	Get victim's SID and hijack his session
How	
Solution	
OWASP Top 10	
Violated Principle	

# Session Hijacking



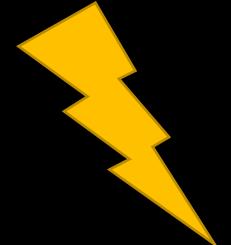
Goal	Get victim's SID and hijack his session
How	Guessable SID
Solution	
OWASP Top 10	
Violated Principle	

# Session Hijacking



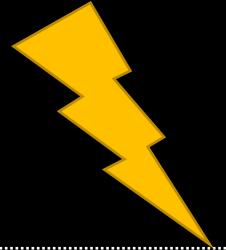
Goal	Get victim's SID and hijack his session
How	Guessable SID
Solution	(Secure) Random generation with enough entropy bare min: $\geq 64$ bit recommended: $\geq 128$ bit Invalidate SIDs after a defined idle time
OWASP Top 10	
Violated Principle	

# Session Hijacking



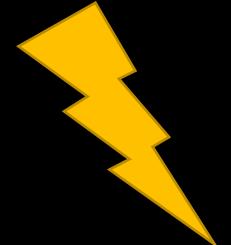
Goal	Get victim's SID and hijack his session
How	Guessable SID MitM at insecure channel Firesheep (2010)
Solution	(Secure) Random generation with enough entropy bare min: $\geq 64$ bit recommended: $\geq 128$ bit Invalidate SIDs after a defined idle time
OWASP Top 10	
Violated Principle	

# Session Hijacking



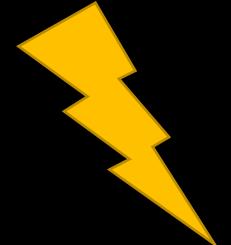
Goal	Get victim's SID and hijack his session
How	Guessable SID MitM at insecure channel Firesheep (2010)
Solution	(Secure) Random generation with enough entropy bare min: $\geq 64$ bit recommended: $\geq 128$ bit Invalidate SIDs after a defined idle time HTTPS for whole communication not just login Secure flag Set-Cookie: SID=<ID>; Secure
OWASP Top 10	
Violated Principle	

# Session Hijacking



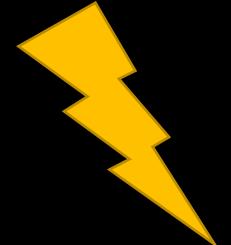
Goal	Get victim's SID and hijack his session
How	Guessable SID MitM at insecure channel Firesheep (2010) SID in GET Parameter Logfiles, Browser history, Referrer header
Solution	(Secure) Random generation with enough entropy bare min: $\geq 64$ bit recommended: $\geq 128$ bit Invalidate SIDs after a defined idle time HTTPS for whole communication not just login Secure flag Set-Cookie: SID=<ID>; Secure
OWASP Top 10	
Violated Principle	

# Session Hijacking



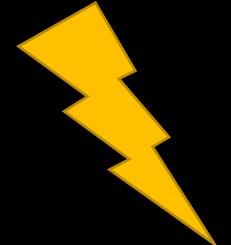
Goal	Get victim's SID and hijack his session
How	Guessable SID MitM at insecure channel Firesheep (2010) SID in GET Parameter Logfiles, Browser history, Referrer header
Solution	(Secure) Random generation with enough entropy bare min: $\geq 64$ bit recommended: $\geq 128$ bit Invalidate SIDs after a defined idle time HTTPS for whole communication not just login Secure flag Set-Cookie: SID=<ID>; Secure Send SID only in header or body
OWASP Top 10	
Violated Principle	

# Session Hijacking



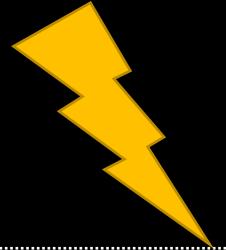
Goal	Get victim's SID and hijack his session
How	<ul style="list-style-type: none"><li>Guessable SID</li><li>MitM at insecure channel<ul style="list-style-type: none"><li>Firesheep (2010)</li></ul></li><li>SID in GET Parameter<ul style="list-style-type: none"><li>Logfiles, Browser history, Referrer header</li></ul></li><li>Steal SID via XSS<ul style="list-style-type: none"><li>we will cover this later</li></ul></li></ul>
Solution	<ul style="list-style-type: none"><li>(Secure) Random generation with enough entropy<ul style="list-style-type: none"><li>bare min: <math>\geq 64</math> bit</li><li>recommended: <math>\geq 128</math> bit</li></ul></li><li>Invalidate SIDs after a defined idle time</li><li>HTTPS for whole communication<ul style="list-style-type: none"><li>not just login</li></ul></li><li>Secure flag<ul style="list-style-type: none"><li>Set-Cookie: SID=&lt;ID&gt;; Secure</li></ul></li><li>Send SID only in header or body</li></ul>
OWASP Top 10	
Violated Principle	

# Session Hijacking

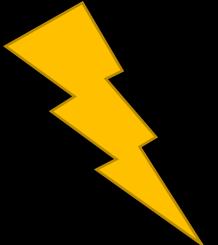


Goal	Get victim's SID and hijack his session
How	<ul style="list-style-type: none"><li>Guessable SID</li><li>MitM at insecure channel<ul style="list-style-type: none"><li>Firesheep (2010)</li></ul></li><li>SID in GET Parameter<ul style="list-style-type: none"><li>Logfiles, Browser history, Referrer header</li></ul></li><li>Steal SID via XSS<ul style="list-style-type: none"><li>we will cover this later</li></ul></li></ul>
Solution	<ul style="list-style-type: none"><li>(Secure) Random generation with enough entropy<ul style="list-style-type: none"><li>bare min: <math>\geq 64</math> bit</li><li>recommended: <math>\geq 128</math> bit</li></ul></li><li>Invalidate SIDs after a defined idle time</li><li>HTTPS for whole communication<ul style="list-style-type: none"><li>not just login</li></ul></li><li>Secure flag<ul style="list-style-type: none"><li>Set-Cookie: SID=&lt;ID&gt;; Secure</li></ul></li><li>Send SID only in header or body</li><li>Pay close attention to our XSS lecture later ;-)</li></ul>
OWASP Top 10	
Violated Principle	

# Session Hijacking



Goal	Get victim's SID and hijack his session
How	<ul style="list-style-type: none"><li>Guessable SID</li><li>MitM at insecure channel<ul style="list-style-type: none"><li>Firesheep (2010)</li></ul></li><li>SID in GET Parameter<ul style="list-style-type: none"><li>Logfiles, Browser history, Referrer header</li></ul></li><li>Steal SID via XSS<ul style="list-style-type: none"><li>we will cover this later</li></ul></li></ul>
Solution	<ul style="list-style-type: none"><li>(Secure) Random generation with enough entropy<ul style="list-style-type: none"><li>bare min: <math>\geq 64</math> bit</li><li>recommended: <math>\geq 128</math> bit</li></ul></li><li>Invalidate SIDs after a defined idle time</li><li>HTTPS for whole communication<ul style="list-style-type: none"><li>not just login</li></ul></li><li>Secure flag<ul style="list-style-type: none"><li>Set-Cookie: SID=&lt;ID&gt;; Secure</li></ul></li><li>Send SID only in header or body</li><li>Pay close attention to our XSS lecture later ;-)</li></ul>
OWASP Top 10	A07:2021-Identification and Authentication Failures
Violated Principle	



# Session Hijacking

Goal	Get victim's SID and hijack his session
How	<ul style="list-style-type: none"><li>Guessable SID</li><li>MitM at insecure channel<ul style="list-style-type: none"><li>Firesheep (2010)</li></ul></li><li>SID in GET Parameter<ul style="list-style-type: none"><li>Logfiles, Browser history, Referrer header</li></ul></li><li>Steal SID via XSS<ul style="list-style-type: none"><li>we will cover this later</li></ul></li></ul>
Solution	<ul style="list-style-type: none"><li>(Secure) Random generation with enough entropy<ul style="list-style-type: none"><li>bare min: <math>\geq 64</math> bit</li><li>recommended: <math>\geq 128</math> bit</li></ul></li><li>Invalidate SIDs after a defined idle time</li><li>HTTPS for whole communication<ul style="list-style-type: none"><li>not just login</li></ul></li><li>Secure flag<ul style="list-style-type: none"><li>Set-Cookie: SID=&lt;ID&gt;; Secure</li></ul></li><li>Send SID only in header or body</li><li>Pay close attention to our XSS lecture later ;-)</li></ul>
OWASP Top 10	A07:2021-Identification and Authentication Failures
Violated Principle	„Identify sensitive data and how they should be handled.“



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

Solution

OWASP Top 10

(Primary)  
Violated Principle



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

This only works in specific scenarios with very specific circumstances

Solution

OWASP Top 10

(Primary)  
Violated Principle



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

This only works in specific scenarios with very specific circumstances  
application also assigns SID to anonymous users (before login)  
attacker can retrieve this SID from the user  
e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.

Solution

OWASP Top 10

(Primary)  
Violated Principle



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

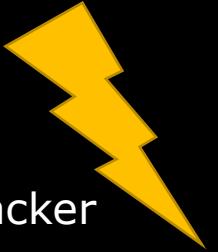
How

This only works in specific scenarios with very specific circumstances  
application also assigns SID to anonymous users (before login)  
attacker can retrieve this SID from the user  
e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.  
or attacker is able to set a SID in the user's browser  
e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.

Solution

OWASP Top 10

(Primary)  
Violated Principle



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

This only works in specific scenarios with very specific circumstances  
application also assigns SID to anonymous users (before login)  
attacker can retrieve this SID from the user  
e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.  
or attacker is able to set a SID in the user's browser  
e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.

application doesn't change SID after successful authentication  
SID known by the attacker is now authenticated

Solution

OWASP Top 10

(Primary)  
Violated Principle



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

This only works in specific scenarios with very specific circumstances  
application also assigns SID to anonymous users (before login)  
attacker can retrieve this SID from the user  
e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.  
or attacker is able to set a SID in the user's browser  
e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.

application doesn't change SID after successful authentication  
SID known by the attacker is now authenticated

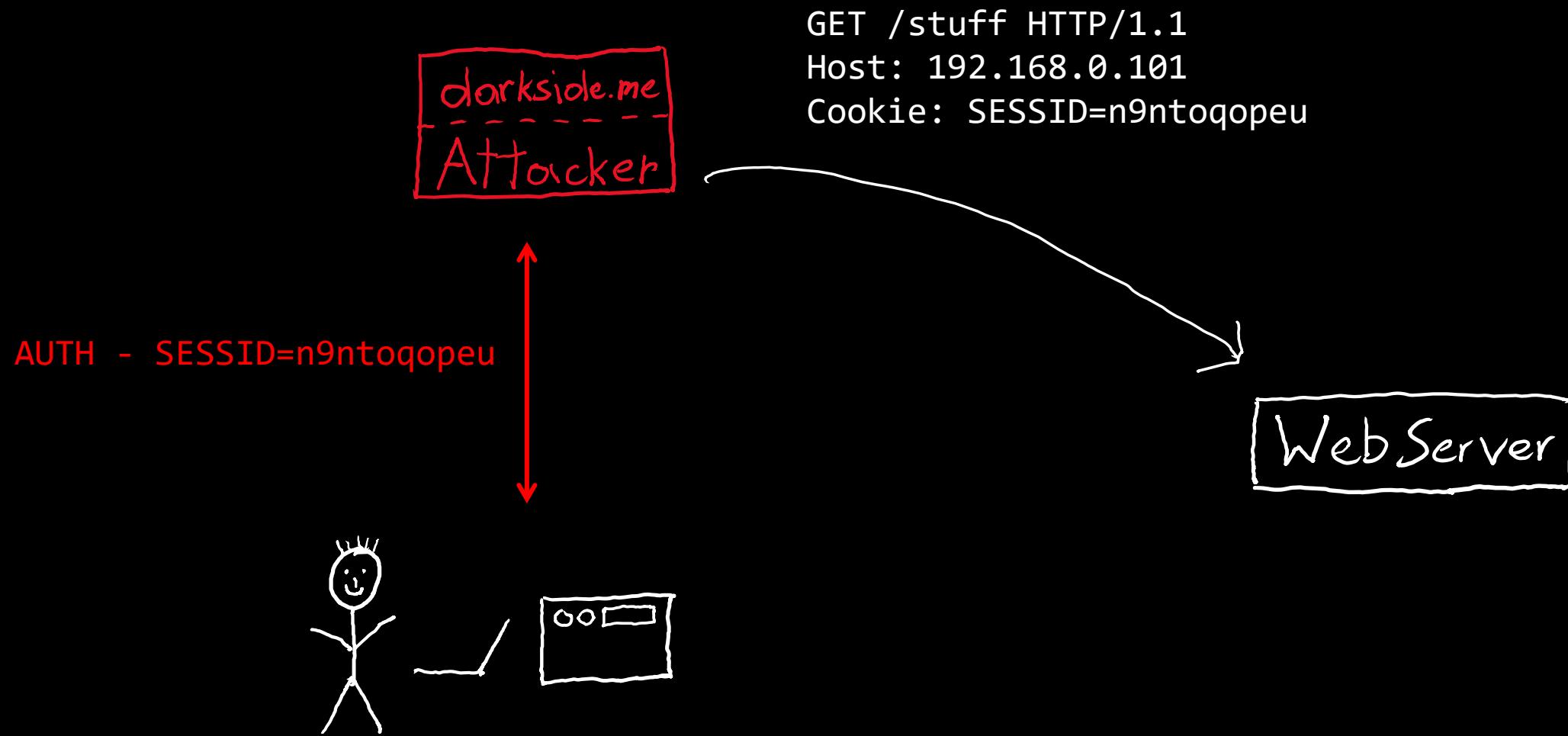
Quite common several years ago, but sometimes developers design custom authentication-schemes which have similar problems

Solution

OWASP Top 10

(Primary)  
Violated Principle







# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

This only works in specific scenarios with very specific circumstances  
application also assigns SID to anonymous users (before login)  
attacker can retrieve this SID from the user  
e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.  
or attacker is able to set a SID in the user's browser  
e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.

application doesn't change SID after successful authentication  
SID known by the attacker is now authenticated

Quite common several years ago, but sometimes developers design custom authentication-schemes which have similar problems

Solution

Change SID after login (or after every change of privilege level)

OWASP Top 10

(Primary)  
Violated Principle



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

This only works in specific scenarios with very specific circumstances  
application also assigns SID to anonymous users (before login)  
attacker can retrieve this SID from the user  
e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.  
or attacker is able to set a SID in the user's browser  
e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.

application doesn't change SID after successful authentication  
SID known by the attacker is now authenticated

Quite common several years ago, but sometimes developers design custom authentication-schemes which have similar problems

Solution

Change SID after login (or after every change of privilege level)

OWASP Top 10

A07:2021-Identification and Authentication Failures

(Primary)  
Violated Principle



# Session Fixation

Goal

User authenticates a SID which is already known by the attacker, so the attacker can hijack the session

How

This only works in specific scenarios with very specific circumstances

application also assigns SID to anonymous users (before login)  
attacker can retrieve this SID from the user  
e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.  
or attacker is able to set a SID in the user's browser  
e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.

application doesn't change SID after successful authentication  
SID known by the attacker is now authenticated

Quite common several years ago, but sometimes developers design custom authentication-schemes which have similar problems

Solution

Change SID after login (or after every change of privilege level)

OWASP Top 10

A07:2021-Identification and Authentication Failures

(Primary)  
Violated Principle

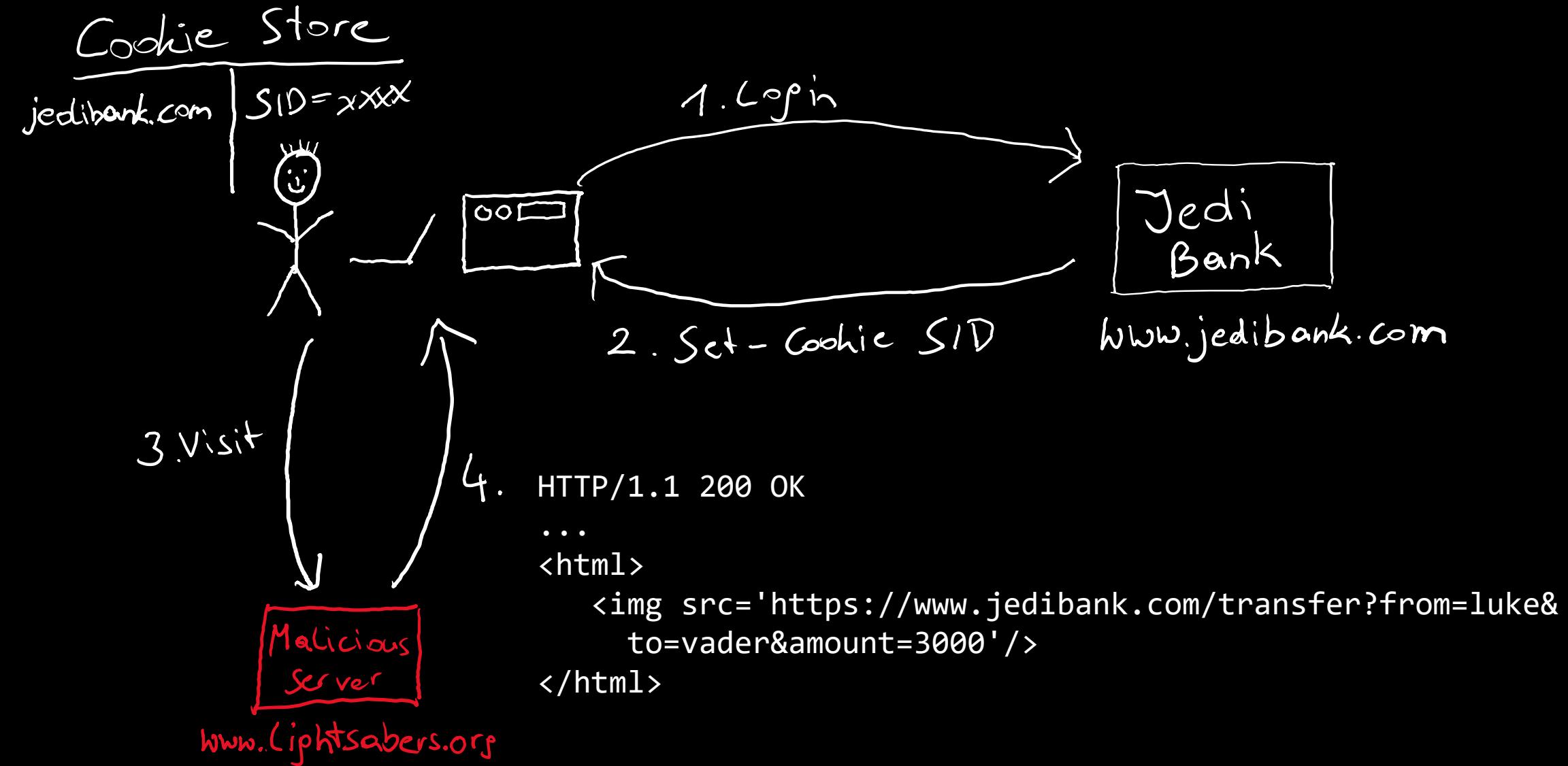
“Identify sensitive data and how they should be handled.”

so the cookie is sent to the server at every subsequent request...

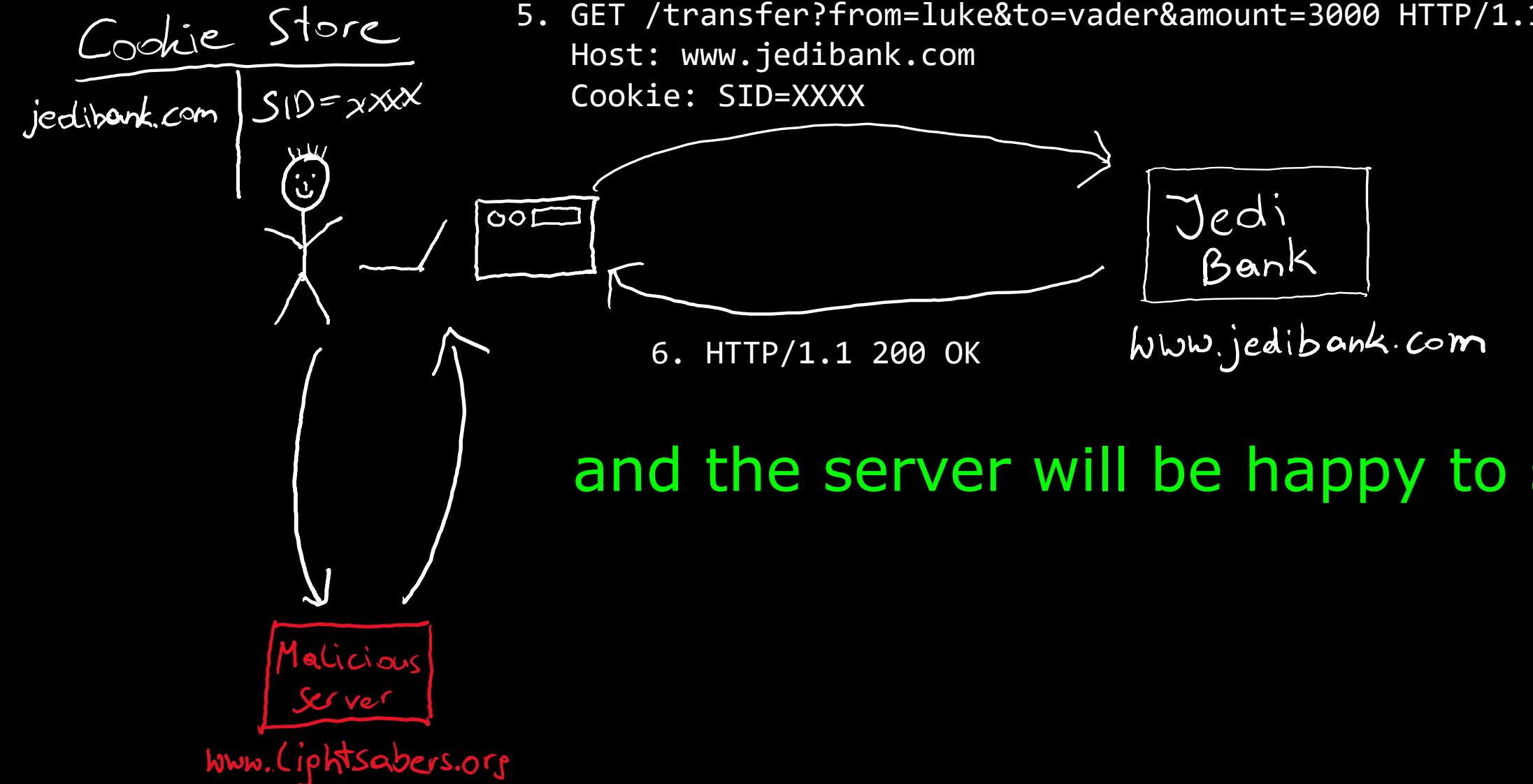
can this be a problem?



# what will the user's browser do?



he will just send the request as he is told...



and the server will be happy to serve

Shouldn't GET  
requests be  
read-only?

# Basic HTML Form

From:

To:

Amount:

POST /transfer HTTP/1.1  
Host: www.jedibank.com  
Cookie: SID=XXXX

from=Luke&to=Han+Solo&amount=40000

```
1 <html>
2   <body>
3     <form method="POST" action="/transfer">
4       From: <input type="text" name="from"/> <br><br>
5       To: <input type="text" name="to"/> <br><br>
6       Amount: <input type="text" name="amount"/><br><br>
7       <input type="submit" value="transfer" />
8     </form>
9   </body>
10 </html>
```

# More realistic attack code

```
csrf2.html > html
1  <html>
2    <head>
3      <title>CSRF 2</title>
4    </head>
5    <body>
6      Nice lightsaber bro...
7      <img src='https://www.jedipedia.net/w/images/e/e6/Darth_Maul%27s_Lightsaber.jpg'>
8      <iframe src='./csrf3.html' style='display: none;'></iframe>
9    </body>
10   </html>
```

```
csrf3.html > html
1  <html>
2    <head>
3      <title>CSRF 3</title>
4    </head>
5    <body>
6      <form method="POST" action="http://www.jedibank.com/transfer">
7        <input name="from" value="luke" />
8        <input name="to" value="vader" />
9        <input name="amount" value="3000" />
10       <input type="submit" value="do it" />
11     </form>
12     <script>
13       document.forms[0].submit()
14     </script>
15   </body>
16   </html>
```



Nice lightsaber bro...

POST /transfer HTTP/1.1  
Host: www.jedibank.com  
Cookie: SID=XXXX  
...  
from=luke&to=vader&amount=3000

# Also possible with JavaScript

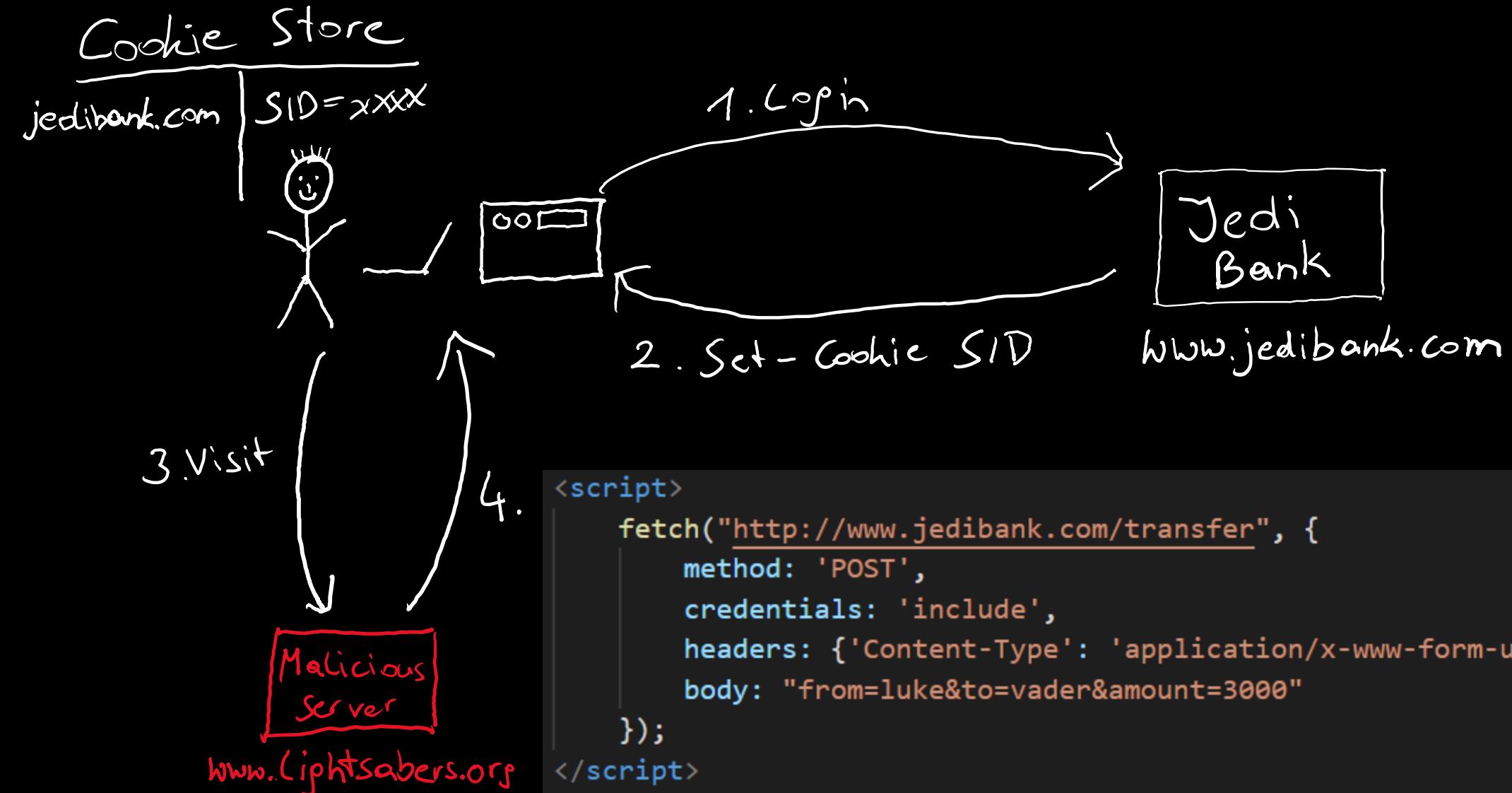
```
1 <html>
2 <head>
3   <title>CSRF 3</title>
4 </head>
5 <body>
6   Nice lightsaber bro...
7   <img src='https://www.jedipedia.net/w/images/e/e6/Darth_Maul%27s_Lightsaber.jpg'>
8 <script>
9   fetch("http://www.jedibank.com/transfer", {
10     method: 'POST',
11     credentials: 'include',
12     headers: {'Content-Type': 'application/x-www-form-urlencoded'},
13     body: "from=luke&to=vader&amount=3000"
14   });
15 </script>
16 </body>
17 </html>
```



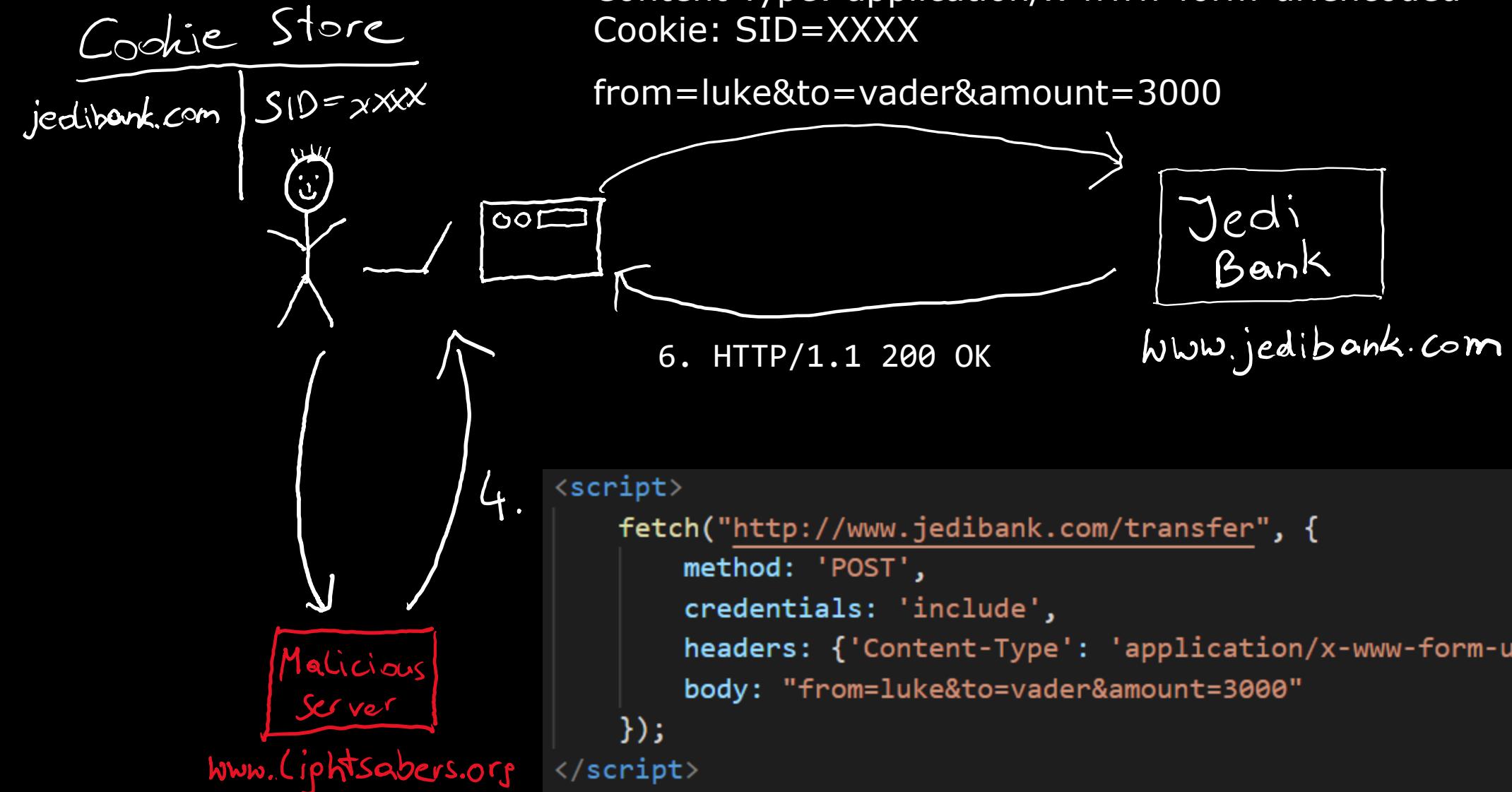
Nice lightsaber bro...

POST /transfer HTTP/1.1  
Host: www.jedibank.com  
Cookie: SID=XXXX  
...

from=luke&to=vader&amount=3000

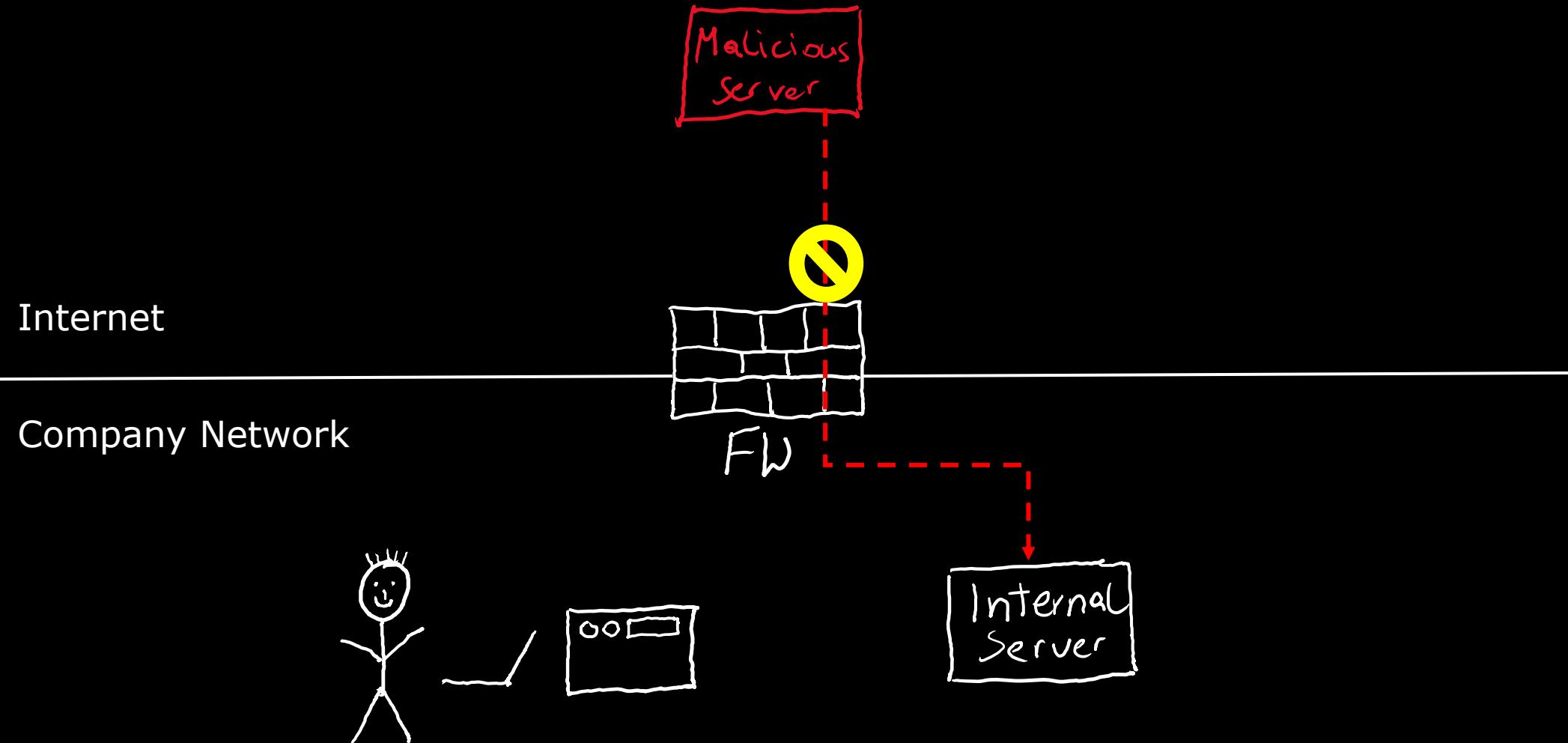


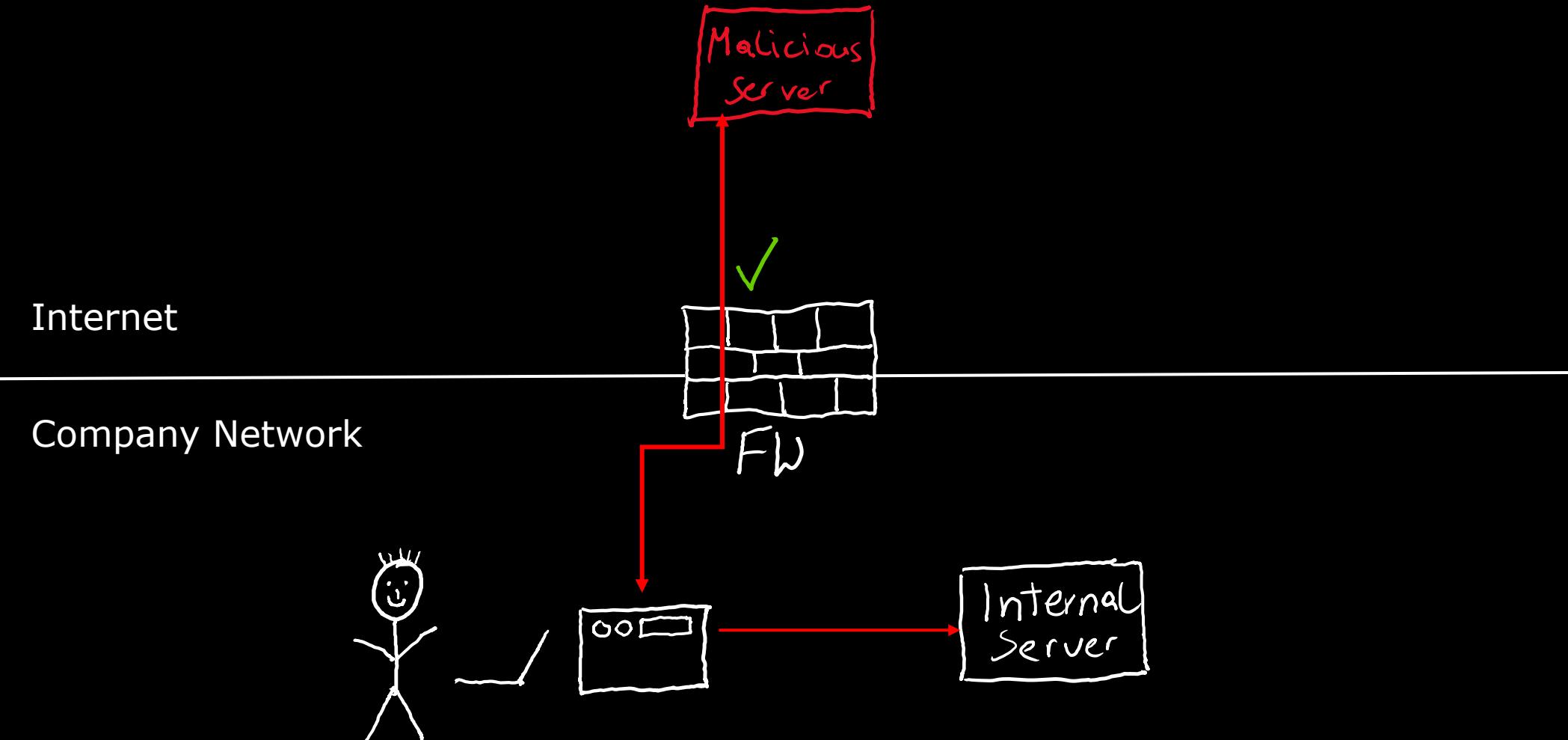
5. POST /transfer HTTP/1.1  
Host: www.jedibank.com  
Content-Type: application/x-www-form-urlencoded  
Cookie: SID=XXXX  
  
from=luke&to=vader&amount=3000



So you can either  
attack the user...

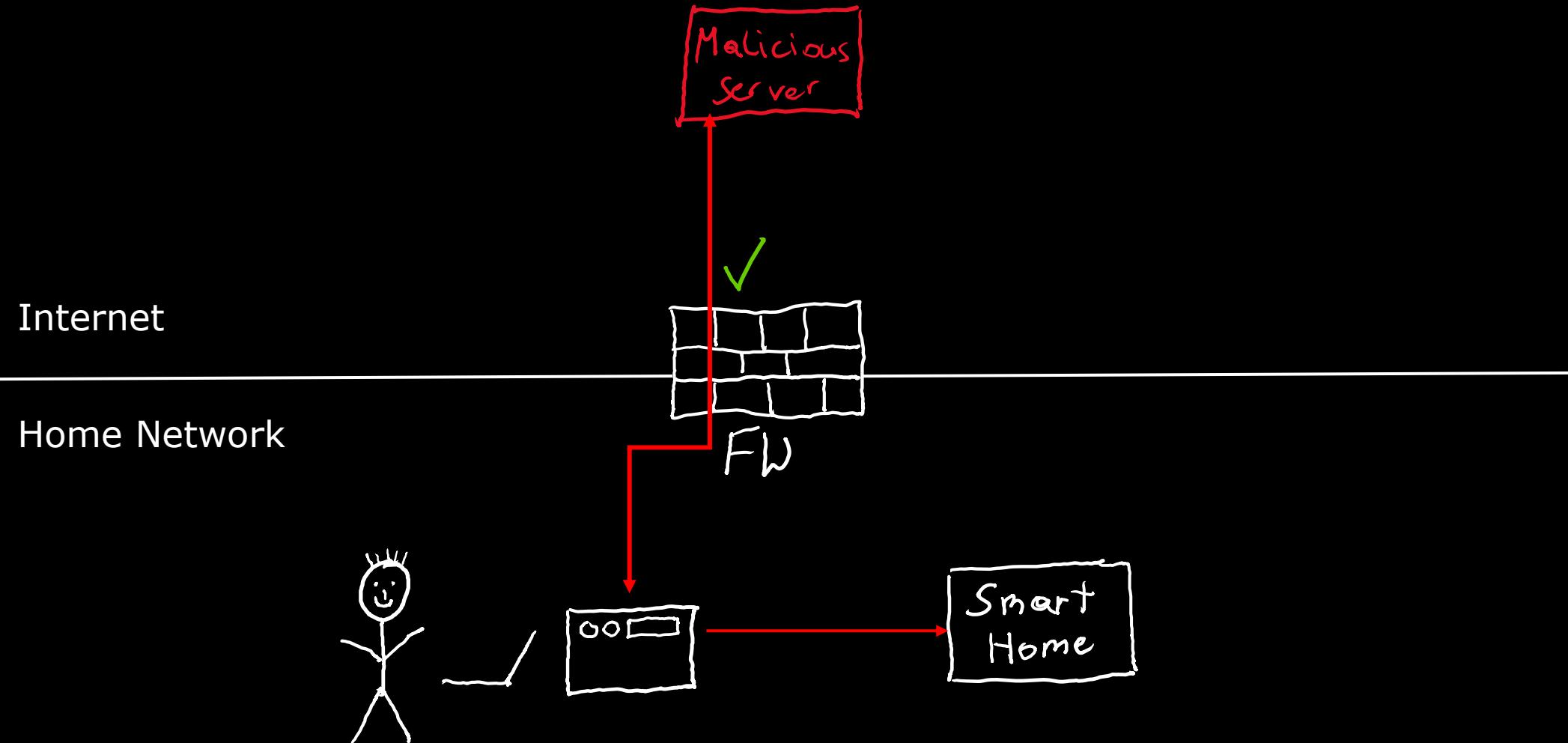
...or use him as a  
„jumphost“

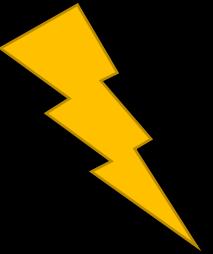




<img src='https://internalserver.intra:3000/somenastystuff' />

especially bad if user is admin...





# Cross-Site Request Forgery (CSRF)

Goal

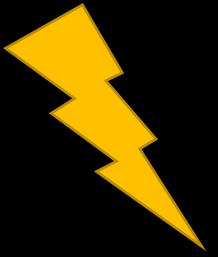
Force user's browser to execute actions in the user's context but without the user's knowledge

How

Solution

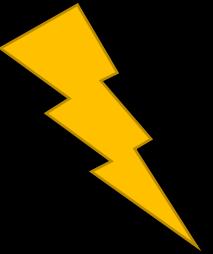
OWASP Top 10

(Primary)  
Violated Principle



# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	
OWASP Top 10	
(Primary) Violated Principle	

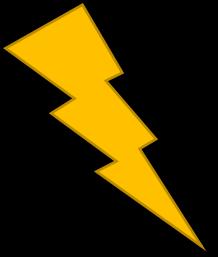


# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	Synchronizer Token Pattern Server adds a random token to every form via hidden form field and checks this token in the user's subsequent request
OWASP Top 10	
(Primary) Violated Principle	

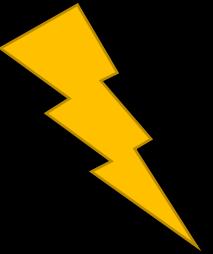
```
1 <html>
2   <body>
3     <form method="POST" action="/transfer">
4       From: <input type="text" name="from"/> <br><br>
5       To: <input type="text" name="to"/> <br><br>
6       Amount: <input type="text" name="amount"/><br><br>
7       <input type="submit" value="transfer" />
8     </form>
9   </body>
10 </html>
```

```
1 <html>
2   <body>
3     <form method="POST" action="/transfer">
4       From: <input type="text" name="from"/> <br><br>
5       To: <input type="text" name="to"/> <br><br>
6       Amount: <input type="text" name="amount"/><br><br>
7       <input type="submit" value="transfer" />
8       <input type='hidden' name='csrf_token' value='bed35f2ad140542241dae8160fe9fff3' />
9     </form>
10   </body>
11 </html>
```



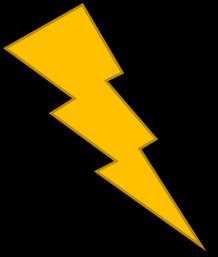
# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	Synchronizer Token Pattern Server adds a random token to every form via hidden form field and checks this token in the user's subsequent request
OWASP Top 10	
(Primary) Violated Principle	



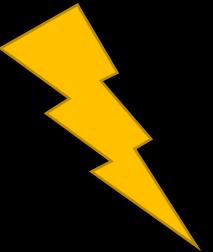
# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	<p><b>Synchronizer Token Pattern</b> Server adds a random token to every form via hidden form field and checks this token in the user's subsequent request</p> <p><b>Double Submit Pattern</b> Server sets a random token as browser cookie and client-side application adds it to every form/header. Server just needs to compare cookie and form-field/header. Actually works but has some pitfalls and is therefore not considered 100% secure</p>
OWASP Top 10	
(Primary) Violated Principle	



# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	<p><b>Synchronizer Token Pattern</b> Server adds a random token to every form via hidden form field and checks this token in the user's subsequent request</p> <p><b>Double Submit Pattern</b> Server sets a random token as browser cookie and client-side application adds it to every form/header. Server just needs to compare cookie and form-field/header. Actually works but has some pitfalls and is therefore not considered 100% secure</p>
OWASP Top 10	<p><b>Same-Site Flag</b> Recommended: SameSite=Lax</p>
(Primary) Violated Principle	



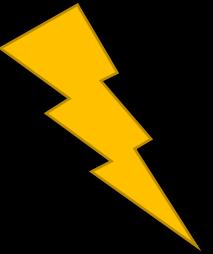
# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	<p><b>Synchronizer Token Pattern</b> Server adds a random token to every form via hidden form field and checks this token in the user's subsequent request</p> <p><b>Double Submit Pattern</b> Server sets a random token as browser cookie and client-side application adds it to every form/header. Server just needs to compare cookie and form-field/header. Actually works but has some pitfalls and is therefore not considered 100% secure</p>
OWASP Top 10	<p><b>Same-Site Flag</b> Recommended: SameSite=Lax</p> <p><b>Complex Requests</b> e.g. if server only accepts special content type (e.g. JSON), custom headers, etc.</p>
(Primary) Violated Principle	

# Complex Requests as CSRF Protection

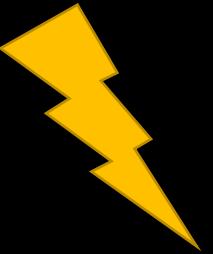
- only works if you really check the content-type on the server side (e.g. application/json)
- otherwise you can fake it:

```
<form method="POST" enctype="text/plain">  
    <input type="hidden" name='{"title": "' value='...', "content": "..."}'>  
</form>
```



# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	<p><b>Synchronizer Token Pattern</b> Server adds a random token to every form via hidden form field and checks this token in the user's subsequent request</p> <p><b>Double Submit Pattern</b> Server sets a random token as browser cookie and client-side application adds it to every form/header. Server just needs to compare cookie and form-field/header. Actually works but has some pitfalls and is therefore not considered 100% secure</p>
OWASP Top 10	Was kicked from Top 10 2017
(Primary) Violated Principle	



# Cross-Site Request Forgery (CSRF)

Goal	Force user's browser to execute actions in the user's context but without the user's knowledge
How	Slip the user some html forms and abuse the fact that session tokens get appended automatically e.g. by phishing, forums, etc.
Solution	<p><b>Synchronizer Token Pattern</b> Server adds a random token to every form via hidden form field and checks this token in the user's subsequent request</p> <p><b>Double Submit Pattern</b> Server sets a random token as browser cookie and client-side application adds it to every form/header. Server just needs to compare cookie and form-field/header. Actually works but has some pitfalls and is therefore not considered 100% secure</p>
OWASP Top 10	Was kicked from Top 10 2017
(Primary) Violated Principle	„Earn or give, but never assume, trust.“

# Similar: JSONP & XSS

- Unfortunately not enough time at this class
- Recommended reading:
  - [https://www.w3schools.com/js/js\\_json\\_jsonp.asp](https://www.w3schools.com/js/js_json_jsonp.asp)
  - <https://www.sjoerdlangkemper.nl/2019/01/02/jsonp/>
  - <https://web.stanford.edu/class/cs253/lectures/Lecture%2005.pdf>
  - <https://www.scip.ch/en/?labs.20160414>
  - <https://medium.com/bugbountywriteup/effortlessly-finding-cross-site-script-inclusion-xssi-jsonp-for-bug-bounty-38ae0b9e5c8a>

# So we can send requests from the user's browser...

```
<body>
|   <img src='http://www.jedibank.com/transfer?from=luke&to=vader&amount=3000' />
</body>
```

```
<script>
|   fetch("http://www.jedibank.com/transfer", {
|       method: 'POST',
|       credentials: 'include',
|       headers: {'Content-Type': 'application/x-www-form-urlencoded'},
|       body: "from=luke&to=vader&amount=3000"
|   });
</script>
```

## What about the responses?



```
soreq.html > html
1  <html>
2    <head><title>Dark Side</title></head>
3    <body>
4      <h1>Welcome to the dark side!</h1>
5      <script>
6        var req = new XMLHttpRequest()
7        req.open("GET", "http://darkside.me/darkcontent.txt", false);
8        req.send()
9        alert(req.responseText);
10       </script>
11     </body>
12   </html>
```



```
soreq.html > html
1  <html>
2    <head><title>Dark Side</title></head>
3    <body>
4      <h1>Welcome to the dark side!</h1>
5      <script>
6        var req = new XMLHttpRequest()
7        req.open("GET", "http://darkside.me/darkcontent.txt", false);
8        req.send()
9        alert(req.responseText);
10       </script>
11     </body>
12   </html>
```



```
coreq.html > html
1  <html>
2    <head><title>Dark Side</title></head>
3    <body>
4      <h1>Welcome to the dark side!</h1>
5      <script>
6        var req = new XMLHttpRequest()
7        req.open("GET", "http://lightside.me/lightcontent.txt", false);
8        req.send()
9        alert(req.responseText);
10       </script>
11     </body>
12   </html>
```

darksiole.me

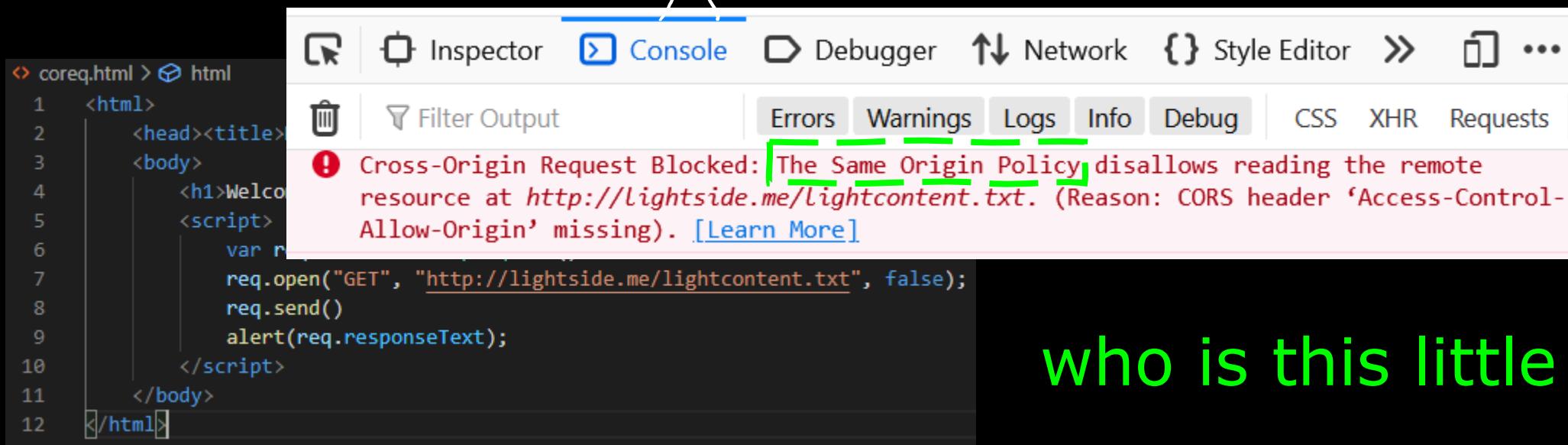
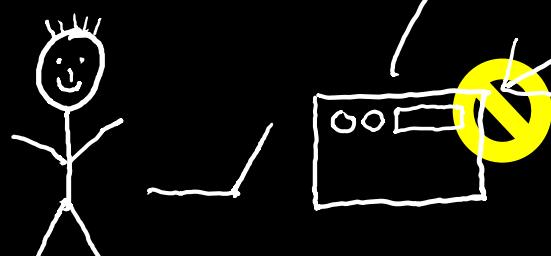
lightside.me

GET /lightcontent.txt HTTP/1.1  
Host: lightside.me

HTTP/1.1 200 OK

...

Hey there, this message comes from  
the light side.



coreq.html > html

```
1 <html>
2   <head><title>
3     <body>
4       <h1>Welco
5       <script>
6         var r
7           req.open("GET", "http://lightside.me/lightcontent.txt", false);
8           req.send()
9           alert(req.responseText);
10      </script>
11    </body>
12  </html>
```

Console Network Style Editor ...

Filter Output Errors Warnings Logs Info Debug CSS XHR Requests

! Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <http://lightside.me/lightcontent.txt>. (Reason: CORS header 'Access-Control-Allow-Origin' missing). [Learn More]

who is this little helper?

# Same Origin Policy

- The web's basic security model
  - Basic idea:
    - Code from one origin (`darkside.me`) is not allowed to interfere with resources of another origin (`lightside.me`)
  - Origin = Protocol + Host + Port
    - `http://darkside.me:80`
    - `http://lightside.me:80`

# Same Origin Policy - Exceptions

- HTML elements & Cookies
  - That's why CSRF, JSONP/XSSI, etc. work
    - Embedded static resources
      - `<img src = ..." />`
      - `<script src = ..."></script>`
      - `<link rel="stylesheet" href="..." />`
    - Embedding iframes
      - but not read/tamper the content
    - Forms (GET & POST requests)
      - but not read the answer
  - Explicit opt-out
    - e.g. `document.domain` and `postMessage` API
    - for more details see:
      - <https://web.stanford.edu/class/cs253/lectures/Lecture%2004.pdf>

# Allowed or not allowed?

http://darkside.me/req -> http://darkside.me/message ✓

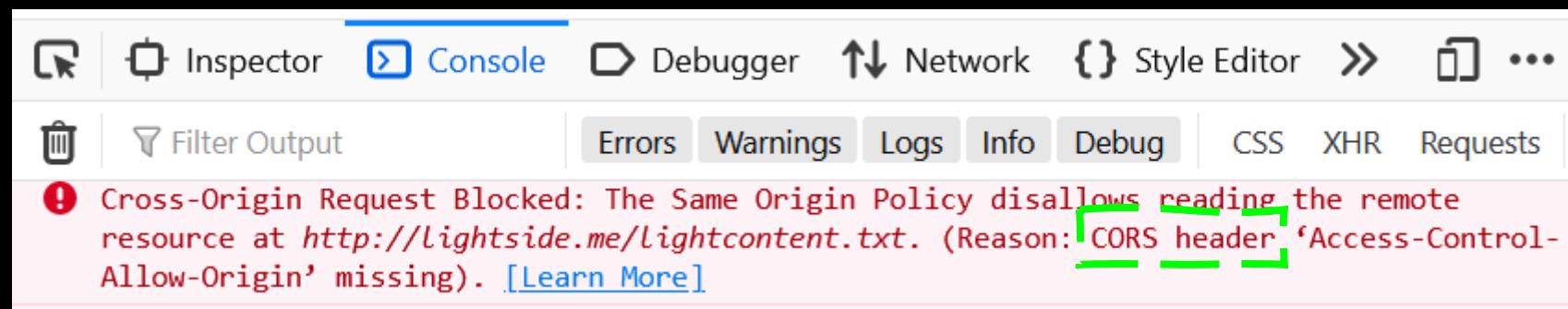
http://darkside.me/req -> http://darkside.me:80/message ✓

http://darkside.me/req -> http://darkside.me:8080/message ✗

http://darkside.me/req -> https /darkside.me/message ✗

http://darkside.me/req -> http://www.darkside.me/message ✗

http://darkside.me/req -> http://lightside.me/message ✗



# Cross Origin Resource Sharing (CORS)

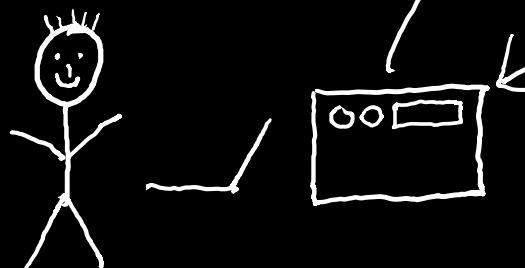
- Sometimes applications just really want to communicate with other origins
- If the other origin is fine with it, why not...
- Just define a CORS header
  - To allow a specific origin to read data
    - Access-Control-Allow-Origin: `http://darkside.me`
  - To allow any origin to read data
    - Access-Control-Allow-Origin: `*`



```
④ acoreq.html > ⚒ html
1   <html>
2     <head><title>Dark Side</title></head>
3     <body>
4       <h1>Welcome to the dark side!</h1>
5       <script>
6         var req = new XMLHttpRequest()
7         req.open("GET", "http://lightside.me/allowedContent.php", false);
8         req.send()
9         alert(req.responseText);
10        </script>
11      </body>
12    </html>
```

darkside.me

GET /allowedContent.php HTTP/1.1  
Host: lightside.me

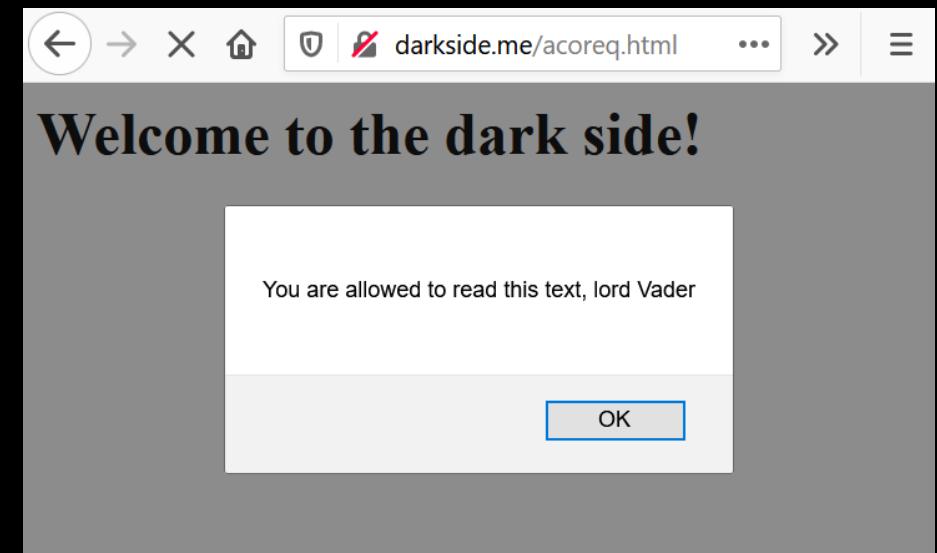


lightside.me

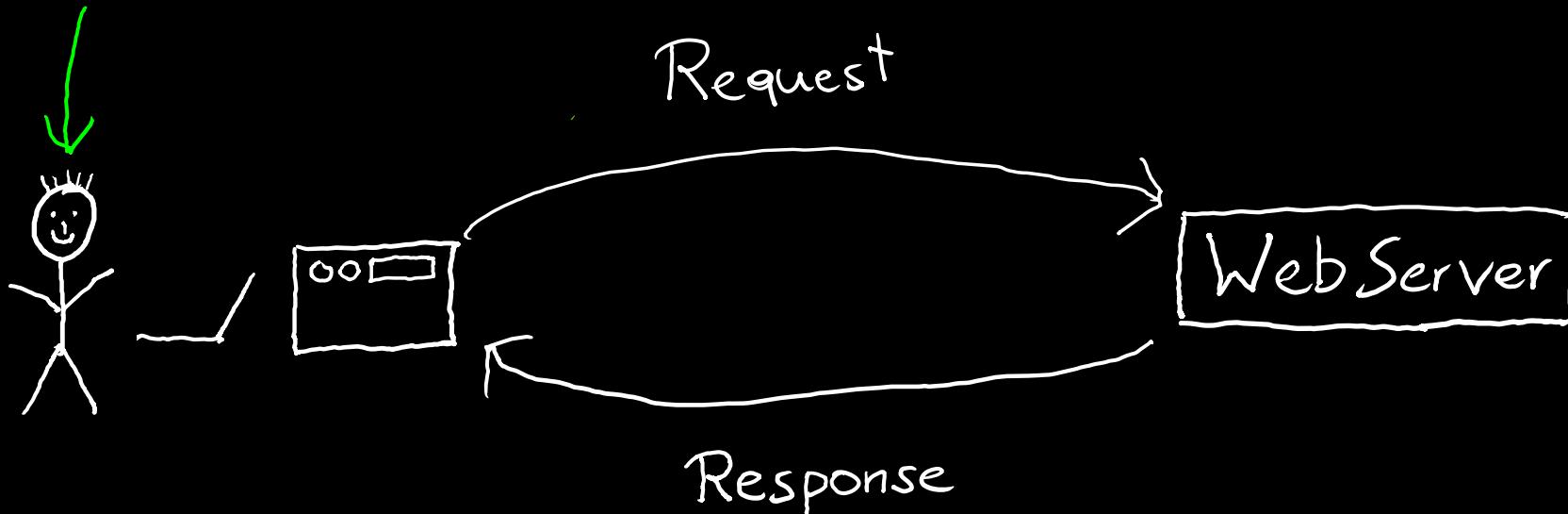
```
allowedContent.php
1 <?php
2 header("Access-Control-Allow-Origin: http://darkside.me");
3 echo("You are allowed to read this text, lord Vader");
4 ?>
```

HTTP/1.1 200 OK  
Access-Control-Allow-Origin: http://darkside.me  
...  
You are allowed to read this text, lord Vader

```
acoreq.html > html
1 <html>
2   <head><title>Dark Side</title></head>
3   <body>
4     <h1>Welcome to the dark side!</h1>
5     <script>
6       var req = new XMLHttpRequest()
7       req.open("GET", "http://lightside.me/allowedContent.php", false);
8       req.send()
9       alert(req.responseText);
10      </script>
11    </body>
12  </html>
```



ok, we now really know who this guy is...



... and we also know how session management works  
but what is he actually allowed to do?

# Key messages

- Use TLS for your whole application
- Store your passwords securely (bcrypt, PBKDF2,...)
- Provide multi factor support and enforce it for critical areas
- Secure your SID
- Never trust the client or anything originating from the client