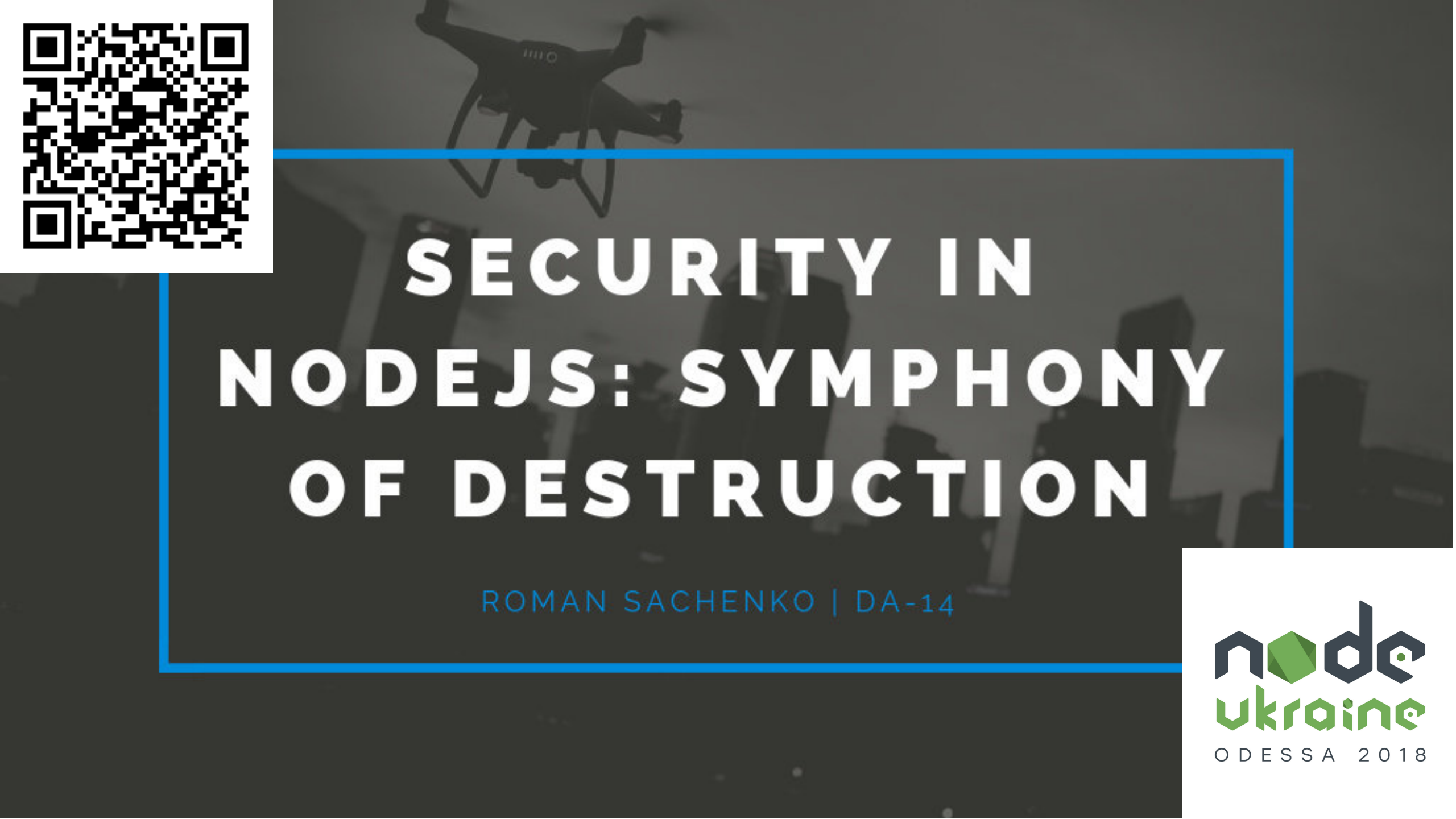
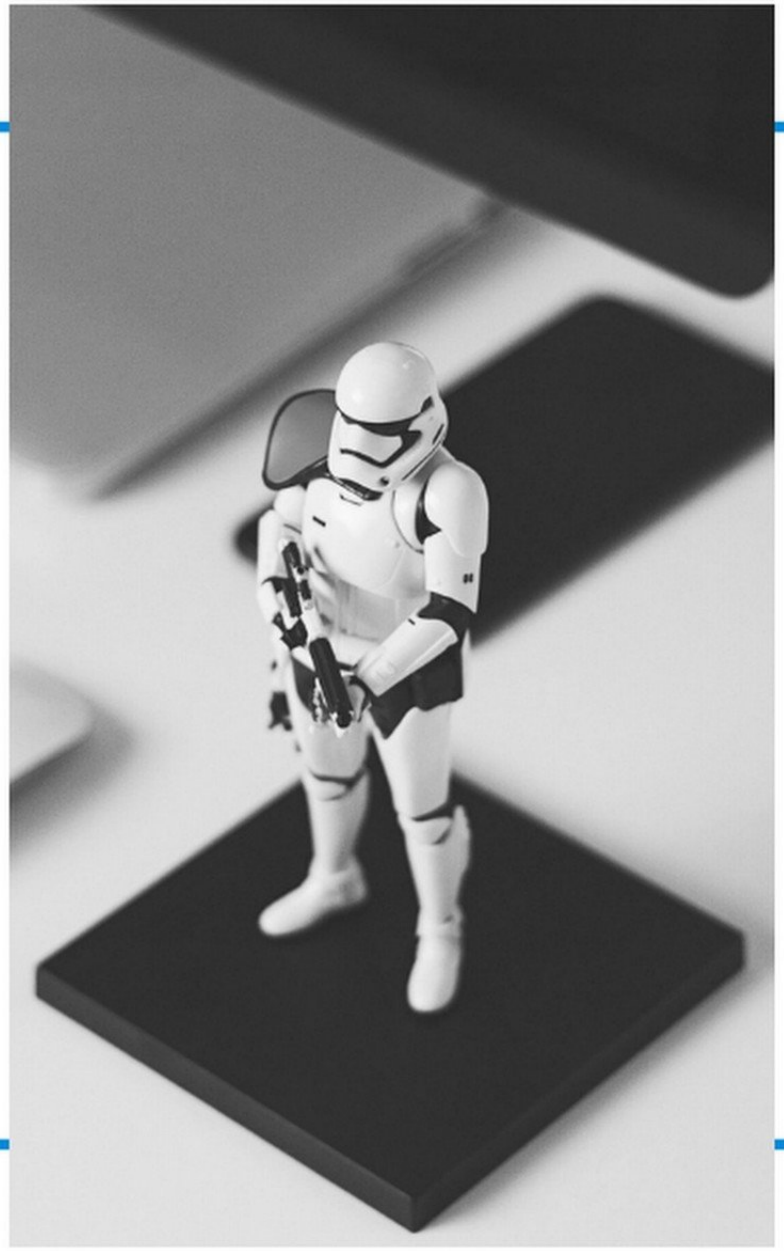


SECURITY IN NODEJS: SYMPHONY OF DESTRUCTION

ROMAN SACHENKO | DA-14

node ukraine
ODESSA 2018



MY BIO

- back-end developer, team lead at DA-14
- 4 years in software development area

CONTENTS

> *It's show time (c) Ben Richards*

Cyber Security - beginning for beginners

- Web Apps Evolution
- Secure App: myth or fact

NodeJS Vulnerabilities

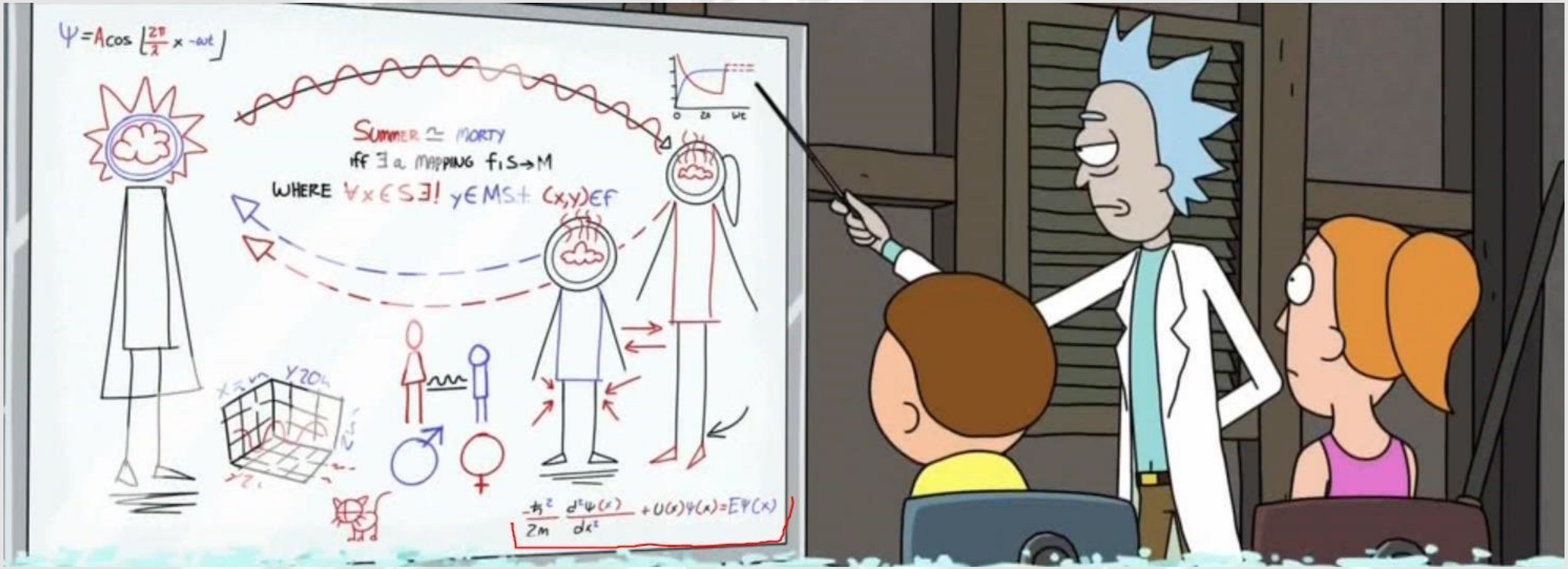
- Brute-Force Attacks
- Database Injections
- Regular Expression DOS
- Memory Leaks
- Hijacking the require chain
- Rainbow Table attack
- Hash Table Collision attack
- Timing attack

Painkiller

- Best Practice
- Helpful Modules

CYBER SECURITY - BEGINNING FOR BEGINNERS

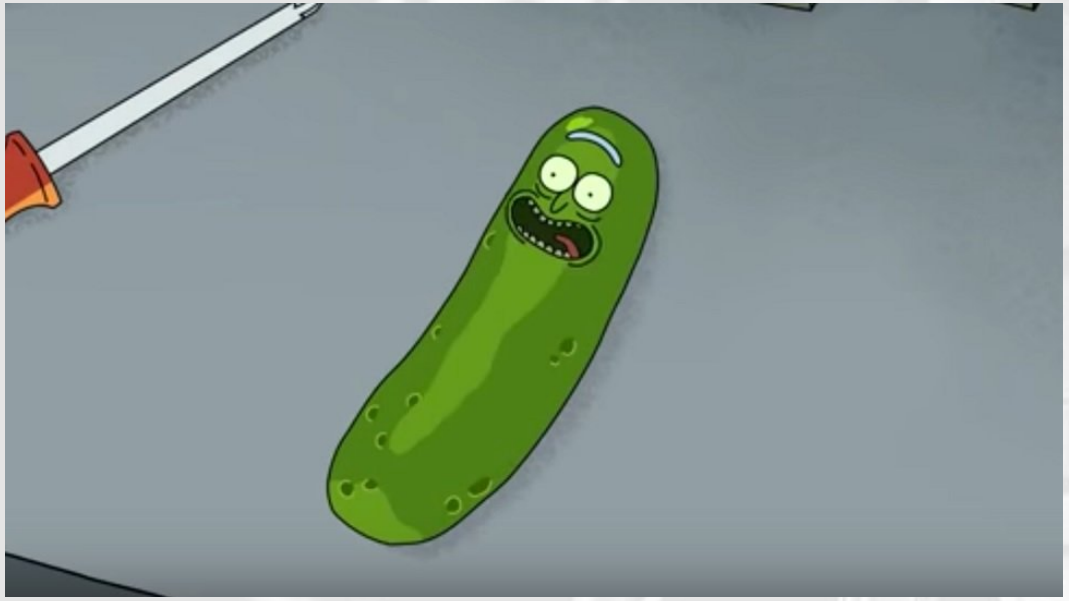
> *Okay, now it's secure enough (c) noone ever*



CYBER SECURITY - BEGINNING FOR BEGINNERS

Application evolution

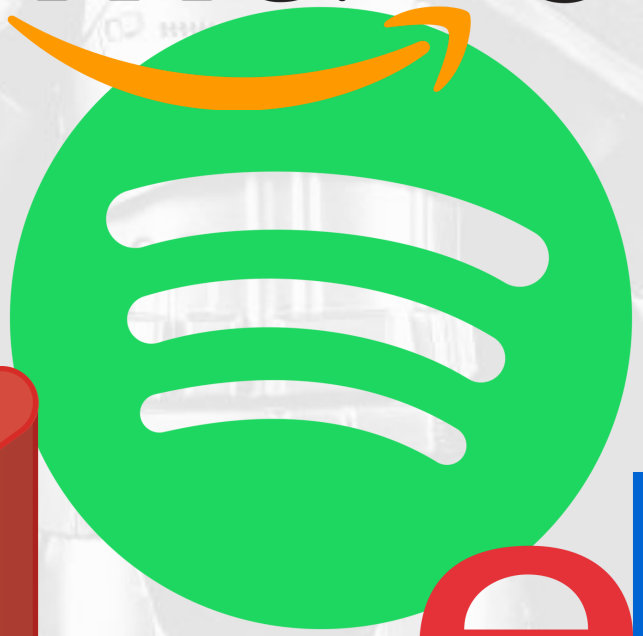
Before



Now

CYBER SECURITY - BEGINNING FOR BEGINNERS

What's now



CYBER SECURITY - BEGINNING FOR BEGINNERS

Is this application secure enough?

- SSL is not the silver bullet

CYBER SECURITY - BEGINNING FOR BEGINNERS

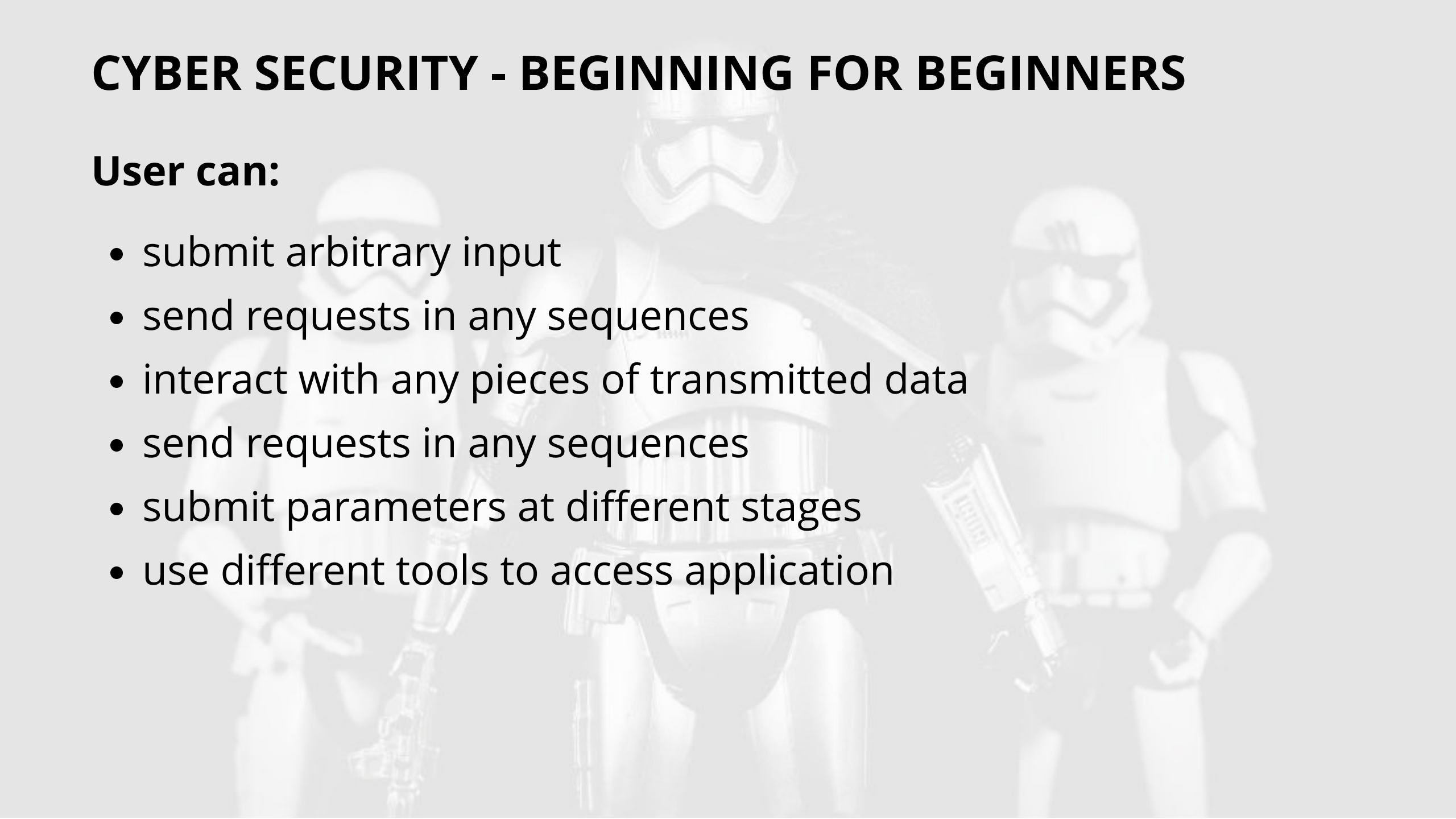
Is this application secure enough?

- Broken authentication
- Broken access controls
- Database injection
- Cross-site scripting
- Information leakage
- Cross-site request forgery

CYBER SECURITY - BEGINNING FOR BEGINNERS

User can:

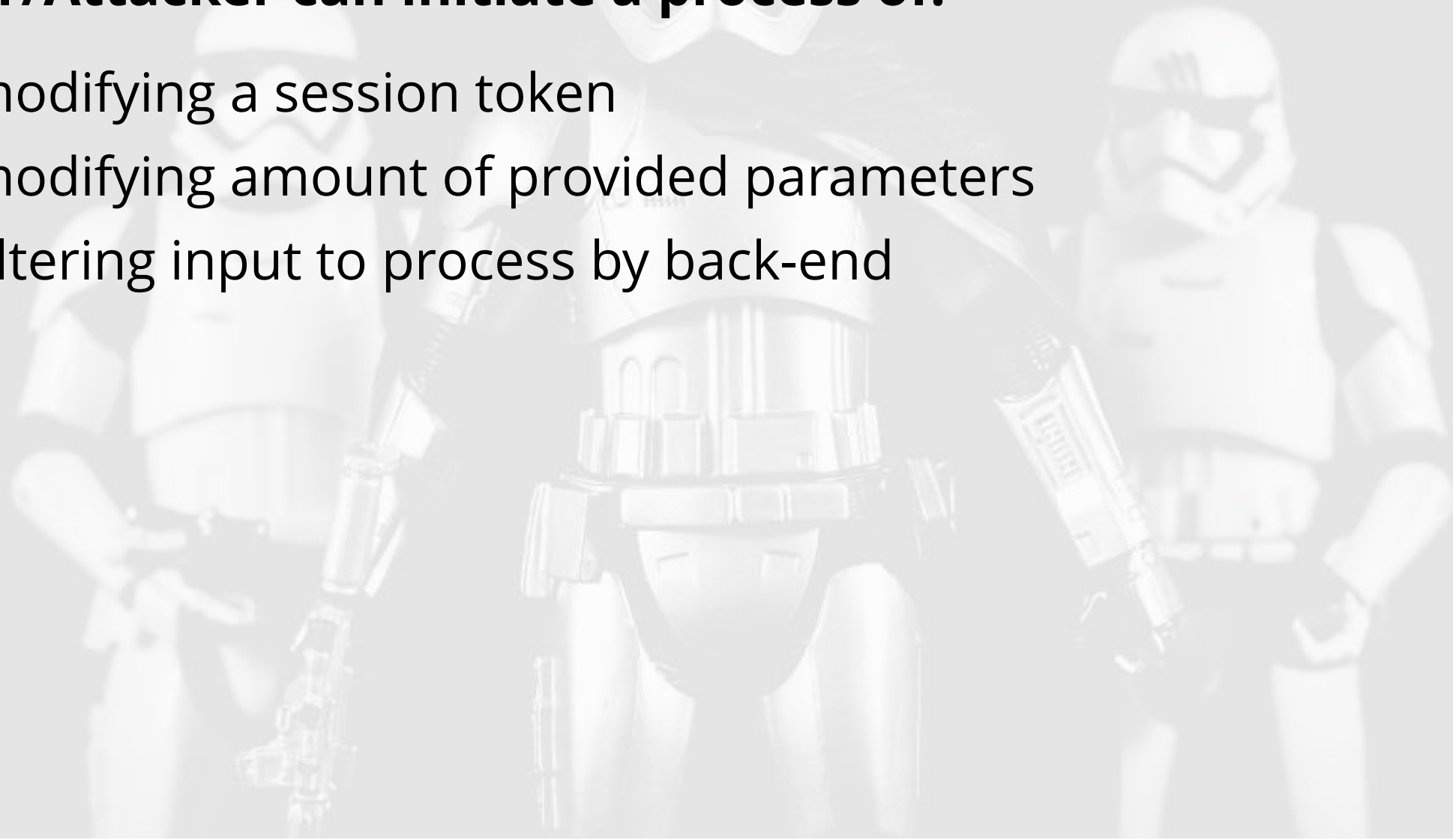
- submit arbitrary input
- send requests in any sequences
- interact with any pieces of transmitted data
- send requests in any sequences
- submit parameters at different stages
- use different tools to access application



CYBER SECURITY - BEGINNING FOR BEGINNERS

User/Attacker can initiate a process of:

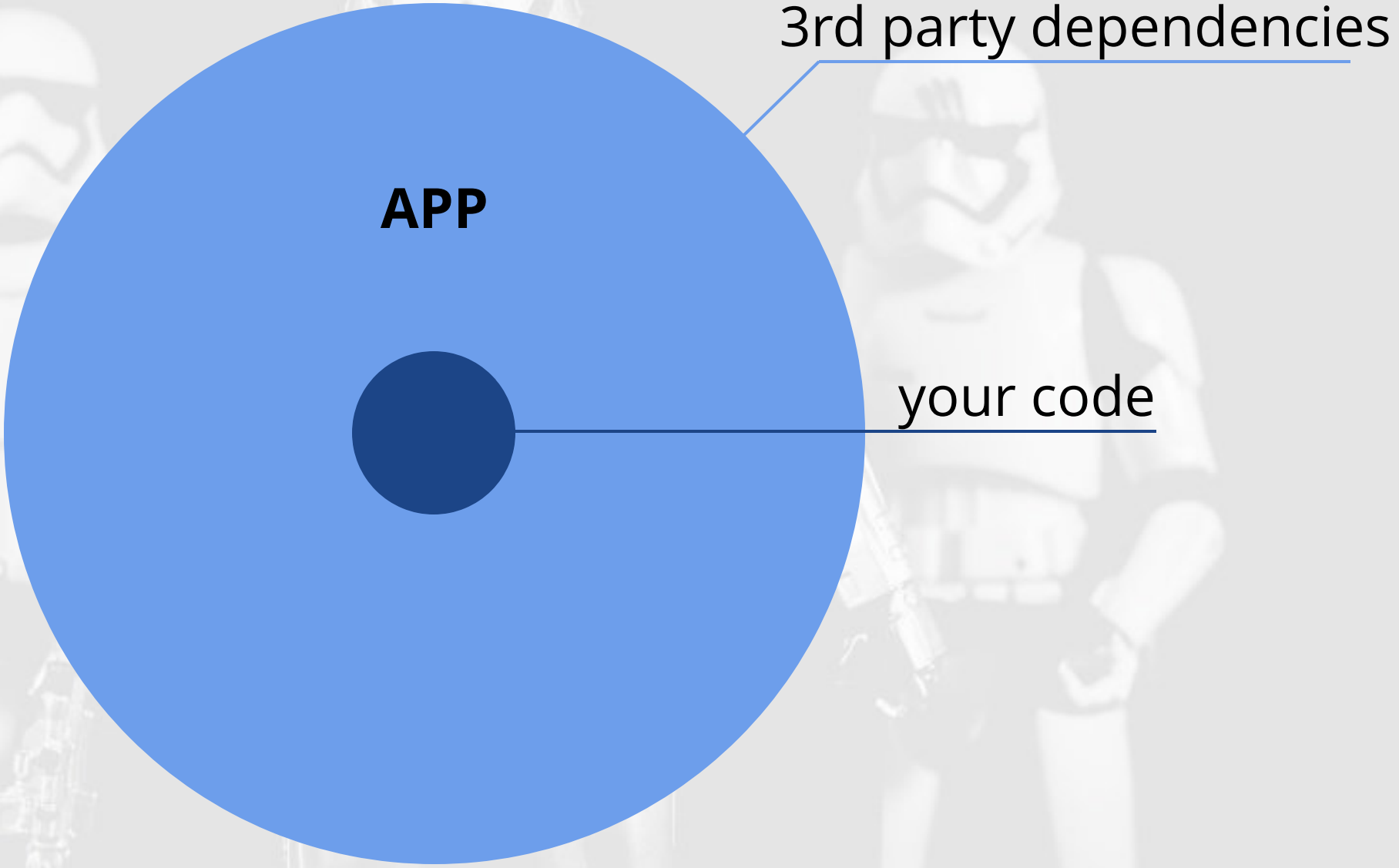
- modifying a session token
- modifying amount of provided parameters
- altering input to process by back-end



BEFORE WE START

> *Good developers code; Best developers copy (c) stack overflow*

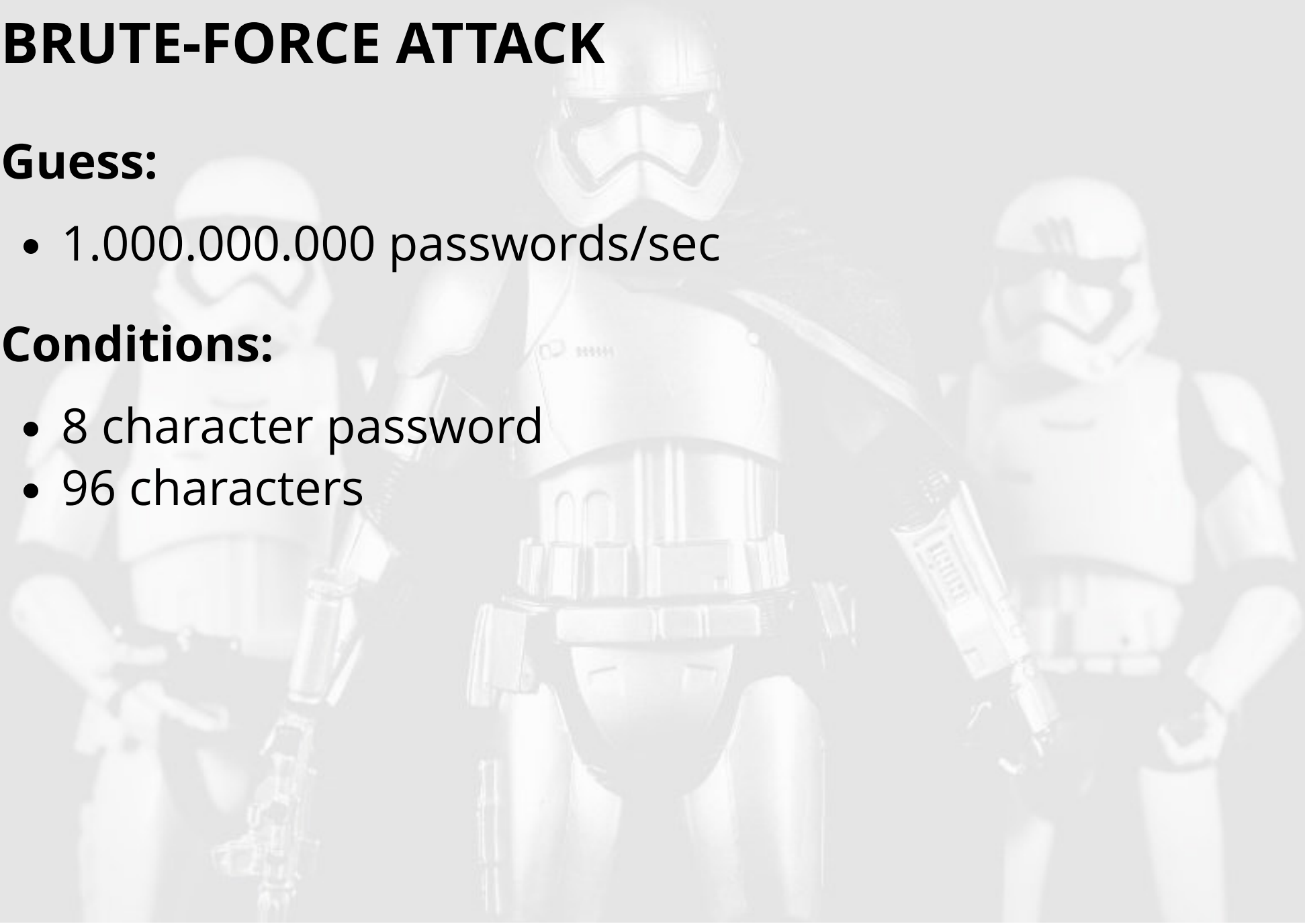
MODERN JAVA SCRIPT APPLICATION



BRUTE-FORCE ATTACK

“ The most guaranteed but time-consuming method to crack a password

BRUTE-FORCE ATTACK



Guess:

- 1.000.000.000 passwords/sec

Conditions:

- 8 character password
- 96 characters

BRUTE-FORCE ATTACK

Guess:

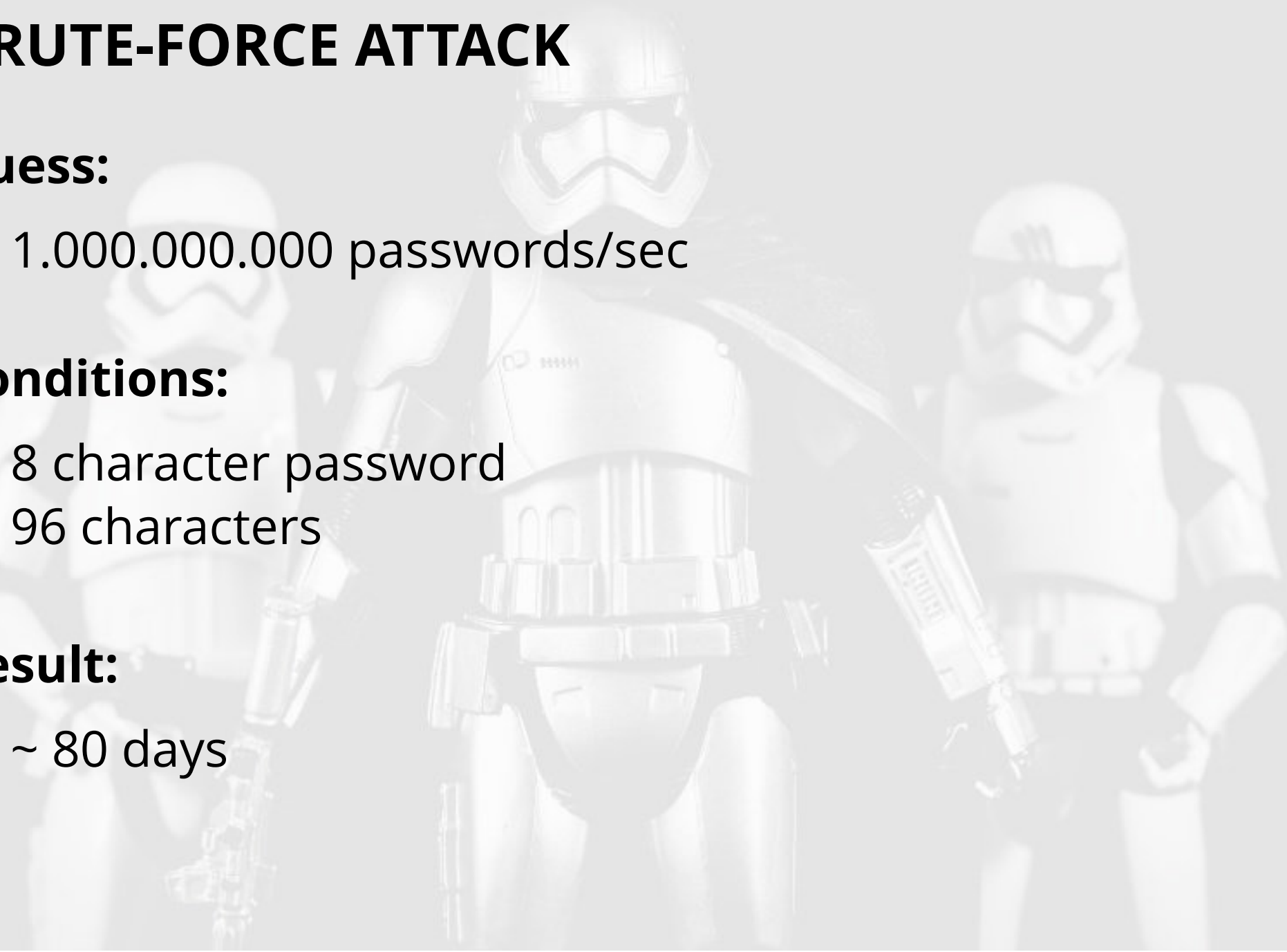
- 1.000.000.000 passwords/sec

Conditions:

- 8 character password
- 96 characters

Result:

- ~ 80 days



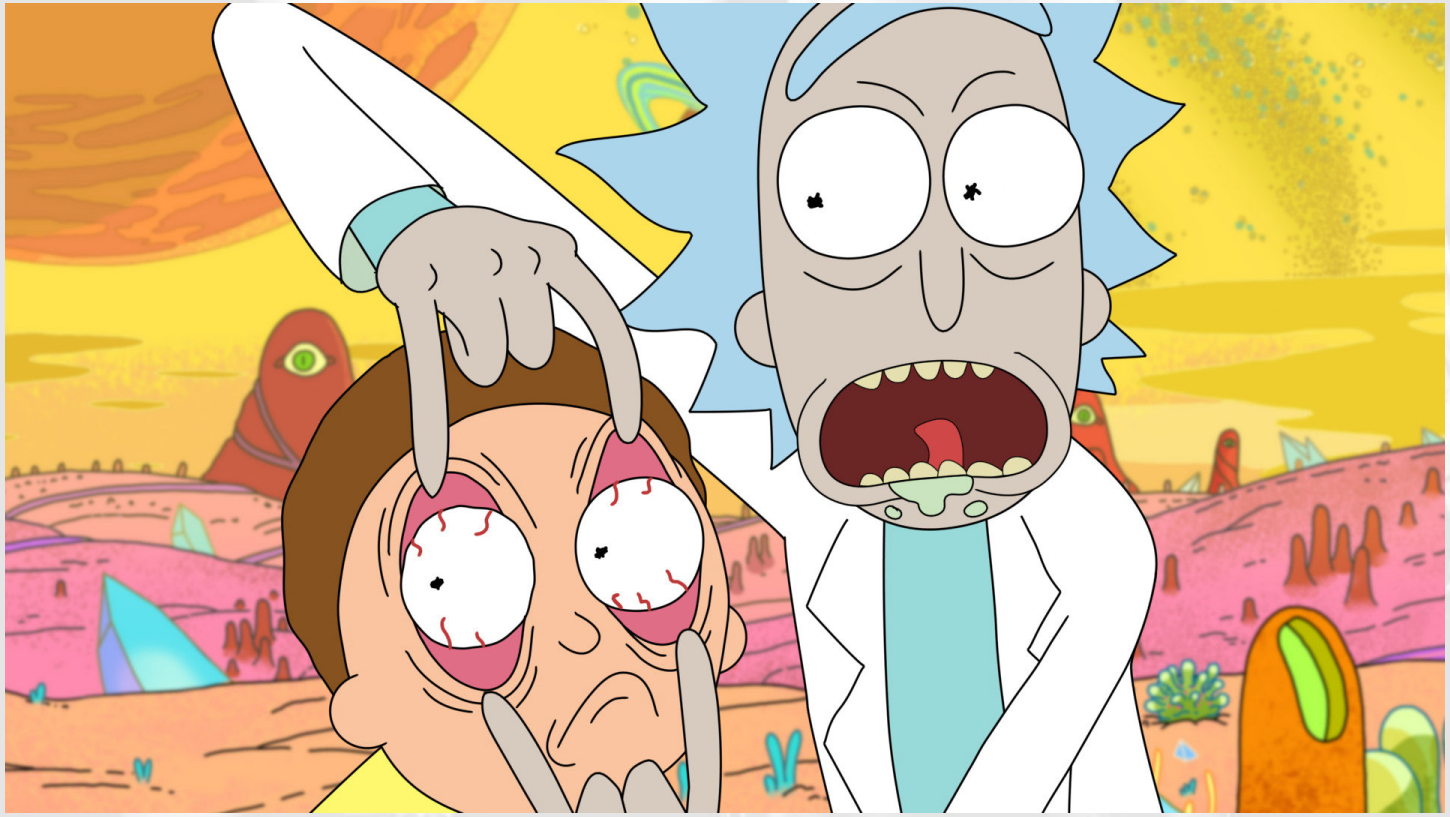
BRUTE-FORCE ATTACK

Guess:

- 400.000.000.000 passwords/sec

Result:

- up to 6 hours



BRUTE-FORCE ATTACK

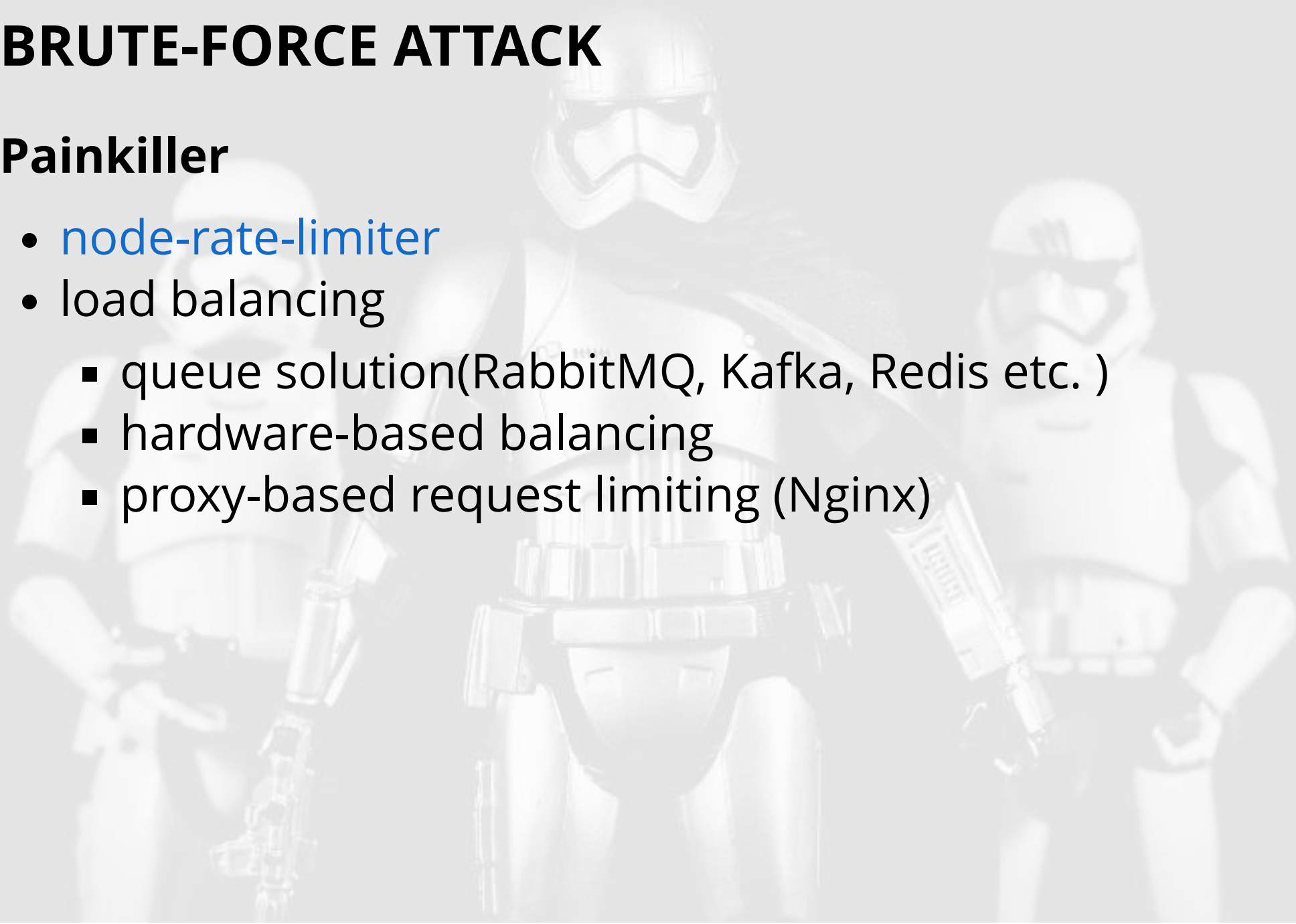
Best Practice:

- Require strong passwords
- Rate-limiting
- Invalid login attempts locker
- Logging
- Secure password recover
  - Requires the old password
  - Secret question
  - Do not reveal the current password

BRUTE-FORCE ATTACK

Painkiller

- node-rate-limiter
- load balancing
  - queue solution(RabbitMQ, Kafka, Redis etc.)
  - hardware-based balancing
  - proxy-based request limiting (Nginx)



DATABASE INJECTIONS

> *DB injection has been around for almost 20 years and is still a big issue*

DATABASE INJECTIONS

Examples:

```
DELETE /users/?id=<userId>

UserModel.remove({ _id: req.query.id });
```

DATABASE INJECTIONS

Examples:

```
DELETE /users/?id=<userId>

UserModel.remove({ _id: req.query.id });
```

```
DELETE /users/?id={ '$exists': true }

UserModel.remove({ _id: { '$exists': true } });
```

DATABASE INJECTIONS

Examples:

```
.find( { $where: "this.<key> == <value>" } );
```

```
.find( { $where: "this.<key> == 'a; sleep(1000000)'" } );
```

DATABASE INJECTIONS

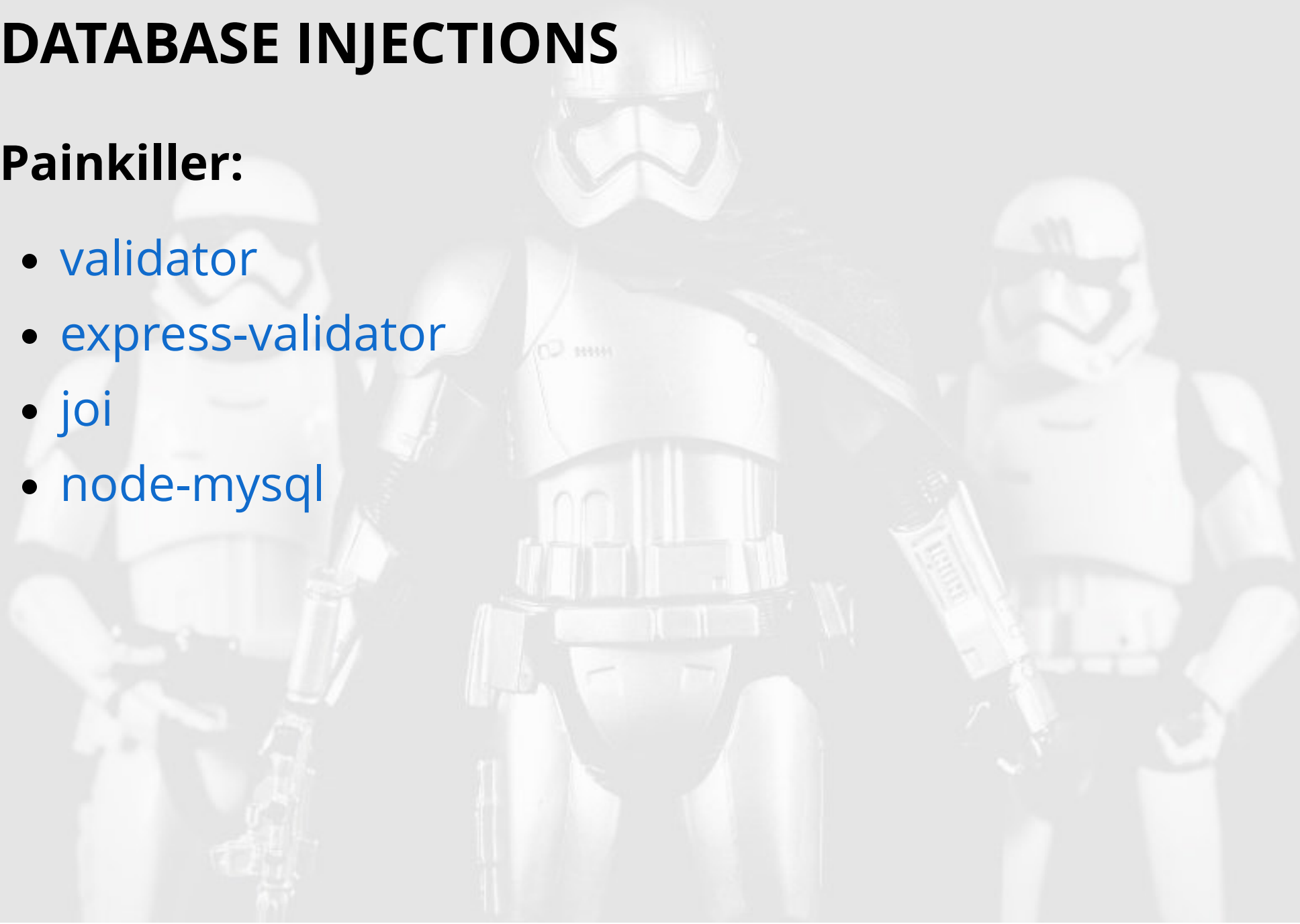
Best Practice:

- validate incoming body and query parameters
- be careful with the executable incoming data
- create strict rules for incoming data
  - validate data type
  - check length
  - use value enumeration if it's possible
- use ORM/ODM for database queries

DATABASE INJECTIONS

Painkiller:

- validator
- express-validator
- joi
- node-mysql

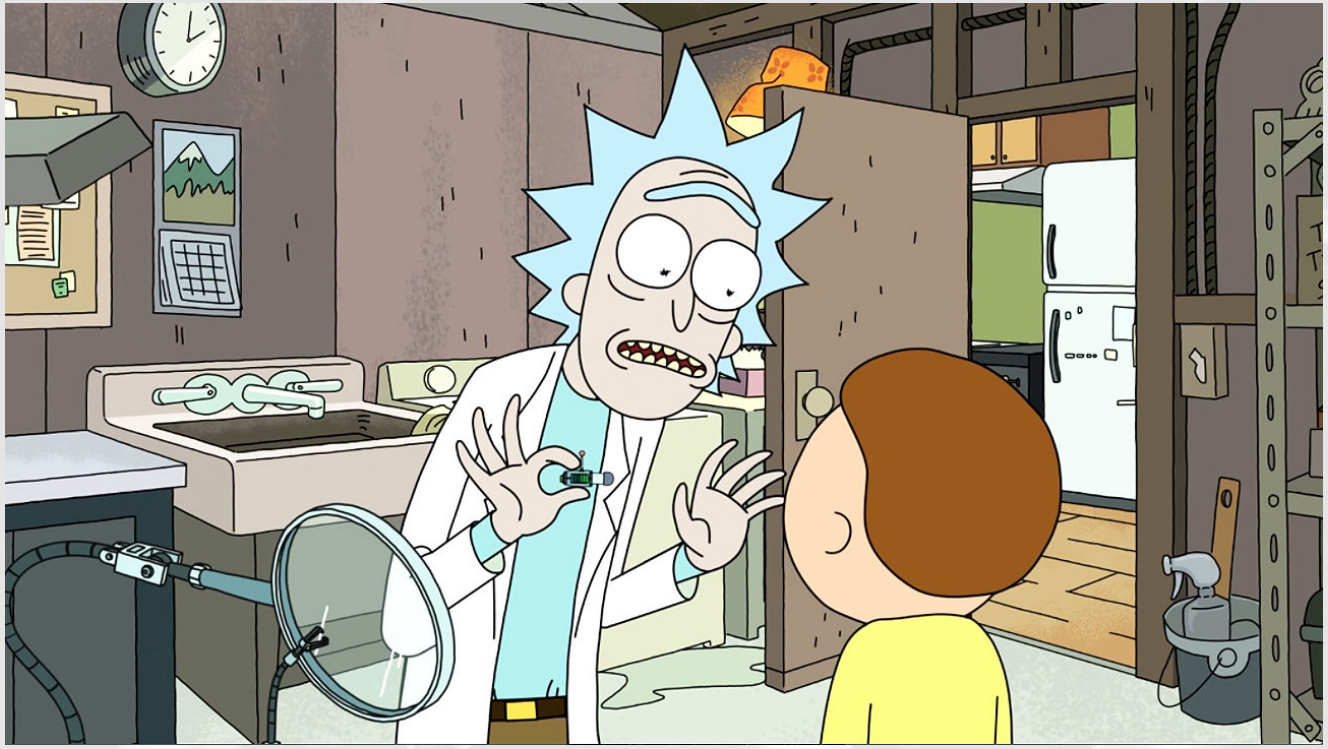


REGULAR EXPRESSION DOS

> *RE implementations may reach extreme situations*

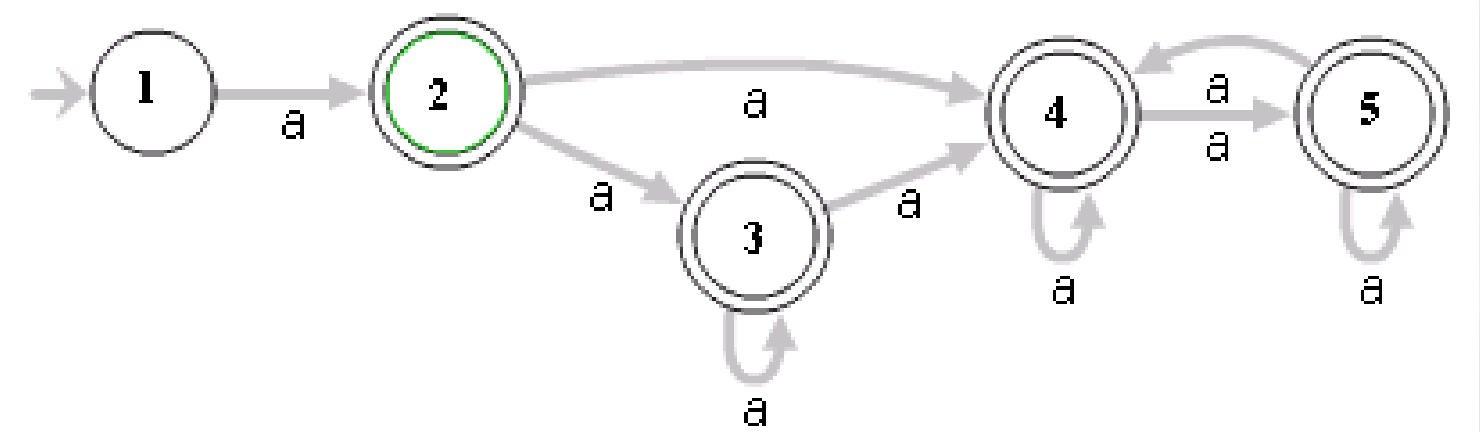
REGULAR EXPRESSION DOS

What the hell is the **state machine** and **NFA**?



REGULAR EXPRESSION DOS

Regular Expression: ^(a+)+$



Result:

aaaaX - 16 possible paths

aaaaaaaaaaaaaaaaX - 65536 possible paths

REGULAR EXPRESSION DOS

Evil Regular Expressions:

- grouping with repetition
- inside the repeated group
  - repetition
  - alternation with overlapping

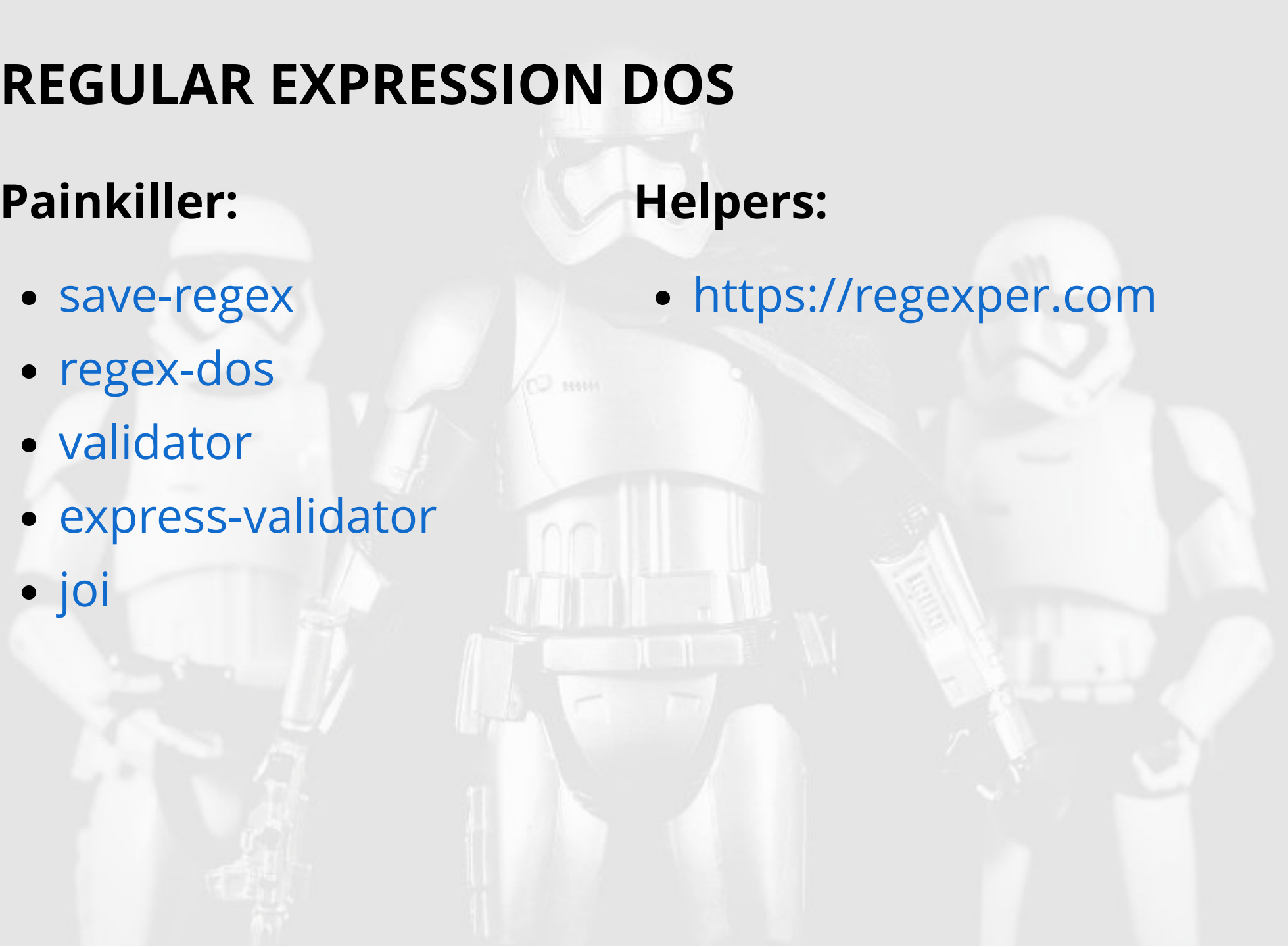
REGULAR EXPRESSION DOS

Painkiller:

- save-regex
- regex-dos
- validator
- express-validator
- joi

Helpers:

- https://regexper.com



MEMORY LEAKS

“ Tracking down memory leaks has always been a challenge

MEMORY LEAKS

heap

resident set

new space

stack

retained size

old space

mark-sweep

shallow size

scavenge

MEMORY LEAKS

Common Reasons:

- forgotten timers or callbacks
- weak closures
- insecure dependencies
- buggy technology

  - new Buffer(size) (deprecated, but still)

MEMORY LEAKS

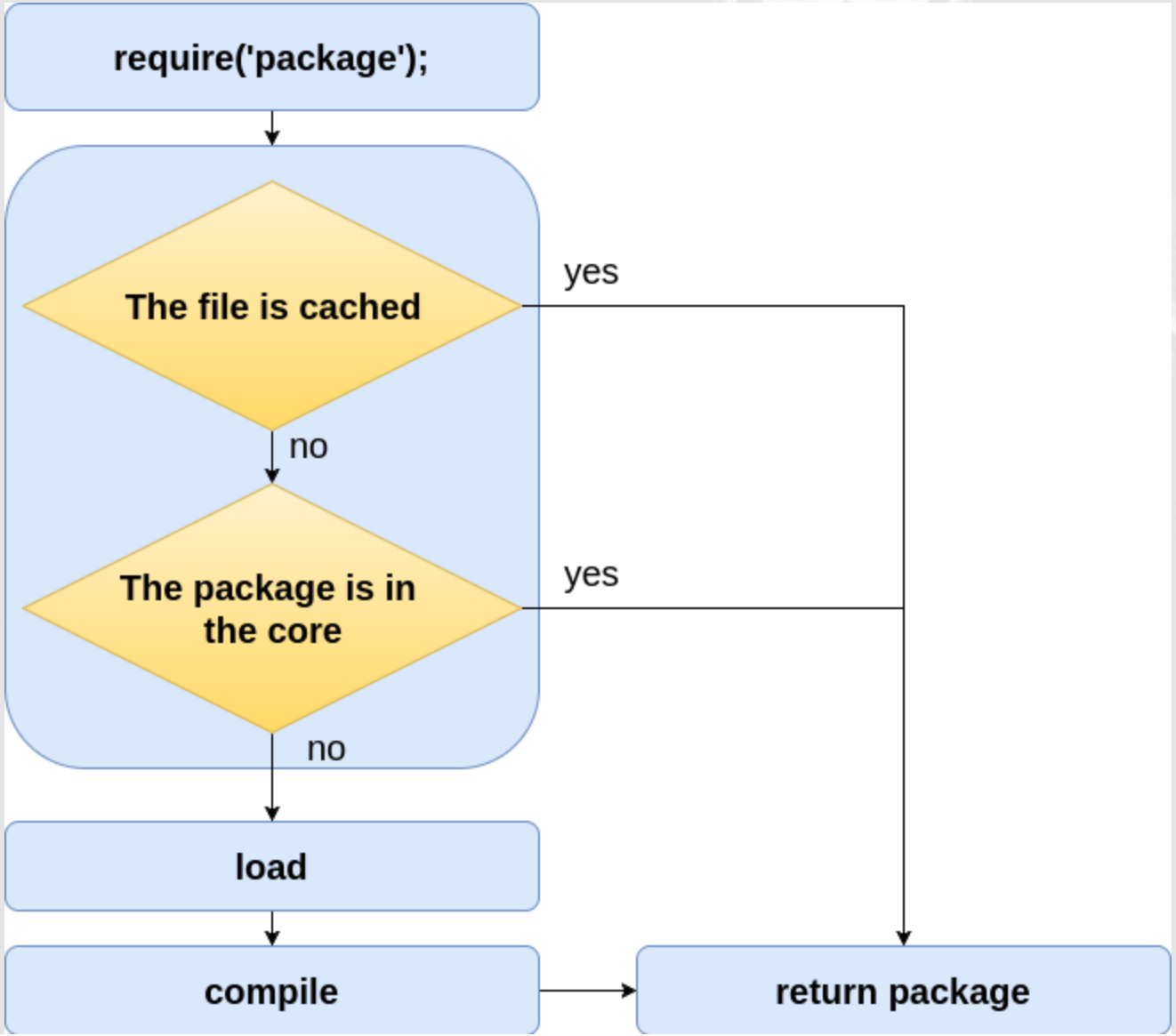
Painkiller:

- heapdump
- memwatch
- node-inpsector

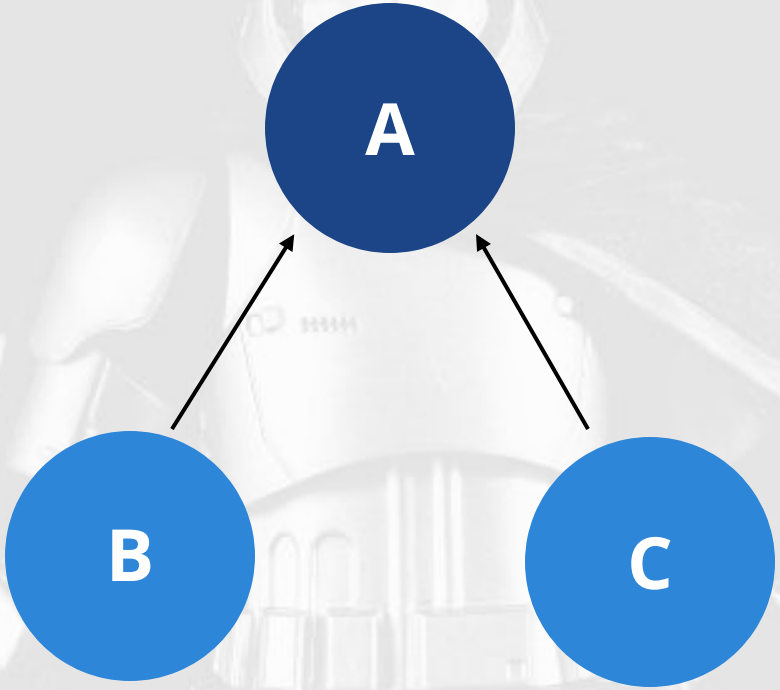
HIJACKING THE REQUIRE CHAIN

> *Hooking all asynchronous core methods is definitely possible*

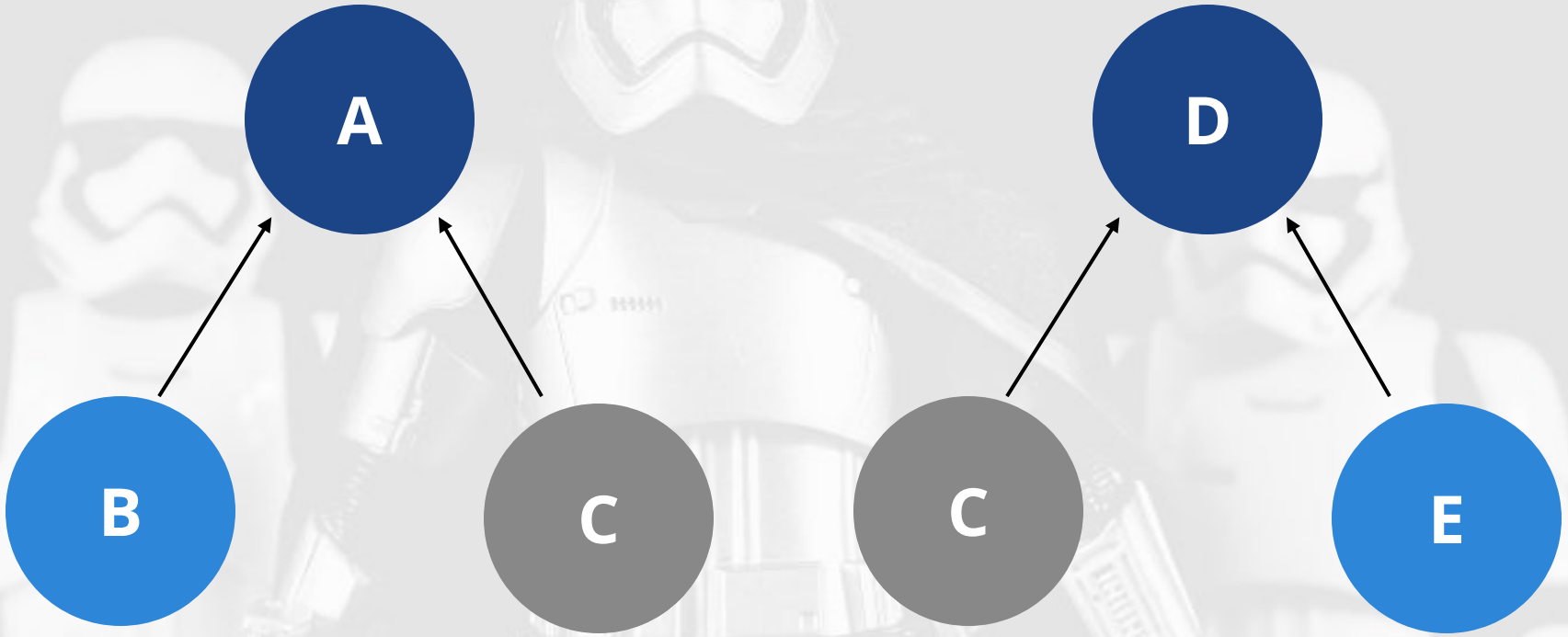
HIJACKING THE REQUIRE CHAIN



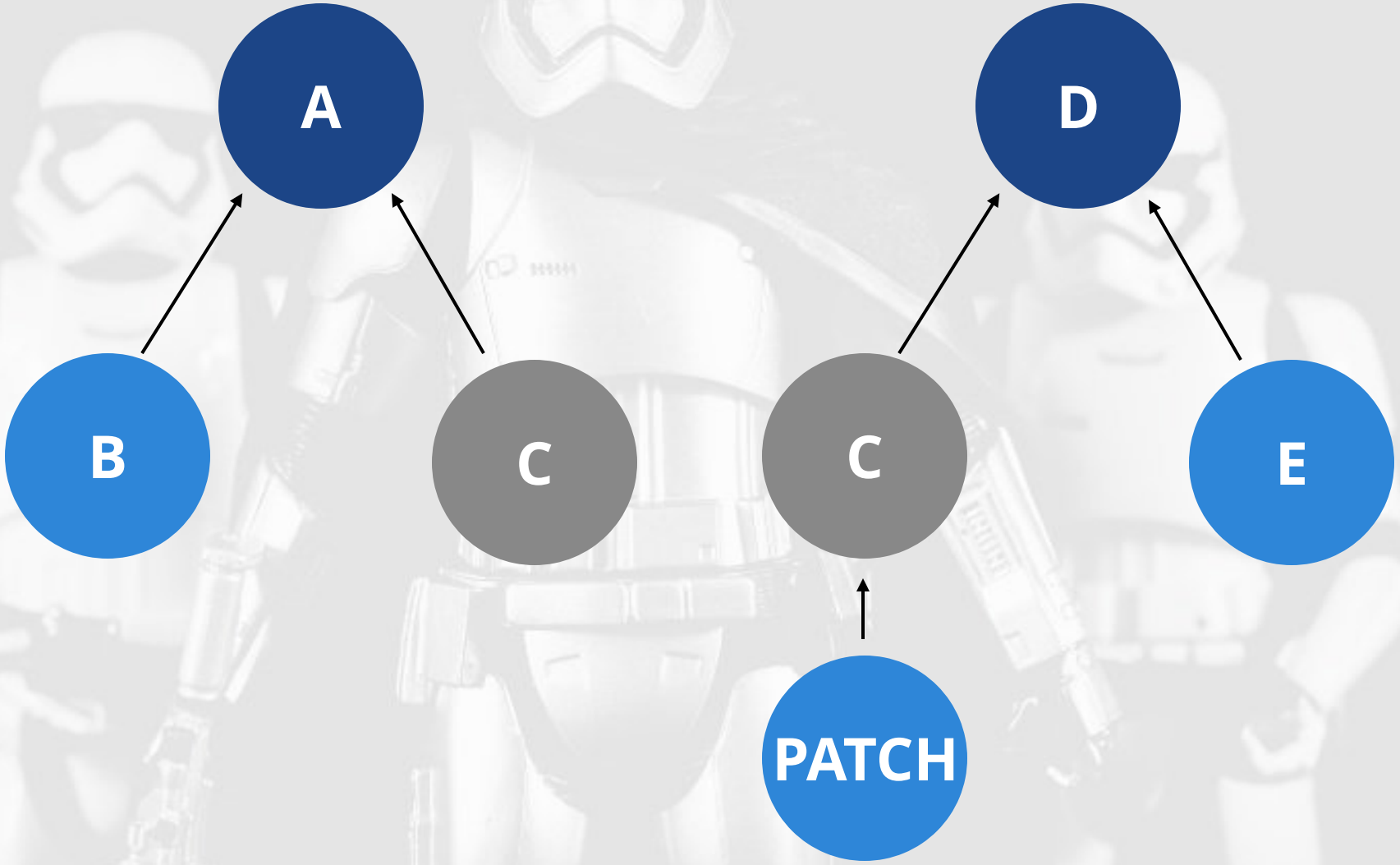
HIJACKING THE REQUIRE CHAIN



HIJACKING THE REQUIRE CHAIN

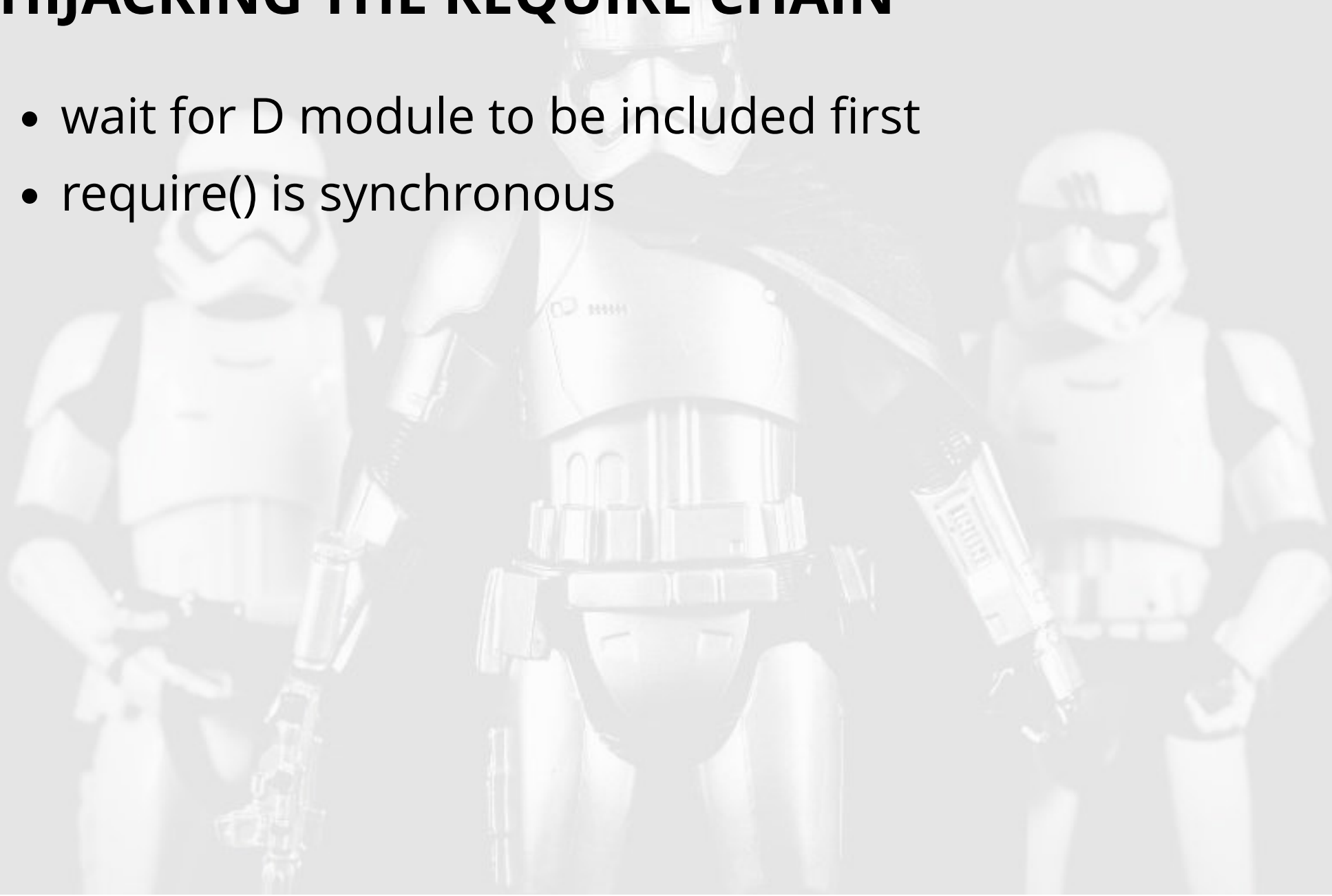


HIJACKING THE REQUIRE CHAIN

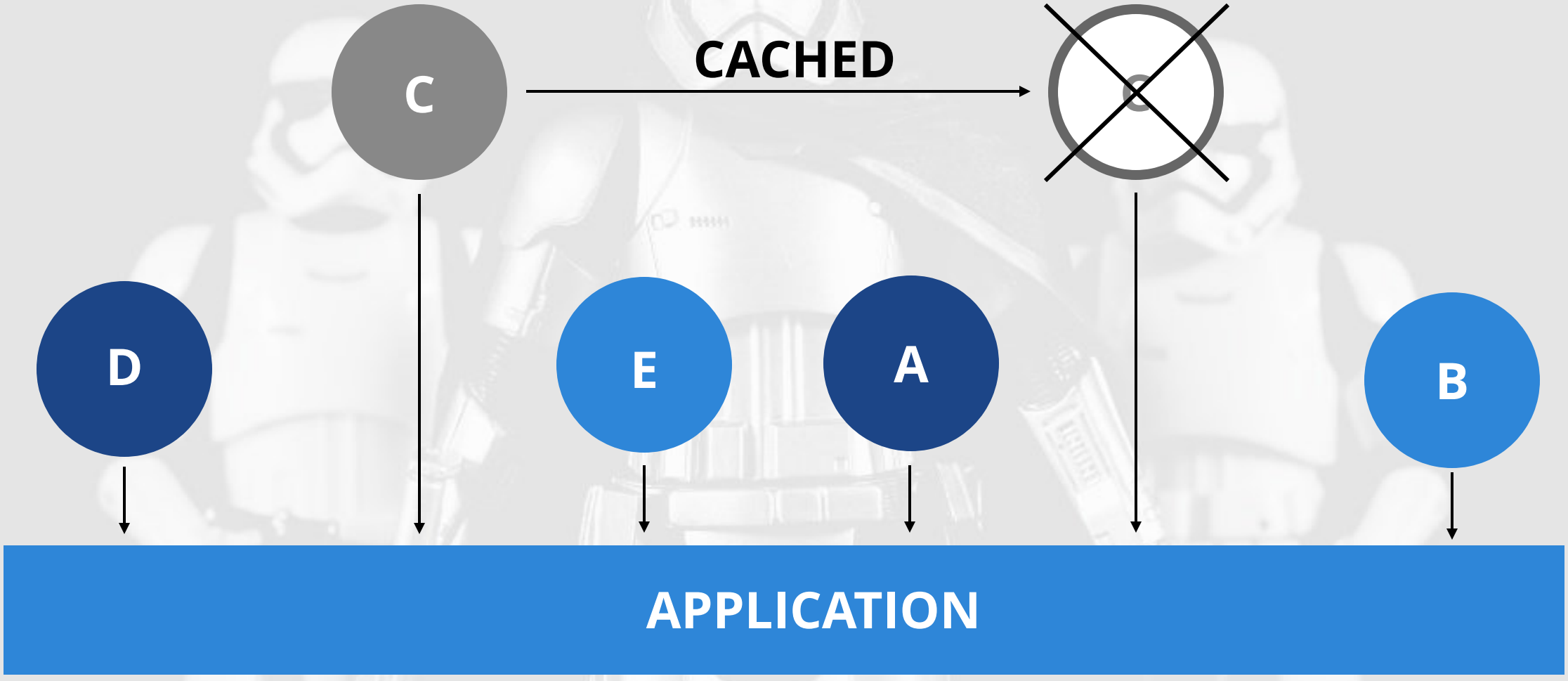


HIJACKING THE REQUIRE CHAIN

- wait for D module to be included first
- require() is synchronous



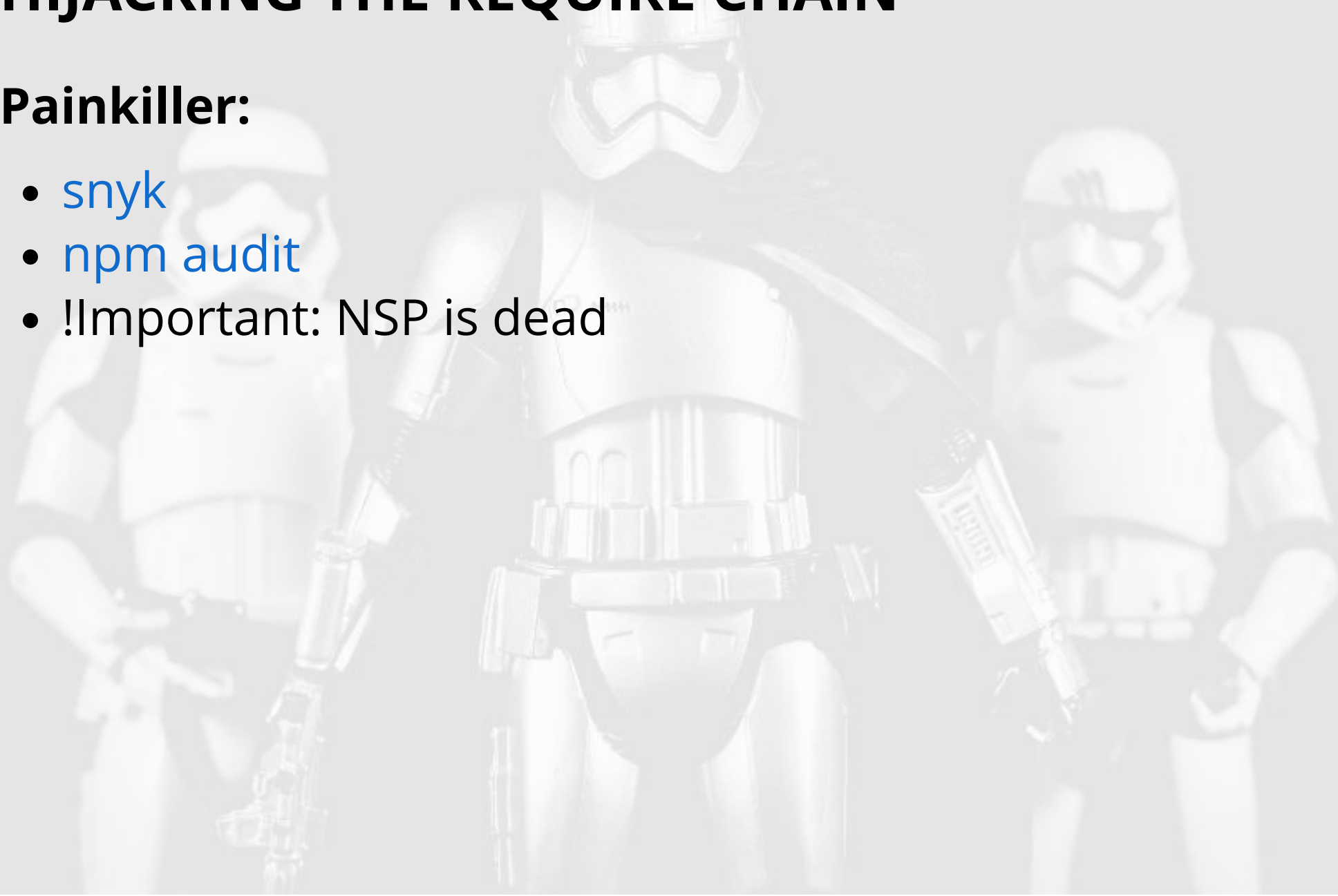
HIJACKING THE REQUIRE CHAIN



HIJACKING THE REQUIRE CHAIN

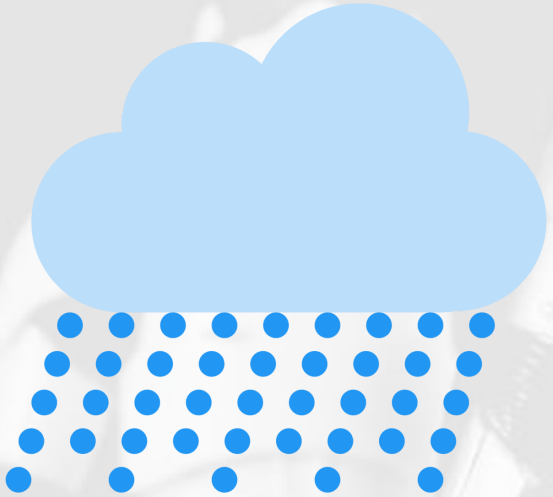
Painkiller:

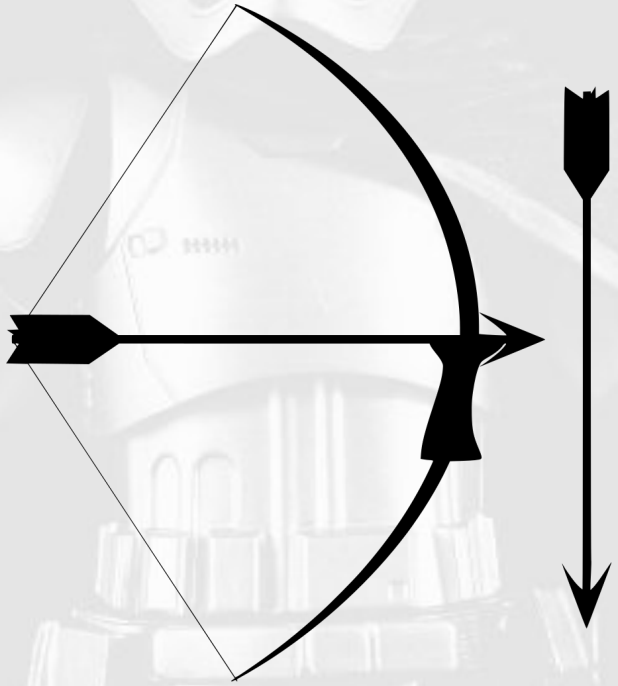
- snyk
- npm audit
- !Important: NSP is dead



RAINBOW TABLE

“ Rain Bow Table attack must be weird enough…

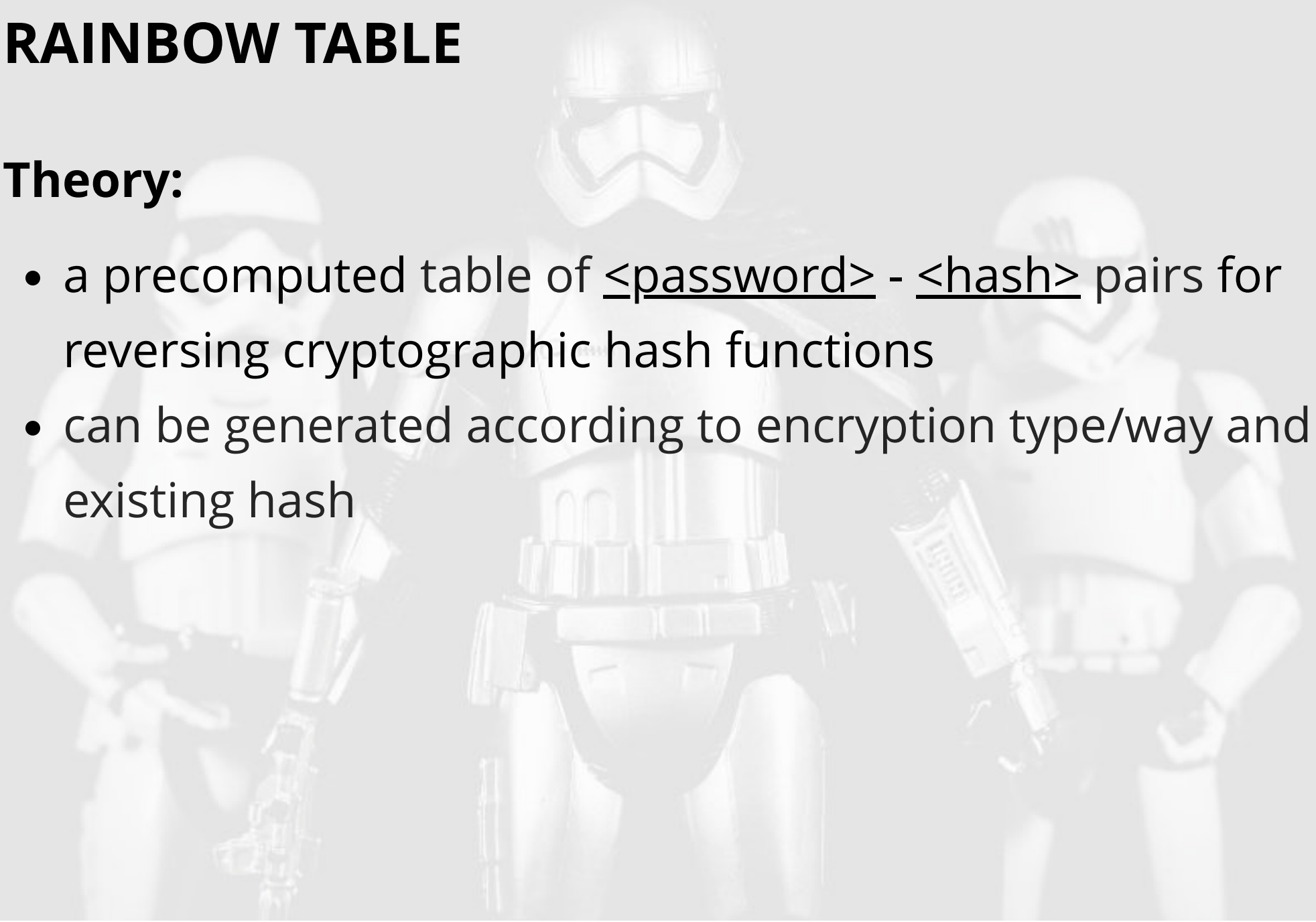




RAINBOW TABLE

Theory:

- a precomputed table of <password> - <hash> pairs for reversing cryptographic hash functions
- can be generated according to encryption type/way and existing hash



RAINBOW TABLE

Best Practice:

- don't use Math.random() to generate a random password
- use personal salt
- use well known cryptographic modules (crypto, bcrypt)
  - !important: crypto vs bcrypt

TIMING ATTACK

> *The fourth letter of your password is wrong, please try again*

TIMING ATTACK

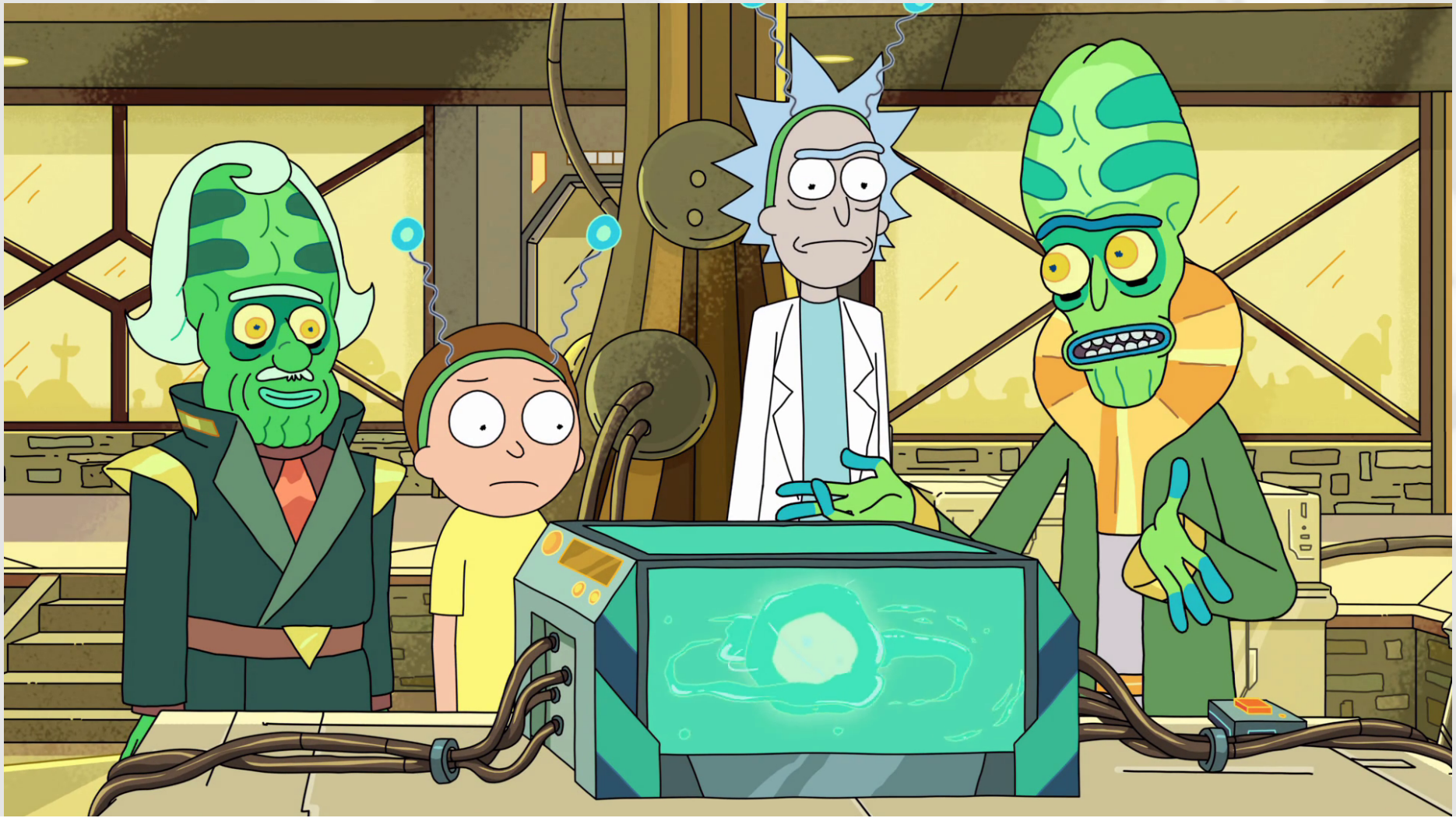
- tired of slides
- just keep talking
- they won't notice

HASH TABLE COLLISION (FLOOD) ATTACK

> *That moment when you realize we're in trouble*

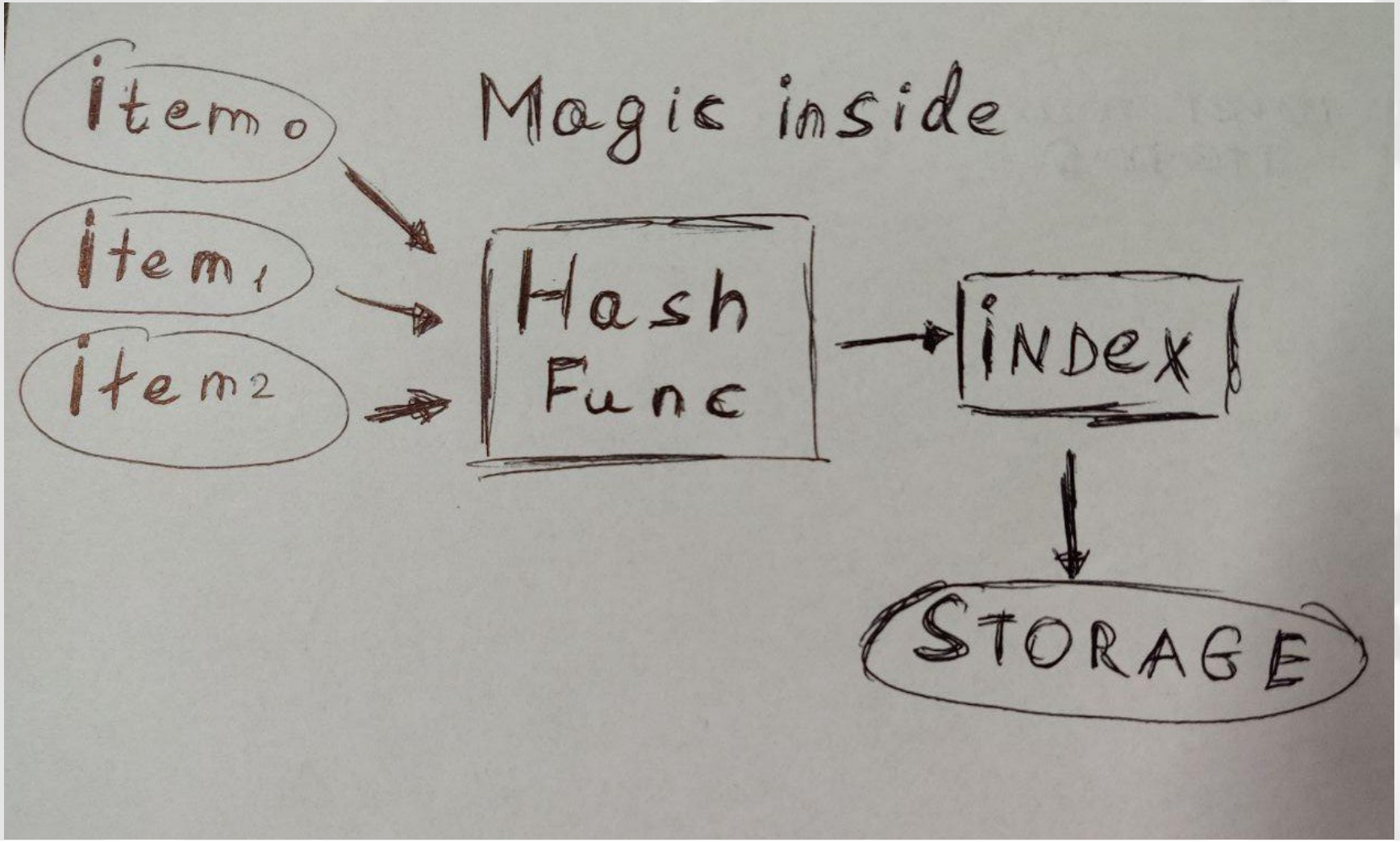
HASH TABLE COLLISION (FLOOD) ATTACK

Hash table?



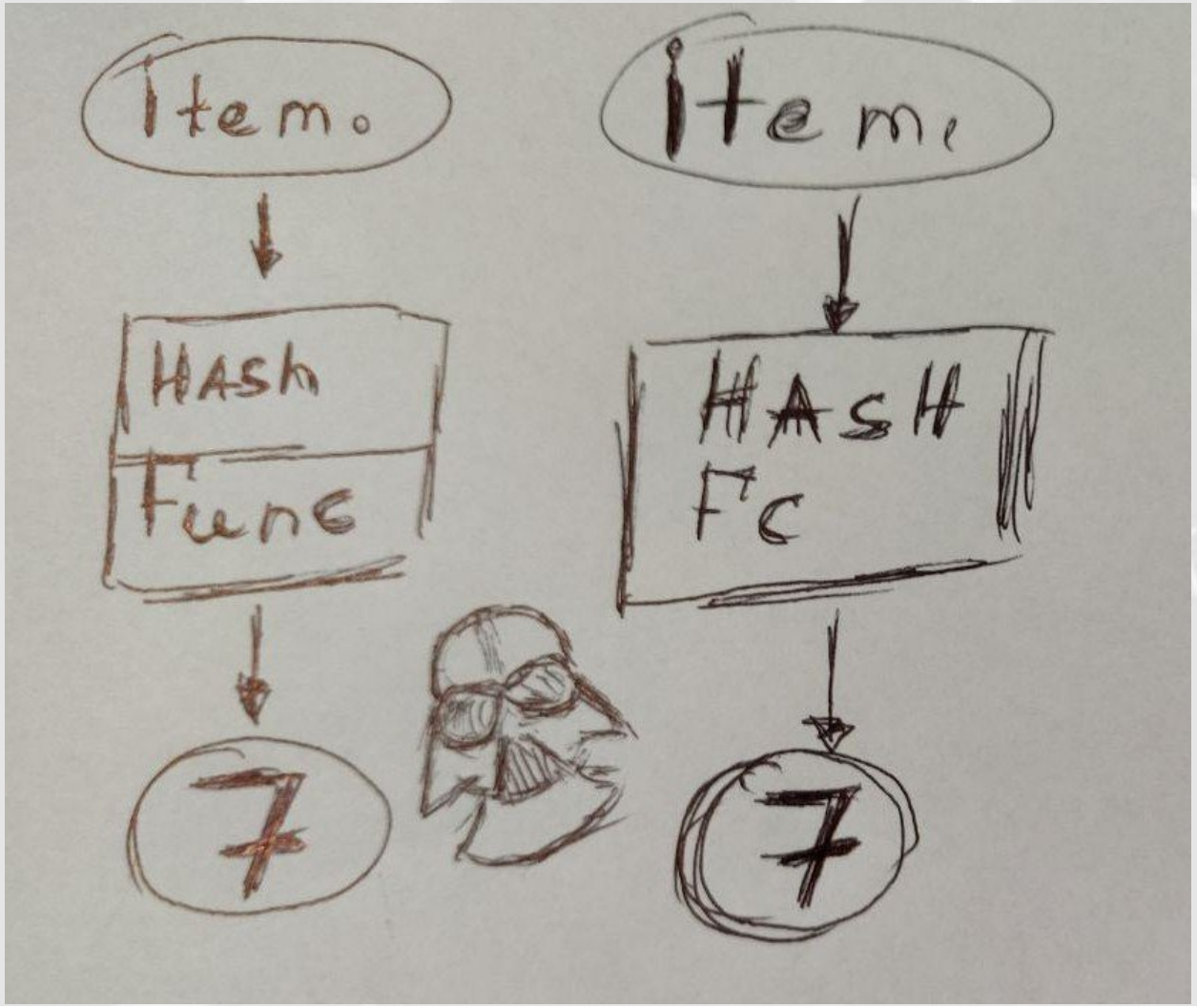
HASH TABLE COLLISION (FLOOD) ATTACK

Hash table?



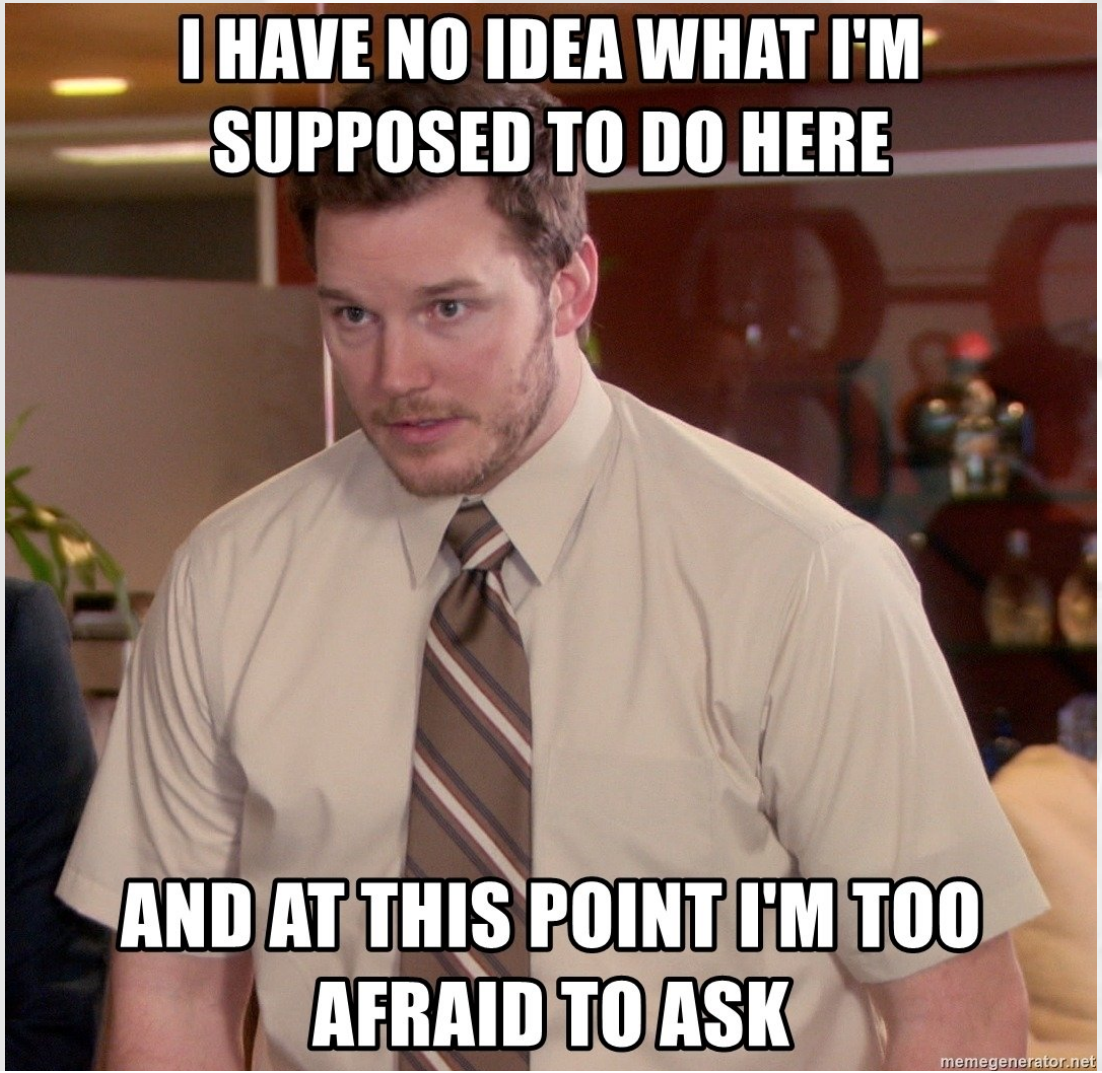
HASH TABLE COLLISION (FLOOD) ATTACK

Hash table collision :(



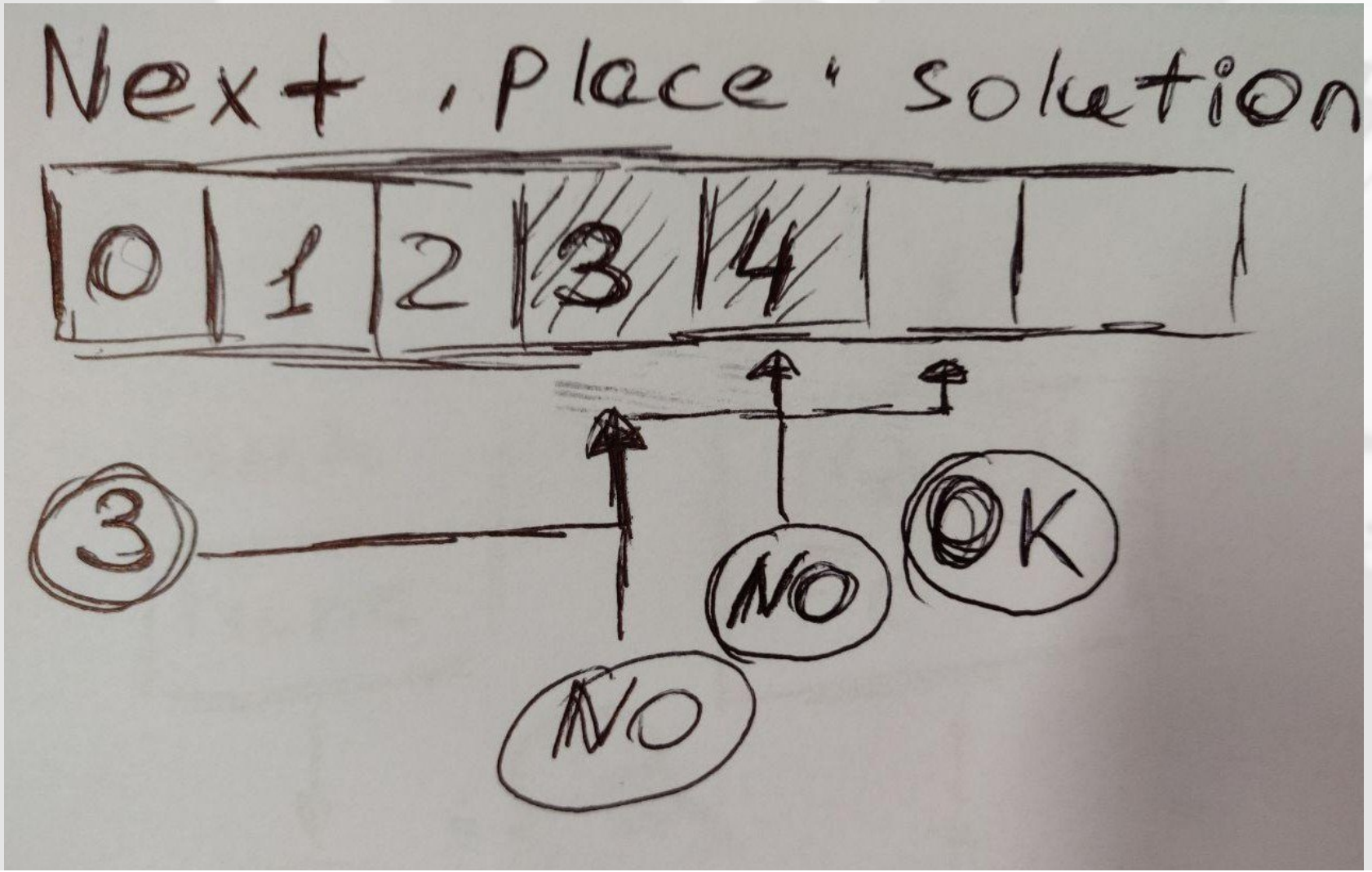
HASH TABLE COLLISION (FLOOD) ATTACK

Hash table collision :(



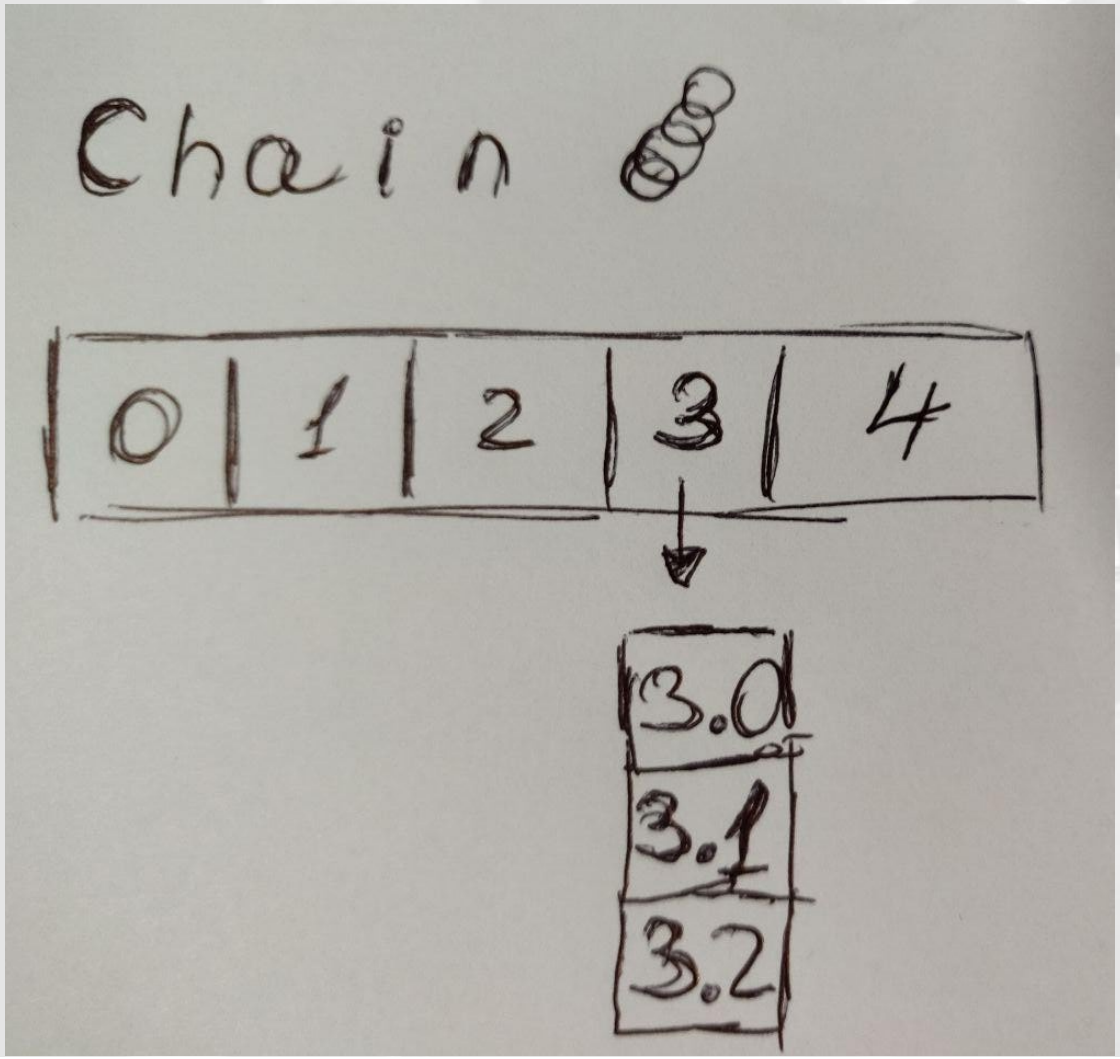
HASH TABLE COLLISION (FLOOD) ATTACK

Hash table collision :(



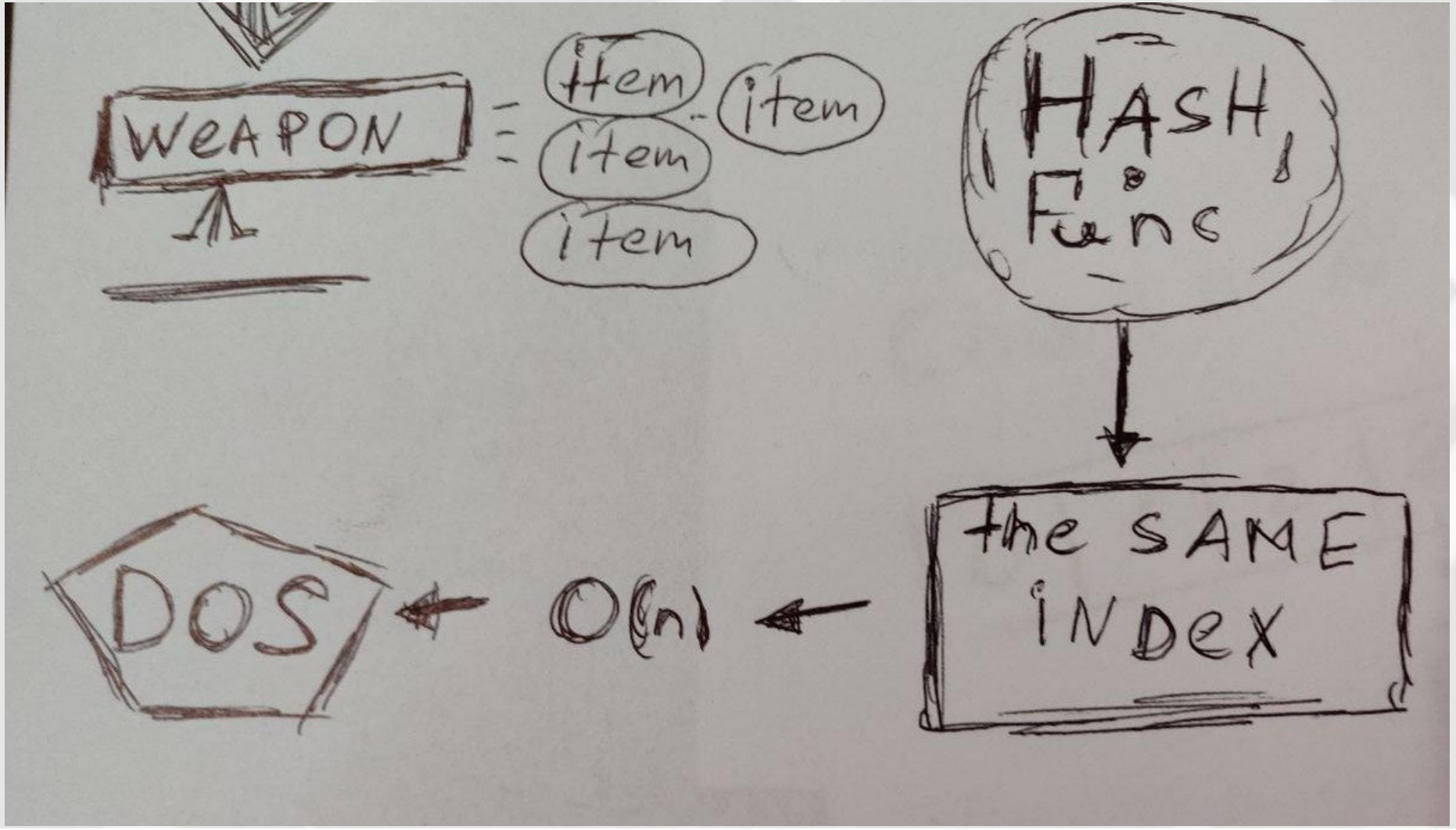
HASH TABLE COLLISION (FLOOD) ATTACK

Hash table collision :(



HASH TABLE COLLISION (FLOOD) ATTACK

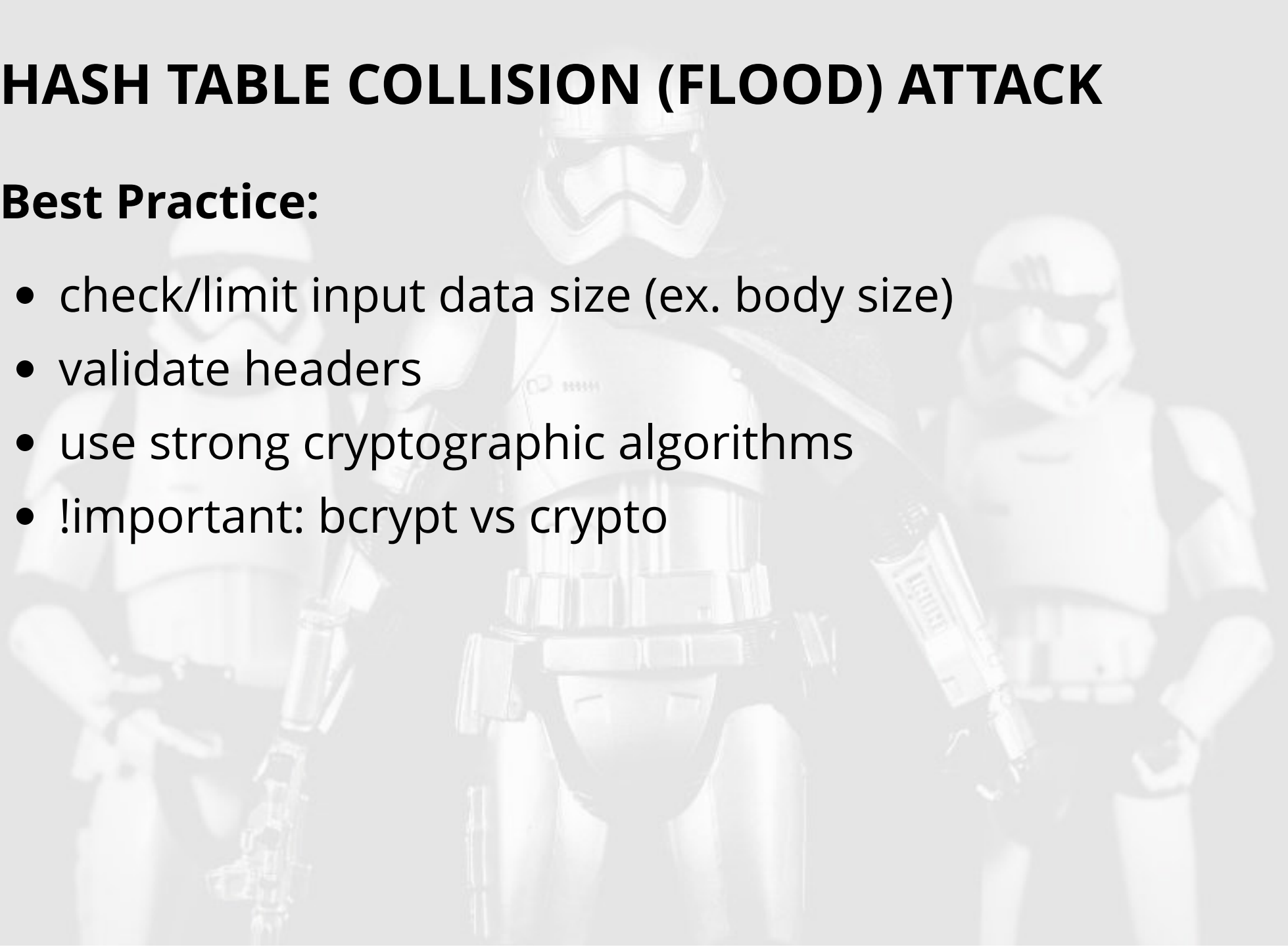
Hash table collision attack?



HASH TABLE COLLISION (FLOOD) ATTACK

Best Practice:

- check/limit input data size (ex. body size)
- validate headers
- use strong cryptographic algorithms
- !important: bcrypt vs crypto



BEFORE YOU GO

> *The thing to show I've got for you (c) Yoda parodist*

BEFORE YOU GO

Security starts from quality

```
// package.json

"config": {
  "ghooks": {
    "pre-push": "npm test && npm audit",
    "pre-commit": "npm run lint"
  }
},
```

BEFORE YOU GO

Security starts from quality

```
rsachenko@HATECREW:~/Projects/Talks/2018_Node_Conf$ npm run lint:fix

> 2018_node_conf@1.0.0 lint:fix /home/rsachenko/Projects/Talks/2018_Node_Conf
> eslint  . *.js --fix


/home/rsachenko/Projects/Talks/2018_Node_Conf/controller.js
  54:1  error  Line 54 exceeds the maximum line length of 180  max-len

✖ 1 problem (1 error, 0 warnings)

npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! 2018_node_conf@1.0.0 lint:fix: `eslint  . *.js --fix`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the 2018_node_conf@1.0.0 lint:fix script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output a

npm ERR! A complete log of this run can be found in:
npm ERR!     /home/rsachenko/.npm/_logs/2018-09-28T12_17_16_779Z-debug.log
```

BEST PRACTICE

" May the 4th be with you

BEST PRACTICE

- Limit requests frequency
- Don't accept or use carefully query params as database query items
- Validate incoming body, query params
- Hide 'X-Powered-By' header
- Set a strong access control system
- Use SSL
- Be careful with expressions
- Use Security check tools
- User secure WebSocket connection
- OWASP top 10 (http://nodegoat.herokuapp.com/tutorial)

HELPFUL MODULES

“ Let off some steam

HELPFUL MODULES

Protection

- helmet
- lusca
- cruf

Validation

- validator
- express-validator
- joi

Package Security Check

- snyk
- npm audit

Rate Limiting

- node-rate-limiter

Memory Check

- heapdump
- memwatch
- node-inspector

Other

- cors
- safe-regex
- regex-dos

ARTICLES

“ Do they speak English in What?

ARTICLES

Security 'musts'

- node-js-security-checklist
- express-best-practice-security

Memory Leak

- a-curious-case-of-memory-leak
- self-detect-memory-leak-node
- memory-leaks-patterns

Require Chain

- easy-way-to-inject-malicious-code

Rainbow Table

- rainbow-table

ReDOS

- regular-expression-dos

Other

- writing-secure-node-js-code

QUESTIONS

*and again I couldn't find a funny meme for this page

CONTACTS

roman.sachenko@gmail.com

roman.sachenko

https://www.facebook.com/rsachenko

https://twitter.com/rsachenko

https://github.com/roman-sachenko/