

The OWASP Top 10

Bernard Nongpoh

Assistant Professor, Department of CSE



ভাৰতীয় প্ৰযুক্তিবিদ্যা প্ৰতিষ্ঠান গুৱাহাটী
भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

What Is OWASP? (1/3)



Open **W**orldwide **A**pplication **S**ecurity **P**roject

- A **non-profit, open community** for software security
- Everything is **free** and **vendor-neutral**: anyone can read it or help

OWASP Foundation; OWASP Top 10:2025, Introduction

What Is OWASP? (2/3)



Open **W**orldwide **A**pplication **S**ecurity **P**roject

- A **non-profit, open community** for software security
- Everything is **free** and **vendor-neutral**: anyone can read it or help
- Known for the **Top 10**, the **Juice Shop**, and many guides

OWASP Foundation; OWASP Top 10:2025, Introduction

What Is OWASP? (3/3)



Open **W**orldwide **A**pplication **S**ecurity **P**roject

- A **non-profit, open community** for software security
- Everything is **free** and **vendor-neutral**: anyone can read it or help
- Known for the **Top 10**, the **Juice Shop**, and many guides

OWASP Foundation; OWASP Top 10:2025, Introduction

OWASP Juice Shop (1/5)



Björn Kimminich
and contributors, since 2014

GitHub:
juice-shop/juice-shop

We need a real web shop to attack. Where can we do that legally?

OWASP Juice Shop (2/5)



Björn Kimminich
and contributors, since 2014

GitHub:
juice-shop/juice-shop

We need a real web shop to attack. Where can we do that legally?

- A web shop that is **insecure on purpose**: Node.js, Express, Angular

OWASP Juice Shop (3/5)



Björn Kimminich
and contributors, since 2014

GitHub:
juice-shop/juice-shop

We need a real web shop to attack. Where can we do that legally?

- A web shop that is **insecure on purpose**: Node.js, Express, Angular
- **118 challenges**, from 1 to 6 stars, tracked on a score board

OWASP Juice Shop; github.com/juice-shop/juice-shop

OWASP Juice Shop (4/5)



Björn Kimminich
and contributors, since 2014

GitHub:
juice-shop/juice-shop

We need a real web shop to attack. Where can we do that legally?

- A web shop that is **insecure on purpose**: Node.js, Express, Angular
- **118 challenges**, from 1 to 6 stars, tracked on a score board
- **Coding challenges**: find the vulnerable line, then pick the fix

OWASP Juice Shop; github.com/juice-shop/juice-shop

OWASP Juice Shop (5/5)



Björn Kimminich
and contributors, since 2014

GitHub:
juice-shop/juice-shop

We need a real web shop to attack. Where can we do that legally?

- A web shop that is **insecure on purpose**: Node.js, Express, Angular
- **118 challenges**, from 1 to 6 stars, tracked on a score board
- **Coding challenges**: find the vulnerable line, then pick the fix

OWASP Juice Shop; github.com/juice-shop/juice-shop

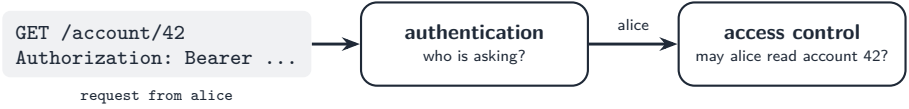
A01 Broken Access Control (1/4)

The user is logged in. May they do *this*?



A01 Broken Access Control (2/4)

The user is logged in. May they do *this*?



A01 Broken Access Control (3/4)

The user is logged in. May they do *this*?



A01 Broken Access Control (4/4)

The user is logged in. May they do *this*?



Three common forms: **change an id** in the URL, **open an admin URL** directly, **a check only in the browser**.

A01 in Juice Shop: View Basket (1/5)

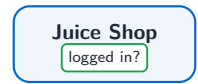
server.ts and routes/basket.ts, Juice Shop

```
431 app.use('/rest/basket/:id', security.
        isAuthorized())
```

```
18 const id = req.params.id
19 const basket = await BasketModel.findOne({
        where: { id }, ... })
```



her basket is 1



Juice Shop: routes/basket.ts, challenge *View Basket*; OWASP Top 10:2025, A01 Broken Access Control

A01 in Juice Shop: View Basket (2/5)

server.ts and routes/basket.ts, Juice Shop

```
431 app.use('/rest/basket/:id', security.
        isAuthorized())
```

```
18 const id = req.params.id
19 const basket = await BasketModel.findOne({
        where: { id }, ... })
```



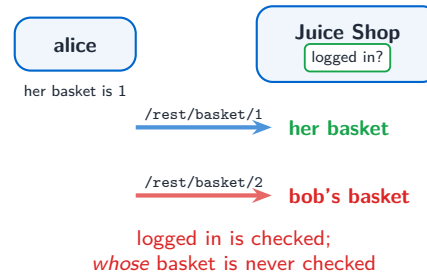
Juice Shop: routes/basket.ts, challenge *View Basket*; OWASP Top 10:2025, A01 Broken Access Control

A01 in Juice Shop: View Basket (3/5)

server.ts and routes/basket.ts, Juice Shop

```
431 app.use('/rest/basket/:id', security.
        isAuthorized())
```

```
18 const id = req.params.id
19 const basket = await BasketModel.findOne({
        where: { id }, ... })
```



Juice Shop: routes/basket.ts, challenge *View Basket*; OWASP Top 10:2025, A01 Broken Access Control

A01 in Juice Shop: View Basket (4/5)

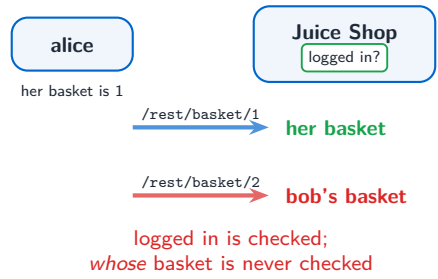
server.ts and routes/basket.ts, Juice Shop

```
431 app.use('/rest/basket/:id', security.
        isAuthorized())
```

```
18 const id = req.params.id
19 const basket = await BasketModel.findOne({
        where: { id }, ... })
```

a fix

```
const user =
  security.authenticatedUsers.from(req)
const basket = await BasketModel.findOne({
  where: { id, UserId: user?.data.id },
  ... })
```



Juice Shop: routes/basket.ts, challenge *View Basket*; OWASP Top 10:2025, A01 Broken Access Control

A01 in Juice Shop: View Basket (5/5)

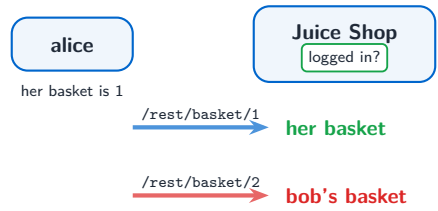
server.ts and routes/basket.ts, Juice Shop

```
431 app.use('/rest/basket/:id', security.
        isAuthorized())
```

```
18 const id = req.params.id
19 const basket = await BasketModel.findOne({
        where: { id }, ... })
```

a fix

```
const user =
  security.authenticatedUsers.from(req)
const basket = await BasketModel.findOne({
  where: { id, UserId: user?.data.id },
  ... })
```



Lab: Open your basket. In Application, Session Storage, change `bid` to another number. Reload.

Juice Shop: routes/basket.ts, challenge *View Basket*; OWASP Top 10:2025, A01 Broken Access Control

Security Misconfiguration (1/5)

The code is correct. Can the settings still open a door?

- **Features left on**: directory listing, debug pages, sample apps

OWASP Top 10:2025, Introduction; Stanford CS253

Security Misconfiguration (2/5)

The code is correct. Can the settings still open a door?

- **Features left on**: directory listing, debug pages, sample apps
- **Defaults kept**: admin accounts with default passwords

OWASP Top 10:2025, Introduction; Stanford CS253

Security Misconfiguration (3/5)

The code is correct. Can the settings still open a door?

- **Features left on**: directory listing, debug pages, sample apps
- **Defaults kept**: admin accounts with default passwords
- **Too much information**: versions in headers, stack traces

OWASP Top 10:2025, Introduction; Stanford CS253

Security Misconfiguration (4/5)

The code is correct. Can the settings still open a door?

- **Features left on**: directory listing, debug pages, sample apps
- **Defaults kept**: admin accounts with default passwords
- **Too much information**: versions in headers, stack traces
- **Cloud storage** readable by everyone; **missing security headers**

OWASP Top 10:2025, Introduction; Stanford CS253

Security Misconfiguration (5/5)

The code is correct. Can the settings still open a door?

- **Features left on**: directory listing, debug pages, sample apps
- **Defaults kept**: admin accounts with default passwords
- **Too much information**: versions in headers, stack traces
- **Cloud storage** readable by everyone; **missing security headers**

Moved up from place 5 in 2021 to place 2 in 2025.

OWASP Top 10:2025, Introduction; Stanford CS253

A02 in Juice Shop: Files and Error Pages (1/4)

server.ts, Juice Shop

```
294 app.use('/ftp', serveIndexMiddleware,
        serveIndex('ftp', { icons: true }))
```

open /ftp in the browser (abridged)

```
acquisitions.md
coupons_2013.md.bak
incident-support.kdbx
package.json.bak
... 11 entries
```

Juice Shop: server.ts, challenges *Confidential Document*, *Error Handling*

A02 in Juice Shop: Files and Error Pages (2/4)

server.ts, Juice Shop

```
294 app.use('/ftp', serveIndexMiddleware,
        serveIndex('ftp', { icons: true }))
```

```
720 /* Error Handling */
721 app.use(verify.errorHandlingChallenge())
722 app.use(errorhandler())
```

open /ftp in the browser (abridged)

```
acquisitions.md
coupons_2013.md.bak
incident-support.kdbx
package.json.bak
... 11 entries
```

open /rest/no-such-thing

```
OWASP Juice Shop (Express ^4.22.1)
at Layer.handle [as handle_request]
 (/juice-shop/node_modules/express/
  lib/router/layer.js:95:5)
...
```

Juice Shop: server.ts, challenges *Confidential Document*, *Error Handling*

A02 in Juice Shop: Files and Error Pages (3/4)

server.ts, Juice Shop

```
294 app.use('/ftp', serveIndexMiddleware,
        serveIndex('ftp', { icons: true }))
```

```
720 /* Error Handling */
721 app.use(verify.errorHandlingChallenge())
722 app.use(errorhandler())
```

open /ftp in the browser (abridged)

```
acquisitions.md
coupons_2013.md.bak
incident-support.kdbx
package.json.bak
... 11 entries
```

open /rest/no-such-thing

```
OWASP Juice Shop (Express ^4.22.1)
at Layer.handle [as handle_request]
 (/juice-shop/node_modules/express/
  lib/router/layer.js:95:5)
...
```

Juice Shop's own fix: **remove** `/ftp`; show a **plain error page** in production.

Juice Shop: server.ts, challenges *Confidential Document*, *Error Handling*

A02 in Juice Shop: Files and Error Pages (4/4)

server.ts, Juice Shop

```
294 app.use('/ftp', serveIndexMiddleware,
        serveIndex('ftp', { icons: true }))
```

```
720 /* Error Handling */
721 app.use(verify.errorHandlingChallenge())
722 app.use(errorhandler())
```

open /ftp in the browser (abridged)

```
acquisitions.md
coupons_2013.md.bak
incident-support.kdbx
package.json.bak
... 11 entries
```

open /rest/no-such-thing

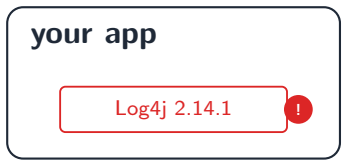
```
OWASP Juice Shop (Express ^4.22.1)
at Layer.handle [as handle_request]
 (/juice-shop/node_modules/express/
  lib/router/layer.js:95:5)
...
```

Juice Shop's own fix: **remove** `/ftp`; show a **plain error page** in production.
Lab: Open `/ftp` and the file `acquisitions.md`. Then open `/rest/no-such-thing`.

Juice Shop: server.ts, challenges *Confidential Document*, *Error Handling*

A03 Software Supply Chain Failures (1/2)

You ship code you did not write. Its bugs become yours.

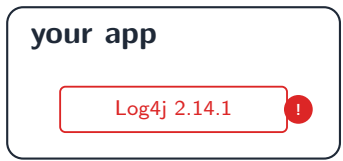


a known bug *inside* the library is a bug **inside your app**

OWASP Top 10:2025

A03 Software Supply Chain Failures (2/2)

You ship code you did not write. Its bugs become yours.

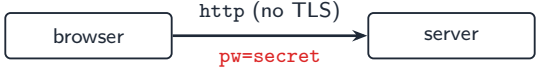


a known bug *inside* the library is a bug **inside your app**

Example: Log4Shell, 2021.

A04 Cryptographic Failures (1/2)

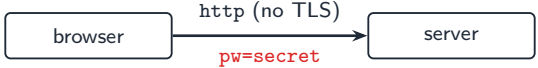
Secrets are not protected well enough, in transit or at rest.



OWASP Top 10:2025

A04 Cryptographic Failures (2/2)

Secrets are not protected well enough, in transit or at rest.



Example: password over plain HTTP, or stored as MD5.

A04: TLS and MD5 (1/3)

Two ways cryptography fails: in transit, and at rest.

TLS protects data *in transit*



- the **S** in HTTP**S**: encrypts the traffic

MD5 (mis)protects data *at rest*



- a **hash**: a fixed fingerprint of the password

OWASP: TLS; OWASP: Password Storage

A04: TLS and MD5 (2/3)

Two ways cryptography fails: in transit, and at rest.

TLS protects data *in transit*



- the **S** in HTTP**S**: encrypts the traffic
- also **proves the server** and detects tampering

MD5 (mis)protects data *at rest*



- a **hash**: a fixed fingerprint of the password
- **Failure**: **too fast** (billions/sec guessed), **broken**, often unsalted

OWASP: TLS; OWASP: Password Storage

A04: TLS and MD5 (3/3)

Two ways cryptography fails: in transit, and at rest.

TLS protects data *in transit*



- the **S** in HTTP**S**: encrypts the traffic
- also **proves the server** and detects tampering
- **Failure**: plain `http`, so anyone on the network reads it

MD5 (mis)protects data *at rest*



- a **hash**: a fixed fingerprint of the password
- **Failure**: **too fast** (billions/sec guessed), **broken**, often unsalted
- **Fix**: a slow, salted hash: `bcrypt`, `scrypt`, `Argon2`

Rule: **HTTPS everywhere**, and never store a password with a fast hash.

OWASP: TLS; OWASP: Password Storage

A05 Injection (1/2)

User text is read as code by another interpreter.

```
WHERE email='' OR 1=1--'
```

the typed text became SQL: logged in as admin

A05 Injection (2/2)

User text is read as code by another interpreter.

```
WHERE email='' OR 1=1--'
```

the typed text became SQL: logged in as admin

Example: SQL, shell, or HTML injection.

OWASP Top 10:2025

A05: Code Injection (1/3)

Injection is *code injection*: code and data share one channel.



database: SQL injection

```
q = "... email = '" + input + "'"
input = ' OR 1=1--
⇒ email = '' OR 1=1--' (always true)
```

OS shell: command injection

```
exec("ping " + input)
input = ; cat /etc/passwd
⇒ ping ...; cat /etc/passwd
```

A05: Code Injection (2/3)

Injection is *code injection*: code and data share one channel.



database: SQL injection

```
q = "... email = '" + input + "'"
input = ' OR 1=1--
⇒ email = '' OR 1=1--' (always true)
```

OS shell: command injection

```
exec("ping " + input)
input = ; cat /etc/passwd
⇒ ping ...; cat /etc/passwd
```

Same bug in the **browser** too, reading HTML: that is XSS.

A05: Code Injection (3/3)

Injection is *code injection*: code and data share one channel.



database: SQL injection

```
q = "... email = '" + input + "'"
input = ' OR 1=1--
⇒ email = '' OR 1=1--' (always true)
```

OS shell: command injection

```
exec("ping " + input)
input = ; cat /etc/passwd
⇒ ping ...; cat /etc/passwd
```

Same bug in the **browser** too, reading HTML: that is XSS.
Fix: keep data **apart** from code: SQL parameters, shell arg lists, HTML escaping.

A06 Insecure Design (1/2)

No bug in the code. The rule itself is wrong.



600 seats,
one attacker,
no limit

OWASP Top 10:2025

A06 Insecure Design (2/2)

No bug in the code. The rule itself is wrong.



600 seats,
one attacker,
no limit

Example: a cinema with no cap on a booking.

OWASP Top 10:2025

A07 Authentication Failures (1/2)

Weak passwords, no limit on guesses, or a skipped check.

`admin123` weak, and no lockout

try 1, 2, 3, . . . until it works

OWASP Top 10:2025

A07 Authentication Failures (2/2)

Weak passwords, no limit on guesses, or a skipped check.

`admin123` weak, and no lockout

try 1, 2, 3, . . . until it works

Example: brute-forcing the admin.

OWASP Top 10:2025

A08 Software or Data Integrity Failures (1/2)

Code or data is trusted without checking where it came from.



OWASP Top 10:2025

A08 Software or Data Integrity Failures (2/2)

Code or data is trusted without checking where it came from.



Example: an unsigned update, or insecure deserialization.

OWASP Top 10:2025

A09 Security Logging and Alerting Failures (1/2)

The attack happens, but nobody records it or is warned.



no record,
no alert:
found years later

OWASP Top 10:2025

A09 Security Logging and Alerting Failures (2/2)

The attack happens, but nobody records it or is warned.



no record,
no alert:
found years later

Example: a breach noticed after years.

OWASP Top 10:2025

A10 Mishandling of Exceptional Conditions (1/2)

After an error, the program is left in an unsafe state.



OWASP Top 10:2025

A10 Mishandling of Exceptional Conditions (2/2)

After an error, the program is left in an unsafe state.



Example: a transfer cut half-way, repeated.

OWASP Top 10:2025

Quiz: Which Category? (1/5)

Which OWASP category fits each case?

1. A user adds `"role": "admin"` to their sign-up request, and the server makes them an admin

2. The live production site shows full developer error pages (stack traces) to visitors

3. The app still runs Log4j `2.14.1`, a **vulnerable** version (Log4Shell, CVE-2021-44228)

4. The login page sends the password over plain `http`

Quiz: Which Category? (2/5)

Which OWASP category fits each case?

1. A user adds `"role": "admin"` to their sign-up request, and the server makes them an admin — `A01 access control`

2. The live production site shows full developer error pages (stack traces) to visitors

3. The app still runs Log4j `2.14.1`, a **vulnerable** version (Log4Shell, CVE-2021-44228)

4. The login page sends the password over plain `http`

Quiz: Which Category? (3/5)

Which OWASP category fits each case?

1. A user adds `"role": "admin"` to their sign-up request, and the server makes them an admin — `A01 access control`

2. The live production site shows full developer error pages (stack traces) to visitors — `A02 misconfiguration`

3. The app still runs Log4j `2.14.1`, a **vulnerable** version (Log4Shell, CVE-2021-44228) — ____

4. The login page sends the password over plain `http` — ____

Quiz: Which Category? (4/5)

Which OWASP category fits each case?

1. A user adds `"role": "admin"` to their sign-up request, and the server makes them an admin — `A01 access control`

2. The live production site shows full developer error pages (stack traces) to visitors — `A02 misconfiguration`

3. The app still runs Log4j `2.14.1`, a **vulnerable** version (Log4Shell, CVE-2021-44228) — `A03 supply chain`

4. The login page sends the password over plain `http` —

Quiz: Which Category? (5/5)

Which OWASP category fits each case?

1. A user adds `"role": "admin"` to their sign-up request, and the server makes them an admin — `A01 access control`

2. The live production site shows full developer error pages (stack traces) to visitors — `A02 misconfiguration`

3. The app still runs Log4j `2.14.1`, a **vulnerable** version (Log4Shell, CVE-2021-44228) — `A03 supply chain`

4. The login page sends the password over plain `http` — `A04 cryptographic`