Secure Coding Practices

Bernard Nongpoh

Assistant Professor, Department of CSE



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

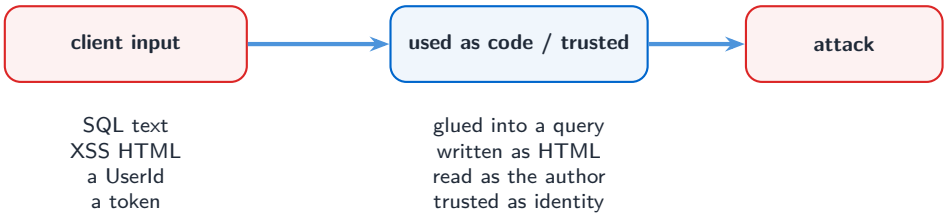
One Root Cause (1/3)

Every attack today was the same mistake, wearing different clothes.



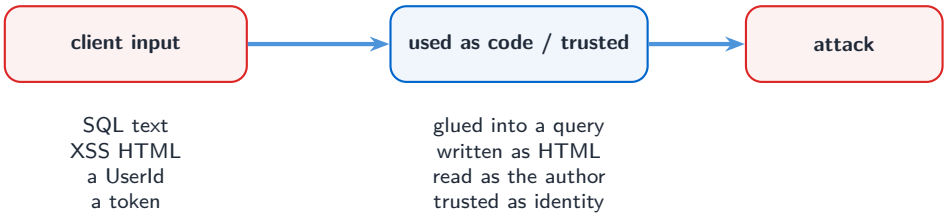
One Root Cause (2/3)

Every attack today was the same mistake, wearing different clothes.



One Root Cause (3/3)

Every attack today was the same mistake, wearing different clothes.



The server trusted what the client sent. Different inputs, one root cause.

The Fix Pattern (1/3)

Two habits fix almost everything we saw.

1. Keep data as data

never build code by gluing in input: parameters for SQL, escaping for HTML

The Fix Pattern (2/3)

Two habits fix almost everything we saw.

1. Keep data as data

never build code by gluing in input: parameters for SQL, escaping for HTML

2. Decide trust on the server

identity, price, author, role come from the server, never from the request

OWASP Cheat Sheet Series

The Fix Pattern (3/3)

Two habits fix almost everything we saw.

1. Keep data as data

never build code by gluing in input: parameters for SQL, escaping for HTML

2. Decide trust on the server

identity, price, author, role come from the server, never from the request

Now we find each broken line in Juice Shop and choose the fix. Keep the **score board** open.

Using the Coding Challenges (1/2)



Juice Shop shows you its own code for each bug you solved.

1. On the **score board**, a solved challenge with a <> icon has a coding challenge
2. **Find it**: the real source appears; click the **vulnerable line**
3. **Fix it**: choose the right fix from several options; Juice Shop checks it
4. For a wrong fix, read **why it fails**, then try again

Using the Coding Challenges (2/2)



Juice Shop shows you its own code for each bug you solved.

1. On the **score board**, a solved challenge with a <> icon has a coding challenge
2. **Find it**: the real source appears; click the **vulnerable line**
3. **Fix it**: choose the right fix from several options; Juice Shop checks it
4. For a wrong fix, read **why it fails**, then try again

We do four together: SQL injection, XSS, authentication, misconfiguration.

SQL Injection: Find the Line, Fix It (1/3)



Challenge *Login Admin*: the login query.

```
vulnerable
query("...email = '" + email + "'...")
```

input glued into the SQL text

Juice Shop *Login Admin* coding challenge; OWASP SQLi Prevention

SQL Injection: Find the Line, Fix It (2/3)



Challenge *Login Admin*: the login query.

```
vulnerable
query("...email = '" + email + "'...")
```

input glued into the SQL text

```
fix
query("...email = :email...",
  { replacements: { email } })
```

input travels as a parameter, never code

Juice Shop *Login Admin* coding challenge; OWASP SQLi Prevention

SQL Injection: Find the Line, Fix It (3/3)



Challenge *Login Admin*: the login query.

```
vulnerable
query("...email = '" + email + "'...")
```

input glued into the SQL text

```
fix
query("...email = :email...",
  { replacements: { email } })
```

input travels as a parameter, never code

Juice Shop uses a template string; same mistake. Fix: a **parameterized query**.

Juice Shop *Login Admin* coding challenge; OWASP SQLi Prevention

Cross-Site Scripting: Find the Line, Fix It (1/3)



Challenge *DOM XSS*: the search box handler.

```
vulnerable
searchValue =
  sanitizer.bypassSecurityTrustHtml(q)
```

turns off the framework's escaping

Juice Shop *DOM XSS* coding challenge; OWASP XSS Prevention

Cross-Site Scripting: Find the Line, Fix It (2/3)



Challenge *DOM XSS*: the search box handler.

vulnerable

```
searchValue =
  sanitizer.bypassSecurityTrustHtml(q)
```

turns off the framework's escaping

fix

```
searchValue = q
  // the framework escapes it
```

q is shown as text, not HTML

Juice Shop *DOM XSS* coding challenge; OWASP XSS Prevention

Cross-Site Scripting: Find the Line, Fix It (3/3)



Challenge *DOM XSS*: the search box handler.

vulnerable

```
searchValue =
  sanitizer.bypassSecurityTrustHtml(q)
```

turns off the framework's escaping

fix

```
searchValue = q
  // the framework escapes it
```

q is shown as text, not HTML

The bug is **switching protection off**. The fix is to leave it **on**: escape by default.

Juice Shop *DOM XSS* coding challenge; OWASP XSS Prevention

Weak Authentication: Find the Line, Fix It (1/3)



Challenge *Password Strength*: how passwords are stored.

```
vulnerable
hash = md5(password)
admin seeded "admin123"
```

fast hash, weak default, no lockout

Juice Shop *Password Strength* coding challenge; OWASP Password Storage

Weak Authentication: Find the Line, Fix It (2/3)



Challenge *Password Strength*: how passwords are stored.

```
vulnerable
hash = md5(password)
admin seeded "admin123"
```

fast hash, weak default, no lockout

```
fix
hash = bcrypt(password)
+ strength rules + lockout
```

slow hash, strong rules, limited tries

Juice Shop *Password Strength* coding challenge; OWASP Password Storage

Weak Authentication: Find the Line, Fix It (3/3)



Challenge *Password Strength*: how passwords are stored.

```
vulnerable
hash = md5(password)
admin seeded "admin123"
```

fast hash, weak default, no lockout

```
fix
hash = bcrypt(password)
+ strength rules + lockout
```

slow hash, strong rules, limited tries

MD5 is fast, so guessing is fast. A slow hash and a lockout make guessing impractical.

Juice Shop *Password Strength* coding challenge; OWASP Password Storage

Misconfiguration: Find the Line, Fix It (1/3)



Challenge *Confidential Document*: a setting, not a logic bug.

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

lists every file in the folder

Juice Shop *Confidential Document* coding challenge; OWASP Hardening

Misconfiguration: Find the Line, Fix It (2/3)



Challenge *Confidential Document*: a setting, not a logic bug.

vulnerable

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

lists every file in the folder

fix

```
// remove the /ftp listing
serve only the files you mean to
```

nothing extra is exposed

Juice Shop *Confidential Document* coding challenge; OWASP Hardening

Misconfiguration: Find the Line, Fix It (3/3)



Challenge *Confidential Document*: a setting, not a bug.

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

lists every file in the folder

```
fix
// remove the /ftp listing
serve only the files you mean to
```

nothing extra is exposed

The code was correct; the **setting** opened the door. Turn off what you do not need.

Juice Shop *Confidential Document* coding challenge; OWASP Hardening

Every Fix Is the Same Idea (1/6)

Attack	Root cause	Fix
SQL injection	input glued into a query	parameterized query

Every Fix Is the Same Idea (2/6)

Attack	Root cause	Fix
SQL injection	input glued into a query	parameterized query
XSS	input written as HTML	escape output; no bypass

Every Fix Is the Same Idea (3/6)

Attack	Root cause	Fix
SQL injection	input glued into a query	parameterized query
XSS	input written as HTML	escape output; no bypass
broken access control	trusting a client id	check the owner on the server

Every Fix Is the Same Idea (4/6)

Attack	Root cause	Fix
SQL injection	input glued into a query	parameterized query
XSS	input written as HTML	escape output; no bypass
broken access control	trusting a client id	check the owner on the server
weak auth	fast hash, weak default	slow hash, strong rules, lockout

Every Fix Is the Same Idea (5/6)

Attack	Root cause	Fix
SQL injection	input glued into a query	parameterized query
XSS	input written as HTML	escape output; no bypass
broken access control	trusting a client id	check the owner on the server
weak auth	fast hash, weak default	slow hash, strong rules, lockout
misconfiguration	a feature left on	turn it off

Every Fix Is the Same Idea (6/6)

Attack	Root cause	Fix
SQL injection	input glued into a query	parameterized query
XSS	input written as HTML	escape output; no bypass
broken access control	trusting a client id	check the owner on the server
weak auth	fast hash, weak default	slow hash, strong rules, lockout
misconfiguration	a feature left on	turn it off
session hijacking	readable, long-lived token	HttpOnly, short lifetime

Two habits cover the table: **keep data as data**, and **decide trust on the server**.

Summary (1/5)

You attacked Juice Shop all day. Now you can fix it.

- **Keep data as data**: parameters for SQL, escaping for HTML

Summary (2/5)

You attacked Juice Shop all day. Now you can fix it.

- **Keep data as data**: parameters for SQL, escaping for HTML
- **Decide trust on the server**: identity, price, author, role

Summary (3/5)

You attacked Juice Shop all day. Now you can fix it.

- **Keep data as data**: parameters for SQL, escaping for HTML
- **Decide trust on the server**: identity, price, author, role
- **Store secrets well**: slow password hashes, protected tokens

Summary (4/5)

You attacked Juice Shop all day. Now you can fix it.

- **Keep data as data**: parameters for SQL, escaping for HTML
- **Decide trust on the server**: identity, price, author, role
- **Store secrets well**: slow password hashes, protected tokens
- **Turn off what you do not need**: every feature is attack surface

OWASP Cheat Sheet Series; Pwning OWASP Juice Shop

Summary (5/5)

You attacked Juice Shop all day. Now you can fix it.

- **Keep data as data**: parameters for SQL, escaping for HTML
- **Decide trust on the server**: identity, price, author, role
- **Store secrets well**: slow password hashes, protected tokens
- **Turn off what you do not need**: every feature is attack surface

Hands-on next: open the coding challenges you solved today and fix each one.

OWASP Cheat Sheet Series; Pwning OWASP Juice Shop