IERG4210 Web Programming and Security

Course Website: https://course.ie.cuhk.edu.hk/~ierg4210/
Live FB Feedback Group: https://fb.com/groups/ierg4210.2015spring/

Web Application Security II
Lecture 9

Dr. Adonis Fung
phfung@ie.cuhk.edu.hk

Information Engineering, CUHK
Product Security Engineering, Yahoo!

Agenda

- Web Application Vulnerabilities
  - A1-Injection Flaws
    - Malicious inputs executed as code
    - Examples: SQL Injection, OS Command Injection, A2-Cross-Site Scripting (covered), File-based XSS Injection, CSS Injection, A10-Unvalidated Redirects and Forwards, Dynamic Code Execution
  - Parameter Tampering Attacks
    - Malicious user inputs to bypass logic
    - Examples: A4-Insecure Direct Object References, Path Traversal Vulnerability/(2007-A3)Malicious File Execution, A7-Missing_Function_Level_Access_Control, Bypassing Client-side Restrictions

Note: A[1-10] refers to the items in OWASP Top 10 Web Application Security Risks, 2013

INJECTION FLAWS

OWASP Top 10 Application Security Risks

2010	2013
A1-Injection	**A1-Injection**
A2-Cross Site Scripting (XSS)	A2-Broken Authentication and Session Management
A3-Broken Authentication and Session Management	A3-Cross-Site Scripting (XSS)
A4-Insecure Direct Object References	A4-Insecure Direct Object References
A5-Cross Site Request Forgery (CSRF)	A5-Security Misconfiguration
A6-Security Misconfiguration	A6-Sensitive Data Exposure
A7-Insecure Cryptographic Storage	A7-Missing Function Level Access Control
A8-Failure to Restrict URL Access	A8-Cross-Site Request Forgery (CSRF)
A9-Insufficient Transport Layer Protection	A9-Using Components with Known Vulnerabilities
A10-Unvalidated Redirects and Forwards	A10-Unvalidated Redirects and Forwards

- References: https://www.owasp.org/index.php/Top_10_2010-Main https://www.owasp.org/index.php/Top_10_2013

Injection Flaws

- General Cause:
  - Some special characters from user inputs evaluated as executable commands instead of textual values
- General Defense:
  - Apply Rigorous Whitelist Validations
- Many different kinds of injection
  1. SQL Injection
  2. File-based XSS Injection
  3. Shell Command Injection
  4. CSS Injection
  5. A10-Unvalidated Redirects and Forwards
  6. Dynamic Code Execution
  - Covered: XSS;
  - Others: HTTP Headers, Cookies, XPath, SMTP, etc

1. SQL Injection

- Database holds a lot of sensitive data
  - For example, users' privacy, passwords, credit card numbers
  - Makes it easily become an attractive target
- Cause: Using unvalidated user-supplied input with SQL
- Example Vulnerability:
  - As simple as directly concatenating user input with SQL statement:

```
db.query("SELECT * FROM products
            WHERE catid = " + req.params.catid);
```

- Consequences:
  - Confidentiality: Information Leakage
  - Integrity: Modifying/Deleting DB Data

1. Some SQL Injection Attacks

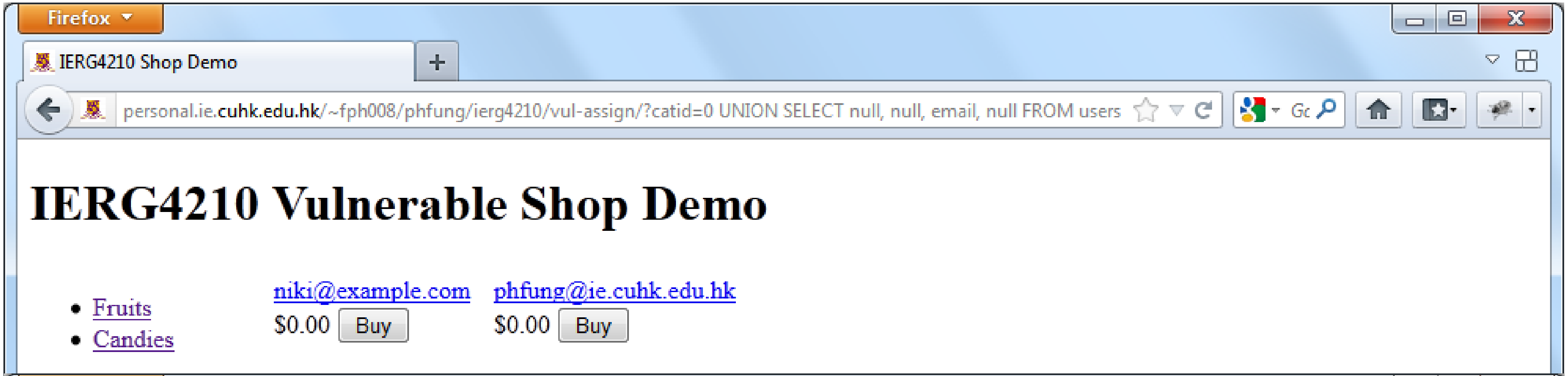
- Given a vulnerability that directly concats user-input:
  - To get all products: use `1 OR 1=1` in `$_GET["catid"]`
    - `SELECT * FROM products WHERE catid = 1 OR 1=1`
  - To steal users' privacy:
    1. Brute-force the number of columns in-use by the original table:
       - Use: `0 UNION SELECT null, null` (append `,null` until no error)

       -
         ```
         SELECT * FROM products WHERE catid =
                 0 UNION SELECT null, null, null, null
         ```
    2. Guess table and column names (No need to guess for open-source proj.)
    3. In our example, only the third field is a textfield:

       -
         ```
         SELECT * FROM products WHERE catid =
             0 UNION SELECT null, null, email, null FROM users
         ```

       -
         ```
         SELECT * FROM products WHERE catid =
             0 UNION SELECT null, null, password, null FROM users
         ```

1. Capturing Email and Password



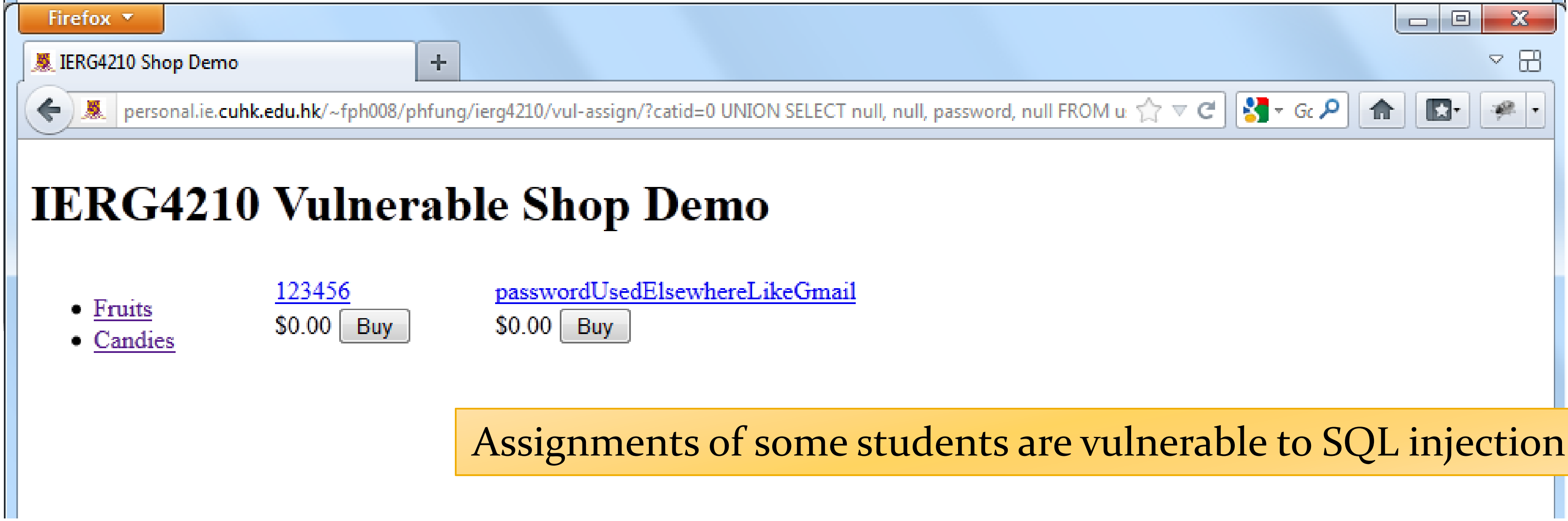
personal.ie.cuhk.edu.hk/~fph008/phfung/ierg4210/vul-assign/?catid=0 UNION SELECT null, null, email, null FROM users

IERG4210 Vulnerable Shop Demo

- Fruits
- Candies

niki@example.com $0.00

phfung@ie.cuhk.edu.hk $0.00



personal.ie.cuhk.edu.hk/~fph008/phfung/ierg4210/vul-assign/?catid=0 UNION SELECT null, null, password, null FROM u

IERG4210 Vulnerable Shop Demo

- Fruits
- Candies

123456 $0.00

passwordUsedElsewhereLikeGmail $0.00

Assignments of some students are vulnerable to SQL injection!

1. Other SQL Injection Attacks

- Attackers can do a lot more than that
  - Even database-specific attacks: SQLite, MySQL, MSSQL, Oracle
- More Examples:
  - Commenting the statement using `--` after the injection point
    - `SELECT * FROM products WHERE pid = 0 UNION SELECT 1, 1, email, 1 FROM users ;-- LIMIT 1`
  - Breaking out from double/single/grave accent quotes
  - Destroying tables using `DROP TABLE users`
  - Exposing database schema
  - Dumping all data into a single file for easy download
  - Many more…

Reference: D. Stuttard, and M. Pinto, "The Web Application Hacker's Handbook", Wiley, 2nd Edition, 2011. Chapter 9

1. Defending SQL Injection

- Use Prepared Statements properly for every SQL call
  - Avoid concatenating the statement with user-supplied parameters
  - Make all data properly quoted and escaped at placeholders (?), hence no chance to inject SQL commands

```
db.query('INSERT INTO categories (name) VALUES (?)',
    [req.body.name],
    function (error, result) {/** process the results **/}
);
```

- What cannot be prepared: Table, Column names, ASC, DESC, etc
  - In some cases, apps may need to vary them
  - MAP user inputs to some hardcoded SQL before concatenation:

```
var order = req.params.order && req.params.order == 'desc' ? ' DESC' : '';
db.query('SELECT * FROM products ORDER BY price' + order /*, function(){...} */});
```

  - Use rigorous whitelist validations

1. Defense-in-Depth for SQL Injections

- Least-privilege approach
  - For the DB user account used by public-facing/risky applications,
    - No root privilege, and as restrictive as possible
    - Minimized permissions for specific tables
- Compartmentalization / Separation of Privilege
  - For data that have higher security needs,
    - Separate databases of different sensitivity, accessible by different DB users
- Promote Privacy
  - Always encrypt or apply one-way hash functions for sensitive data
- Others
  - Backup
  - Upgrade to the latest DB version; No unnecessary packages/extensions
  - (A9-Using Components with Known Vulnerabilities)

2. File-based XSS Injection (1/3)

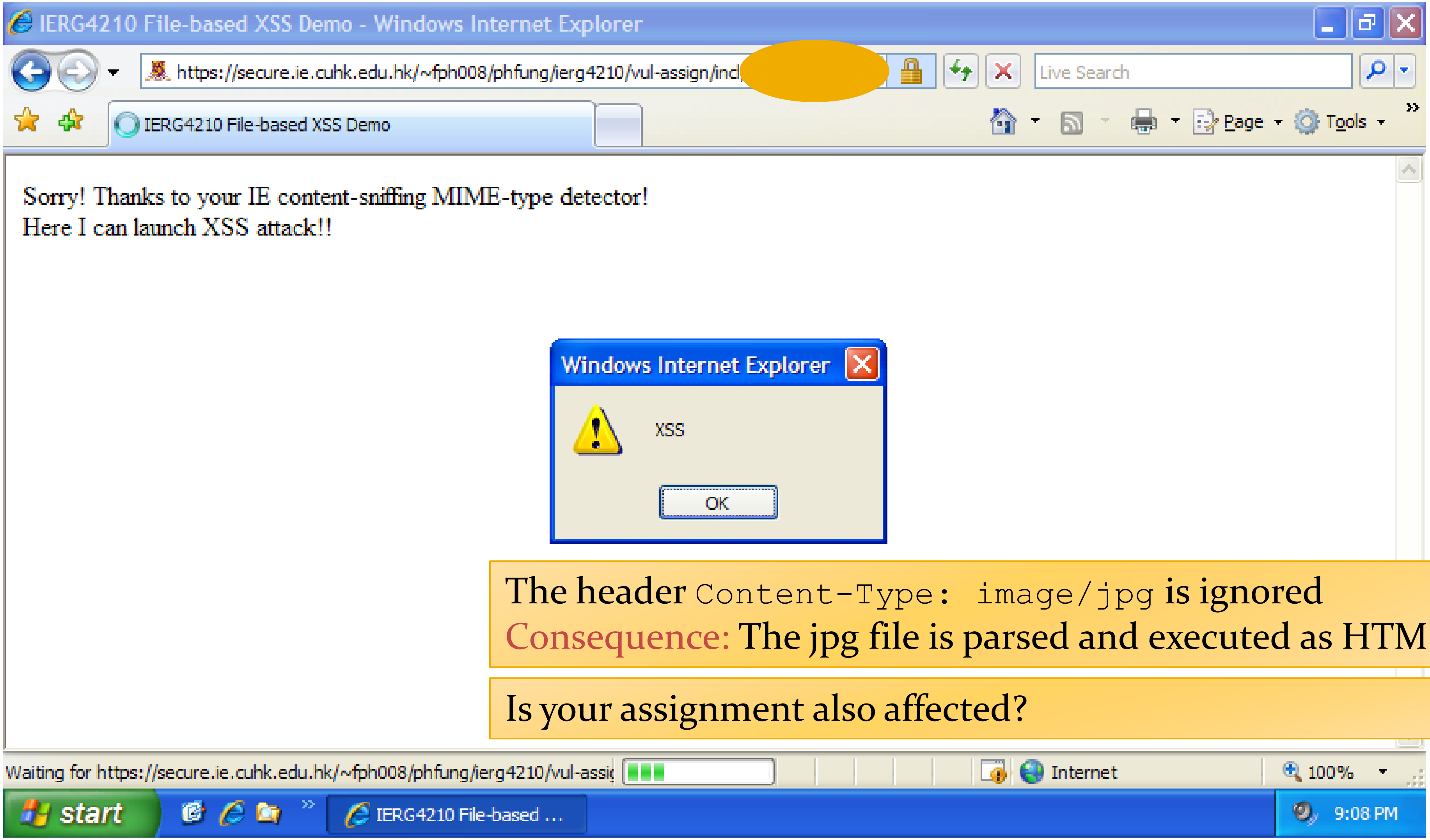
- Cause: An application is vulnerable if it allows file upload yet does not check the MIME-type at server

- Your assignments are very likely vulnerable:
  - Attacker's goal: Run HTML file under your domain/origin
  - Image type check can often be easily bypassed
    - file.mimetype == 'image/jpeg' // is unreliable
  - Attacker can bypass it by:
    - Browser sends Content-Type based on file extension (e.g., .html > .jpg)
    - Attacker can craft his own HTTP request

How about deducting the marks for "No XSS Vulnerability"? :)
Anybody aware of this before?

(Demo & Code)
https://gist.github.com/adon-at-work/26c8a8e0a1aee5ded03c

2. File-based XSS Injection (2/3)


IERG4210 File-based XSS Demo - Windows Internet Explorer
https://secure.ie.cuhk.edu.hk/~fph008/phfung/ierg4210/vul-assign/incl
Live Search
IERG4210 File-based XSS Demo
Page
Tools
Windows Internet Explorer
OK
Waiting for https://secure.ie.cuhk.edu.hk/~fph008/phfung/ierg4210/vul-assig
Internet
100%
start
IERG4210 File-based ...
9:08 PM

Sorry! Thanks to your IE content-sniffing MIME-type detector!
Here I can launch XSS attack!!

XSS

The header `Content-Type: image/jpg` is ignored
Consequence: The jpg file is parsed and executed as HTML

Is your assignment also affected?

2. File-based XSS Injection (3/3)

- Cause:
  - Given no Content-type Response header is set
    - Browser detects it by sniffing the content
    - Even worse, older browsers can even disregard it, and guess a "right" one
- Consequence:
  - User-uploaded file is executed as HTML in victim domain
    - Attacker's can contribute a HTML file (with its extension equals .jpg)
    - Content-Type header can be ignored in IE 7 or below, Firefox 3 or below, Safari 3.1 and older Google Chrome
  - Therefore, file-based XSS

Reference: A. Barth, J. Caballero, and D. Song, "Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves", IEEE S&P, 2009

2. Defending File-based XSS Injection

- As a user:
  - To protect yourself, upgrade to the latest versions of your browser
  - Greet your friends who are fans of Internet Explorer, Good luck!!   :)
- Defense-in-Depth as an application developer:
  - Host user-uploaded content in a separated origin
    - User-supplied file is executed in another origin or IP address
    - Even for users of outdated browsers, these files cannot launch XSS
    - Example 1: Modern webmail all use a separate domain for attachment
      - ymail.com, googleusercontent.com, etc
    - Example 2: Use the Amazon S3 provided location as in the sample code
  - Configure proper response headers
    - Invented by IE: `X-Content-Type-Options: nosniff`
    - Tell the browser not to render but download it
      `Content-type: application/octet-stream`

3. Shell Command Injection

- Example Vulnerability:
  - If you're to write a DNS lookup application
  - Intuitively, you want to use the `nslookup` command
  - Here is an insecure application (demo),

```
var exec = require('child_process').exec;
exec('nslookup ' + req.params.domainName,
  function (error, stdout, stderr) {
    console.log('nslookup result:\n', stdout);
});
```

- Cause:
  - Again, `req.params.domainName` is not properly escaped or validated
- Consequence:
  - Execute commands on behalf of the user who runs `node app.js`
    - Hence, using his same privilege. In beanstalk/EC2, it's `ec2-user`

3. Defending Shell Command Injection

- Defenses:
  - Use rigorous whitelist validations
  - Escape the data. NodeJS's spawn can escape string arguments:

```
var spawn = require('child_process').spawn,
    nslookup = spawn('nslookup', [req.params.domainName]);
nslookup.stdout.on('data', function (data) {
  console.log('nslookup result:\n', data.toString());
});
```

- Shell/OS-related Functions (Avoid whenever possible):
  - `require('child_process')`

4. CSS Injection (1/2)

- Cause: Forcing browsers to parse HTML as CSS
  - HTML Sanitizers may not be helpful
- Consequence: Data leakage across origin
- Example Vulnerability
  - Attacker send an email with subject `{}*{font-family:'` to victim@gmail.com
  - Victim opens the email. The HTML looks like so:

```
<html><body>
…<td>Subject: {}*{font-family:'</td>…
<form action="http://gmail.com/forwardemail" method="POST">
<input type="hidden" name="nonce" value="SD9fsjdf35HE4f">
<input type="submit" value="Forward">
…
</form>
…</body></html>
```

4. CSS Injection (2/2)

- Example Vulnerability (cont.)
  - Victim visits a malicious page (e.g. by clicking a link in the email):

```
<link rel="stylesheet" href="https://gmail.com/inbox"
type="text/css" />
<script>
  document.write(document.body.currentStyle.fontFamily);
</script>
```

  - Vulnerable browsers are told it's a "CSS stylesheet"
  - So, skip `<...>` contents and parse whatever understandable as CSS
  - Given `{}*{font-family:'`, anything beyond is assigned to `fontFamily`, e.g. the CSRF nonce stored in a hidden field
- Affected Browsers:
  - Old IE and some obsolete versions of other browsers

(Midterm/Exam: MIME detection by content-sniffing v.s. CSS injection)

4. Defending CSS Injection

- As a user:
  - To protect yourself, upgrade to the latest versions of your browser
  - Greet your friends who are fans of Internet Explorer, Good luck!! :)
- As a developer:
  - Not much to do in this particular case
  - ALWAYS apply whitelist validation on users' input whenever possible!!
    - Blacklist output sanitization is subject to future unexpected flaws

OWASP Top 10 Application Security Risks

2010	2013
A1-Injection	A1-Injection
A2-Cross Site Scripting (XSS)	A2-Broken Authentication and Session Management
A3-Broken Authentication and Session Management	A3-Cross-Site Scripting (XSS)
A4-Insecure Direct Object References	A4-Insecure Direct Object References
A5-Cross Site Request Forgery (CSRF)	A5-Security Misconfiguration
A6-Security Misconfiguration	A6-Sensitive Data Exposure
A7-Insecure Cryptographic Storage	A7-Missing Function Level Access Control
A8-Failure to Restrict URL Access	A8-Cross-Site Request Forgery (CSRF)
A9-Insufficient Transport Layer Protection	A9-Using Components with Known Vulnerabilities
A10-Unvalidated Redirects and Forwards	**A10-Unvalidated Redirects and Forwards**

- References: https://www.owasp.org/index.php/Top_10_2010-Main
  https://www.owasp.org/index.php/Top_10_2013

5. Unvalidated Redirects and Forwards (1/3)

- Cause: Using unvalidated user-supplied input in web page redirections and forwards
  - E.g., When session expired, records a URL, redirect back after login

- FYI, redirections can be made in various languages
  - In NodeJS: res.location(), res.redirect(), res.set('Location', ...), etc
  - In JS: document.location, document.URL, document.open, window.location.href, window.navigate(), window.open(), window.location.replace(), etc...
  - In HTML, inside <head>:
    <meta http-equiv="Refresh" content="0; url=somewhere.html" />
  - In PHP: header("Location: somewhere.php"), header("Refresh: 0; url=somewhere.php"), etc
  - In Apache: RewriteRule, Redirect, Header, etc

5. Unvalidated Redirects and Forwards (2/3)

- Example Vulnerability 1: Creating Phishy URLs
  - Attacker can email the following URLs to victims:
    - `http://vul.com/login-success?url=%2F%2Fattack.com`
    - `http://vul.com/login-success?url=%2F%2FvuI.com`
    - `http://easyaccess.lib.cuhk.edu.hk/login?url=http://www.google.com.hk/search?q=don%27t+hack+cuhk` :)
  - Victims thought they are visiting vul.com and feel safe
  - But instead they got quickly redirected to `vu`**`I`**`.com` or `attack.com`
- Defenses:
  - AVOID incorporating user-supplied input in page redirections
  - If unavoidable, use rigorous whitelist validation

5. Unvalidated Redirects and Forwards (3/3)

- Example Vulnerability 2: HTTP Response Splitting
  - In `login-success?url=somewhere.php`

    ```
    header("Location: " . $_GET['url']); exit();
    ```

    Note: remember to call `exit()` after redirection headers
  - Attacker uses `%0d%0a` for carriage return (CRLF or \r\n):
    `http://vul.com/login-success?url=somewhere.php%0D%0ASet-Cookie%3A%20token%3Dreplaced`
  - The HTTP response turns out to be:

    ```
    HTTP/1.1 302 Object temporarily moved
    Location: somewhere.php
    Set-Cookie: token=replaced
    ...
    ```

  - Defense: Modern framework fix it by stripping line breaks from HTTP Response Header configurations

6. Dynamic Code Execution Vulnerability

- Cause:
  - Using unvalidated user-supplied input for dynamic code execution
- Consequence:
  - Most language has a evaluate feature, which evaluates a string and run it as program code. JavaScript provides such feature thru `eval()`.
- Defenses
  - Simply avoid eval() whenever possible
  - Make sure no user inputs, or apply vigorous validations

A Threat

Outdated JavaScript tutorials teach people to use `eval('('+json+')')` to parse JSON. Instead, use the native API `JSON.parse(json)` supported by new browsers; even for old ones, use: http://json-sans-eval.googlecode.com/svn/trunk/src/json_sans_eval.js

"IERG4210 students shouldn't easily trust online tutorials"

PARAMETER TAMPERING ATTACKS

OWASP Top 10 Application Security Risks

2010	2013
A1-Injection	A1-Injection
A2-Cross Site Scripting (XSS)	A2-Broken Authentication and Session Management
A3-Broken Authentication and Session Management	A3-Cross-Site Scripting (XSS)
A4-Insecure Direct Object References	**A4-Insecure Direct Object References**
A5-Cross Site Request Forgery (CSRF)	A5-Security Misconfiguration
A6-Security Misconfiguration	A6-Sensitive Data Exposure
A7-Insecure Cryptographic Storage	A7-Missing Function Level Access Control
A8-Failure to Restrict URL Access	A8-Cross-Site Request Forgery (CSRF)
A9-Insufficient Transport Layer Protection	A9-Using Components with Known Vulnerabilities
A10-Unvalidated Redirects and Forwards	A10-Unvalidated Redirects and Forwards

- References: https://www.owasp.org/index.php/Top_10_2010-Main
  https://www.owasp.org/index.php/Top_10_2013

Insecure Direct Object References

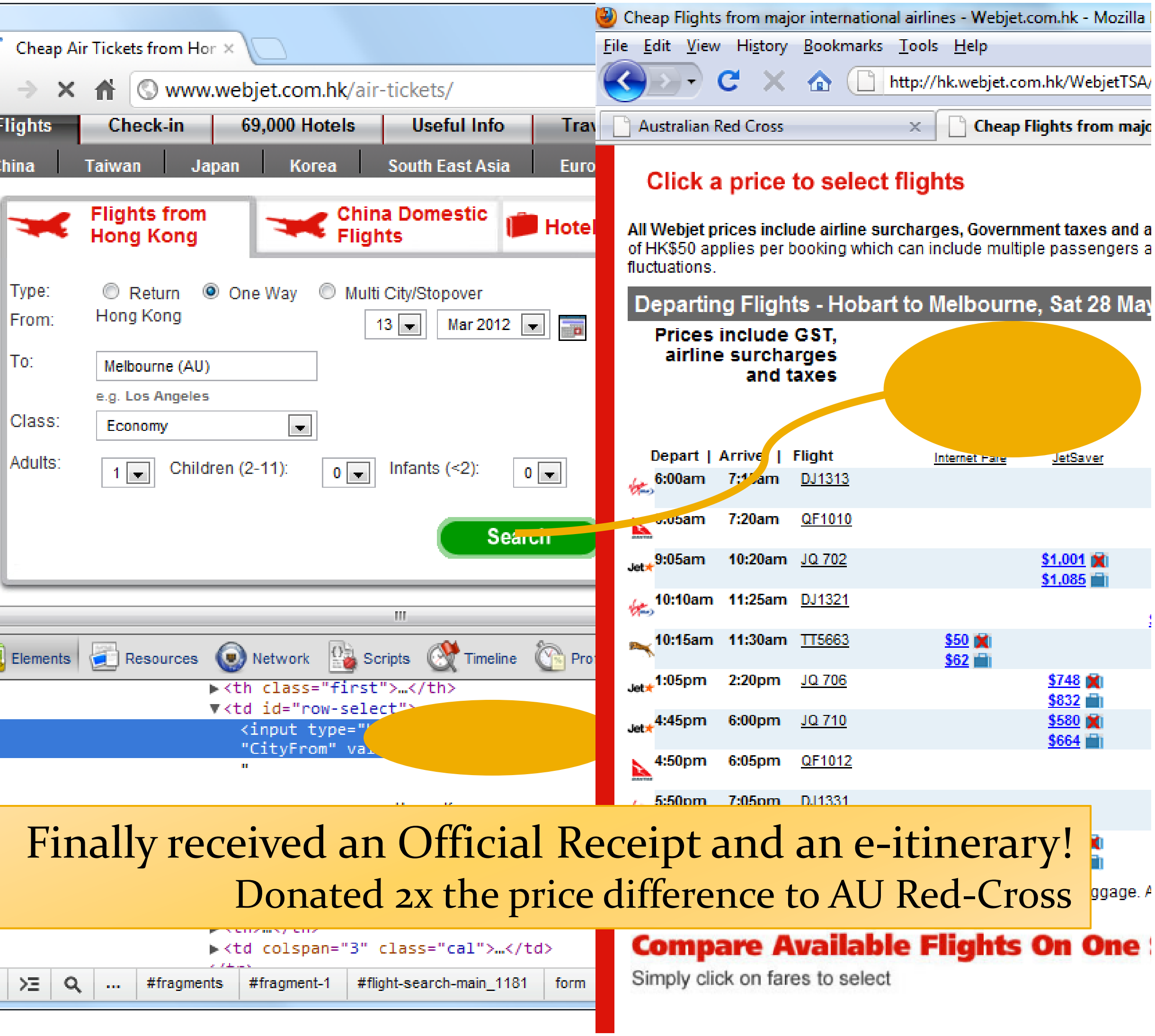
- Cause: Vulnerable applications expose the actual name or key of an object without proper authorization control
  - Object Examples: File/folder, Database Key, Form parameters, etc
- Also known as Parameter Tampering Attack

- Consequence: Attackers can access some internal objects without authorization by parameter tampering
- Many different kinds of vulnerabilities:
  1. Path Traversal Vulnerability/Malicious File Execution
  2. A7-Missing_Function_Level_Access_Control
  3. Bypassing Client-side Restrictions
  - More to be discussed in a later lecture (e-banking case studies)

1. Bypassing Client-side Restrictions

- Cause: Client-side Restrictions and Validations can NEVER be enforced securely
- Attackers can bypass any client-side restrictions with Firebug
- Also known as CWE472: External Control of Assumed-Immutable Web Parameter
- Example Vulnerabilities:
  - Changing values of hidden field, radio button, checkbox or dropdown menu, etc
  - Rewriting/Bypassing some Javascript
- Defenses:
  - Apply server-side validations and sanitizations
  - Map internal objects to indirect object references

1. Tampering a Hidden Field



Cheap Air Tickets from Hor ×

www.webjet.com.hk/air-tickets/

Type: ○ Return ● One Way ○ Multi City/Stopover

From: Hong Kong 13 Mar 2012

To: Melbourne (AU)

e.g. Los Angeles

Class: Economy

Adults: 1 Children (2-11): 0 Infants (<2): 0

```
▶<th class="first">…</th>
▼<td id="row-select">
    <input type="
    "CityFrom" va
    "
▶<td colspan="3" class="cal">…</td>
```

Cheap Flights from major international airlines - Webjet.com.hk - Mozilla

http://hk.webjet.com.hk/WebjetTSA/

Click a price to select flights

All Webjet prices include airline surcharges, Government taxes and a of HK$50 applies per booking which can include multiple passengers a fluctuations.

Departing Flights - Hobart to Melbourne, Sat 28 May

Prices include GST, airline surcharges and taxes

Depart	Arrive	Flight	Internet Fare	JetSaver
6:00am	7:1 am	DJ1313		
.05am	7:20am	QF1010		
9:05am	10:20am	JQ 702		$1,001 $1,085
10:10am	11:25am	DJ1321		
10:15am	11:30am	TT5663	$50 $62	
1:05pm	2:20pm	JQ 706		$748 $832
4:45pm	6:00pm	JQ 710		$580 $664
4:50pm	6:05pm	QF1012		
5:50pm	7:05pm	DJ1331		

ggage. A

Compare Available Flights On One

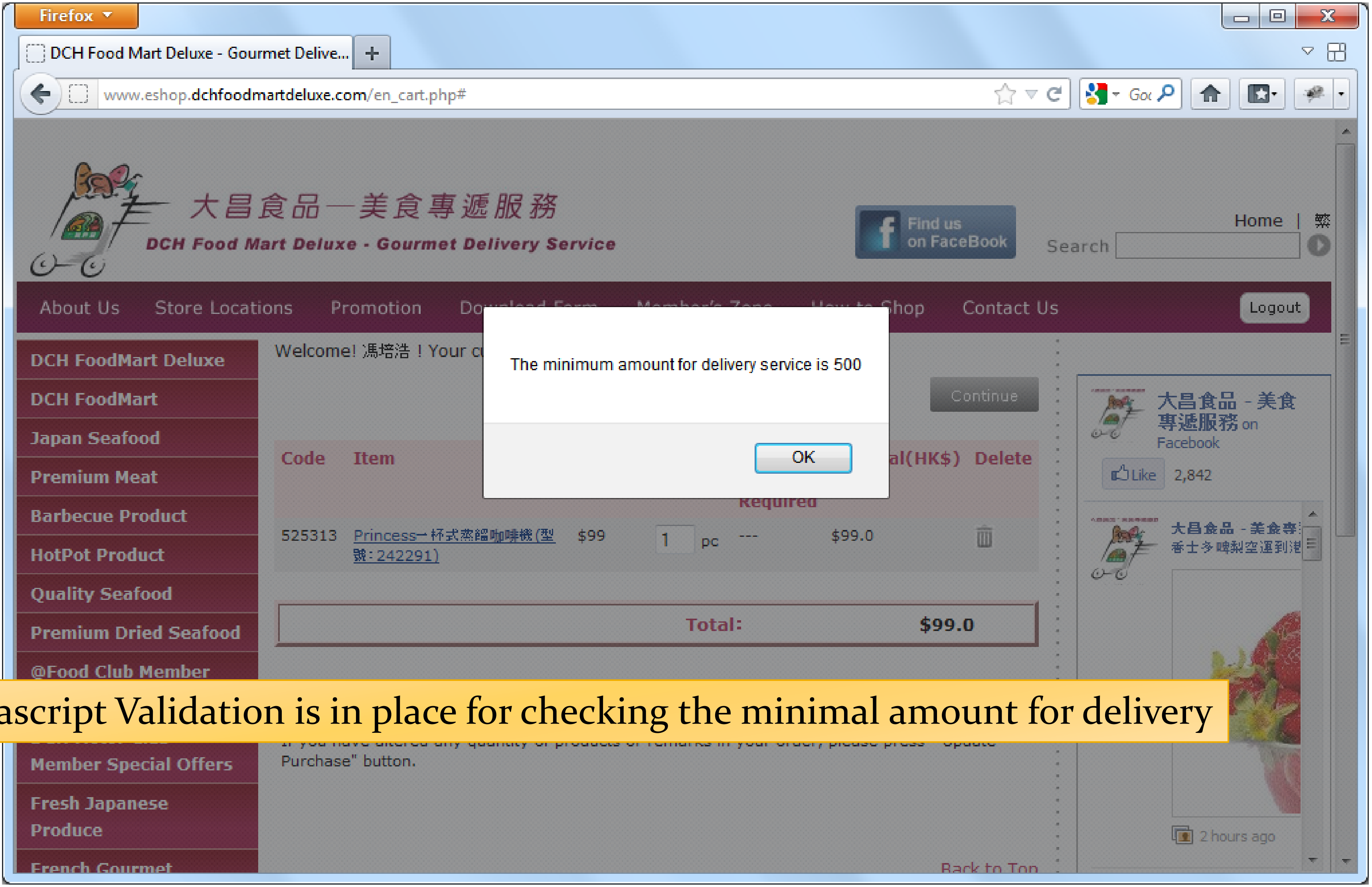
Simply click on fares to select

Finally received an Official Receipt and an e-itinerary!

Donated 2x the price difference to AU Red-Cross

- Uncovered and Patched in 2011
- Tampered a hidden field CityFrom
  - HKG → HBA (Hobert)
- The unexpected value triggered a currency calc. bug
- Purchased a cheaper ticket successfully (was 43.5% cheaper compared to webjet.com.au)

1. Bypassing Javascript Validations (1/3)



Welcome! 馮培浩 ! Your c

The minimum amount for delivery service is 500

OK

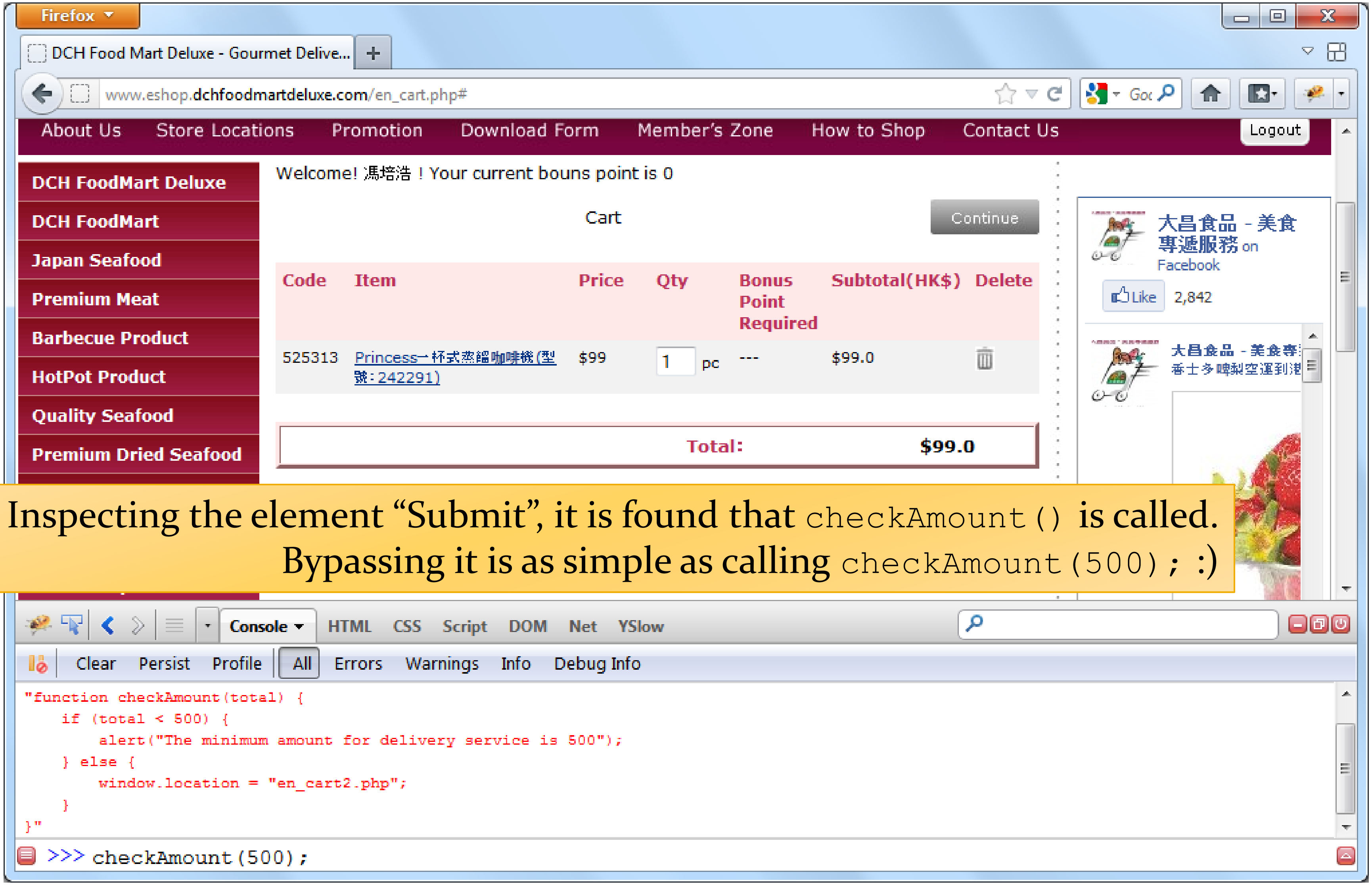
Code	Item				Required	al(HK$)	Delete
525313	Princess一杯式蒸餾咖啡機(型號:242291)	$99	1	pc	---	$99.0	

Total: $99.0

Purchase" button.

Javascript Validation is in place for checking the minimal amount for delivery

1. Bypassing Javascript Validations (2/3)



Welcome! 馮培浩 ! Your current bouns point is 0

Cart

Code	Item	Price	Qty	Bonus Point Required	Subtotal(HK$)	Delete
525313	Princess一杯式蒸餾咖啡機(型號:242291)	$99	1 pc	---	$99.0	

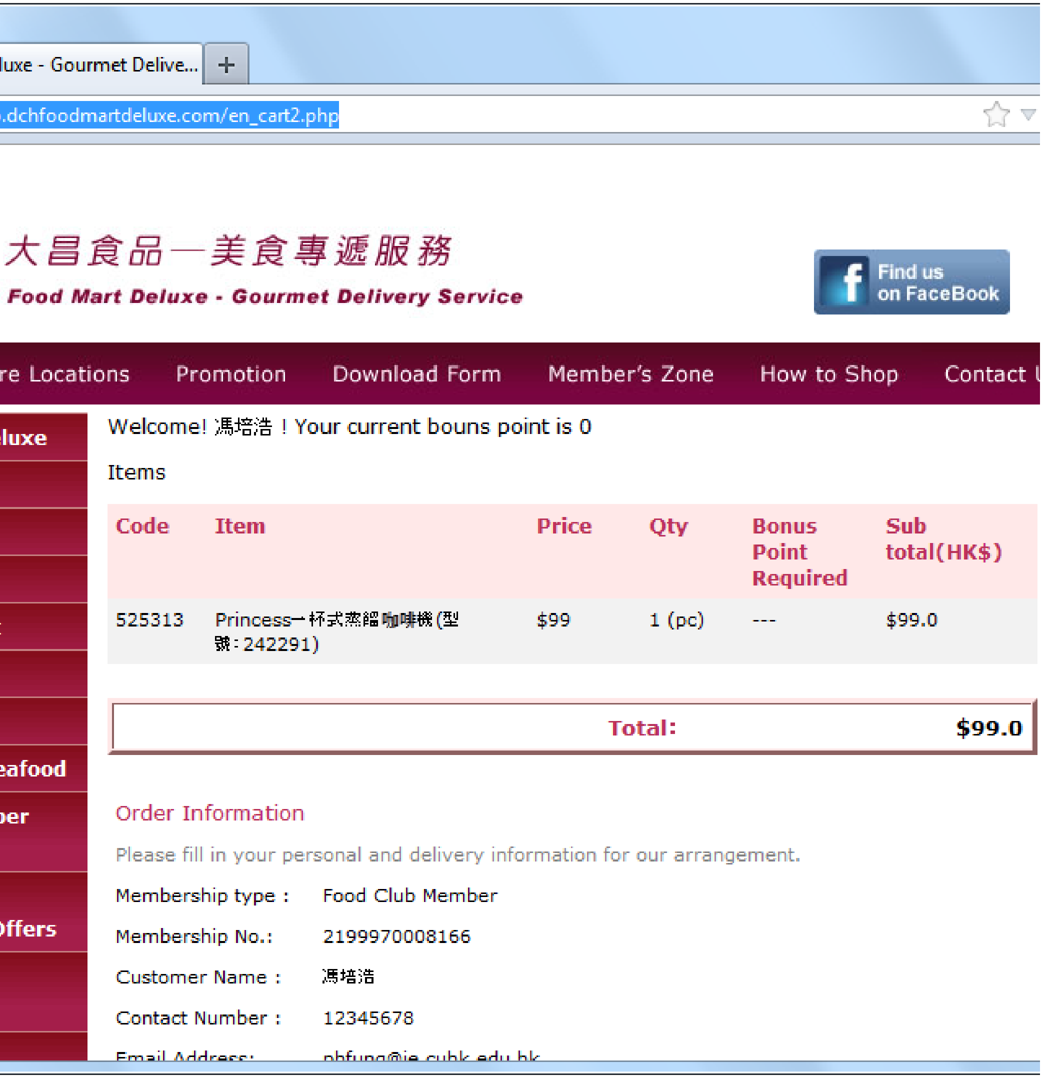
Total: $99.0

Inspecting the element "Submit", it is found that `checkAmount()` is called. Bypassing it is as simple as calling `checkAmount(500);` :)

```
"function checkAmount(total) {
    if (total < 500) {
        alert("The minimum amount for delivery service is 500");
    } else {
        window.location = "en_cart2.php";
    }
}"
```

```
>>> checkAmount(500);
```

1. Bypassing Javascript Validations (3/3)



Welcome! 馮培浩！Your current bouns point is 0

Items

Code	Item	Price	Qty	Bonus Point Required	Sub total(HK$)
525313	Princess一杯式蒸餾咖啡機(型號：242291)	$99	1 (pc)	---	$99.0

Total:	$99.0

Order Information

Please fill in your personal and delivery information for our arrangement.

Membership type :	Food Club Member
Membership No.:	2199970008166
Customer Name :	馮培浩
Contact Number :	12345678
Email Address:	phfung@ie.cuhk.edu.hk

- This example is good:
  - JS rewriting is simple and possible
  - Also demonstrated the A8-Failure to Restrict URL Access; can go straight to this page without knowing JS
  - Vulnerability: No minimum amount check at server-side and in this page
- This example is bad:
  - Limited impact; that's why I can show you. :)
  - Will likely be noticed upon delivery??

2. Path Traversal Vulnerability

- Cause: Using unvalidated user-supplied input in file path
  - Also known as Malicious File Execution ranked No. 3 in OWASP Top 10, 2007
- Example Vulnerabilities:

```
fs.createReadStream(req.params.lib);
require(req.params.lib);
```

- Attackers can supply the following for the `lib` parameter:
  - Traversing to the root and referencing an interesting file
    `../../../../../../../../../../etc/passwd`
    Note: OS often tolerate excessive use of traversal sequences ../
  - (Midterm/Exam) can `userInput.replace(/\.\.\//g, '')` solve this? Consider what will be stripped from `....//.`
  - When attacking Windows, attacker uses `..\` instead of `../`

2. Defending Path Traversal Vulnerability

- Resolve the file path before validation
  - https://nodejs.org/api/path.html#path_path_resolve_from_to
  - Hence, `path.resolve('incl/../../../../../../../../etc/passwd')` will return `/etc/passwd`
  - Much easier to validate whether it is still within the expected scope
- Use rigorous whitelist validation and avoid input sanitization
  - For instance, `/^\w+$/.test(req.params.lib)`
- Map user-supplied parameters to some hardcoded path
  - For instance,
    `req.params.lib = req.params.lib ? 'somewhere' : 'elsewhere’`

OWASP Top 10 Application Security Risks

2010	2013
A1-Injection	A1-Injection
A2-Cross Site Scripting (XSS)	A2-Broken Authentication and Session Management
A3-Broken Authentication and Session Management	A3-Cross-Site Scripting (XSS)
A4-Insecure Direct Object References	A4-Insecure Direct Object References
A5-Cross Site Request Forgery (CSRF)	A5-Security Misconfiguration
A6-Security Misconfiguration	A6-Sensitive Data Exposure
A7-Insecure Cryptographic Storage	**A7-Missing Function Level Access Control**
A8-Failure to Restrict URL Access	A8-Cross-Site Request Forgery (CSRF)
A9-Insufficient Transport Layer Protection	A9-Using Components with Known Vulnerabilities
A10-Unvalidated Redirects and Forwards	A10-Unvalidated Redirects and Forwards

- References: https://www.owasp.org/index.php/Top_10_2010-Main https://www.owasp.org/index.php/Top_10_2013

3. A7-Missing Function Level Access Control

- Cause: Vulnerable applications implemented insufficient authorization checks
- Attackers then access the following w/some educated guess:
  - "Hidden" admin pages: admin.html, admin.php, or /admin
    - Obfuscation does not guarantee security
  - Horizontal privilege escalation: profile?uid=4
    - User is supposed to access his own profile only, but attacker tampered uid
  - Hardcoded admin access: login?admin=1 or `Cookie: admin=1;`
    - Privilege escalated to admin rights
- Defenses:
  - Ensure EVERY page has implemented access right checks at server-side
  - Avoid hardcoding access right policies, e.g.,`if(DEBUG)admin=1`

Review on 1st Course Evaluation

- Thank you for the first evaluation
  - It is only examined by the instructor
  - Thanks for the opinions, good or bad

- Some Common and Known Opinions:
  - Heavy-loaded / “chur” in Cantonese :)
  - This is as expected and already made clear in the first lecture
  - Insufficient tutorials or assignment guidelines
  - It’s a painful step that every programmer (incl. me) must go through; Can understand your difficulties, but surely you will be rewarded