



---

# SECURE SOFTWARE DEVELOPMENT

---

Threat Modelling, Design and Implementation of a Secure Website



APRIL 8, 2015

WAYNE CARUANA  
1BSC2S

## Table of Contents

Task 1 – Definition of requirements, Threat Modelling and Design (P2.1, P2.2) .....	3
Assets .....	3
Trust Levels .....	4
DFD Level 0.....	4
Entry Points .....	5
Threats .....	6
Task 2 – DFDs in Threat Modelling Document (M1.4) .....	9
Task 3 – Implementation of Secure Website (P3.1).....	10
Task 4 – Documentation of security patterns and practices (P3.2, P4.3) .....	10
Task 5 – Implementation of Further security techniques in website (M1.5) .....	10
Task 6 – Documentation of alternative security patterns (M1.3) .....	11
PayPal SDK Alternative.....	11
Stripe .....	11
Custom Authorization Alternative .....	12
Token.....	12
Action Filters .....	13
Task 7 – Security Patterns – Argue on what you cannot protect against (D1.2) .....	14
Authentication .....	14
Accessing unencrypted access token.....	14
Facebook data gets stolen .....	14
Facebook Cookies .....	14
Secure Session Management.....	15
Key logger.....	15
Viruses.....	15
Physical Attempts.....	15
Task 8 – Testing and Reviewing a 3rd Party application (P4.1, P4.2, P4.4) .....	16
Black Box Testing .....	16
Code Review.....	20
Strong Point in Code .....	20
Weak Point in Code.....	21
Task 9 – Perform Attacks on a third party application and document them (D1.3).....	22
A1-Injection.....	22
A2 – Broken Authentication and Session Management .....	23
A3 – Cross Site Scripting (XSS).....	24
A4- Insecure Direct Object Reference.....	25

A6 – Sensitive Data Exposure.....	26
Bibliography .....	27

## Task 1 – Definition of requirements, Threat Modelling and Design (P2.1, P2.2)

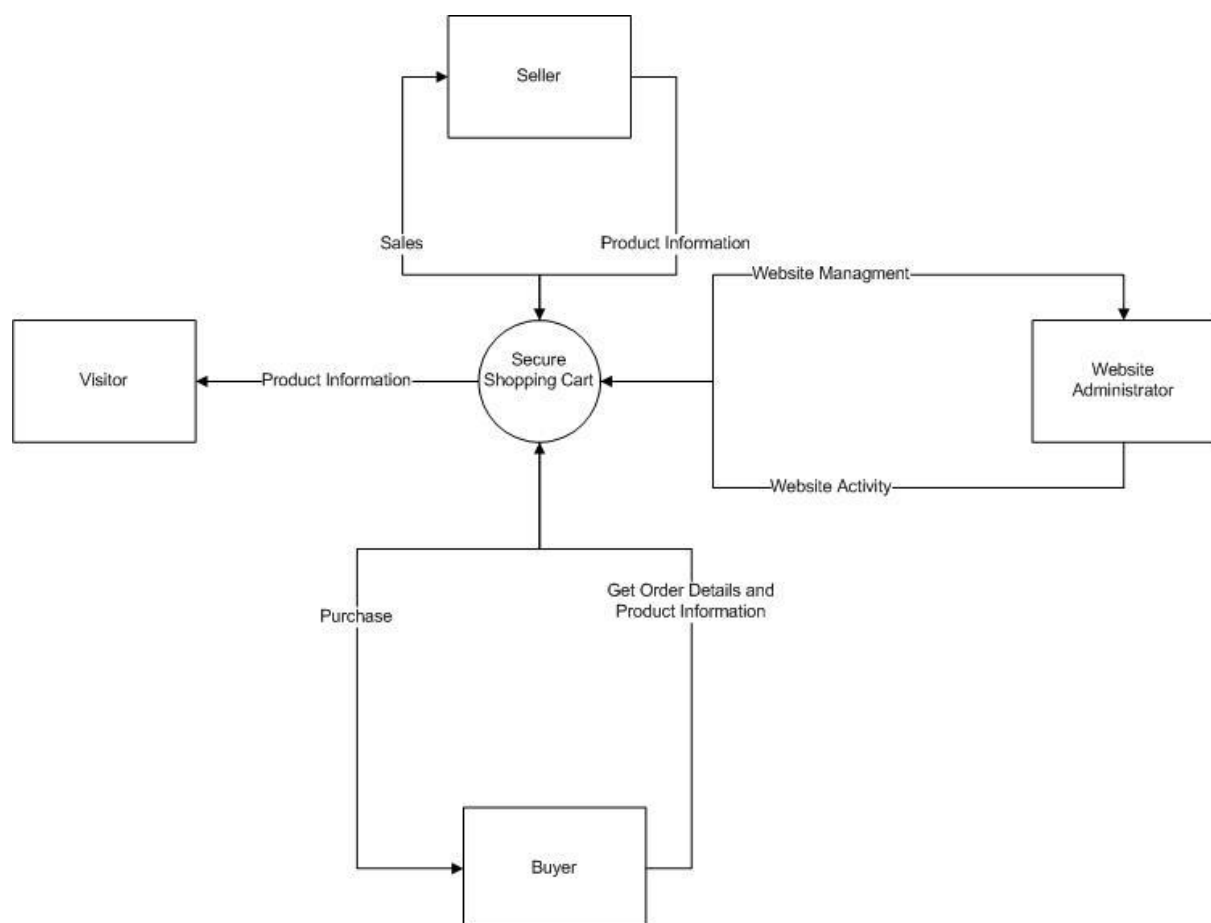
### Assets

ID	Name	Description	Trust Level
<b>A1</b>	User	Assets that relate to a website user	
<b>A1.1</b>	User's Login data	Users' credentials username and password. This asset needs protection because if it is stolen another user would be able to do anything which the user can do.	T2 Authenticated user T3 Website administrator
<b>A1.2</b>	User's Personal Data	User's personal data including contact information. This needs protection because some personal data might be important such as telephone number.	T2 Authenticated user T3 Website administrator
<b>A1.3</b>	User's public and private key	User public and private key are very valuable since one can easily decrypt valuable data	T3 Website administrator
<b>A2</b>	Backend Database	Backend database includes all the data of all users registered into the system together with all the product and order details. All data can be found here.	T3 Website administrator
<b>A3</b>	Product	All the products details that can be found in the database	T1 Remote anonymous user T2 Authenticated user (Buyer) T3 Website Administrator T4 Authenticated user (Seller)
<b>A3.1</b>	Products Details	All the product details that are found in the database. This includes all the product information such as prices and quantity bought.	T1 Remote anonymous user T2 Authenticated user (Buyer) T3 Website Administrator T4 Authenticated user (Seller)
<b>A3.2</b>	Product key and iv	The product key and iv which are used in order to decrypt encrypted files	T3 Website Administrator
<b>A4</b>	Order Details	All the details that are inserted into the database that relates to the buyer. This includes all the information regarding the order such as the product bought, the price and the buyer who bought it	T2 Authenticated user(Buyer) T3 Website Administrator

## Trust Levels

ID	Name	Description
T1	Remote anonymous user	A user who has not yet authenticated to the website
T2	Authenticated user (Buyer)	A registered user who has valid credentials has only the right to buy certain items.
T3	Website administrator	User who can do any operations on the website such as updating, deleting and inserting any products.
T4	Authenticated user (Seller)	A registered user who can upload, edit or delete his own items in order to put them on the market.

## DFD Level 0



## Entry Points

ID	Name	Description	Trust level
<b>E1</b>	Web server listening port	Port on which the web server listens. All web pages are layered on this port.	T1 Remote anonymous user T2 Authenticated user (Buyer)
<b>E1.1</b>	Login Page	The page where a user can login or register.	T1 Remote anonymous user
<b>E1.1.1</b>	Register Method	Create a new user login	T1 Remote anonymous user
<b>E1.1.2</b>	Login Method	Compares user credentials to those stored in database and session is created.	T1 Remote anonymous user
<b>E2</b>	Store Page	The page where all the products are listed	T1 Remote anonymous user T2 Authenticated user (Buyer)
<b>E2.1</b>	Add to Cart Method	Add a product to shopping cart	T1 Remote anonymous user T2 Authenticated user (Buyer)
<b>E3</b>	Manage Products Page	The page where one can add or edit products	T3 Website Administrator T4 Authenticated user (Seller)
<b>E3.1</b>	Create new Product Method	Creating new product. Stores a new product in the database with all its details including the file.	T3 Website Administrator T4 Authenticated user (Seller)
<b>E3.2</b>	Edit an existing Product	Edit new product. Updating an existing product in the database with all its details including the file.	T3 Website Administrator T4 Authenticated user (Seller)

## Threats

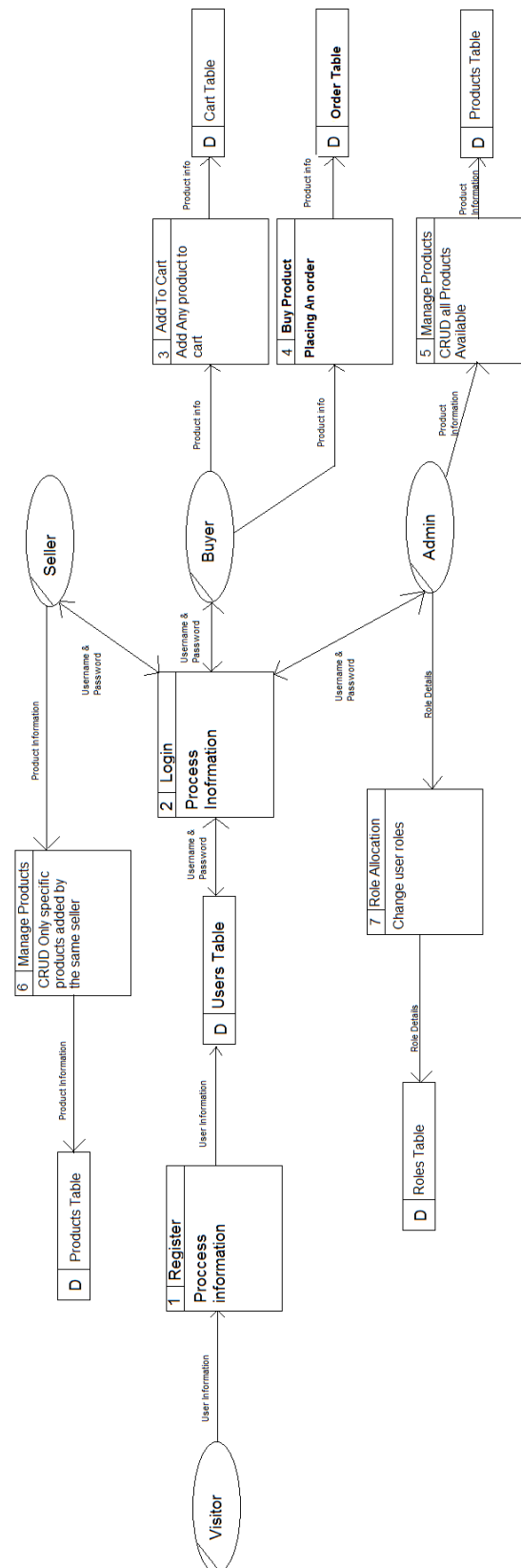
<b>ID</b>	TR1
<b>Name</b>	Adversary tries to supply malicious data when logging in
<b>Description</b>	Adversary tries to input special characters to be able to pose as another user, or logs in without having an appropriate username and password. Handling of data is critical in this regards.
<b>STRIDE</b>	Tempering, Elevation of Privileges
<b>Entry Points</b>	(E1.1) Login Page
<b>Assets</b>	(A1.2) User's personal data, (A2) Backend database
<b>Mitigation/Strategy</b>	Using Stored Procedures, parameterized queries or LINQ
<b>ID</b>	TR2
<b>Name</b>	Adversary manages to get a valid username and password.
<b>Description</b>	If an Adversary manages to get a valid username and password he can do what that particular user can do.
<b>STRIDE</b>	Information disclosure, Tempering, Elevation of Privileges
<b>Entry Points</b>	(E1.1) Login Page
<b>Assets</b>	(A1.2) User's personal data, (A1.1) User's Login Data
<b>Mitigation/Strategy</b>	Strong passwords should be enforced, encryption should take place, and database should be protected from external access.
<b>ID</b>	TR3
<b>Name</b>	Adversary tries to change URL in order to get other user information.
<b>Description</b>	When an attacker tries to change a URL in a way to get information of other users. For example changing id.
<b>STRIDE</b>	Information disclosure, Tempering, Elevation of Privileges
<b>Entry Points</b>	(E1.1) Login Page
<b>Assets</b>	(A1.2) User's personal data, (A3.1) Product Details
<b>Mitigation/Strategy</b>	URL Rewriting, Authorization

<b>ID</b>	<b>TR4</b>
<b>Name</b>	Adversary tries SQL Injection
<b>Description</b>	Adversary tries to input some SQL in any of the entry points in order steal data.
<b>STRIDE</b>	Information disclosure, Tempering, Elevation of Privileges
<b>Entry Points</b>	(E1.1) Login Page, (E3) Manage Products Page
<b>Assets</b>	(A1.2) User's personal data, (A3.1) Product Details
<b>Mitigation/Strategy</b>	Using LINQ, Using Stored Procedures
<b>ID</b>	<b>TR5</b>
<b>Name</b>	Attacker Tries to upload a malicious file
<b>Description</b>	Attacker tries to upload a bad file that might contain something to break the system
<b>STRIDE</b>	Information disclosure, Elevation of Privileges
<b>Entry Points</b>	(E3) Manage Products Page
<b>Assets</b>	(A1.2) User's personal data, (A3.1) Product Details, (A4) Order Details
<b>Mitigation/Strategy</b>	Using specialized tools to detect bad files, filtering of data
<b>ID</b>	<b>TR6</b>
<b>Name</b>	Attacker tries to use cross-site scripting to steal information
<b>Description</b>	When an attacker use an entry point in order to steal valuable information
<b>STRIDE</b>	Information disclosure, Tempering of data
<b>Entry Points</b>	(E1.1) Login Page, (E3) Manage Products Page
<b>Assets</b>	(A1.2) User's personal data, (A3.1) Product Details
<b>Mitigation/Strategy</b>	Using specialized tools to detect bad files, filtering of data
<b>ID</b>	<b>TR7</b>
<b>Name</b>	An attack tries to hack your system using backdoors
<b>Description</b>	When the Adversary will access your system with default data such as testing username and password
<b>STRIDE</b>	Information disclosure, Elevation of Privileges, Tempering of data
<b>Entry Points</b>	(E1.1) Login Page
<b>Assets</b>	(A1.2) User's personal data, (A3.1) Product Details, (A4) Order Details
<b>Mitigation/Strategy</b>	Make sure that any default username and passwords are removed, do not allow any weak passwords



ID	TR8
<b>Name</b>	An adversary tries to skip the client side validation
<b>Description</b>	When an attacker tries to skip validation by disabling java script form the browser in order to try and eliminate validations
<b>STRIDE</b>	Information disclosure, Elevation of Privileges, Tempering of data
<b>Entry Points</b>	(E1.1) Login Page, (E3) Manage Products Page
<b>Assets</b>	(A1.2) User's personal data, (A3.1) Product Details, (A4) Order Details
<b>Mitigation/Strategy</b>	Make sure to also include server side validations.

## Task 2 – DFDs in Threat Modelling Document (M1.4)



### Task 3 – Implementation of Secure Website (P3.1)

This Task can be found on the CD attached at the back of the assignment.

### Task 4 – Documentation of security patterns and practices (P3.2, P4.3)

The Presentation can be found attached at the back of the assignment.

### Task 5 – Implementation of Further security techniques in website (M1.5)

This Task can be found on the CD attached at the back of the assignment.

## Task 6 – Documentation of alternative security patterns (M1.3)

### PayPal SDK Alternative

#### Stripe

One alternative in relation to paying with PayPal is to use other payment gateways such as stripe. In this case stripe is a payment gateway similar to PayPal SDK which allows you to send payments in just only few steps which are:

#### 1. Embed the checkout

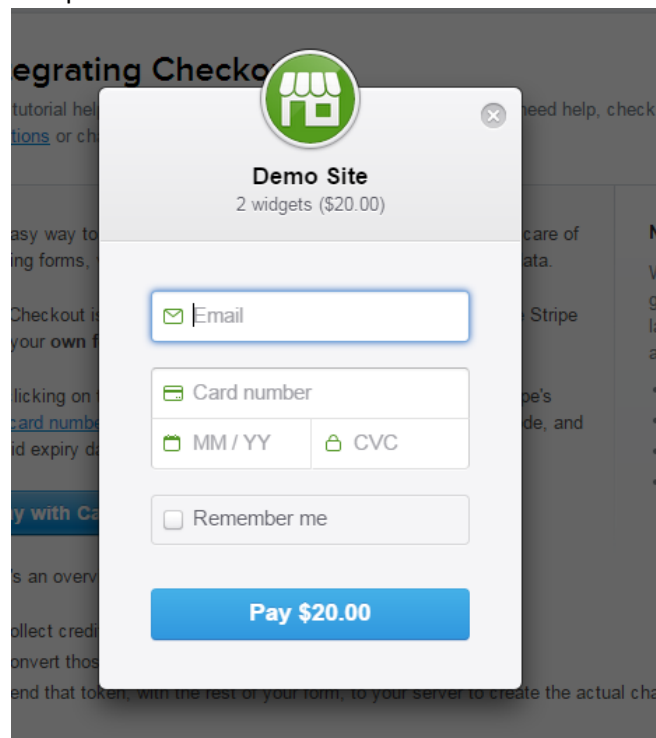
In this case the developer will be provided with a piece of code in order to be able to integrate it in the system.

```
<form action="" method="POST">
  <script
    src="https://checkout.stripe.com/checkout.js" class="stripe-button"
    data-key="pk_test_6pRNASCoBOKtIshFeQd4XMuH"
    data-amount="2000"
    data-name="Demo Site"
    data-description="2 widgets ($20.00)"
    data-image="/128x128.png">
  </script>
</form>
```

#### 2. Sending tokens to your server

In this step this alternative will be eliminating any security issues that might be present if a developer implement his own payment gateway using credit card details and other sensitive data. With this implementation your system is not going to handle any sensitive data but it is the responsibility of who is providing this implementation. One can also add that your system will only have the need to process any tokens passed.

A screen shot of this implementation can be found below:



## Custom Authorization Alternative

### Token

An alternative to custom authorization can be that of implementing a system with a build in implemented token similar to what local banks do. Such system works by providing a different token to the user every time s/he tries to make use or access any the system. Apart from that the token feature can also be used when a user tries to download a bought file in order to make sure and verify that the user is actually who is claiming to be.

Furthermore, one can also implement this system by:

1. Allow the user to login in using his username and password.
2. After the system verifies both username and password the system will send a token to the user's mobile phone while system redirects to a page where the same user can enter the token received.
3. After the user enters the token received the system will redirect the user to the allocated functionalities to be able to make use of them.
4. Lastly, one also has to mention that if a user enters the token incorrectly or claim to not receiving the token s/he must contact the website administrator in order to try and solve any present issues.

A screenshot of how the system may look like can be found below:



To conclude one can also mention that such implementation apart from when a user logs in, it can also be used when the user tries to access sensitive data or use functionalities that might influence any profits to both the system and any sellers available such as:

- Deleting or modifying an existing order
- Deleting or modifying any products
- Deleting any user accounts
- Managing user roles

## Action Filters

Another alternative to custom authorization is action filters. In this case they work by calling the filter both before and after a particular action start executing and when the same functionality has executed. Apart from that one can also mention that any pre and post processing logic can also be put in such filters.

Furthermore, one can also mention that to implement such action filters first a custom filter attribute class must be created and then the `IActionFilter` filter interface must be implemented. After implementing such interface, it will provide us with two other methods which are:

1. **OnActionExecuting** – This will be executed before the action take place.
2. **OnActionExecuted** – This will be executed after the action take place.

Code Snippet:

```
public class CustomActionAttribute : FilterAttribute, IActionFilter
{
    void IActionFilter.OnActionExecuted(ActionExecutedContext filterContext)
    {
        filterContext.Controller.ViewBag.OnActionExecuted = "IActionFilter.OnActionExecuted
filter called";
    }

    void IActionFilter.OnActionExecuting(ActionExecutingContext filterContext)
    {
        filterContext.Controller.ViewBag.OnActionExecuting =
"IActionFilter.OnActionExecuting filter called";
    }
}
```

## Task 7 – Security Patterns – Argue on what you cannot protect against (D1.2)

NB: The two security patterns in relation to both authentication and secure session management can be found implemented in the solution found in the CD.

### Authentication

#### Accessing unencrypted access token

When using functionalities such as Facebook login provided by third party companies one can keep in mind that some security issues might be present. In this case one must make sure that any tokens provided must be stores in an encrypted format. In relation to Facebook login researchers found that Facebook SDK Library stores the same token in an unencrypted format. Apart from that it was also added that this unencrypted token can easily be accessed from a number of different devices.

One can conclude that with this vulnerability the system implemented will not be 100% secure and the developer must also take any security measures to make it even more secure.

#### Facebook data gets stolen

Another problem that the developer does not have control on is that if an attacker will successfully hack Facebook. With such problem one can mention that all Facebook Data might be stolen and the attacker can access any website using Facebook login implementation. In this case in order to mitigate any issues one can only permit buyers to login with Facebook, while sellers and administrators must use the system safe login functionality.

#### Facebook Cookies

The last problem which is also considered as a vulnerability is that in this case Facebook store cookies for every user that is logged in, furthermore, one can also mention that if such cookies aren't well secured and not encrypted it will be very easy for an attacker to access any account.

## Secure Session Management

### Key logger

One factor that you can't protect against when using secure session management or Web APIs is Key Loggers. In this case this can be both a pen drive or any other device or a software installed on your computer that as the name implies will be able to steal and record all the key strokes typed when using your computer. Apart from that one must emphasize that all the keystrokes will be available for the attacker such as tokens, passwords and other sensitive data.

### Viruses

Another factor similar to key loggers is any viruses or spyware found on the machine which will be able to send any sensitive data back to the attacker. Similar to key loggers in this case sensitive data might be maliciously used in order to access any Web API's.

### Physical Attempts

The last factor that in this case a developer cannot cater for is that when an attacker physically steals sensitive data by monitoring any user inputs and try to remember them for later time. In this case this will be very hard to detect the attacker since the same attacker will be using/attacking the system or in this case the Web API under the name of that particular user.



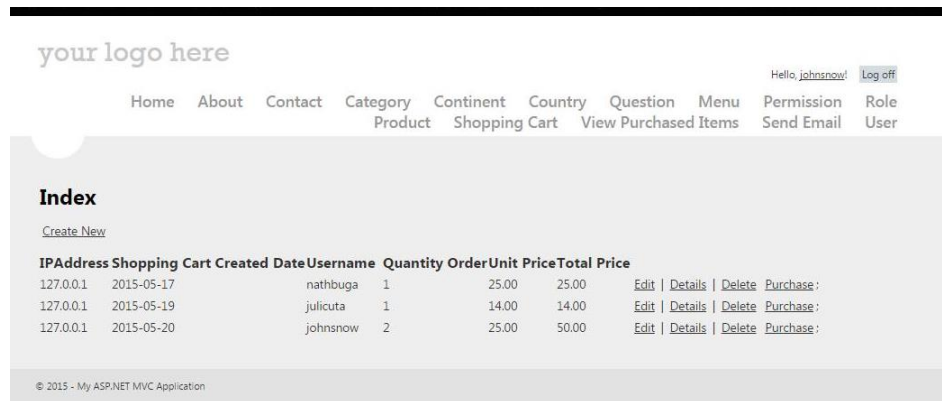
## Black Box Testing

Secure Software Development | Wayne Caruana | 1BSC2S



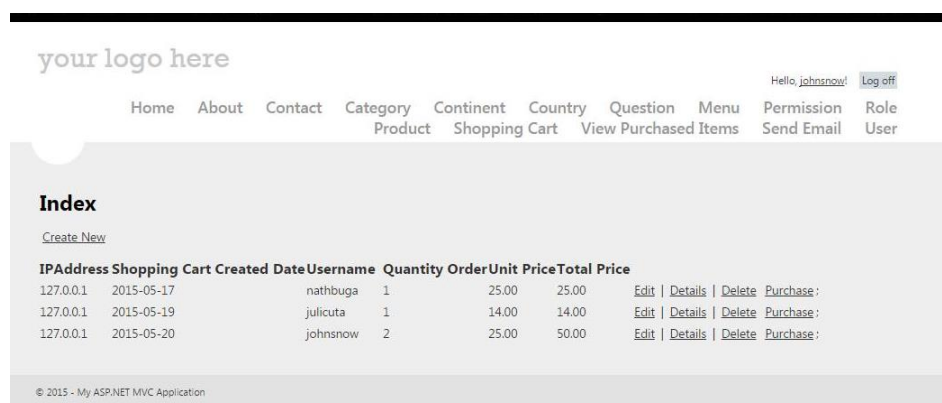
Function No: 3	Functionality: Shopping Cart			
Test No.	Data Input	Expected Output	Actual Output	Success Fail
1	Click add to shopping cart	Product is added to shopping cart	Product is added to shopping cart	Success

**Screenshot:**



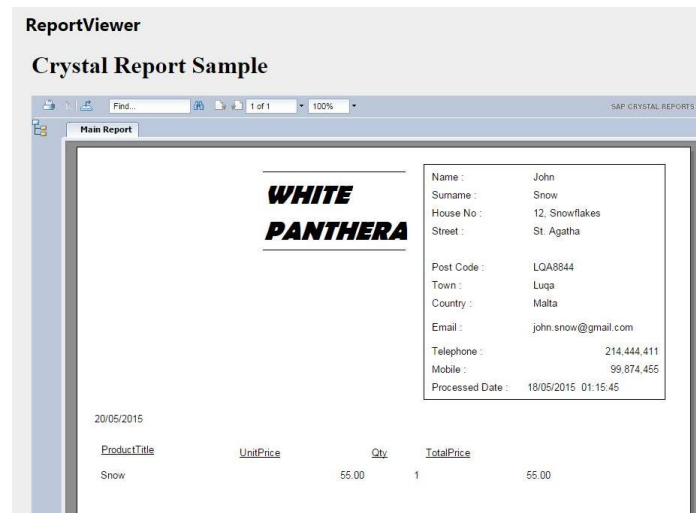
Function No: 4	Functionality: Shopping Cart			
Test No.	Data Input	Expected Output	Actual Output	Success Fail
1	Add an existing item to shopping cart	The system will not allow the product to be added	The product is added to shopping cart	Partial Success

**Screenshot:**

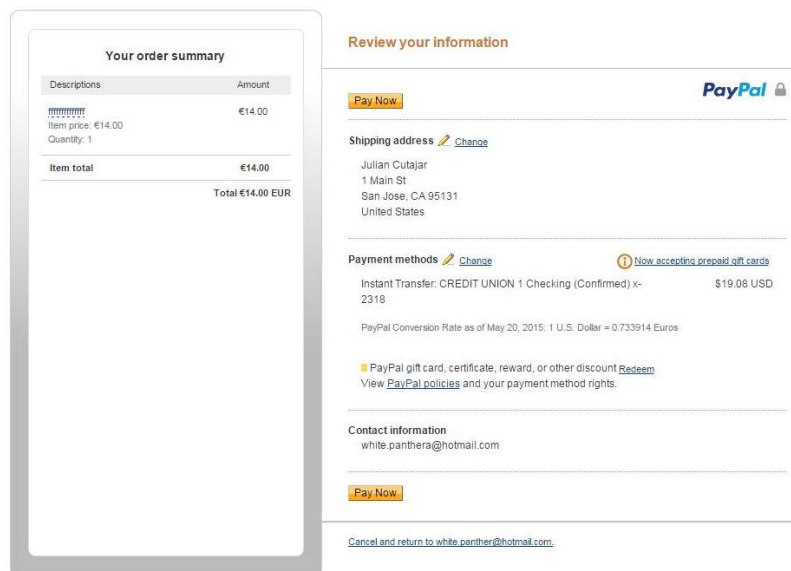


<b>Function No: 5</b>	<b>Functionality: Checkout</b>			
<b>Test No.</b>	<b>Data Input</b>	<b>Expected Output</b>	<b>Actual Output</b>	<b>Success Fail</b>
1	Click checkout button with products in shopping cart	System will be redirected to PayPal	System is redirected to PayPal	Success

#### Screenshots:



white.panther@hotmail.com



## Code Review

Student: Julian Cutajar

### Strong Point in Code

One strong point found in Julian's code was that he made use of custom authorization in his code. With such improvement he can easily change any roles that can access particular web pages. Furthermore, one can also mention that with such improvement he can also do the above mentioned in real time.

```
protected override bool AuthorizeCore(HttpContextBase httpContext)
{
    string email = httpContext.User.Identity.Name;
    string page = httpContext.Request.RawUrl;

    //here add another table with url and role and check
    List<Role> userRoles = new RoleBL().GetUserRoles(email).ToList();
    List<Role> pageRoles = new RoleBL().GetPageRoles(page).ToList();

    foreach (var pr in pageRoles)
    {
        foreach (var ur in userRoles)
        {
            if (pr.Name == ur.Name)
            {
                return true;
            }
        }
    }

    return false;
}
```

## Weak Point in Code

On the other hand one weak point found in Julian's code was that as shown in the screenshot below he rarely use any comments. With such weak point one can also mention that he or anyone using or updating the system in the future might find it more difficult to use and understand any code if it is not well commented.

```
BusinessLayer.ShoppingCartBL shoppingCartBL = new BusinessLayer.ShoppingCartBL();

tblShoppingCart shopping = shoppingCartBL.SearchShoppingCartByShoppingCartID(Convert.ToInt32(val));

int productID = shopping.ProductID.Value;
int qty = shopping.Qty.Value;
decimal price = shopping.UnitPrice.Value;
decimal totalPrice = shopping.TotalPrice.Value;

//Product
BusinessLayer.ProductBL productBL = new BusinessLayer.ProductBL();

tblProduct pro = productBL.SearchProductByProductID(shopping.ProductID.Value);

//Checkout
BusinessLayer.CheckoutBL checkOutBL = new BusinessLayer.CheckoutBL();

tblCheckout checkOut = new tblCheckout();

checkOut.CheckoutDate = DateTime.Now;
checkOut.Username = HttpContext.User.Identity.Name;
checkOut.TotalAmount = totalPrice;

tblCheckout last = checkOutBL.AddCheckoutValue(checkOut);

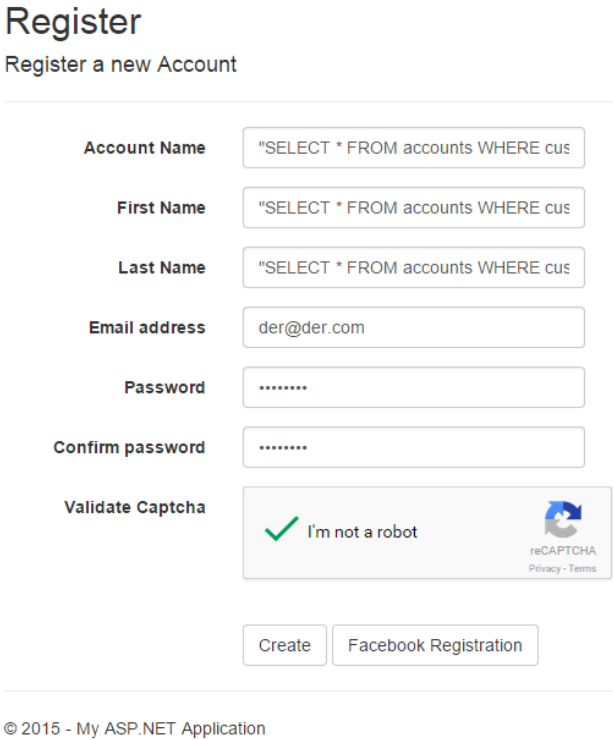
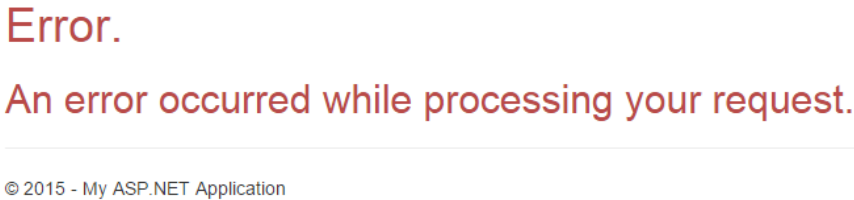
checkoutID = last.CheckoutID;

BusinessLayer.CheckoutListBL checkOutItemsBL = new BusinessLayer.CheckoutListBL();
```

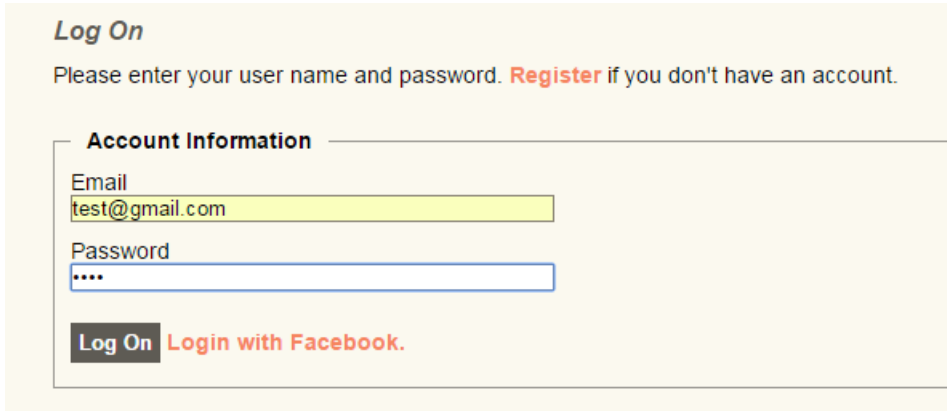
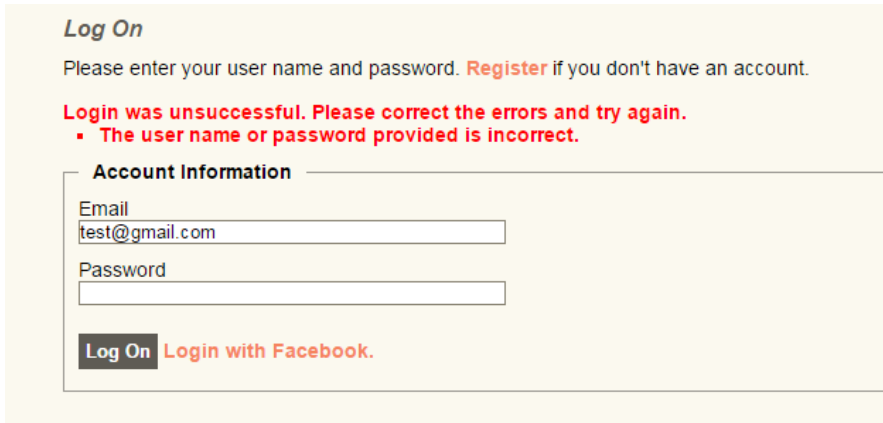
## Task 9 – Perform Attacks on a third party application and document them (D1.3)

NB: Student: Derrick Agius

### A1-Injection

<b>Test</b>	A1- Injection
<b>Outcome</b>	<p>The first type of test type of risk tested for is SQL injection. The application passed this test as the system was able to resist all attacks.</p> <p>Apart from the screenshots below the application was also tested with different type of SQL injection method such as 'or '1'=1</p>
<b>Test input</b>	
<b>Test outcome</b>	

## A2 – Broken Authentication and Session Management

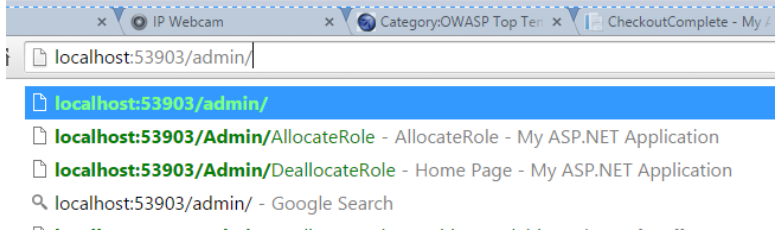
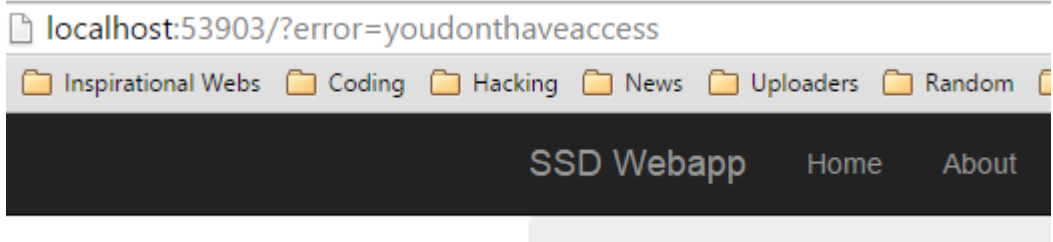
<b>Test</b>	A2 – Broken Authentication and Session Management
<b>Outcome</b>	<p>This type of attack is all about trying to access any account that might be considered as a test account.</p> <p>One of the most common test account that might be present and not removed from the system is:</p> <p>Email: <a href="mailto:test@gmail.com">test@gmail.com</a>          Password: test</p>
<b>Test input</b>	 <p>The screenshot shows a 'Log On' form with the title 'Log On' and a subtitle 'Please enter your user name and password. Register if you don't have an account.' Below this is a section titled 'Account Information' containing two input fields: 'Email' with the value 'test@gmail.com' and 'Password' with four dots. At the bottom of the form is a 'Log On' button and a link 'Login with Facebook.'</p>
<b>Test outcome</b>	 <p>The screenshot shows the same 'Log On' form, but with an error message displayed in red text: 'Login was unsuccessful. Please correct the errors and try again.' Below this message is a red bullet point: 'The user name or password provided is incorrect.' The input fields for 'Email' and 'Password' are now empty.</p>



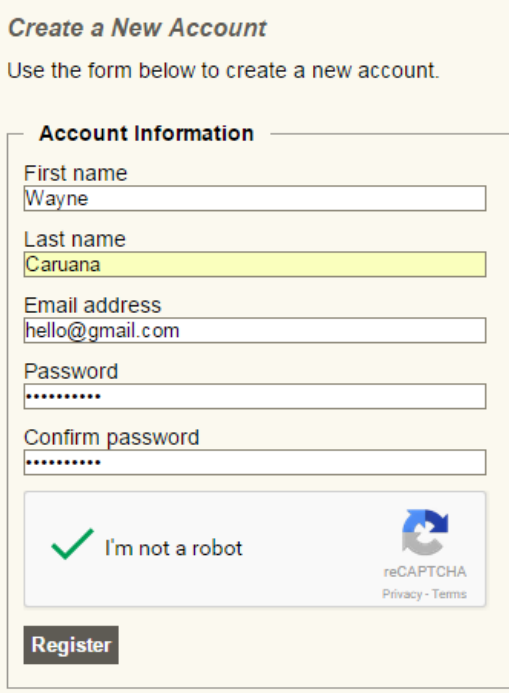

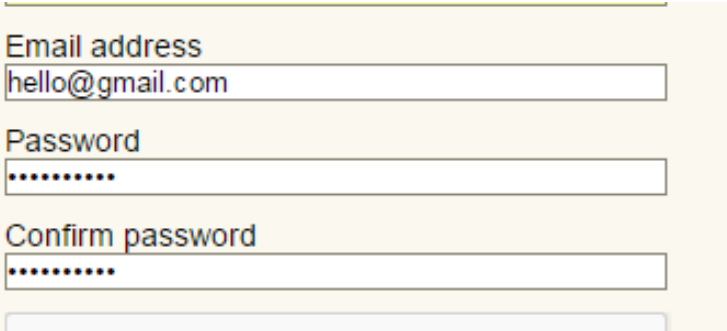
### A3 – Cross Site Scripting (XSS)

<b>Test</b>	A3 – Cross Site Scripting (XSS)
<b>Outcome</b>	<p>In this case a simple code is going to be inserted in a random entry point and see how the system reacts. In this case non persistent XSS was tested.</p> <p>Sample Code used:</p> <pre>&lt;?php  \$name = \$_GET['name'];  echo "Welcome \$name&lt;br&gt;";  echo "&lt;a href='http://xssattackexamples.com/'&gt;Click to Download&lt;/a&gt;";  ?&gt;</pre>
<b>Test input</b>	
<b>Test outcome</b>	General failure

## A4- Insecure Direct Object Reference

<b>Test</b>	A4 – Insecure Direct Object Reference
<b>Outcome</b>	Trying to access a page which I do not have authority/ access to open it by path. The user was redirect to the homepage as he doesn't have access
<b>Test input</b>	
<b>Test outcome</b>	

## A6 – Sensitive Data Exposure

<b>Test</b>	A6 – Sensitive Data Exposure
<b>Outcome</b>	In this case I was trying to search from any sensitive data which might not be hidden for the users to see such as for example password masking.
<b>Test input</b>	 <p><b>Create a New Account</b></p> <p>Use the form below to create a new account.</p> <p><b>Account Information</b></p> <p>First name Wayne</p> <p>Last name Caruana</p> <p>Email address hello@gmail.com</p> <p>Password .....</p> <p>Confirm password .....</p> <p><input checked="" type="checkbox"/> I'm not a robot  reCAPTCHA Privacy - Terms</p> <p><b>Register</b></p>
<b>Test outcome</b>	 <p>Email address hello@gmail.com</p> <p>Password .....</p> <p>Confirm password .....</p>

## Bibliography

- Best Practices: Security Vulnerability Testing*. (n.d.). Retrieved from SoapUI:  
<http://www.soapui.org/testing-dojo/best-practices/security-vulnerability-testing.html>
- Hack Your API First – learn how to identify vulnerabilities in today’s internet connected devices with Pluralsight*. (2014, September 04). Retrieved from Troy Hunt:  
<http://www.troyhunt.com/2014/09/hack-your-api-first-learn-how-to.html>
- Kumar, M. (2014, July 03). *Facebook SDK Vulnerability Puts Millions of Smartphone Users' Accounts at Risk*. Retrieved from The Hacker News: <http://thehackernews.com/2014/07/facebook-sdk-vulnerability-puts.html>
- Singh, R. R. (2013, April 15). *A Beginner's Tutorial for Understanding Filters and Attributes in ASP.NET MVC*. Retrieved from Code Project: <http://www.codeproject.com/Articles/577776/Filters-and-Attributes-in-ASPNET-MVC>
- stripe. (n.d.). *Integrating Checkout*. Retrieved from stripe:  
<https://stripe.com/docs/tutorials/checkout>
- The State of API Security*. (n.d.). Retrieved from SoapUI: <http://www.soapui.org/testing-dojo/world-of-api-testing/state-of-api-security.html>