

Peephole Documentation

This is the documentation paper for the Peephole solution. In this document you can find information on how to extend Peephole Bank and also how to add additional projects into Peephole solution.

Peephole is created as a bachelor project at NTNU Aalesund, Norway. We wanted to create a legal testing environment for information security issues for the ASP.NET platform.

If you have any questions, please feel free to contact us.

Table of Contents

1	Virtual Machine Setup	3
1.1	Requirements	3
1.2	Installation	4
2	Extending Peephole solution	8
2.1	Create new project	8
2.2	Adding link to project in the Frontpage	9
2.3	Default Project in Solution	11
3	Extending Peephole Bank	12
3.1	Adding modules / Areas	12
3.1.1	MVC Area Specifics	12
3.1.2	Webforms Area Specifics	12
3.1.3	Adding the Area/Module To Menu	13
4	Using the application	14
4.1	Injection	14
4.2	Cross-Site Scripting (XSS)	16
4.3	Broken authentication and Session Management	21
4.4	Insecure Direct Object References Jens	21
4.5	Cross-Site Request Forgery	22
4.6	Security Misconfiguration	22
4.7	Insecure Cryptographic Storage	23
4.8	Failure to Restrict URL Access Jens	24
4.9	Insufficient Transport Layer Protection	24
4.10	Unvalidated Redirects and Forwards	25
5	Known bugs	27
5.1	Site.Master changes	27

1 Virtual Machine Setup

1.1 Requirements

The requirements for the host computer varies depending on the which OS the VM client is running. Below are the minimum requirements for resources that needs to be free and available to the VM:

VM running Windows 8.1

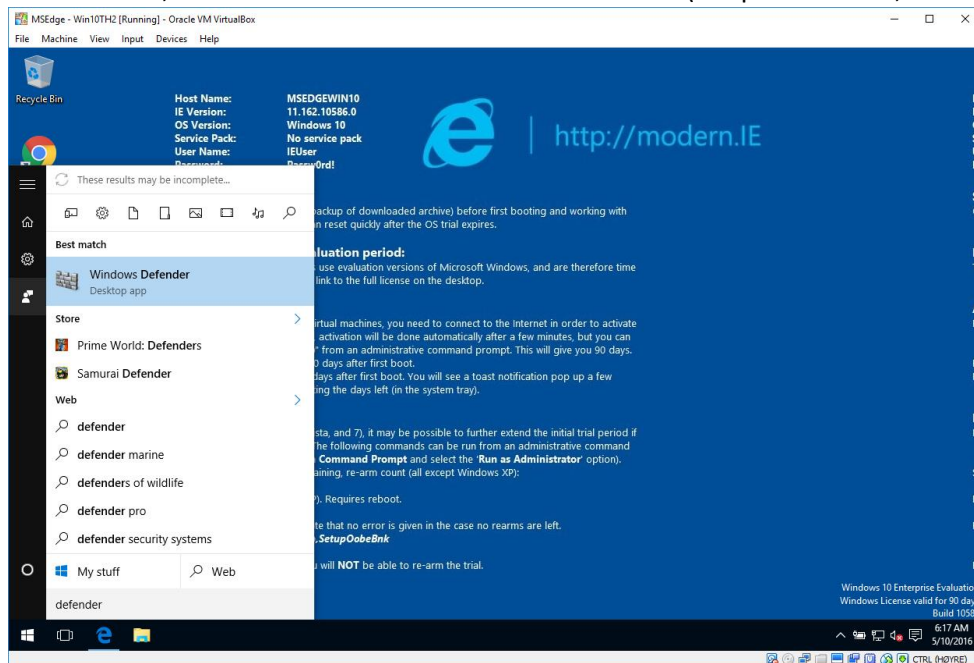
Available disk space	> 20 GB
RAM	> 2 GB
CPU logical cores	Half of total

VM running Windows 10

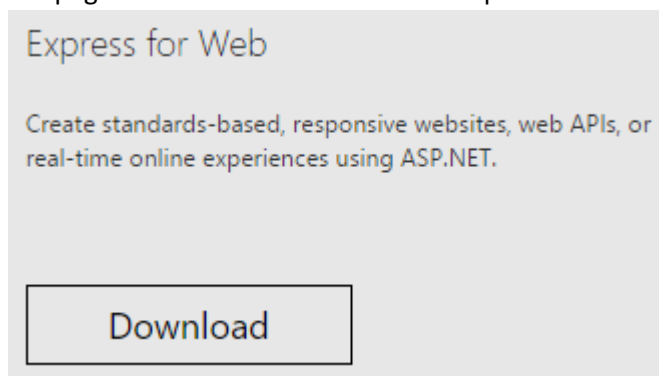
Available disk space	> 20 GB
RAM	> 4 GB
CPU logical cores	Half of total

1.2 Installation

1. If you plan to run the Peephole project from existing Windows installation, please proceed to step 6.
2. Download Virtual Machine from <https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/windows/>
3. Unzip downloaded VM and import in Virtualization software. For VirtualBox, change CPU to 2, memory 4096.
4. Once booted, disable Windows Defender Real-time scan (For performance, can be skipped)

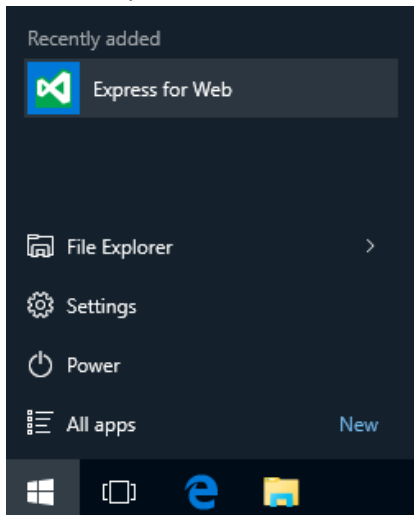


5. Inside the VM, go to <https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx>, scroll down on the page and download Visual Studio Express for Web

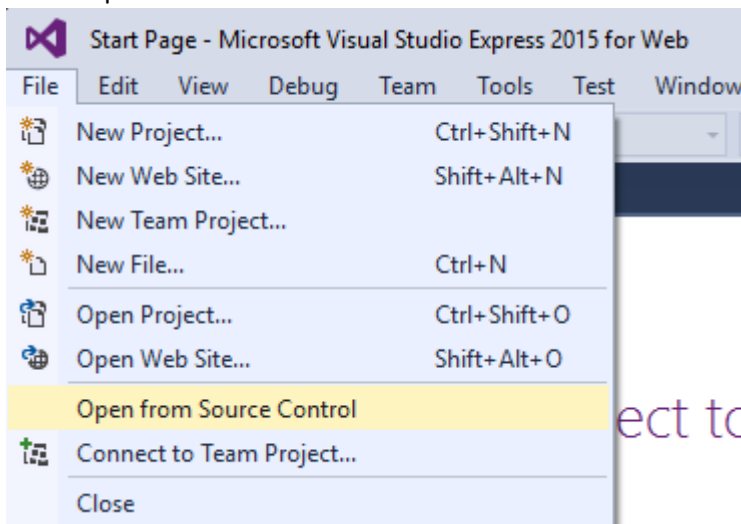


6. If you plan to run the Peephole project from existing Visual Studio installation, please proceed to step 9.
7. Install Visual Studio. This is a time consuming operation, approximately 1 hour depending on system performance.

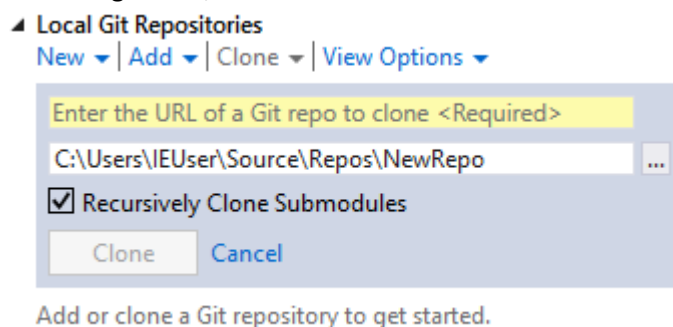
8. Start -> Express for Web



9. File -> Open from source control



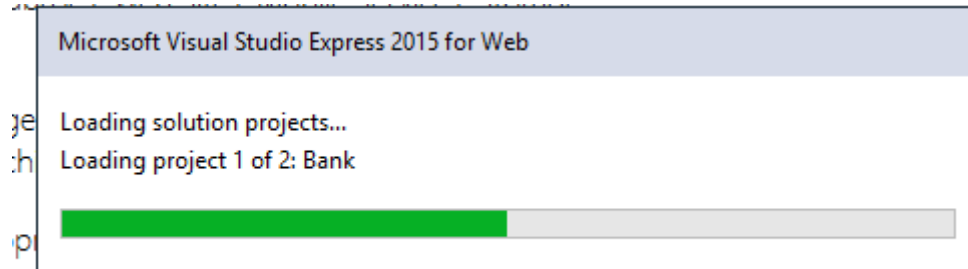
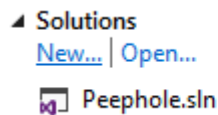
10. On the right side, select "Clone"



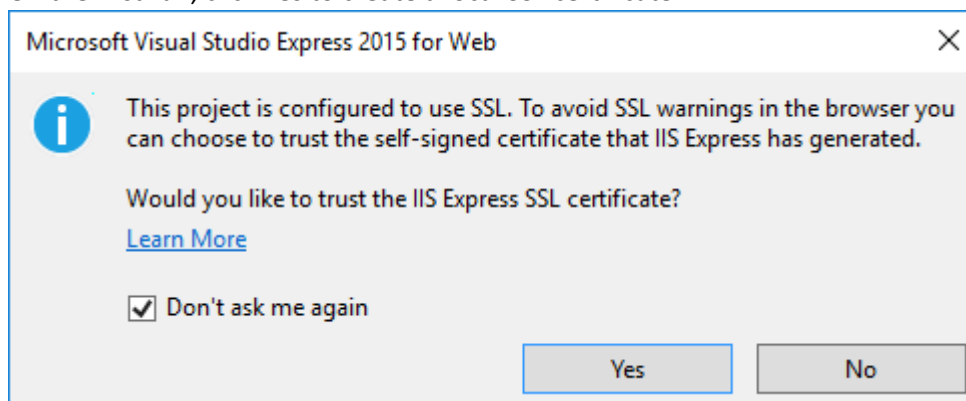
11. Enter the Peephole repository

<https://github.com/UrDar/Peephole.git>

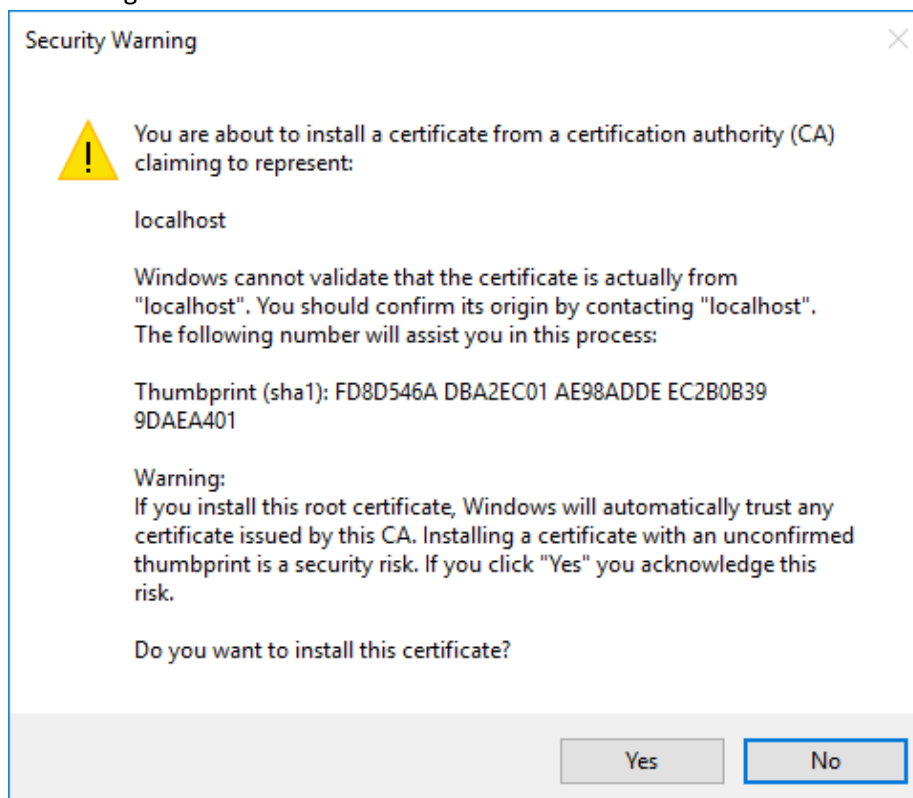
12. Select Peephole.sln and the project will load



13. On the first run, click Yes to create a local SSL certificate



14. And Yes again




15. A browser will be opened showing the Peephole Frontpage

PEEPHOLE

Peephole is a ASP.NET/MSSQL web application that is design to be vulnerable.
The main goal is for beginners and developers to test their skills in an legal environment. We want professional, students and other people to better their understanding in web security to make the web a safer place.

[ABOUT US](#)
[DOCUMENTATION](#)
[INFORMATION](#)
[CONTACT US](#)



The peephole bank is a fictive bank created by the creators of Peephole.

Your project

The next project that appears here could be yours.

2 Extending Peephole solution

2.1 Create new project

The solution can be extended with additional projects.

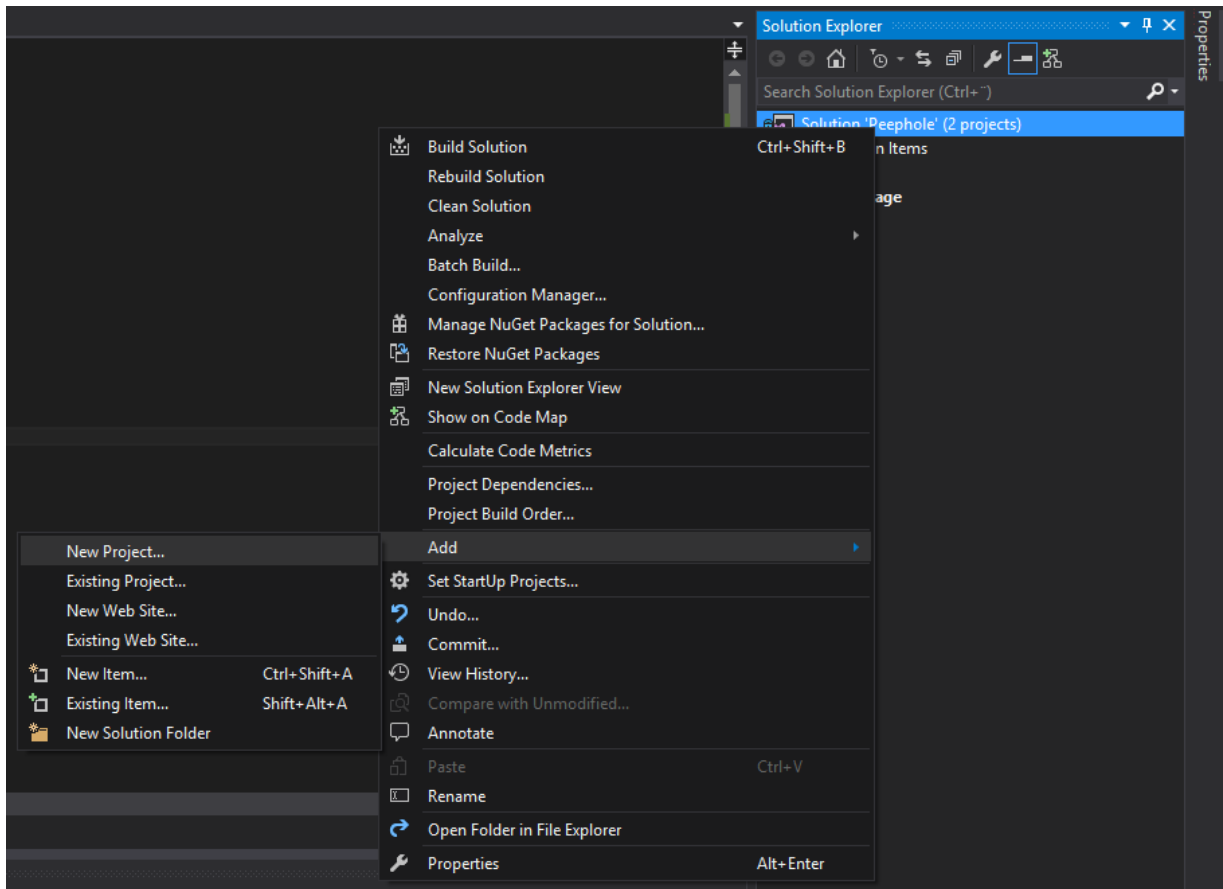


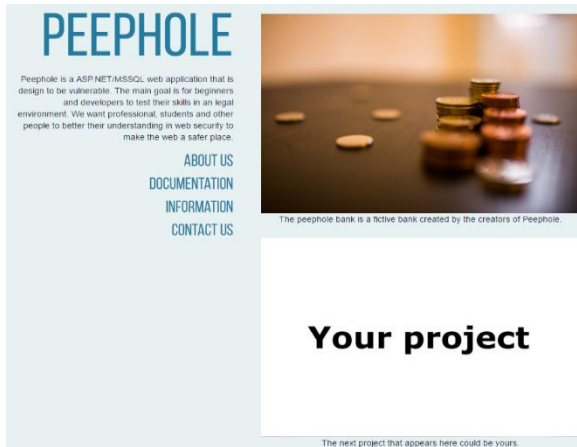
Figure 2.1: Adding Project To Solution

Step by step:

1. Right click the solution
2. Choose "Add"
3. Choose "New project"
4. Select the type of project you want to create
5. Change the application settings
 - a. Right click the project
 - b. Choose "Properties"
 - i. *Application*: set the namespace
 - ii. *Web*: set the localhost address and start-up page)
6. For accessibility, add a link to the new project from the Frontpage-project ()

2.2 Adding link to project in the Frontpage

When adding a new project to the solution you should also make a link to that project in the Frontpage. The Frontpage is the start-up project for the solution and functions as a hub where projects are accessed.



Each of the application have one a clickable picture each that shows what kind of application it is. When hovering over the picture, you'll get the application name and a short description of the project. When adding a new project, you'll have to either edit or add a few links of HTML code to make a new picture for your project. In this code you also have to edit the header, project name and a description of the of the project.

Figure 2.2: How the frontpage and the different project looks like



In the bank application

Header: Peephole

Project: Bank

Description: The peephole bank is a fictive bank created by the creators of Peephole.

Figure 2.3: Project image when hovering

```

<div class="content col-md-9 col-sm-9 col-lg-9">
  <a href="http://localhost:3988/" class="portfolio-box">
    <!-- Add image-->
    
    <div class="portfolio-box-caption">
      <div class="portfolio-box-caption-content">
        <div class="project-category text-faded">
          <!-- Header-->
          Peephole
        </div>
        <!-- Project name-->
        <div class="project-name">
          Bank
        </div>
      </div>
    </div>
  </a>
  <span class="description col-md-12 col-sm-12 col-lg-12">
    The Peephole Bank is a fictive bank within the Peephole solution.
  </span>
</div>

```

Figure 2.4: Code to create project image in Frontpage / Index.html

Figure 1.4 shows the code that makes a new project picture with the text needed. This uses some of the bootstrap framework to style the pictures and the structure.

If you want to use another image, this is also where this is edited. It's possible to add an image from the web, but it's preferable to save the image locally.

2.3 Default Project in Solution

In the solution you can also choose which project should run when you start the solution.

1. Right click the solution
2. Choose “properties”
3. Choose “Startup Project” under “Common Properties”
4. Now you can choose which project should start when you press the Start button in Visual Studio

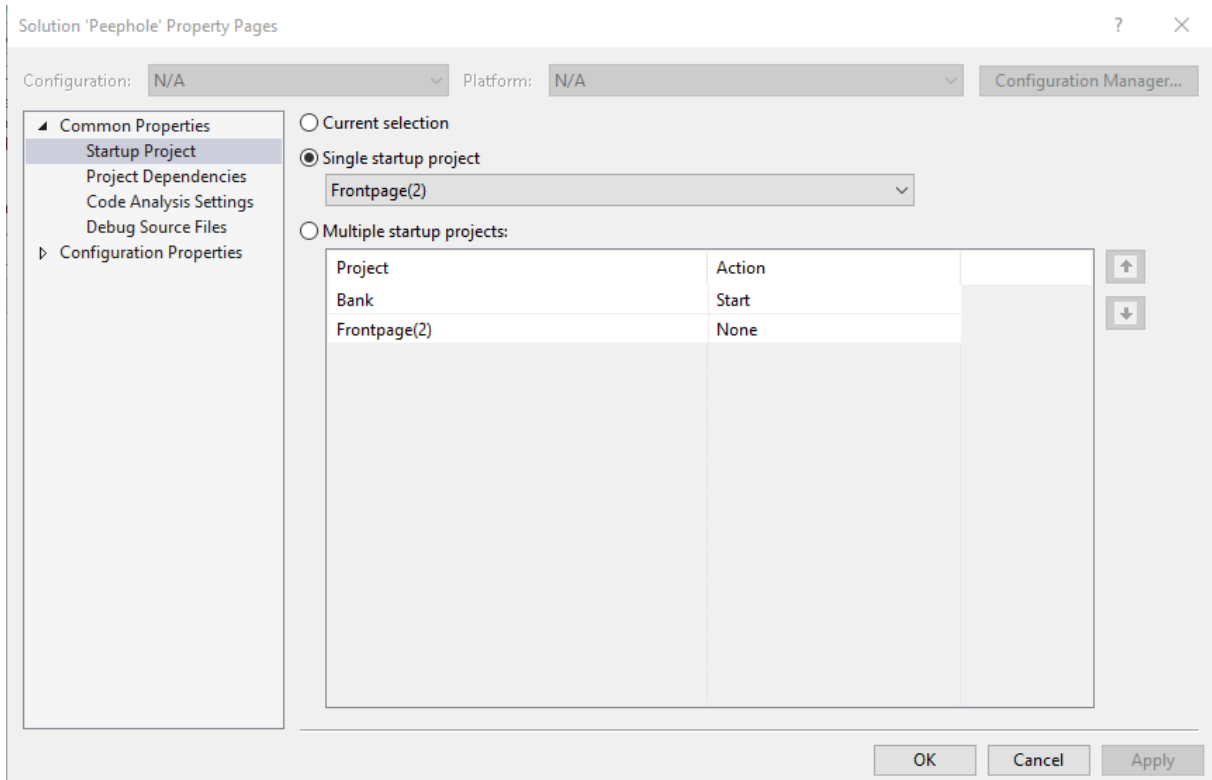


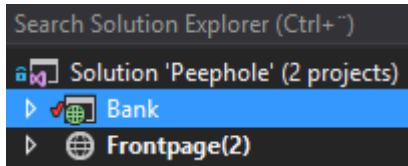
Figure 2.5: Choosing which project should start by default

3 Extending Peephole Bank

Peephole Bank is the first project in the Peephole Solution.

3.1 Adding modules / Areas

Extending “Peephole Bank” with Areas



1. View the file structure of the project
2. Right click on “Areas”
3. Choose “Add”
4. Choose “Area”
5. Write desired area name, which must be unique.
6. The new Area will have MVC structure by default. If Webforms is to be preferred,

3.1.1 MVC Area Specifics

To use shared design in Site.master, in your controllers, you have to replace

```
| return View();
```

with

```
| return this.RazorView();
```

3.1.2 Webforms Area Specifics

If you intend to use Webforms for the new Area, please delete the folders “Controllers”, “Models” and “Views”. These are only used when using the MCV structure.

Leave the *AreaRegistration.cs file intact. The AreaRegistration file will be adapted into registering the Area into the Peephole database / menu.

An example Webforms area is provided in Bank/Areas/_WebformsTemplate.

3.1.3 Adding the Area/Module To Menu

In the `*AreaRegistration.cs` file you need to add a line of code to make it appear the in the menu.

Beneath the code below

```
context.MapRoute(
    "MVCTemplate_default",
    "MVCTemplate/{controller}/{action}/{id}",
    new { action = "Index", id = UrlParameter.Optional }
);
```

Add this line of code:

```
AreasHandling.insertInMenu(context.AreaName);
```

The result would be:

```
public override void RegisterArea(AreaRegistrationContext context)
{
    context.MapRoute(
        "MVCTemplate_default",
        "MVCTemplate/{controller}/{action}/{id}",
        new { action = "Index", id = UrlParameter.Optional }
    );
    AreasHandling.insertInMenu(context.AreaName);
}
```

This applies to both MVC and Webforms Areas.

4 Using the application

4.1 Injection

The injection in our bank application is an SQL-injection done when you search for employees, but you insert a *malicious* query which the database interprets like a query to do something else than searching.

EMPLOYEES

Here you can search for employees by their ID. If you don't have the ID you have to option to show everyone.

Search for an employee

Search by ID

EmployeeID	Firstname	Lastname	Email
1	Malina	Peterson	malina@peephole.com
2	Allan	Vang	Allan@peephole.com
3	Sharyl	Akers	Sharyl@peephole.com
4	Margo	Duke	Margo@peephole.com
5	Mona	Scarlett	Mona@peephole.com
6	Allyn	Brooks	Allyn@peephole.com
7	Dex	Eustis	Dex@peephole.com
8	Havie	Presley	Harvie@peephole.com
9	Roar	Rolvsson	Roar@peephole.com
10	Keir	Aterbury	Keir@peephole.com

Screenshot 1: Show all employees

When using the “Employee”-site normally, you’ll get a list of all the employees with their ID, firstname, lastname and email. This is what you would expect from this kind of site.

Since this website is vulnerable the search-field could be inserted with malicious queries

which makes the database change or display information which is not intended. This is because the code is poorly written.

```
var sqlString = "SELECT * FROM Users WHERE ID = " + ID;
```

This is how the query on the server-side looks like. This means that if you write the number “1” in the search-field, you would return only “Malina Peterson” like this:

Search by ID

EmployeeID	Firstname	Lastname	Email
1	Malina	Peterson	malina@peephole.com

Screenshot 2: Search for specific employee

The database would read the following as `SELECT * FROM Users WHERE ID = 1`

But if you find out names of the other columns in the table, the attacker can retrieve anything from that database.

Search for an employee

Search by ID

EmployeeID	Firstname	Lastname	Email
1	3fc0a7acf087f549ac2b266baf94b8b1	Peterson	malina@peephole.com
2	e807f1fcf82d132f9bb018ca6738a19f	Vang	Allan@peephole.com
3	482c811da5d5b4bc6d497ffa98491e38	Akers	Sharyl@peephole.com
4	36311f1daffcf2f3adfec3e715bda92	Duke	Margo@peephole.com
5	4297f44b13955235245b2497399d7a93	Scarlett	Mona@peephole.com
6	b93eb53158e81c584c92bdcec579a4fe	Brooks	Allyn@peephole.com
7	3c0a7034e8bdc5422433a077a4993516	Eustis	Dex@peephole.com
8	7c7db8f086af5c959bea70596a804d9b	Presley	Harvie@peephole.com
9	daa02a9cc1810b5e359094924568f602	Rolvsson	Roar@peephole.com
10	94d755e45f49f9722b8ff3031cbaba7c	Aterbury	Keir@peephole.com

Screenshot 3: SQL-injection. Firstname is swapped with password

The database would read the following as `SELECT * FROM Users WHERE ID = 1; UPDATE Users SET Firstname = Password`

Now the attacker has the passwords hashed. Since this is hashed with MD5, cracking it would take seconds by using a rainbow table.

This could be fixed by checking if the input is other than numbers in this example (if the search is done by text, filter the text by symbols like ;'@-).

```
var positiveIntRegex = new Regex(@"^0*[1-9][0-9]*$");
```

```

if (!positiveIntRegex.IsMatch(ID))
{
    //Code
}
else {
    //Code
}

```

This code allows only number to be put into the search-field, making an SQL-injection harder to do.

Since all the tables is stored on the same database, it's possible to retrieve information from other tables because they're in the same database and you can input anything into the search field.

Search by ID	1; update users set firstname = Content from	Search	Show all employees
EmployeeID	Firstname	Lastname	Email
1	Try posting a comment with scripts insde	Peterson	malina@peephole.com
2	Try posting a comment with scripts insde	Vang	Allan@peephole.com
3	Try posting a comment with scripts insde	Akers	Sharyl@peephole.com
4	Try posting a comment with scripts insde	Duke	Margo@peephole.com
5	Try posting a comment with scripts insde	Scarlett	Mona@peephole.com
6	Try posting a comment with scripts insde	Brooks	Allyn@peephole.com
7	Try posting a comment with scripts insde	Eustis	Dex@peephole.com
8	Try posting a comment with scripts insde	Presley	Harvie@peephole.com
9	Try posting a comment with scripts insde	Rolvsson	Roar@peephole.com
10	Try posting a comment with scripts insde	Aterbury	Keir@peephole.com

Screenshot 4: Content from another table

There also a lot of new ways to display information from a database. One of these way is using ORM which is a technique converting data between type systems and object-oriented programming languages. The most commonly used in ASP.NET is Entity Framework¹. This eliminates the need for most of the data-access code for the developers and that means it's less vulnerable because of the lack of human mistakes.

EF is one of the most secure ways of displaying and using database information, but there's other ways:

- LINQ²
- Parametrized queries³
- Regular expression⁴ (which is used in the example code above)

All these have been documented by Microsoft with tutorials and examples.

4.2 Cross-Site Scripting (XSS)

In the OWASP Top 10 report XSS made it to a third place and a second place in WhiteHats report. Based on these reports we chose to implement the XSS vulnerability in our web application.

As mentioned above, there are three types of XSS. In our web application we chose to implement two of them; Reflected and Stored XSS. This decision was made due to time limitations and that we already strongly represented XSS in our application.

We are making the application look more like a real example and less like a "go-to-vulnerability" kind of application. The user is presented with links and functionality usually found in a bank, so we had to

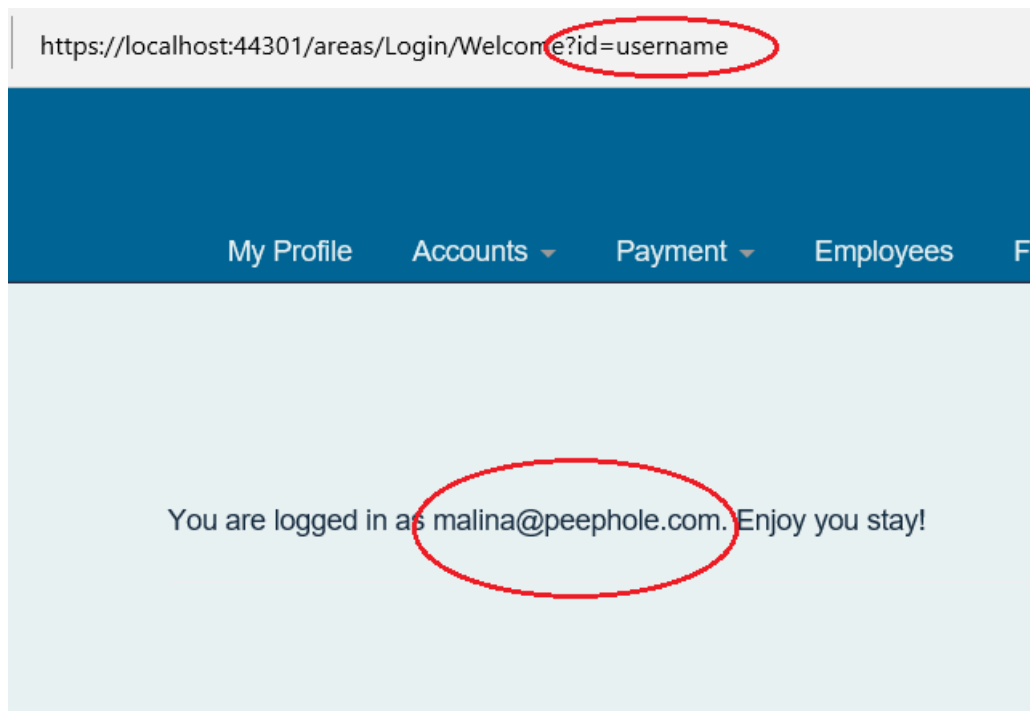
¹ <http://www.asp.net/entity-framework>

² <https://msdn.microsoft.com/en-us/library/bb907622%28v=vs.100%29.aspx?f=255&MSPPErrors=-2147217396>

³ <http://www.asp.net/web-forms/overview/data-access/accessing-the-database-directly-from-an-aspnet-page/using-parameterized-queries-with-the-sqldatasource-vb>

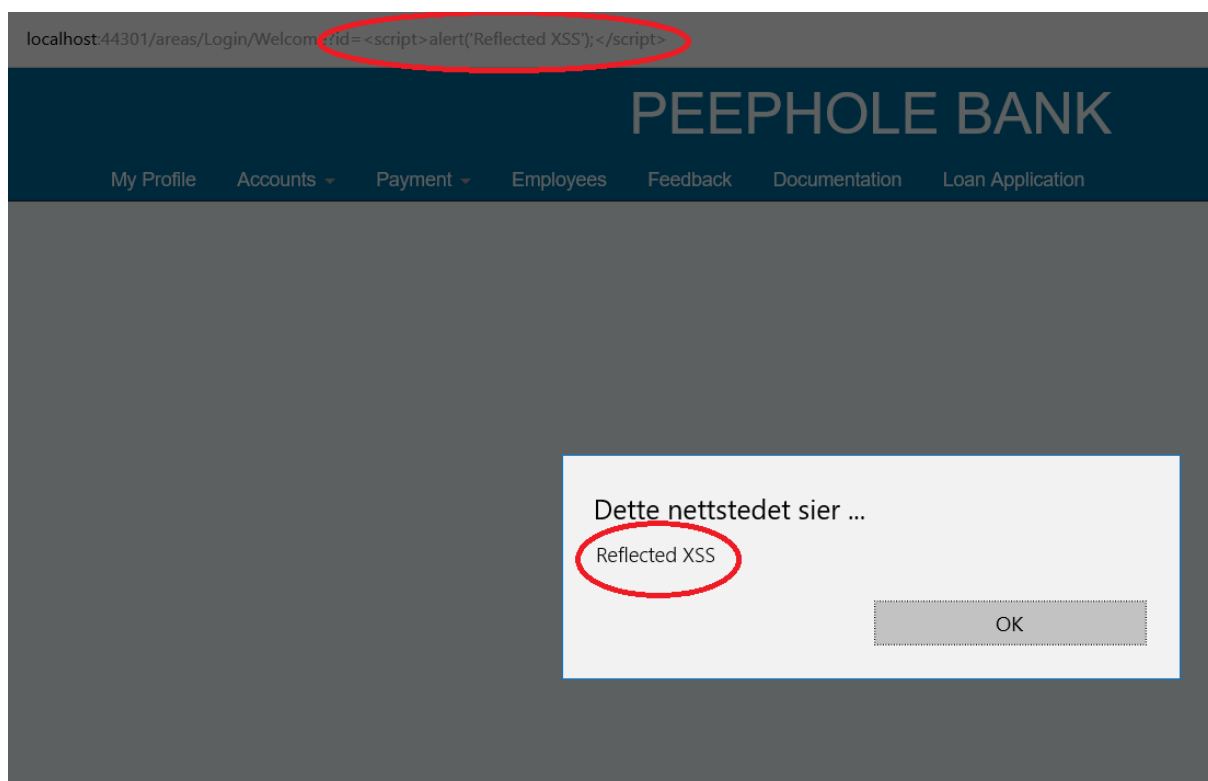
⁴ <https://msdn.microsoft.com/en-us/library/ms972966.aspx>

find natural points to implement the vulnerabilities. Reflected XSS can be found and only accessed after a user has logged in. After the user has been authenticated he is redirected to a welcome page, this is where we show a static welcome message along with the username in the form of a variable seen in the URL. Look at the below picture for an example:



Screenshot 5- Screenshot showing that the username is reflected back on the site through a variable in the URL

In order to exploit this vulnerability, you can put malicious code directly into the URL field. Most of the big browsers have their own filtering of such vulnerabilities by checking the URL input and filtering out specific input. Some browser however, does not filter such input. Among those are Internet Explorer and Microsoft Edge. This is why the examples are shown in Microsoft Edge.



Screenshot 6- Screenshot showing what happens when we change the variable in the URL to a script

In the picture above we can see that the reflected XSS was successful. The script used here and that can be seen some of in the URL is `<script>alert('Reflected XSS');</script>`. This is just a simple alert box, but with such a vulnerability you can do an enormous amount of damage. You can for example steal credentials in non-HTTPOnly cookies, send requests to a server with the user's credentials, make the user download content, display a password input, log keystrokes, and send the result to a site of your choosing. These are just a few of the things you can do with reflected XSS.

Performing the same example in for example Google Chrome would require to either disable the filtering itself. This however, does not mean it is completely secure. The input can be encoded in a way not yet known to the filter implemented in the browser for example, and the script will run. Protecting against XSS is mainly the developers' responsibility, but most of the big browsers have acknowledged the vulnerability and taken steps to help reduce the risk.

The stored XSS can be found under customer feedback in our application, this is where customers and visitors can leave a comment about the bank or their experience with the bank service. The user input is handled and displayed back to the web application without any kind of validation. Here is an example of a normal comment:

The screenshot shows a web interface for a comment system. At the top, there is a form with three input fields: 'Your Name' (placeholder: 'Enter Name'), 'Subject' (placeholder: 'Enter subject'), and 'Content' (a larger text area). Below these fields are two buttons: a green 'Submit' button and an orange 'Reset Database' button. Under the form, two example comments are displayed. The first comment is from a 'Developer' with the subject 'Example' and content 'Try posting a comment with scripts insde'. The second comment is from a 'Customer' with the subject 'Feeback' and content 'I am concerned about the security of this online bank.'

Your Name

Enter Name

Subject

Enter subject

Content

Submit Reset Database

Name: Developer
Subject: Example
Content: Try posting a comment with scripts insde

Name: Customer
Subject: Feeback
Content: I am concerned about the security of this online bank.

Screenshot 7- A normal comment form displaying comments back on the site.

In the picture above we see two normal comments with no harmful code. We also see a button called "Reset Database". This is implemented so that users can try out the vulnerability without concern for breaking the application, as it will be restored to a default state once you push the reset database button. In the next picture we will see what happens when we use the same script as we did in the reflected XSS but with different text inside: `<script>alert('Stored XSS');</script>`

We in Peephole bank would love to hear from you. Please leave a note.

Your Name

Subject

Content

Name: Developer
Subject: Example
Content: Try posting a comment with scripts inside

Melding fra nettside X
! Stored XSS

Screenshot 8- Here we make a comment with a script. The result is that the script is run on the site.

Here we can see that we get the same result as in the reflected XSS. This time the script is stored into a database and then displayed back onto the site. This means that everyone visiting this specific page after the exploit has been done successfully will run the script.

To prevent XSS in .NET, there are two main countermeasures:

- Constrain input
- Encode output

The .NET framework provides some methods to help prevent XSS, both for constraining input and for encoding output. Using the provided methods where they are needed should help secure against XSS. In ASP.NET version 4.5, Microsoft included AntiXSS based methods to help developer protect against XSS, whereas earlier versions of .NET have an AntiXSS library. Fully preventing cross site scripting is harder than it may seem. OWASP has a list of over 80 vectors that can be targeted using XSS.

Following is an example of XSS on a big social network:

A few years ago there was an XSS vulnerability on Myspace, which a user found out about and created the XSS worm called Samy [1]. MySpace had a filter for “javascript” and a lot of “<tags>” to protect against XSS, but these could be bypassed by altering the format of the words, like “javascript = java\nscript” would not be filtered. The script itself was not that harmful, but worked as a great example of how dangerous such a vulnerability could be. The XSS worm carried a payload that would display the string “but most of all, samy is my hero” on a victim’s MySpace profile page. When a user viewed that profile page, the payload would then be replicated to their own profile page which resulted in a continuing distribution of the worm. Within 20 hours of its release, over one million users had run the payload, making Samy the fastest spreading virus of all time.

4.3 Broken authentication and Session Management

Broken authentication and session management encompasses security flaws to all aspects of user authentication and session management, including but not limited to: Password strength, password storage, weak session IDs, accessible account lists on the website and browser caching.

This vulnerability is represented on several points in our application. When making a default ASP.NET application, Microsoft has a default login with strong password requirements. These requirements included special characters, letters, password length of 8+ characters and upper and lower case characters. For the sake of the application, we chose to make our own custom register and login to better show the security threat with weak passwords, insufficient hashing and storing of passwords along with weak protection. With our weak password requirements, the user can make passwords that are easily guessed as seen in this list of most common passwords of 2015⁵. This list also indicates that there are still many sites out there with a weak password policy.

4.4 Insecure Direct Object References Jens

Insecure Direct Object Reference is implemented as a part of the user being able to edit their profile from “My Profile” page accessible from the main menu.

An attacker would create an account, access the edit profile page and inspect the packet sent when submitting the form. The attacker will easily spot the user ID being submitted and suspect the possibility for editing other users accounts simply by changing the ID. The packet can be replayed for ID 1, 2, 3, N, setting the password to whatever he prefers. Once the password has been changed the attacker can simply log in to the victims account and transfer the available money to a bank account of choice.

We’ve made a YouTube which contains how this works⁶

⁵ <http://www.computerworld.com/article/3024404/security/worst-most-common-passwords-for-the-last-5-years.html>

⁶ <https://youtu.be/jWGMPw2JizI?t=3m22s>

4.5 Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) is a highly relevant and underestimated vulnerability that we have implemented or rather, not added protection against, on the account money transfer form. The defence against this in .NET is a helper called "AntiForgeryToken". This helper generates a unique ID for each form, which is impossible for an attacker site to guess. This means that when an attacker site tries to submit a form with the victim's permissions, the targeted site will see that the form comes from an untrusted source since it lacks the unique ID.

Even though our application is vulnerable against CSRF we do not yet have a good way of simulating it. To simulate CSRF you need to have an attacker site, which we have not prioritized due to the restricted development time we have spent on the application.

4.6 Security Misconfiguration

Security misconfiguration is a term that describes when any one part of our application stack has not been hardened against possible security vulnerabilities. In OWASP Security misconfiguration is listed at number 5 of their top 10 most critical web application security flaws. When developing an application in .NET as we did, ASP.NET applications can be configured to produce debug binaries which may be extremely helpful when developing, however, when releasing an application, such a configuration can give extremely helpful information to attackers since it displays direct information about the backend of the system. An example to this is improper error handling. When an application crashes without handling the exception, it will display a stack trace and information not meant for users. This information may prove very useful for an attacker wanting to learn more about the system. Our application produces such an error when you try to search for a user and search with special characters.

Serverfeil i programmet /.

Invalid column name '#'.

Beskrivelse: Det oppstod et ubehandlet unntak under kjøring av gjeldende webforespørsel. Gå gjennom stakkspringen hvis du vil ha mer informasjon om fe

Unntaksdetaljer: System.Data.SqlClient.SqlException: Invalid column name '#'.
Kildefil:

```
Linje 55:         var command = new SqlCommand(sqlString, conn);
Linje 56:         command.Connection.Open();
Linje 57:         userInfo.DataSource = command.ExecuteReader();
Linje 58:         userInfo.DataBind();
Linje 59:     }
```

Kildefil: C:\Users\Per-Olav\Documents\peephole\Peephole\Areas\SQLInjection\SQL.aspx.cs **Linje:** 57

Stakkspring:

```
[SqlException (0x80131904): Invalid column name '#'.]
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection,
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakCor
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Bool
System.Data.SqlClient.TdsParser.TryRun(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDe
System.Data.SqlClient.SqlDataReader.TryConsumeMetaData() +58
System.Data.SqlClient.SqlDataReader.get_MetaData() +89
System.Data.SqlClient.SqlCommand.FinishExecuteReader(SqlDataReader ds, RunBehavior runBehavi
System.Data.SqlClient.SqlCommand.RunExecuteReaderTds(CommandBehavior cmdBehavior, RunBehavio
System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior r
System.Data.SqlClient.SqlCommand.RunExecuteReader(CommandBehavior cmdBehavior, RunBehavior r
System.Data.SqlClient.SqlCommand.ExecuteReader(CommandBehavior behavior, String method) +246
System.Data.SqlClient.SqlCommand.ExecuteReader() +99
Peephole.Areas.SQLInjection.SQL.ButtonSubmitClick(Object sender, EventArgs e) in C:\Users\F
System.Web.UI.WebControls.Button.OnClick(EventArgs e) +9692746
System.Web.UI.WebControls.Button.RaisePostBackEvent(String eventArgument) +108
```

Screenshot 9- Showing the error produced by incorrect input when searching for users

From the error produced above we can see that we got direct contact with the database in the system, this gives us a good indication that the system is vulnerable to SQL-injection.

4.7 Insecure Cryptographic Storage

In our project we've used MD5 as the hashing algorithm. This means that that we're using a hashing algorithm that should not be used to protect confidential information (like passwords). It should not be used because it's unsalted and it's really fast to decrypt. MD5 is also vulnerable to collision, that means two inputs producing the same hash. In theory that means if the password is "123" that gives a specific hash output, but another string, let's say "Asjdaskdbv879" could give the same hash output and the attacker login to account that has the password "123" with "Asjdaskdbv879".

There's also MD5 rainbow tables available online, which makes cracking MD5 hashes incredible easy (MD5Cracker.org).

If you do an SQL injection like we've done in [Injection](#), you'll get the hashed password as an output. Insert one of those hashes into MD5cracker.org and you'll have the password in clear text. After you have the password, you can proceed to logging in with the username and password.

	Id	Firstname	Lastname	Password	Email
▶	1	Malina	Peterson	3fc0a7acf087f549ac2b266baf94b8b1	malina@peeph...
	2	Allan	Vang	e807f1fcf82d132f9bb018ca6738a19f	Allan@peeph...
	3	Sharyl	Akers	482c811da5d5b4bc6d497ffa98491e38	Sharyl@peeph...
	4	Margo	Duke	36311f1daffcf2f3adfec3e715bda92	Margo@peeph...
	5	Mona	Scarlett	4297f44b13955235245b2497399d7a93	Mona@peeph...
	6	Allyn	Brooks	b93eb53158e81c584c92bdcec579a4fe	Allyn@peeph...
	7	Dex	Eustis	3c0a7034e8bdc5422433a077a4993516	Dex@peephole....
	8	Havie	Presley	7c7db8f086af5c959bea70596a804d9b	Harvie@peeph...
	9	Roar	Rolvsson	daa02a9cc1810b5e359094924568f602	Roar@peeph...
	10	Keir	Aterbury	94d755e45f49f9722b8ff3031cbaba7c	Keir@peephole....

Screenshot 10: The passwords is stored in the database as hashed passwords. This got to be secure, right?

If we try taking the first hashed password into MD5cracker.org we'll see the what the password actually is.

As I mentioned this works for MD5, but there's also rainbow tables for a lot of other hashing algorithms such as SHA1 and LM hash.

3fc0a7acf087f549ac2b266baf94b8b1

✓ md5cracker.org
result: qwerty123

✗ [TMT0\[dot\]ORG](mailto:TMT0[dot]ORG)
error: not found

✓ md5online.net
result: qwerty123

✓ MD5.My-Addr.com
result: qwerty123

Screenshot 11: The result of the MD5 crack. This took less than a second.

The best solution for hashing now is the SHA3, which was released in August 2015 by NIST and FIPS. This is more secure because cracking it takes long a long time and there is no collision.

Microsoft has not yet created a SHA3 hashing method, but they have one for SHA2-256⁷.

⁷ <https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256%28v=vs.110%29.aspx?f=255&MSPPErr=-2147217396>

4.8 Failure to Restrict URL Access Jens

Failure to restrict URL access is implemented several places in Peephole Bank.

My Profile

When accessing “My Profile” the user is directed to URL

<https://localhost:44301/CustomerInfo/Default/Edit/1>

An attacker can simply change the last number in the URL, a number representing user ID, to edit some other person’s account

Edit bank account

The URL presented to the user when editing a bank account is

<https://localhost:44301/TransferFunds/BankAccounts/Edit/3994045019>

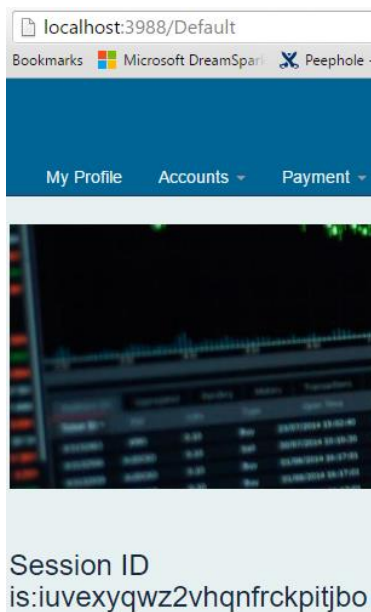
While not being the most serious vulnerability, an attacker could change the bank account name and type by guessing another person’s bank account number in the URL. As all account increment by 1 from 3994.04.5000 other people’s account numbers are easily discoverable and the information could be used for instance to increase the success rate of a spear phishing attack.

4.9 Insufficient Transport Layer Protection

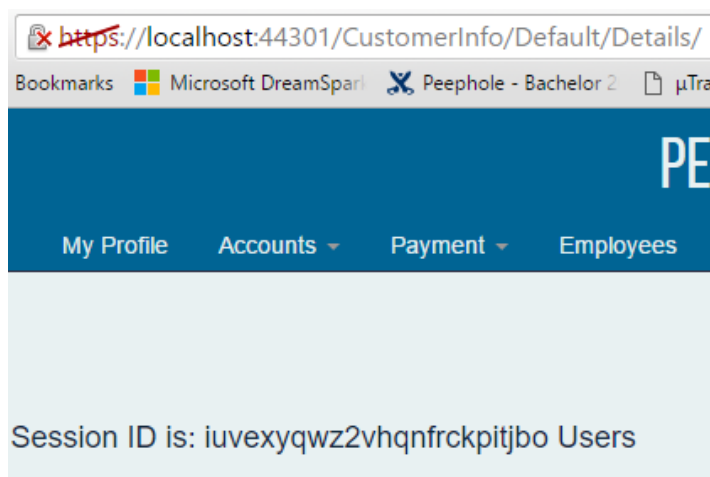
SSL is only partly implemented in Peephole Bank. As the user first enters the bank the session ID is created and transmitted in cleartext over unprotected HTTP. The transmitted information does not get encryption from HTTPS until the user logs in. The session ID remains the same leaving a severe risk of session hijacking.

The attacker could simply capture the transmitted packets when the victim is connected to an unencrypted public Wi-Fi and get the hold of the session ID. Once the victim has authenticated against the bank the attacker holds a valid, authenticated session and can perform whatever actions the victim is authenticated to do.

Two-factor authentication or any other level of secure user authentication would not help as long as the authenticated session ID is up for a grab.



Screenshot 13: Session ID when user is not logged in, HTTP



Screenshot 12: Session ID when is logged in, HTTPS

4.10 Unvalidated Redirects and Forwards

The unvalidated redirect isn't really a website, or kind of it is. Once this site is loaded it just redirect to another site and this pretty much happens instant. That means that the user doesn't even notice it.

What this site does is getting a single parameter from the URL which is redirects to. Here an example

<http://localhost:3988/Redirect/Redirect/Redirect?url=http://www.ntnu.no>

What this does is first accessing the Redirect-module, checking the URL for the parameter called "url" (...?url=http://www.ntnu.no) and redirect to that page. The code that does this is really simple:

```
string url = Request.QueryString["url"];
Response.Redirect(url);
```

But how can this be used by an attacker's points of view? Because the parameter is in the URL, the attacker can change this before sending it to the victim. If the victim trusts the site that redirects (in this case localhost:3988), he/she probably wouldn't check the whole URL. If the attacker changes the parameter to "http://www.malware.com" the victim most probably wouldn't notice.

<http://localhost:3988/Redirect/Redirect/Redirect?url=http://www.malware.com>

From the victim's perspective, this link is pointing towards localhost:3988 and not a malicious site.

The safest way to make this non-existent is using making a redirect page for each redirect you want to do. Instead of requesting the redirecting-URL as a parameter from the URL, you'll put the redirecting URL directly into the Redirect function like this:

```
| Response.Redirect("http://www.safeURL.com");
```

By doing it this way, the attacker cannot change the URL to his advantage.

5 Known bugs

5.1 Site.Master changes

When changing the Site.Master (which contains the structure for the web application), the designer changes the namespace automatically. Why this happens we do not know (guessing we have about 20-30 hours of research, testing and failing on this, but we have not found a way to fix it).

Below you can see what needs to be changed every time there's a saved change in Site.Master. If we're not doing this, the application would not run and you'll get this error:

Server Error in '/' Application.

Parser Error

Description: An error occurred during the parsing of a resource required to service this request. Please review the following specific parse error details and modify your source file appropriately.

Parser Error Message: The type 'System.Web.Mvc.ViewMasterPage' is ambiguous: it could come from assembly 'C:\Users\olemartin\Documents\peephole\Peephole\bin\System.Web.Mvc.DLL' or from assembly 'C:\Users\olemartin\Documents\peephole\Peephole\bin\Peephole.DLL'. Please specify the assembly explicitly in the type name.

Source Error:

```
Line 1: <%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.Master.cs" Inherits="System.Web.Mvc.ViewMasterPage" %>
Line 2:
Line 3: <%@ Register TagPrefix="uc" TagName="uc1" Src="~/SiteMenu.ascx" %>
```

Source File: /Views/Shared/Site.Master Line: 1

Screenshot 14: Error message when changing the Site.Master

How to fix this:

1. This happens when changing the Site.Master file located in ~/Views/Shared/Site.Master. When changing this file, the Site.Master.designer.cs updates automatically. Originally it should be
`namespace Peephole {`
But changes to
`namespace System.Web.Mvc {`
when we edit the Site.Master.Master file.
2. To fix this, go to the designer file and change the System.Web.Mvc to Peephole and it should work.