

Burp Extender

Table of Contents

Burp Extender.....	1
Write the sample application.....	2
Test the application with automated tools.....	6
Vega.....	6
Skipfish.....	7
SQLmap.....	8
Set up the development environment.....	16
Write the extender.....	32
HelloWorld extender.....	36
Correct the checksum.....	38
Convert the hash to hex array.....	42
URL decode the values.....	45

Write the sample application

As a very simple sample we create a login form, which sends the username and password, and the SHA1 checksum of these two values. The sample will contain three files, a `hacktivity.html`, a `hacktivity.php`, and an `internal.php` files. Use for example the following code as the html file:

```
<body>
<script type="text/javascript" src="./jquery-1.10.2.min.js"></script>
<script type="text/javascript" src="./crypto-
js/rollups/sha1.js"></script>
<div id="error"></div>
User Name:<input type="text" id="username" /> <br/>
Password:<input type="password" id="password"/> <br/>
<button id="button">Send</button>
<script>
$("#button").click(function() {

    sr = "username=" + $("#username").val() + "&" +
        "password=" + $("#password").val() + "&" +
        "checksum=" + CryptoJS.SHA1($("#username").val() + $
("#password").val());

    $.ajax({
    url : 'hacktivity.php',
    method:'POST',
    data: sr,
    success: function(data) {
        if (data == 'true'){
            window.location="internal.php";
        }
        else {
            $("#error").html("Wrong username or password");
        }
    }
    });
});
</script>
</body>
```

As we can see this application sends the SHA1 hash of the username and password fields as well, and the server side application will check if it correct. Because the automated tools are not able to figure out the logic, they start to test the fields, but of course will not correct the checksum value to it. It will cause that, the automated tool practically will not test the application at all, because the first if which tests the checksum will fail, and the remaining part of the code contains the problem will never run.

```

hacktivity.html x
1 <body>
2 <script type="text/javascript" src="./jquery-1.10.2.min.js"></script>
3 <script type="text/javascript" src="./crypto-js/rollups/sha1.js"></script>
4 <div id="error"></div>
5 User Name:<input type="text" id="username" /> <br/>
6 Password:<input type="password" id="password"/> <br/>
7 <button id="button">Send</button>
8 <script>
9 $( "#button" ).click(function() {
10
11     sr = "username=" + $( "#username" ).val() + "&" +
12         "password=" + $( "#password" ).val() + "&" +
13         "checksum=" + CryptoJS.SHA1( $( "#username" ).val() + $( "#password" ).val() );
14
15     $.ajax({
16         url : 'hacktivity.php',
17         method:'POST',
18         data: sr,
19         success: function(data) {
20             if (data == 'true'){
21                 window.location="internal.php";
22             }
23             else {
24                 $( "#error" ).html("Wrong username or password");
25             }
26         }
27     });
28 });
29 </script>
30 </body>

```

This html file it will require the crypto-js library, what can be download from:
<https://code.google.com/p/crypto-js/> and requires the jquery library, what can be download from:
<http://jquery.com/download/>

As we can see, it sends the data to the hacktivity.php application, which has the following code (as we can see it contains a nice SQL injection error):

```

<?php
session_start();
$hash = sha1($_POST["username"] . $_POST["password"]);
$con = mysqli_connect("127.0.0.1", "root", "", "a") or die('false');

if (strcasecmp($hash,$_POST["checksum"]) === 0) {
    $sql = "SELECT COUNT(*) FROM tbl1 WHERE username='" .
    $_POST["username"] .
    "' and password='" . $_POST["password"] . "';";
    $res = mysqli_query($con,$sql) or die('false');
    $row = mysqli_fetch_row($res);
    if ($row[0] == 1) {
        echo "true";
        $_SESSION["auth"] = 1;
    }
}

```

```

        else {
            echo "false";
        }
    }
else {
    echo "CEHCKSUM ERROR: " . $hash . " " . $_POST["checksum"] .
        " " . $_POST["username"] . $_POST["password"];
}
mysqli_close($con);
?>

```

```

1  <?php
2  session_start();
3  $hash = sha1($_POST["username"] . $_POST["password"]);
4  $con = mysqli_connect("127.0.0.1", "root", "", "a") or die('false');
5
6  if (strcasecmp($hash, $_POST["checksum"]) === 0) {
7      $sql = "SELECT COUNT(*) FROM tbl1 WHERE username='" . $_POST["username"] .
8          "' and password='" . $_POST["password"] . "'";
9      $res = mysqli_query($con, $sql) or die('false');
10     $row = mysqli_fetch_row($res);
11     if ($row[0] == 1) {
12         echo "true";
13         $_SESSION["auth"] = 1;
14     }
15     else {
16         echo "false";
17     }
18 }
19 else {
20     echo "CEHCKSUM ERROR: " . $hash . " " . $_POST["checksum"] .
21         " " . $_POST["username"] . $_POST["password"];
22 }
23 mysqli_close($con);
24 ?>

```

It redirect us to the internal.php (in reality it is not so important for us):

```

<?php
session_start();
if (!isset($_SESSION["auth"])) {
    header('Location: hacktivity.html');
    die('error');
}
echo "THIS IS THE INTERNAL SITE";
?>

```

```
internal.php x
1  <?php
2  session_start();
3  if (!isset($_SESSION["auth"])) {
4      header('Location: hacktivity.html');
5      die('error');
6  }
7  echo "THIS IS THE INTERNAL SITE";
8  ?>
```

Test the application with automated tools

Now if we try to find the SQL injection error with any automated tool, obviously they will not find the SQL injection vulnerability

The screenshot displays the Netsparker 3.5.8.0 Community Edition interface. The title bar shows the IP address 192.168.168.251 and the version [Community Edition]. The menu bar includes File, View, Reporting, Tools, and Help. The toolbar contains buttons for Start New Scan, Import Links, and Start Proxy.

The **Site Map** panel on the left shows the scanned site structure for 192.168.168.251:80, including folders like hackactivity2014 and files like hackactivity.html, jquery-1.10.2.min.js, crypto.js, and hackactivity.php. The map also lists several detected issues, including Cross-site Scripting and Internal Path Disclosure.

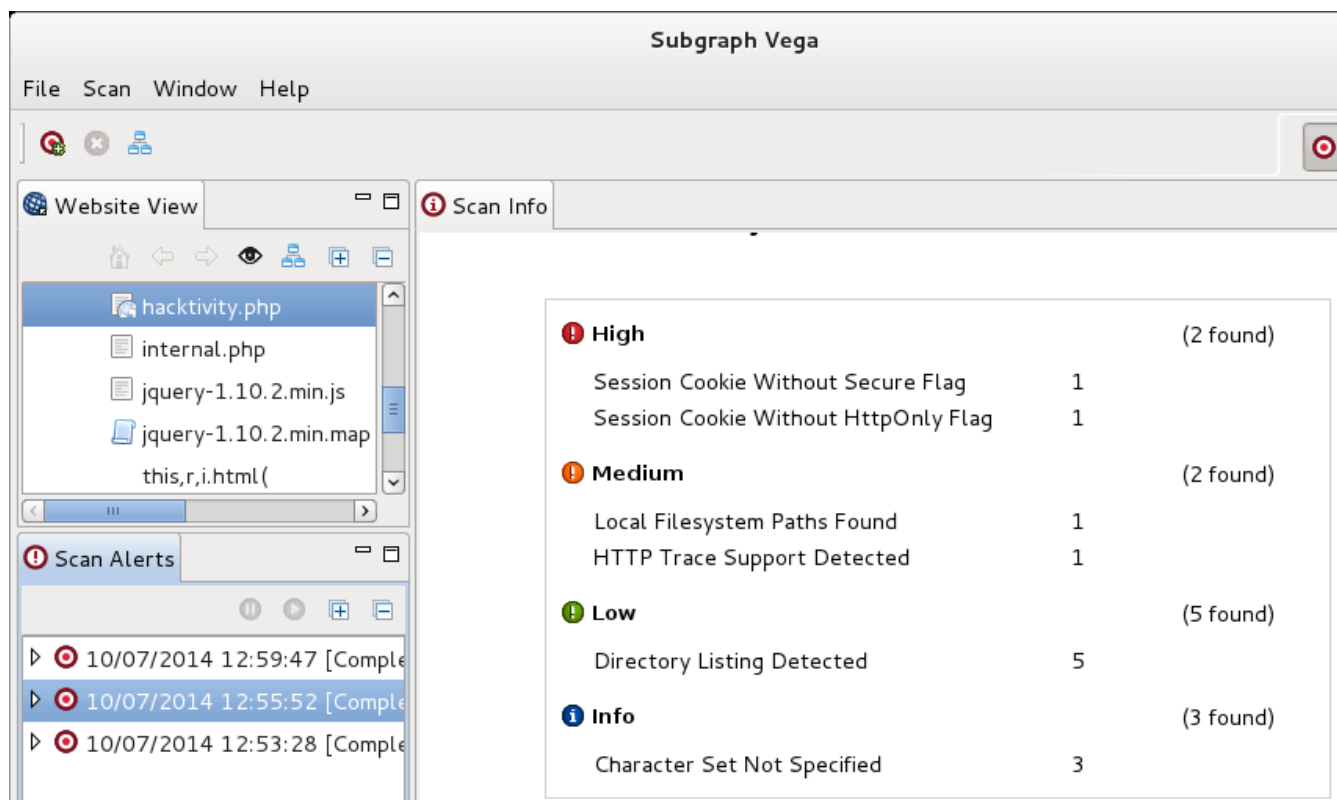
The main panel displays the IP address 192.168.168.251 and a summary of findings: **IMPORTANT (3)**, **LOW (4)**, and **INFORMATION (12)**. To the right of this summary are sliders for **Concurrent Connections** and **Activity**.

The **Dashboard** panel at the bottom left shows the scan status as **Scan Finished** with a 100% progress bar. It also displays scan statistics: 0000 / 0000 requests, a current speed of 77.3 req/sec, an average speed of 40.6 req/sec, 3331 total requests, 0 failed requests, 336 HEAD requests, and an elapsed time of 00:01:22.

The **Issues (19)** panel at the bottom right lists the following issues:

- Cross-site Scripting
- Version Disclosure (OpenSSL)
- Version Disclosure (PHP)
- TRACE/TRACK Method Detected
- Version Disclosure (Apache)
- Forbidden Resource
- E-mail Address Disclosure
- Out-of-date Version (PHP)
- Apache Web Server Identified
- Out-of-date Version (Apache)
- Out-of-date Version (OpenSSL)
- [Possible] Internal Path Disclosure (*nix)
- [Possible] Internal Path Disclosure (Windows)

Vega



Skipfish

No problem, try another one, for example the skipfish, it is included in the kali linux.
Use the following command to strat it:

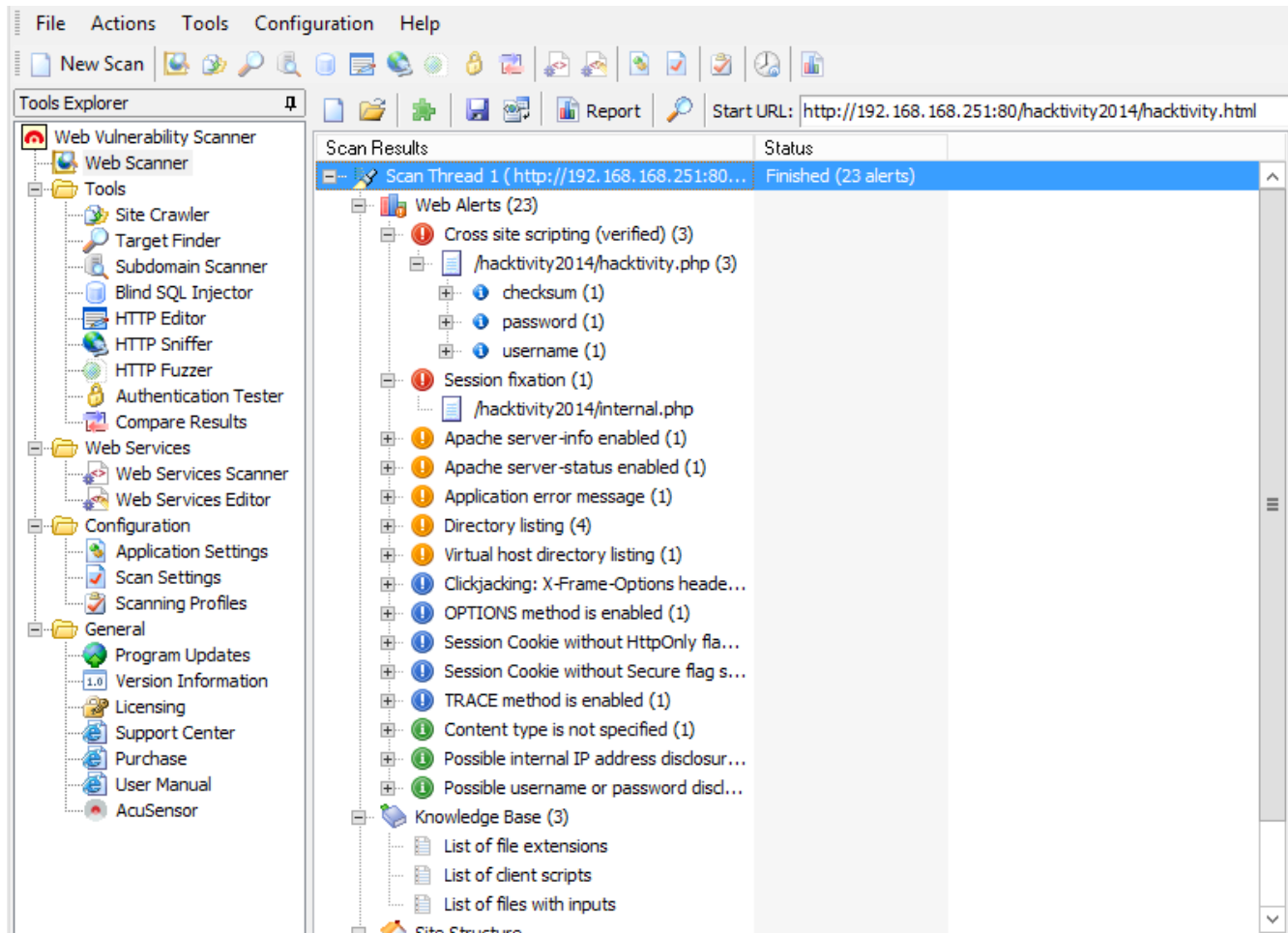
```
skipfish -W ./a.wl -S /usr/share/skipfish/dictionary/minimal.wl
-o ./test http://192.168.168.251/hacktivity2014/hacktivity.html
```

Issue type overview - click to expand:

- 🟡 Interesting server message (2)
- ⚫ Resource fetch failed (5)
- 🟢 Numerical filename - consider enumerating (38)
- 🟢 Incorrect or missing charset (low risk) (70)
- 🟢 Incorrect or missing MIME type (low risk) (3)
- 🟢 Password entry form - consider brute-force (1)
- 🟢 Directory listing enabled (24)
- 🟢 New 404 signature seen (2)
- 🟢 New 'X-*' header value seen (4)
- 🟢 New 'Server' header value seen (1)
- 🟢 New HTTP cookie added (1)

NOTE: 100 samples maximum per issue or document type.

Acunetix



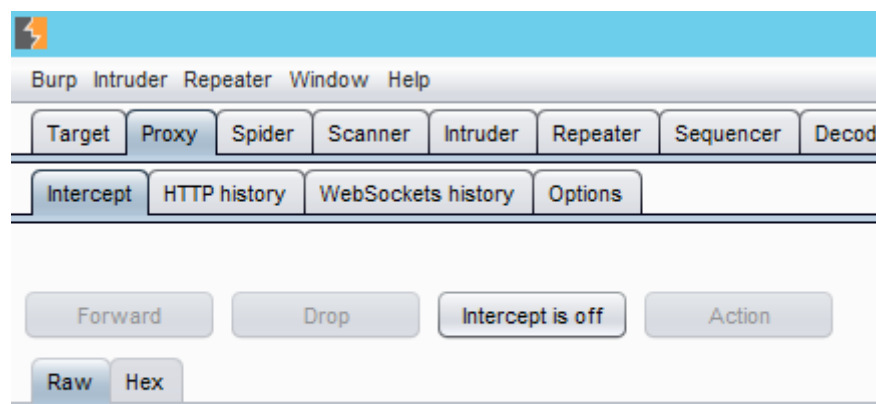
SQLmap


```
Administrator: Command Prompt
[05:45:01] [WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS. You can try to explicitly set it using option '--dbms'
[05:45:03] [INFO] testing 'Generic UNION query (random number) - 1 to 10 columns'
[05:45:05] [INFO] testing 'Generic UNION query (NULL) - 11 to 20 columns'
[05:45:06] [INFO] testing 'Generic UNION query (random number) - 11 to 20 columns'
[05:45:08] [INFO] testing 'Generic UNION query (NULL) - 21 to 30 columns'
[05:45:10] [INFO] testing 'Generic UNION query (random number) - 21 to 30 columns'
[05:45:12] [INFO] testing 'Generic UNION query (NULL) - 31 to 40 columns'
[05:45:13] [INFO] testing 'Generic UNION query (random number) - 31 to 40 columns'
[05:45:15] [INFO] testing 'Generic UNION query (NULL) - 41 to 50 columns'
[05:45:17] [INFO] testing 'Generic UNION query (random number) - 41 to 50 columns'
[05:45:19] [WARNING] User-Agent parameter 'User-Agent' is not injectable
[05:45:19] [CRITICAL] all tested parameters appear to be not injectable. Also, you can try to rerun by providing either a valid value for option '--string' (or '--regexp')

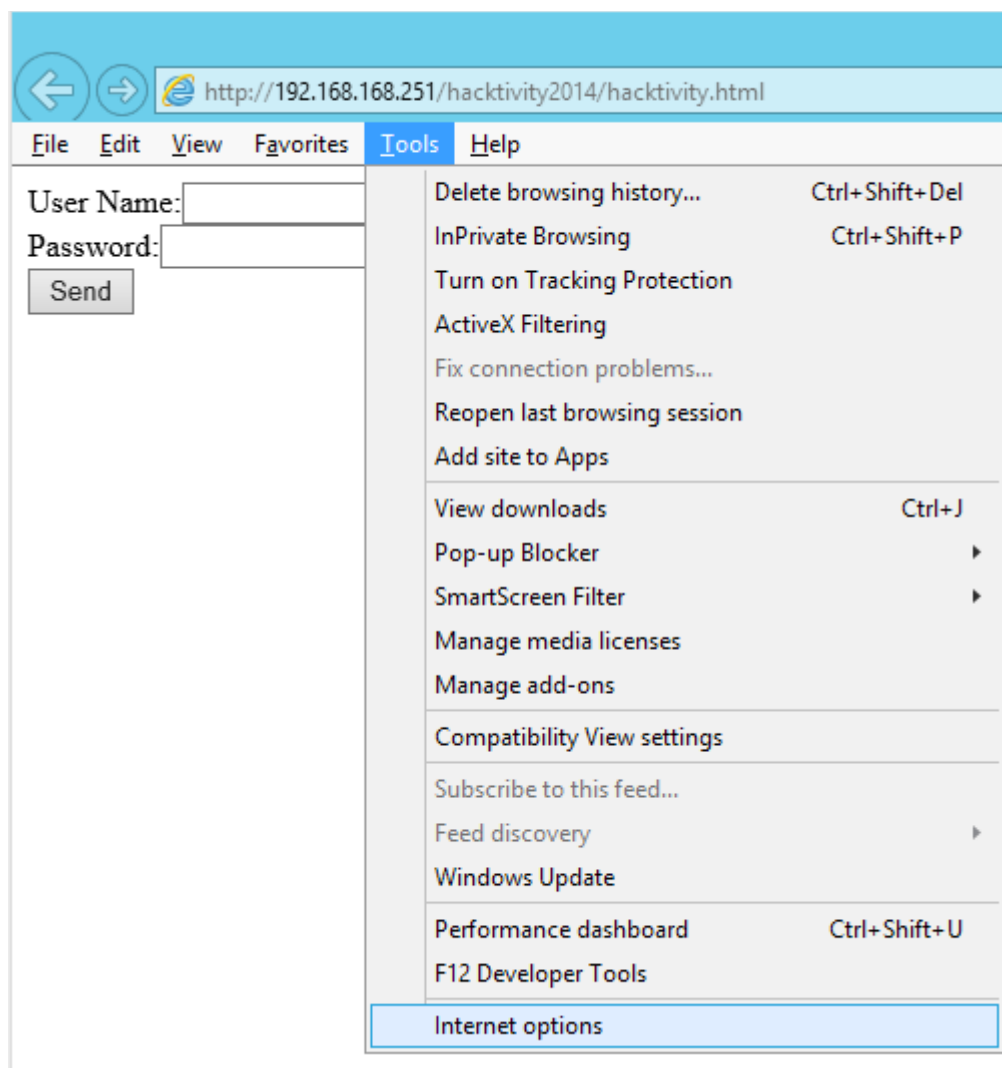
[*] shutting down at 05:45:19

C:\sqlmap>
```

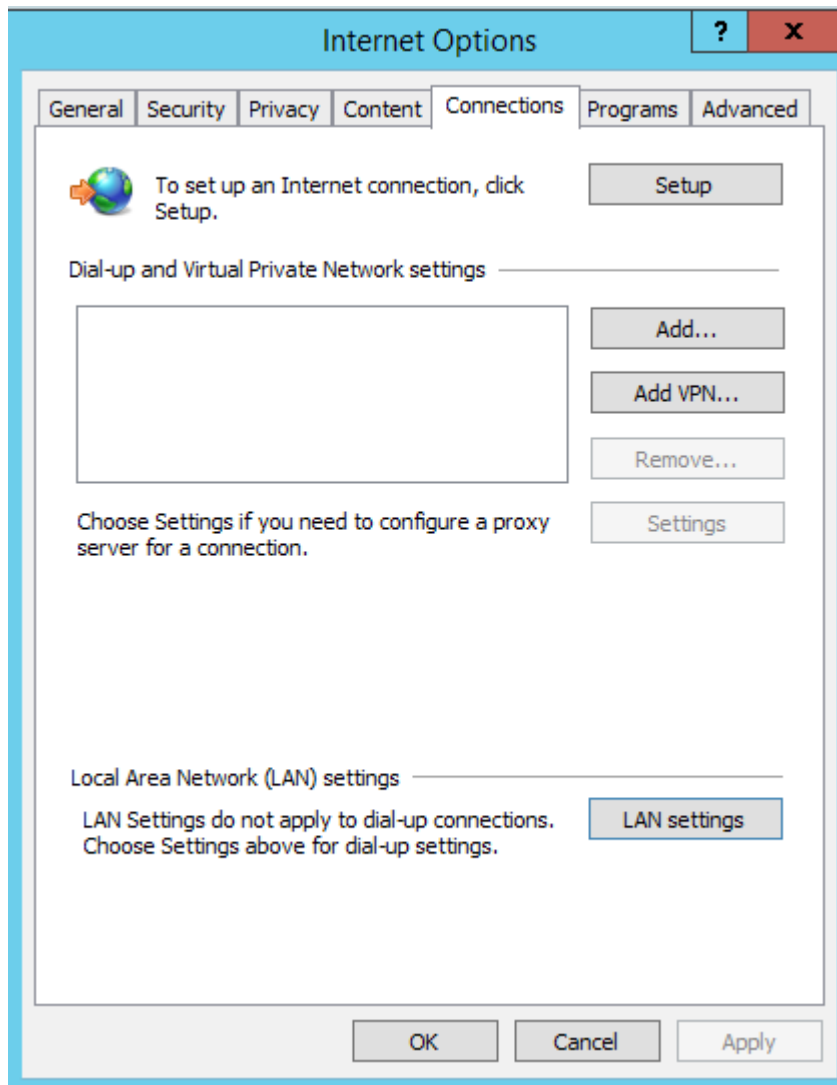
We can use the sqlmap with a request file as well, it may work better. So first capture a request. Start the burp proxy, and turn off the intercept:



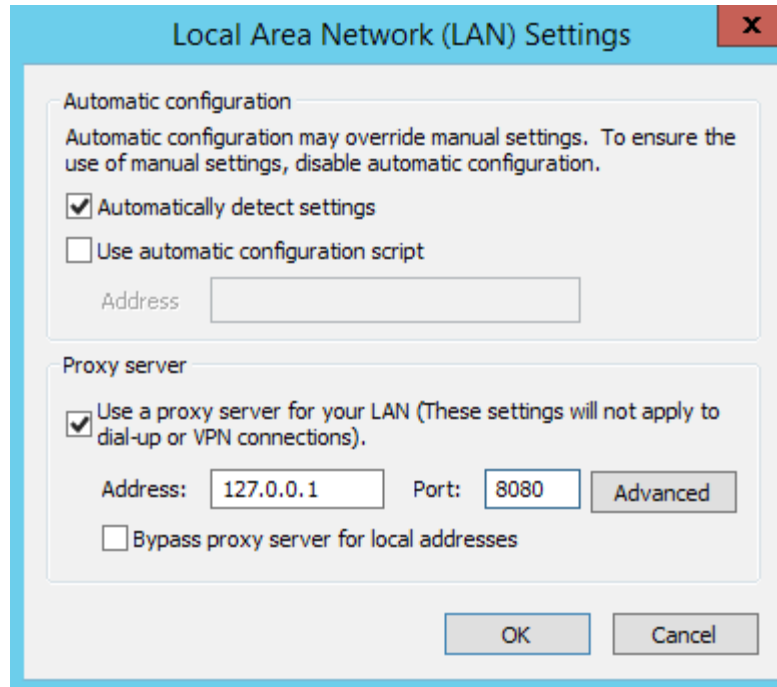
Then set up your browser, to use it as proxy (Tools / Internet options)



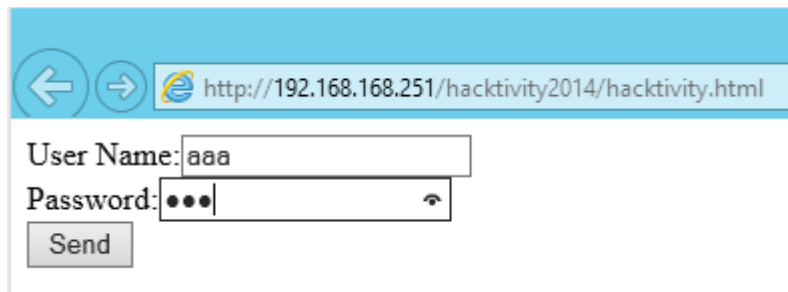
On the Connections tabs click to the LAN settings button.



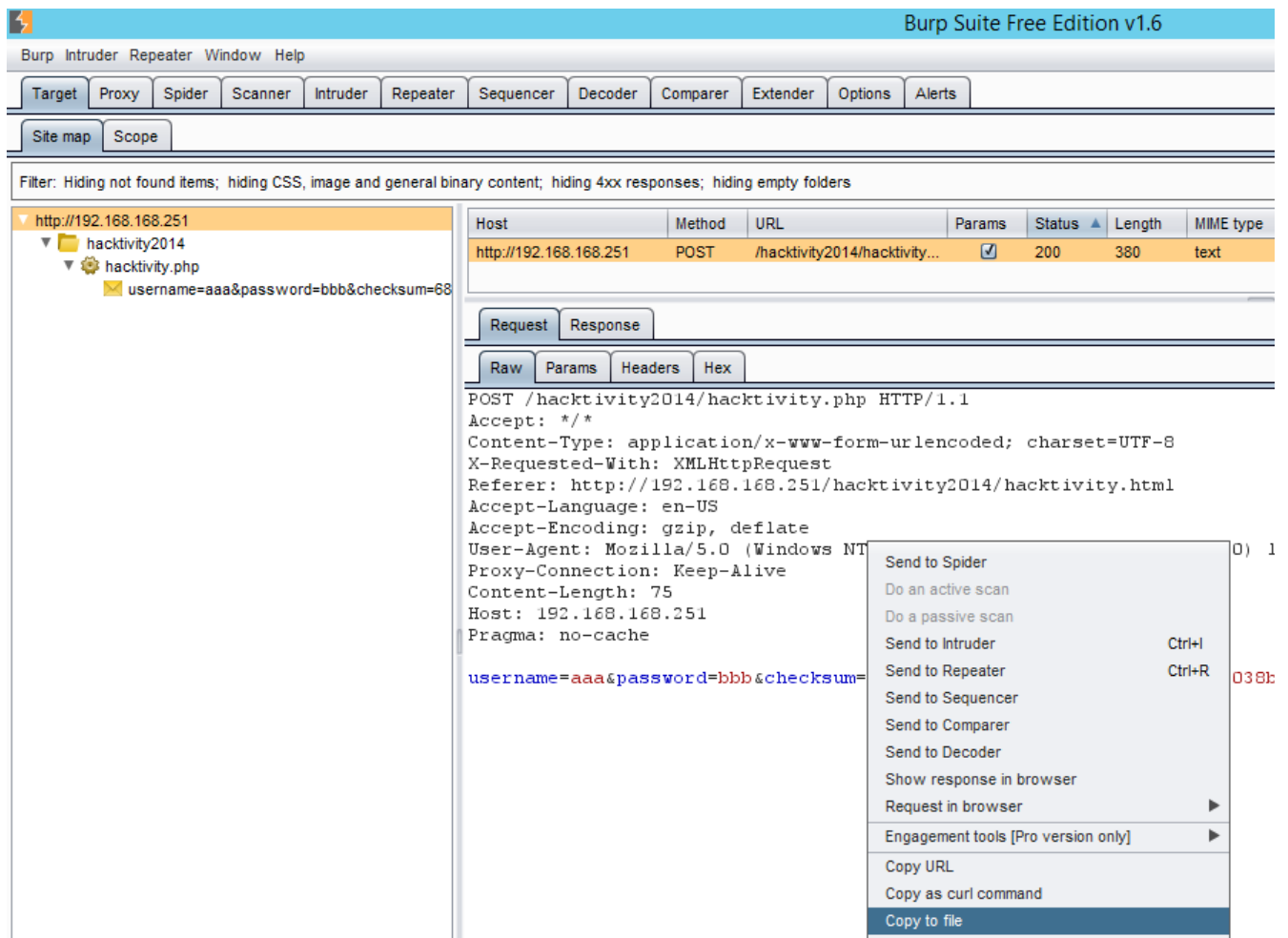
Use the burp proxy (127.0.0.1 and port 8080) as proxy:



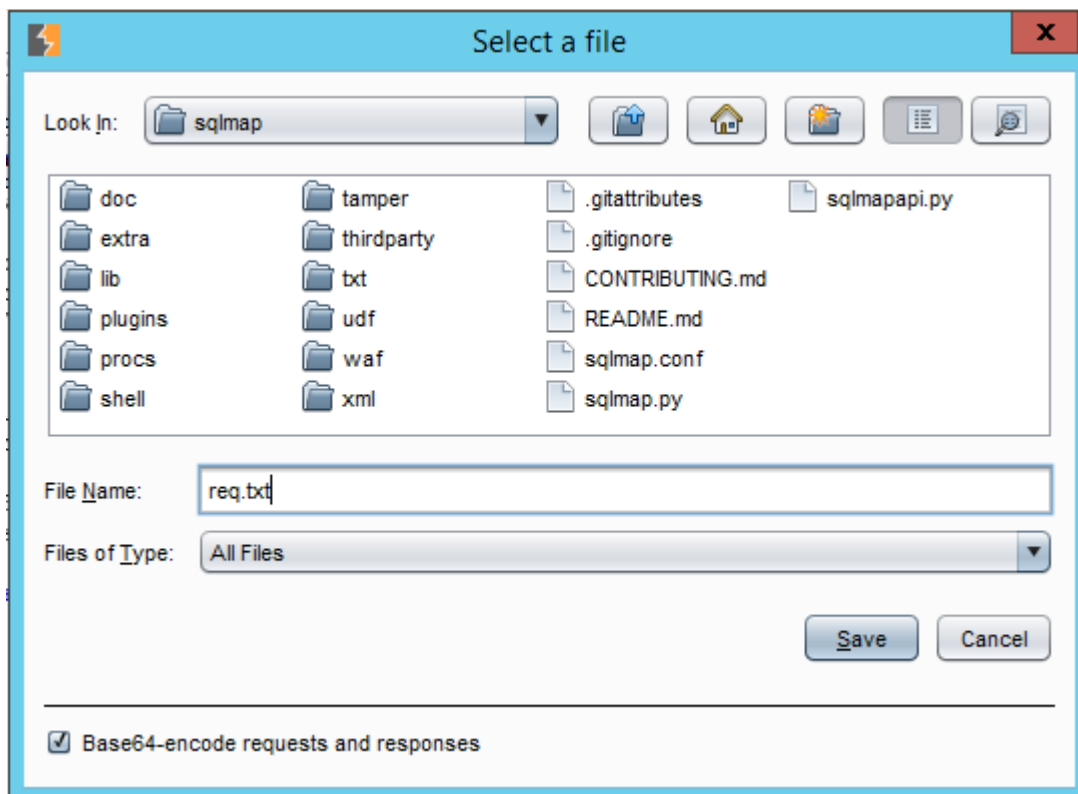
Then go to the login page, and type any username and password:



Find the request in burp, then right click to it, and select the copy to file command from the popup menu:



The save the request as a text file



Open the file with any text editor.

A screenshot of a Notepad window titled "req.txt - Notepad". The window shows the following text:

```
POST /hacktivity2014/hacktivity.php HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://192.168.168.251/hacktivity2014/hacktivity.html
Accept-Language: en-US
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Proxy-Connection: Keep-Alive
Content-Length: 75
Host: 192.168.168.251
Pragma: no-cache

username=aaa&password=bbb&checksum=68d8572c2662b0f06f723d7d507954fb038b8558
```

Then change the username and password to *, because it marks to the sqlmap the testable positions.

```
req.txt - Notepad
File Edit Format View Help
POST /hacktivity2014/hacktivity.php HTTP/1.1
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: http://192.168.168.251/hacktivity2014/hacktivity.html
Accept-Language: en-US
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Proxy-Connection: Keep-Alive
Content-Length: 75
Host: 192.168.168.251
Pragma: no-cache

username=*&password=*&checksum=68d8572c2662b0f06f723d7d507954fb038b8558
```

Then run the sqlmap with the following command:

```
sqlmap -level=5 -risk=3 -r req.txt
```

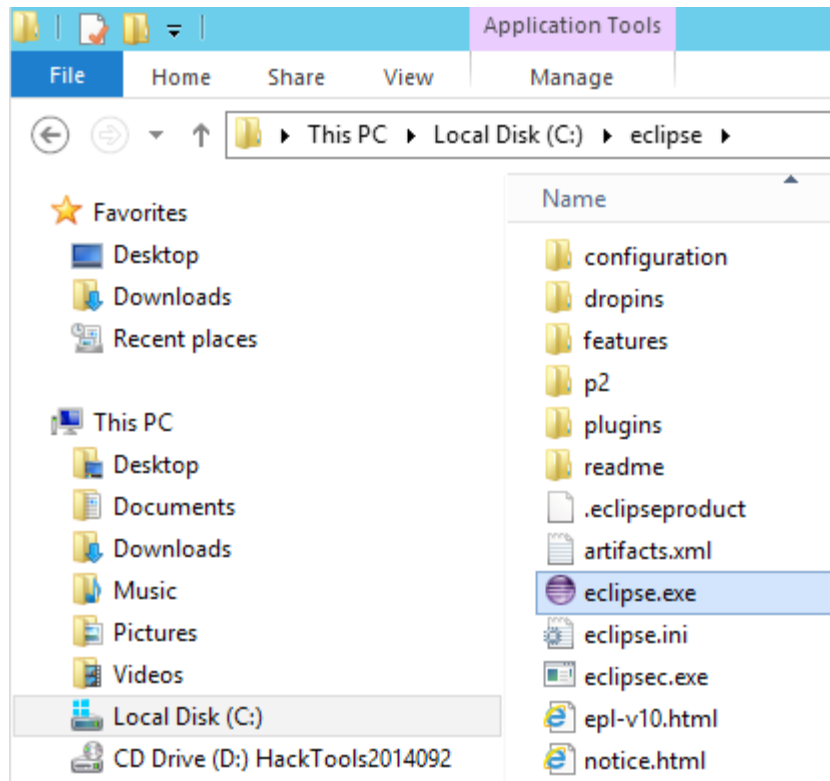
```
Administrator: Command Prompt
[06:05:03] [INFO] testing 'MySQL time-based blind - Parameter replace (ELT)'
[06:05:03] [INFO] testing 'MySQL >= 5.0.11 time-based blind - GROUP BY and ORDER BY clauses'
[06:05:03] [INFO] testing 'MySQL < 5.0.12 time-based blind - GROUP BY and ORDER BY clauses (heavy query)'
[06:05:03] [INFO] testing 'MySQL UNION query (58) - 1 to 10 columns'
[06:05:07] [INFO] testing 'MySQL UNION query (58) - 11 to 20 columns'
[06:05:09] [INFO] testing 'MySQL UNION query (58) - 21 to 30 columns'
[06:05:11] [INFO] testing 'MySQL UNION query (58) - 31 to 40 columns'
[06:05:13] [INFO] testing 'MySQL UNION query (58) - 41 to 50 columns'
[06:05:16] [INFO] testing 'Generic UNION query (58) - 1 to 10 columns'
[06:05:19] [INFO] testing 'Generic UNION query (58) - 11 to 20 columns'
[06:05:24] [INFO] testing 'Generic UNION query (58) - 21 to 30 columns'
[06:05:26] [INFO] testing 'Generic UNION query (58) - 31 to 40 columns'
[06:05:28] [INFO] testing 'Generic UNION query (58) - 41 to 50 columns'
[06:05:34] [WARNING] User-Agent parameter 'User-Agent' is not injectable
[06:05:34] [CRITICAL] all tested parameters appear to be not injectable. Also, you can try to rerun by providing either a valid value for option '--string' (or '--regexp')

[*] shutting down at 06:05:34

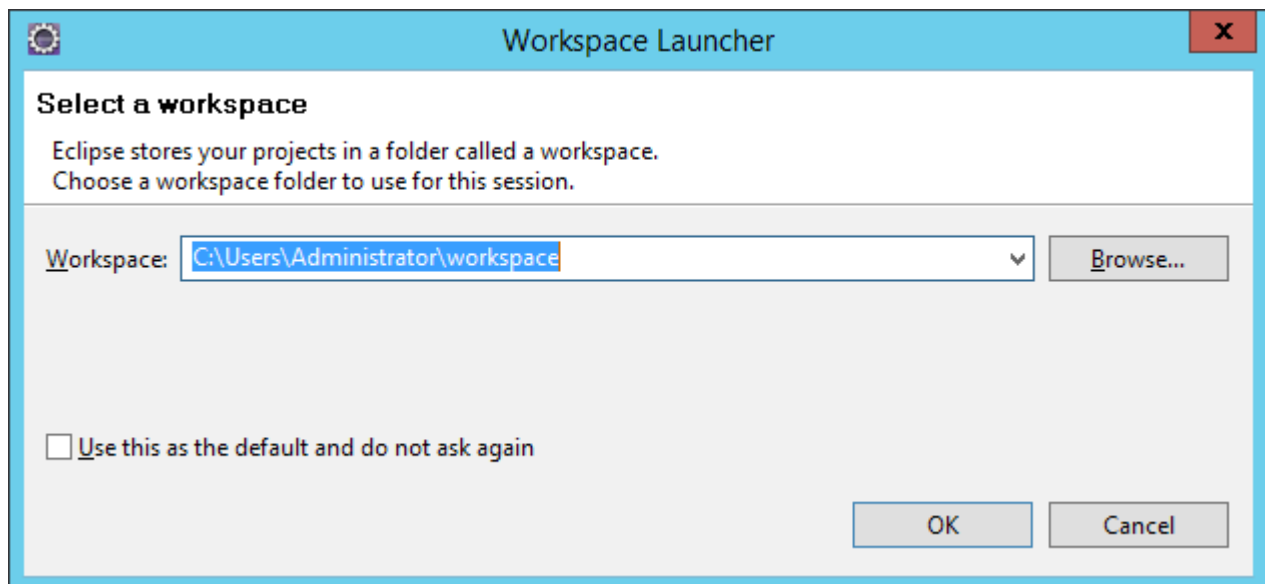
C:\sqlmap>
```

Set up the development environment

The eclipse is a portable application, so just unzip the downloaded eclipse IDE (www.eclipse.org), and copy the whole directory to your machine. I copied it to the eclipse directory, and start here the eclipse.exe



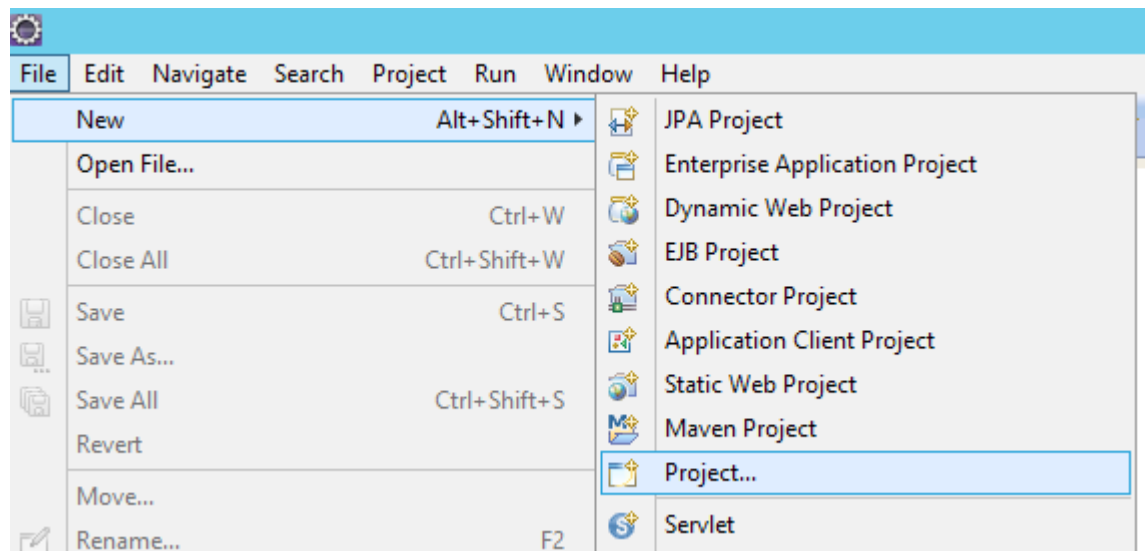
When it asks for a workspace position just accept the default one, and click to the OK button:



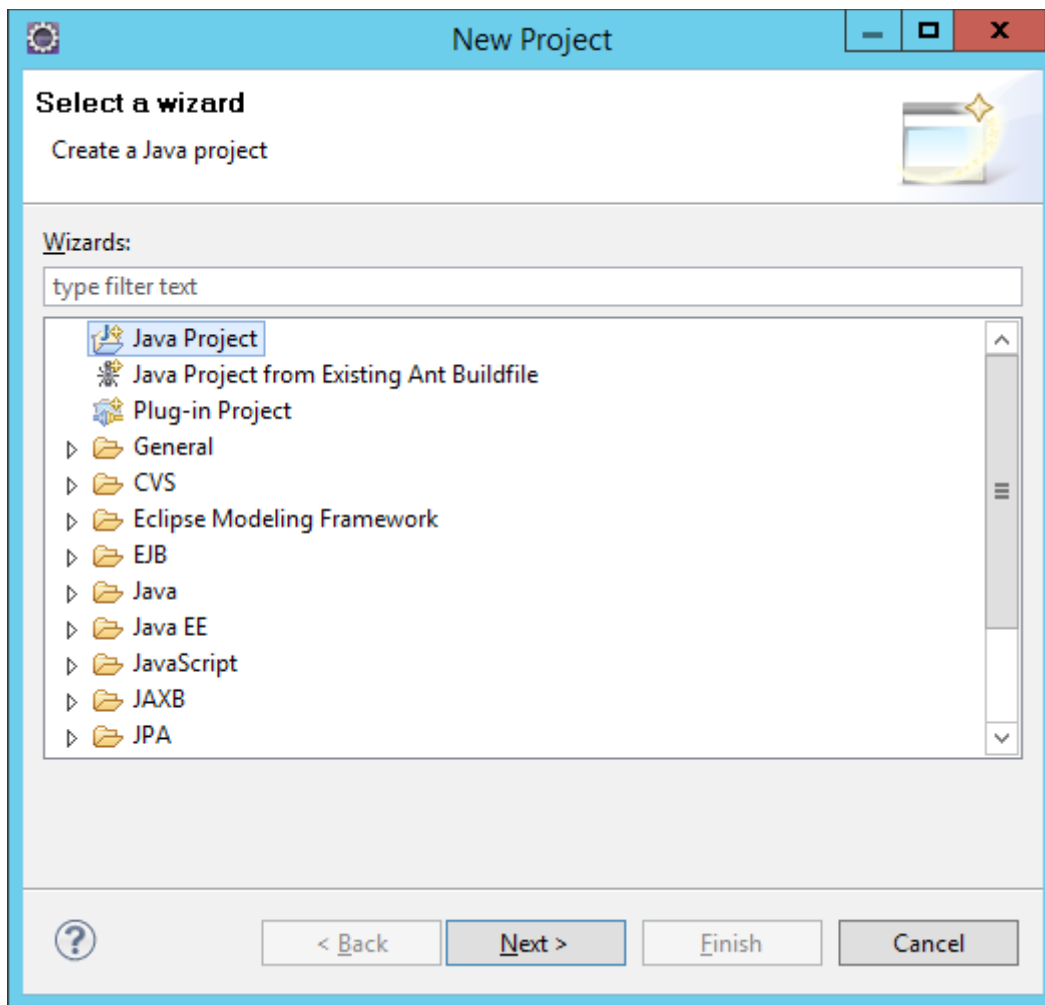
Close the welcome screen




Then select the File / New / Project... command.




In the popup window select Java project as project type, then click to the next button:



Set the project specific JRE for me the 1.8.0_20 was installed so I used that one. Then click to the Next button:

 **New Java Project** [minimize] [maximize] [close]

Create a Java Project
Create a Java project in the workspace or in an external location. 

Project name:

☒ Use default location

Location:

JRE

☐ Use an execution environment JRE:

☒ Use a project specific JRE:

☐ Use default JRE (currently 'jre1.8.0_20') [Configure JREs...](#)

Project layout


☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

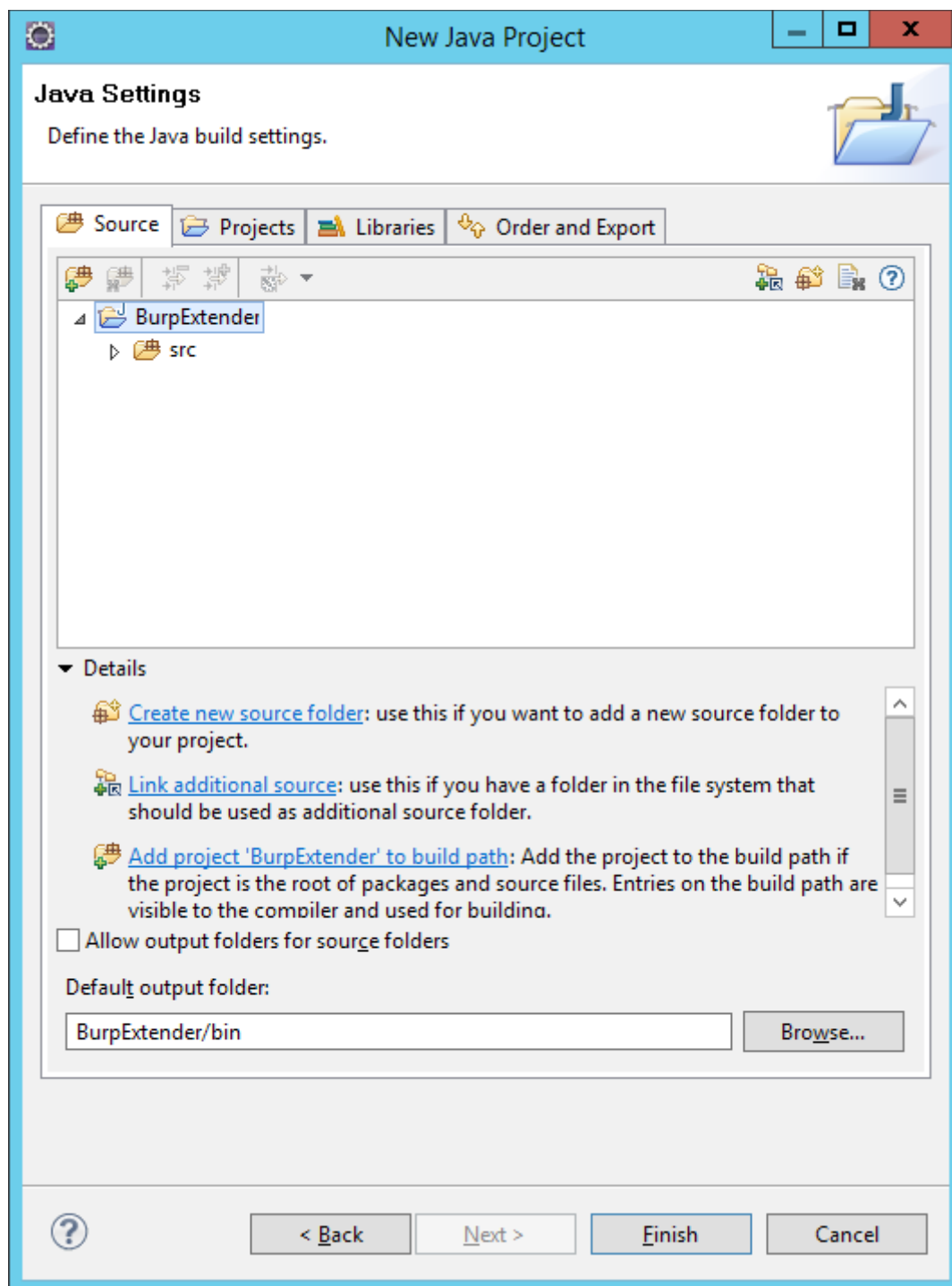
Working sets

☐ Add project to working sets

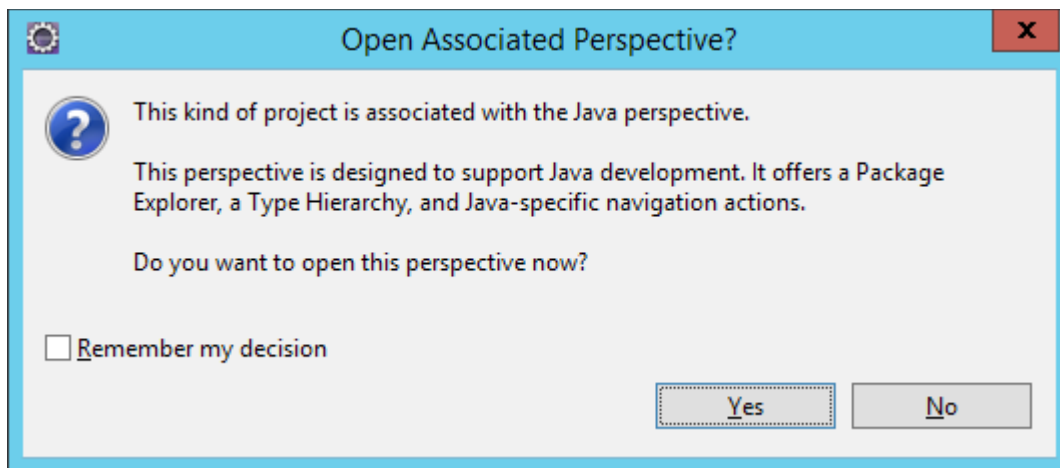
Working sets:



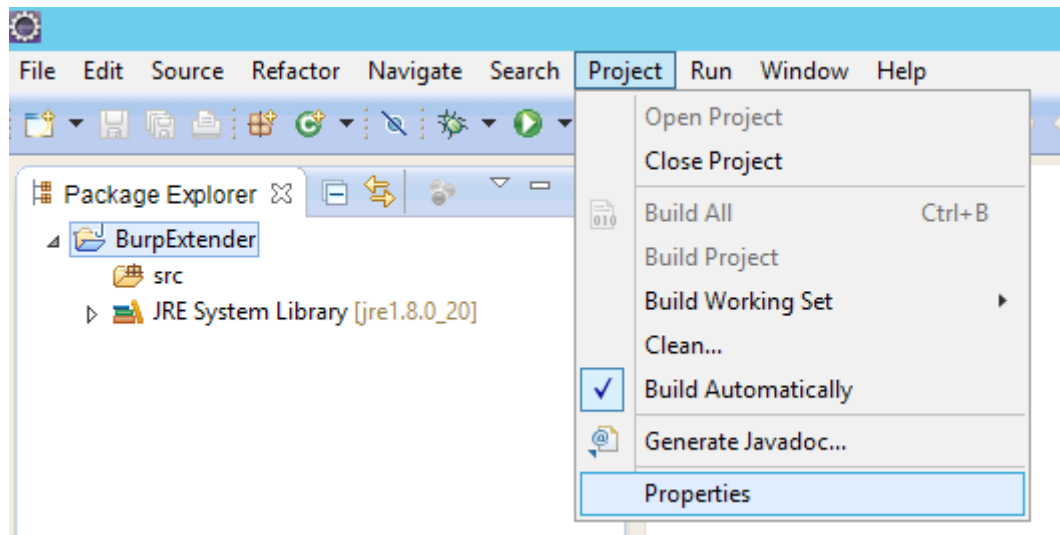
Then click to the Finish button:



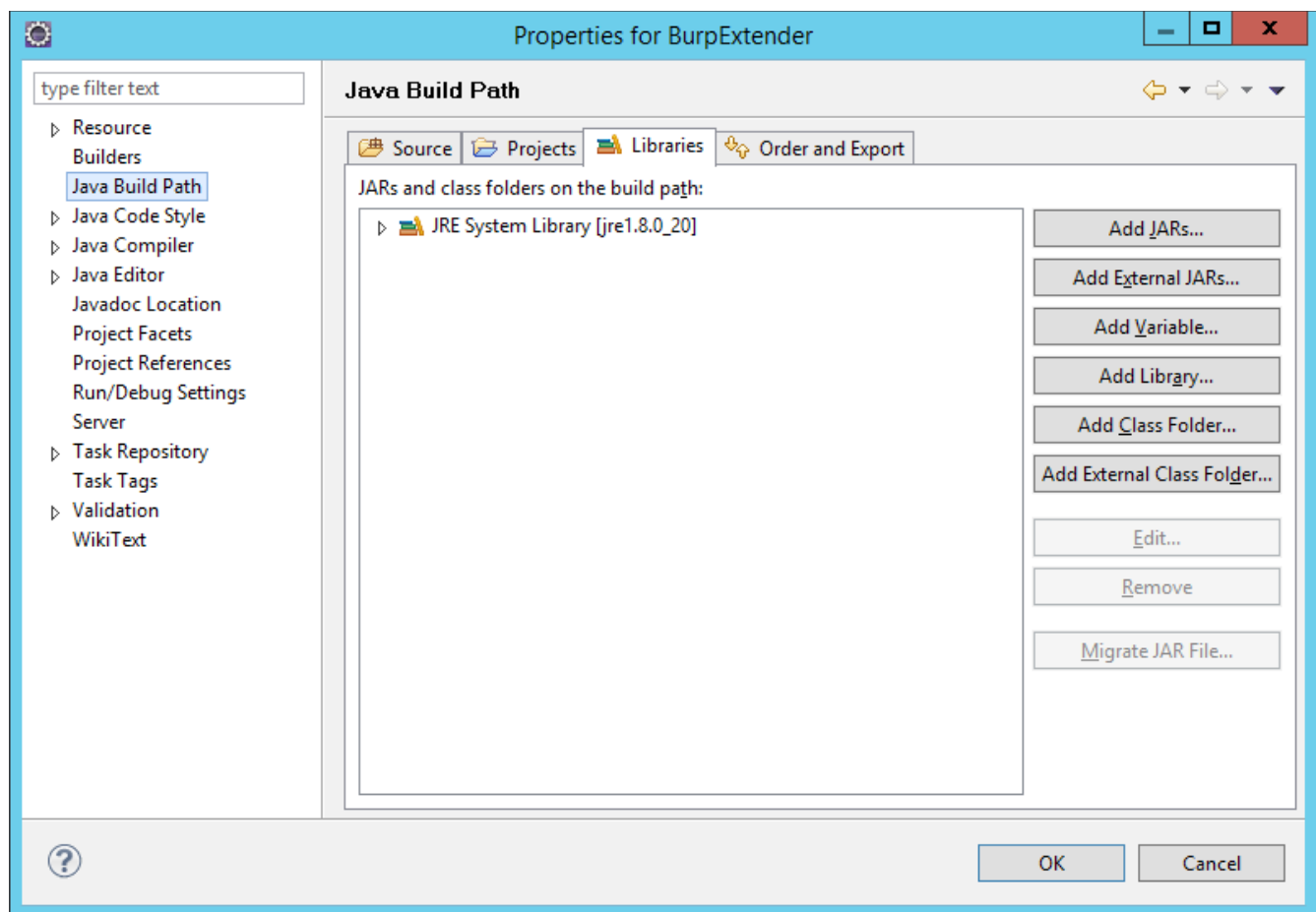
If you want to use the perspective click to the Yes button.



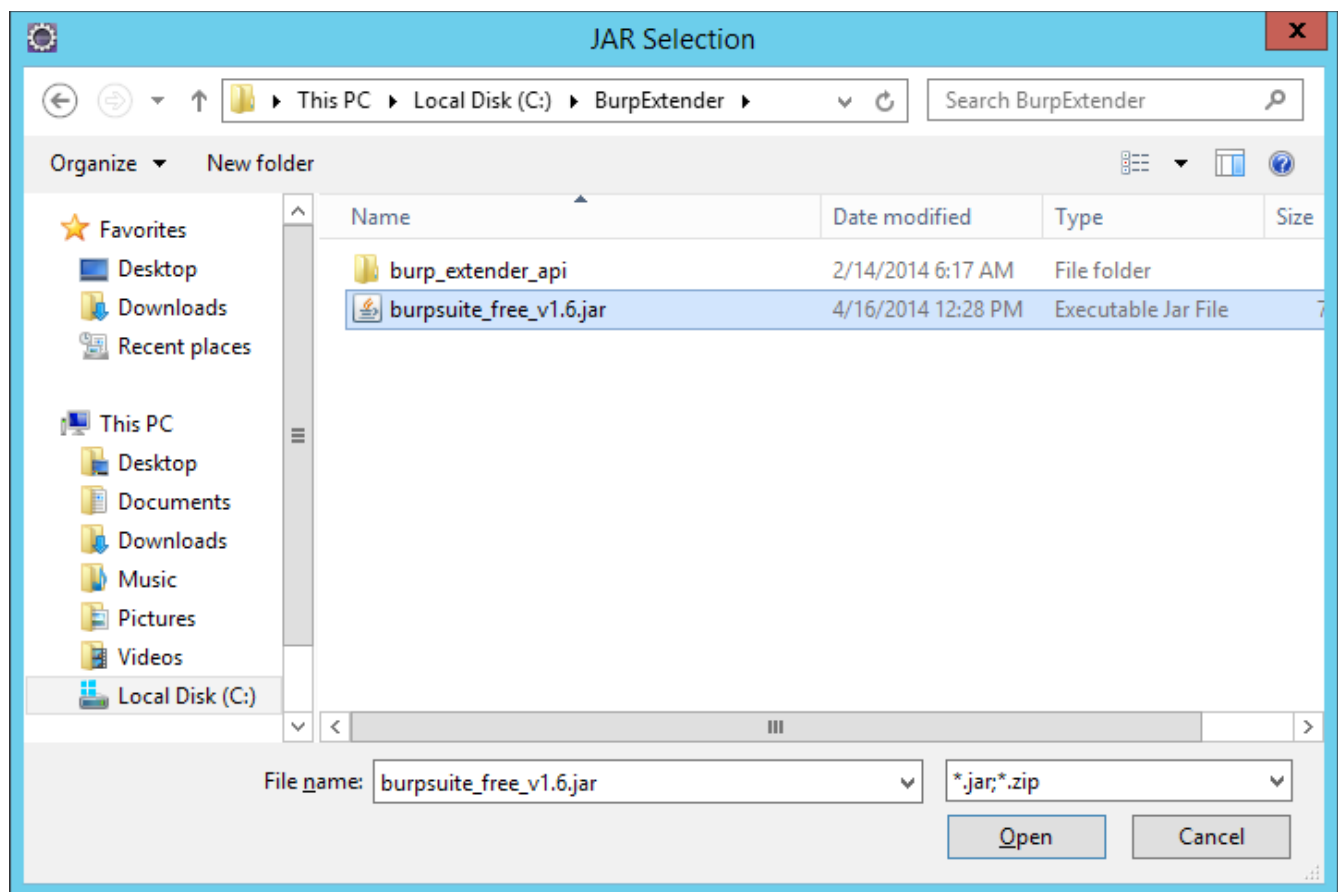
Select the Project / Properties menu:



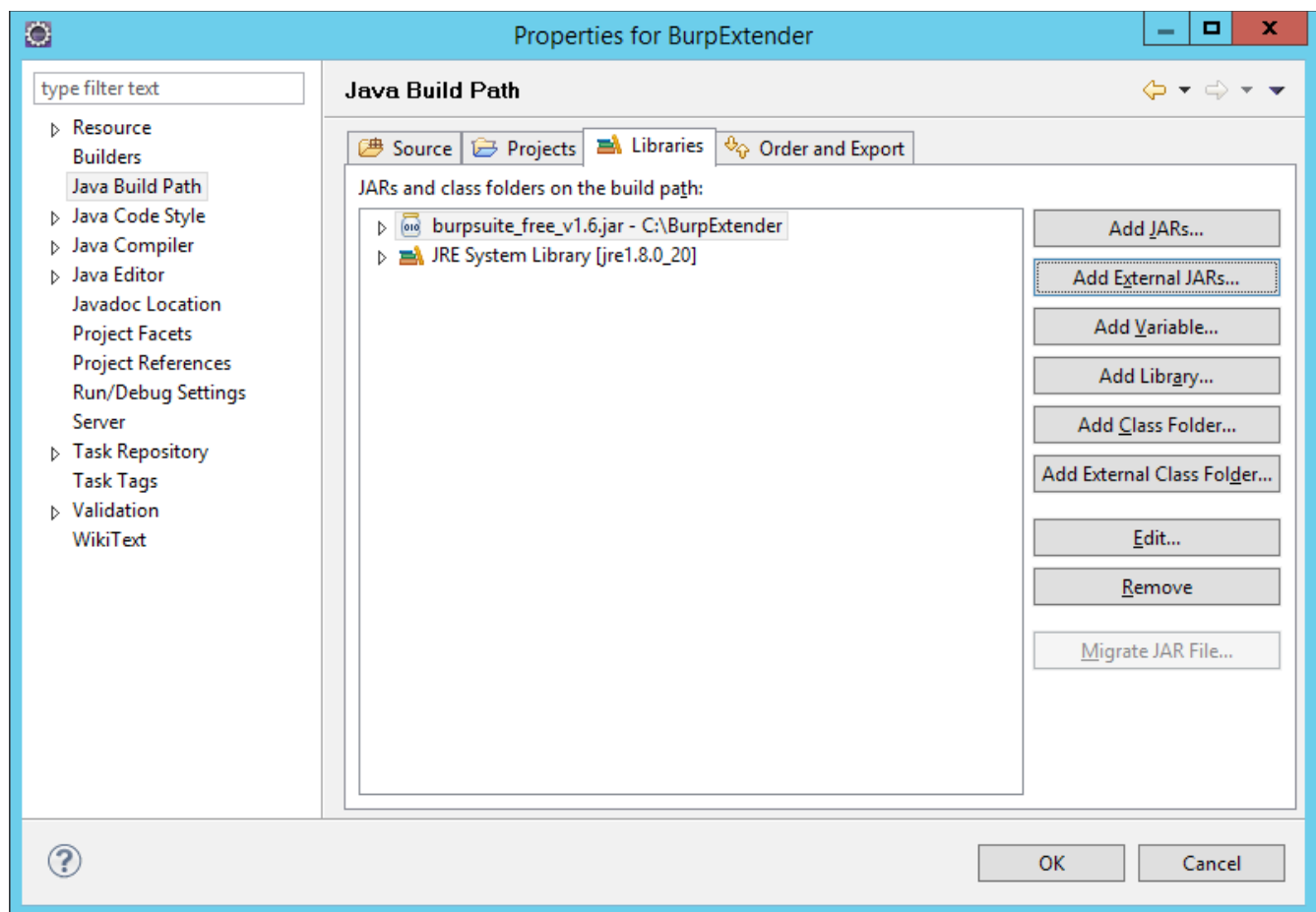
On the popup window go to the "Java Build Path", and select the Libraries tab, and click to the Add External JARs... button.



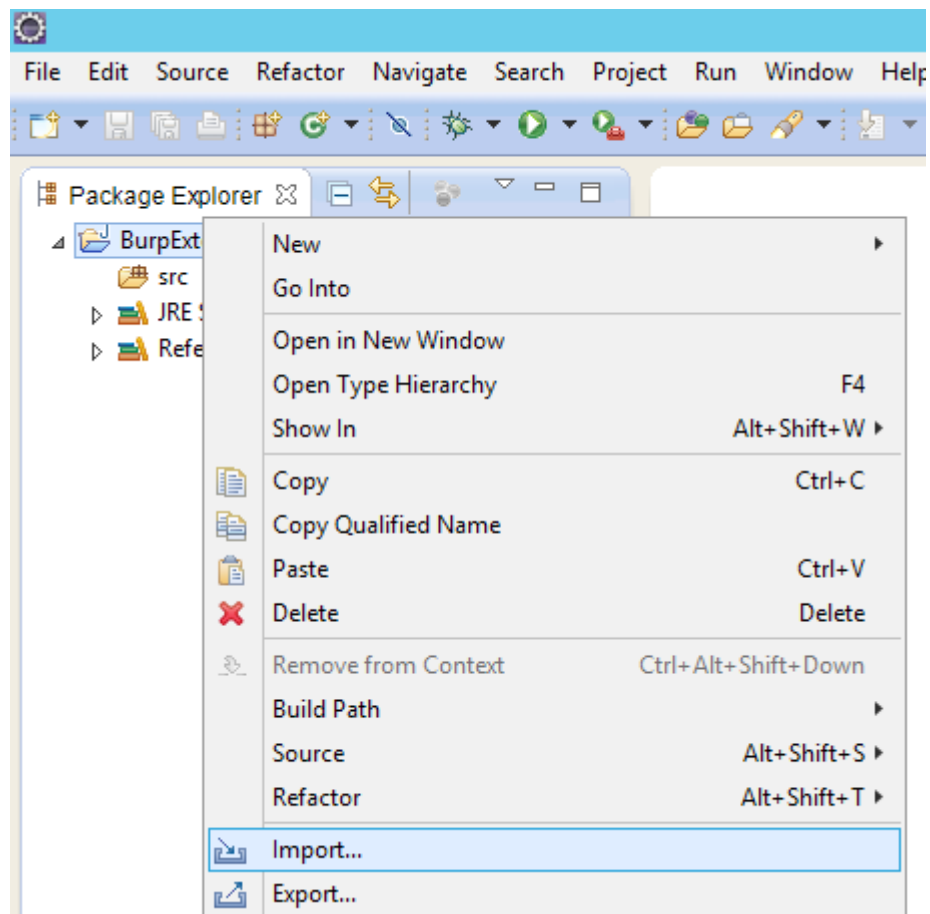
Select the burpsuite_free_v1.6.jar, then click to the open button



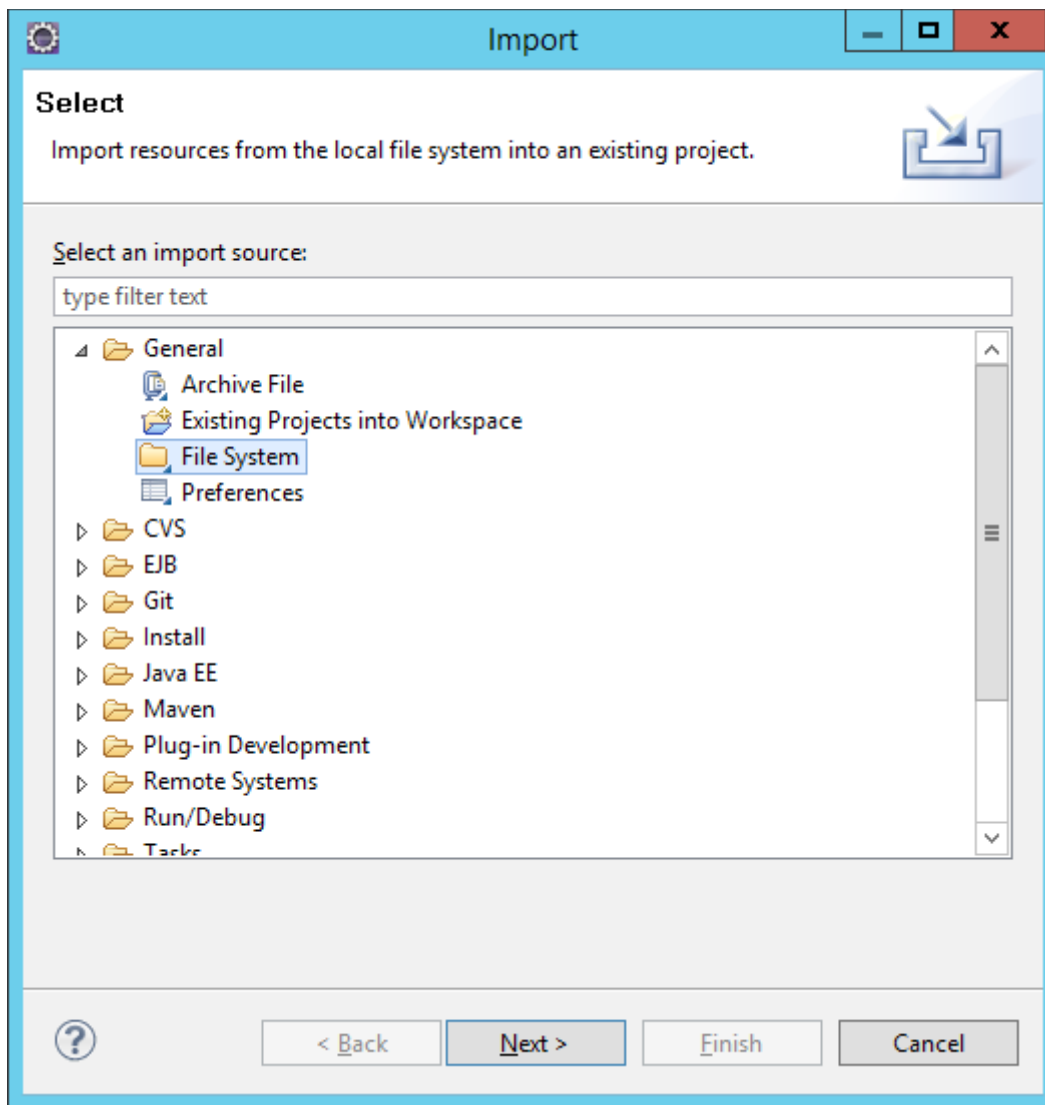
Then click to the OK button:



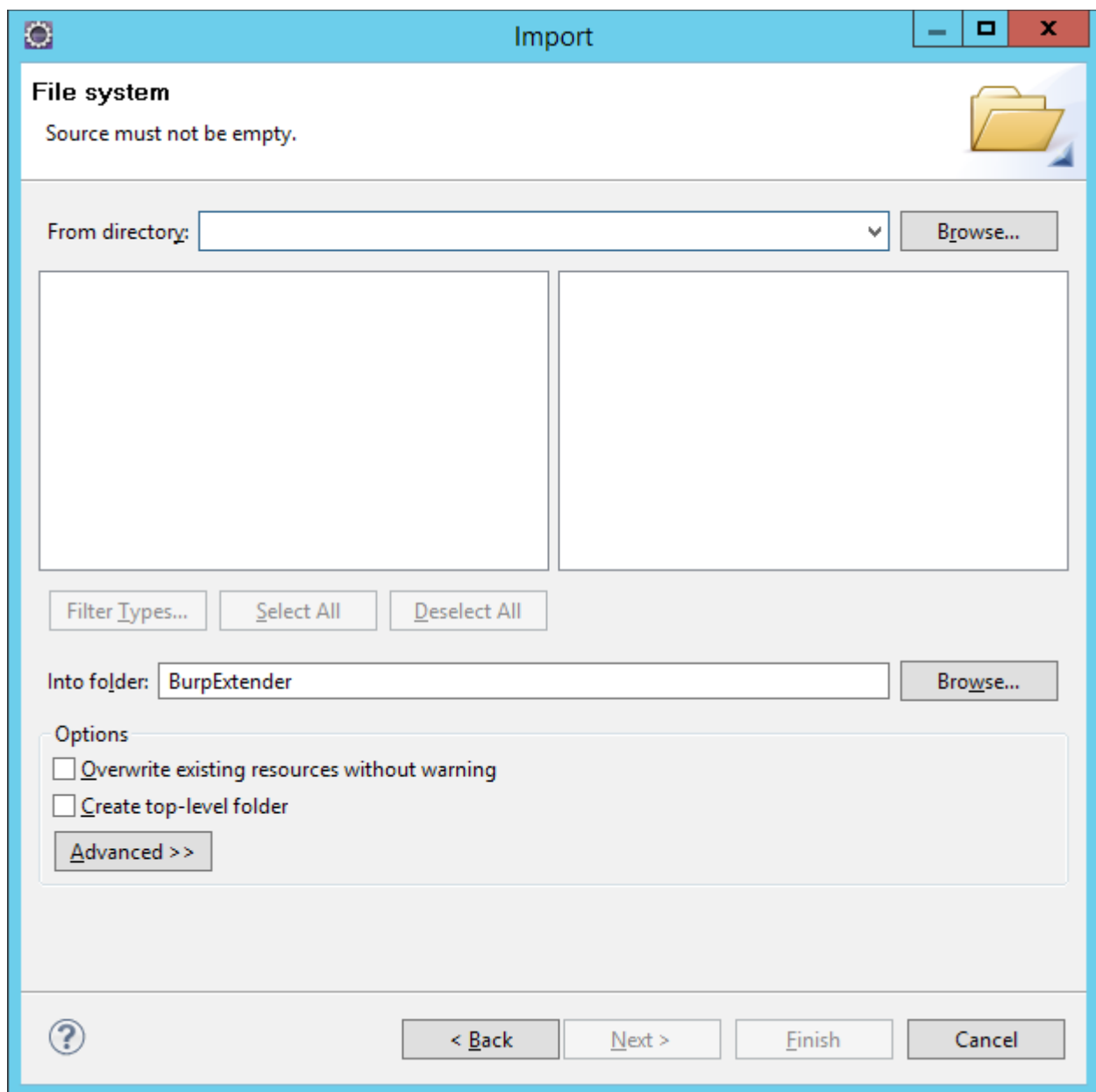
Right click to the Burp Extender package, and select import... from the popup menu:



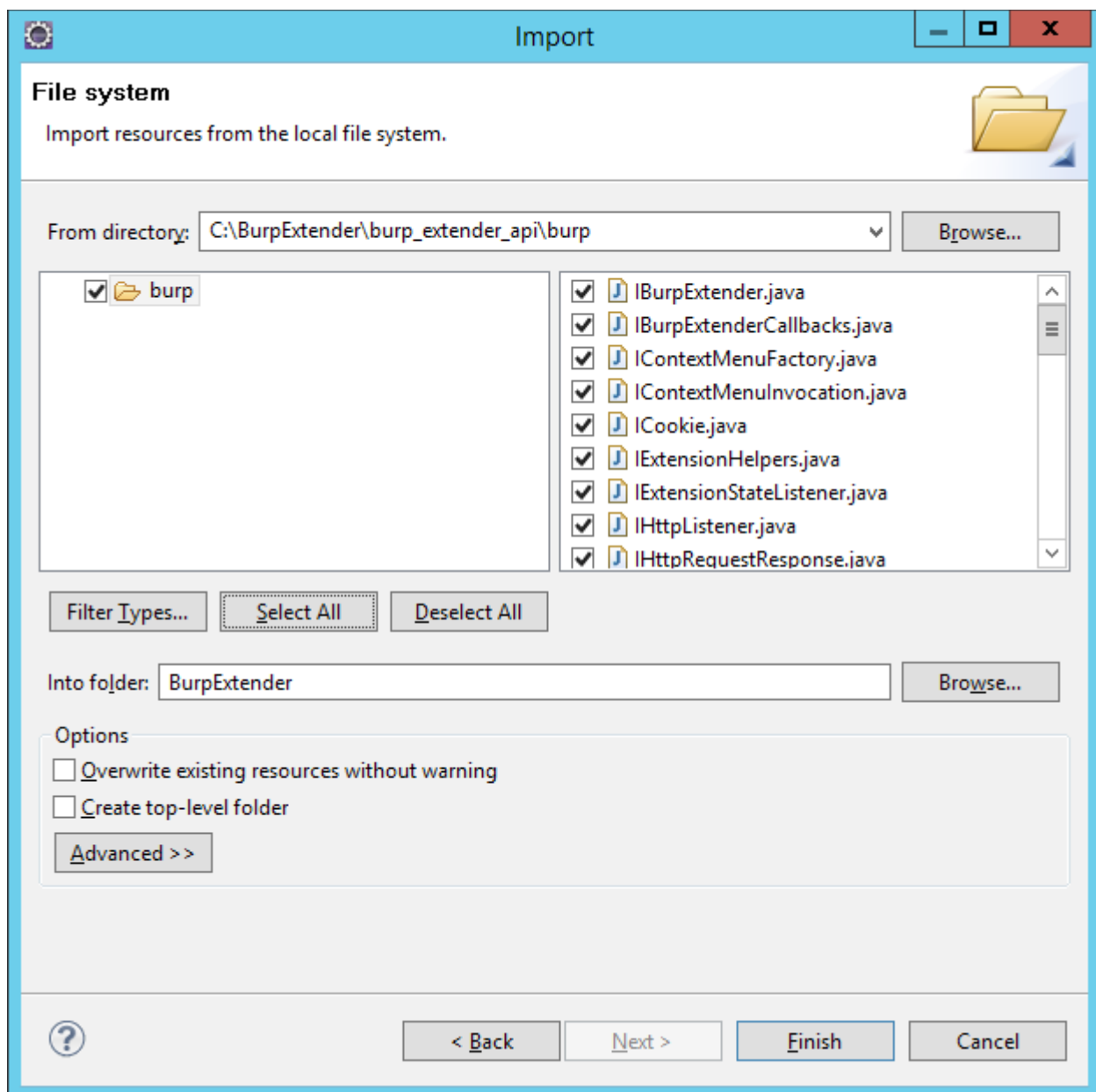
On the popup window select general / file system, then click to the next button



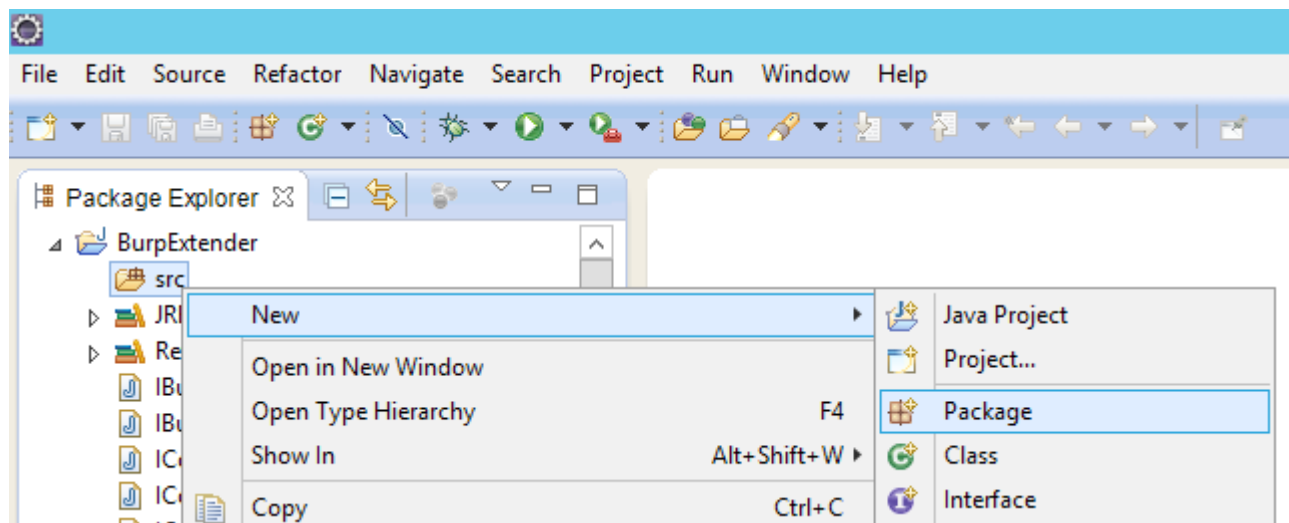
Find the burp extender api \ burp directory. The burp extender API can be download from the <http://portswigger.net/burp/extender/> webpage.



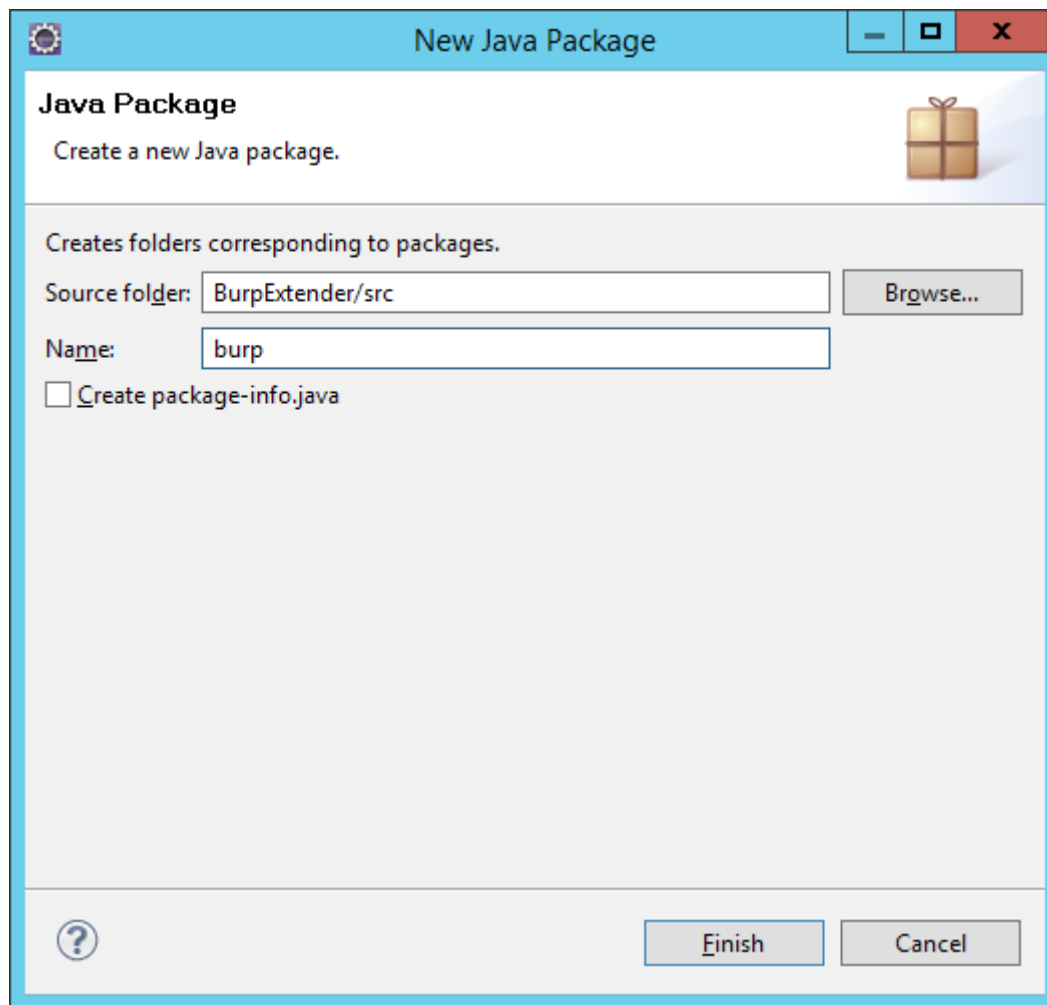
Select the required java files (I just add all the files), then click to the Finish button:



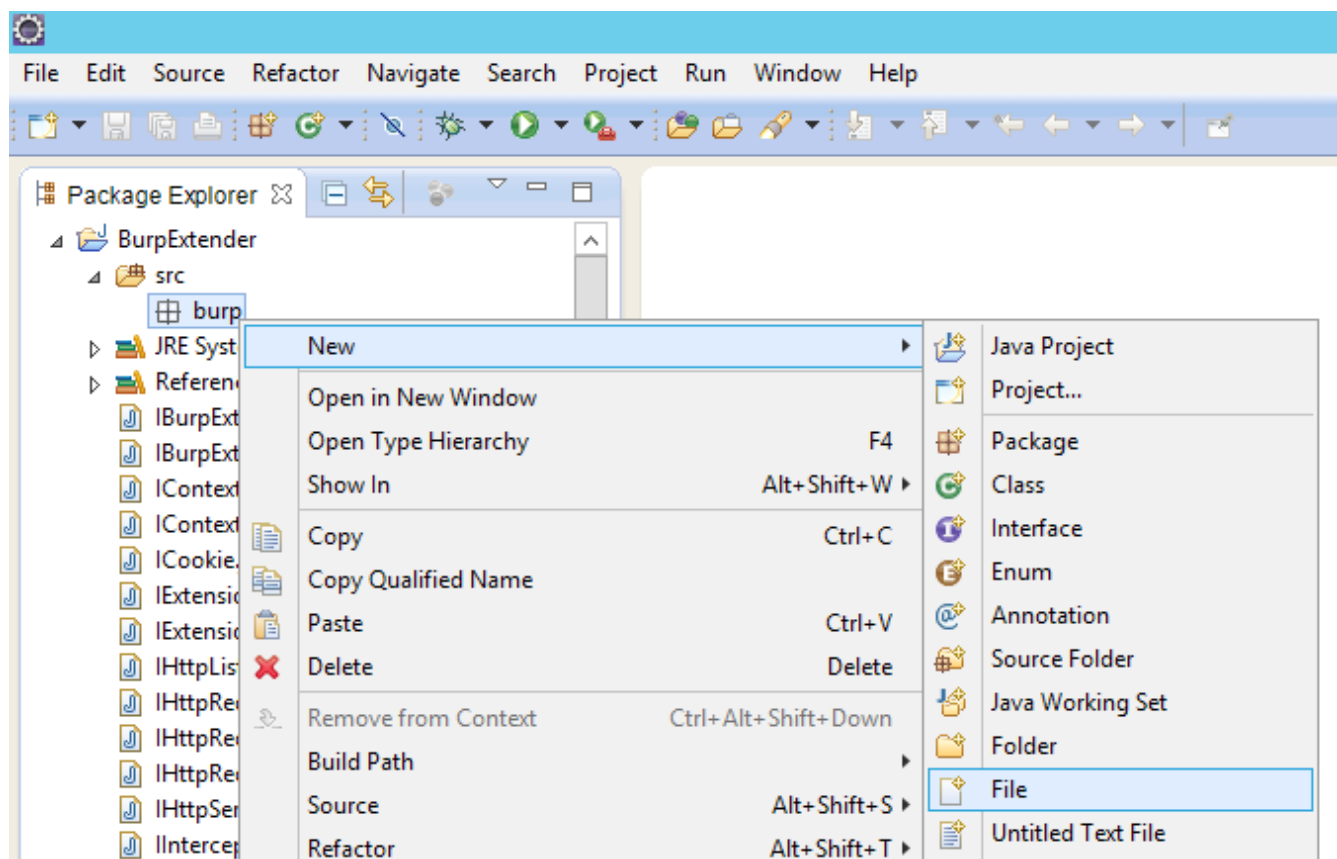
Then right click to the BurpExtender/src and from the popup menu select New / Package.



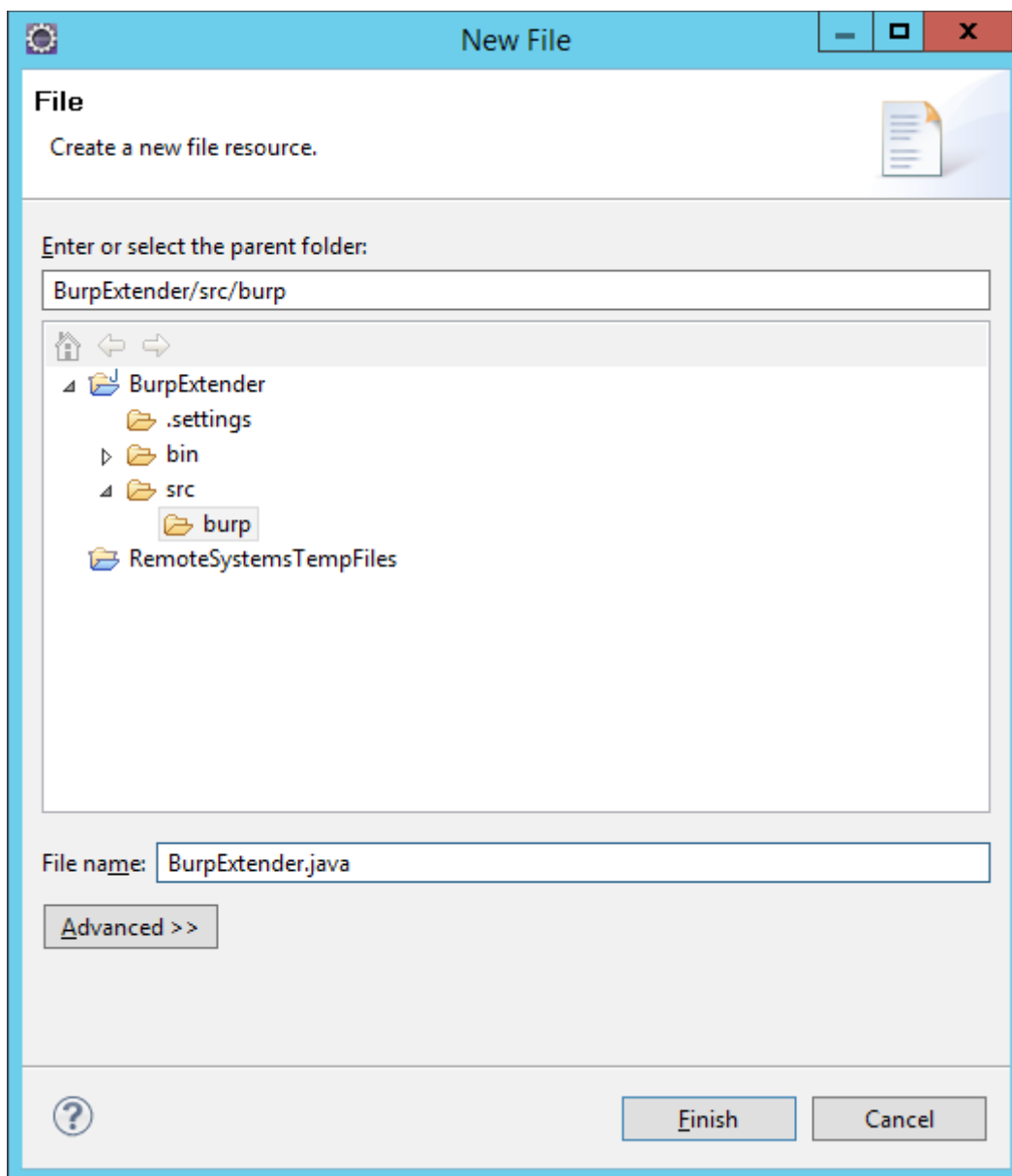
Give the package the burp name (all letter small), then click to the Finish button.



Then right click to the burp package and from the popup menu select New / File command



Call it as BurpExtender.java (take care the small, and capital letters) then click to the Finish button.

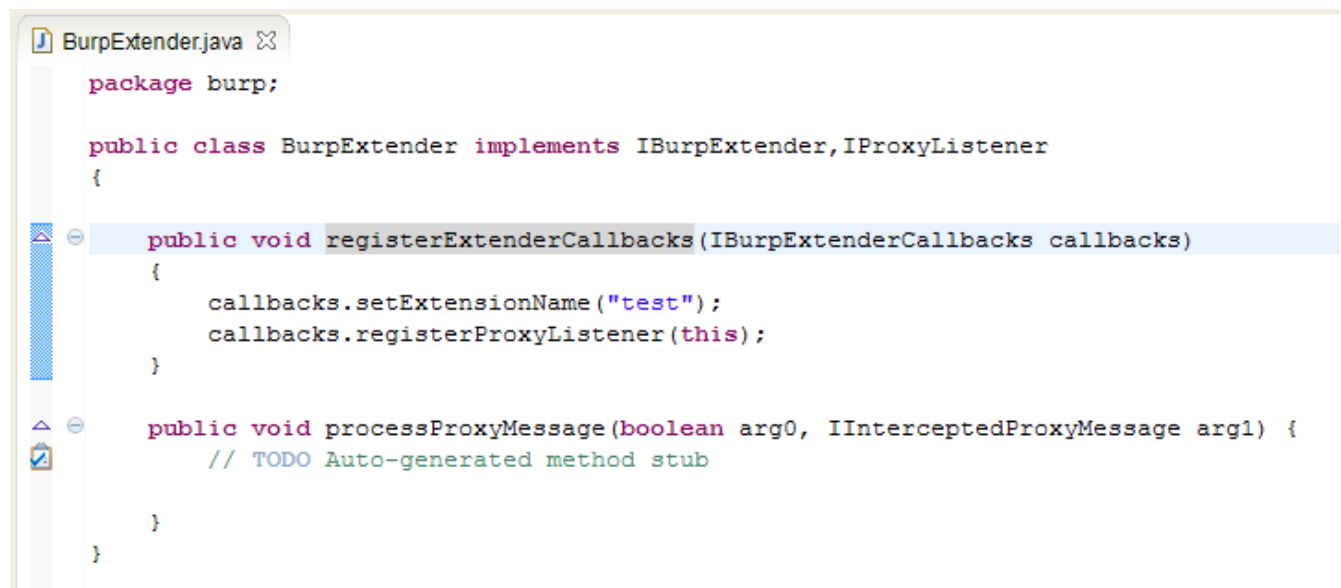


Write the extender

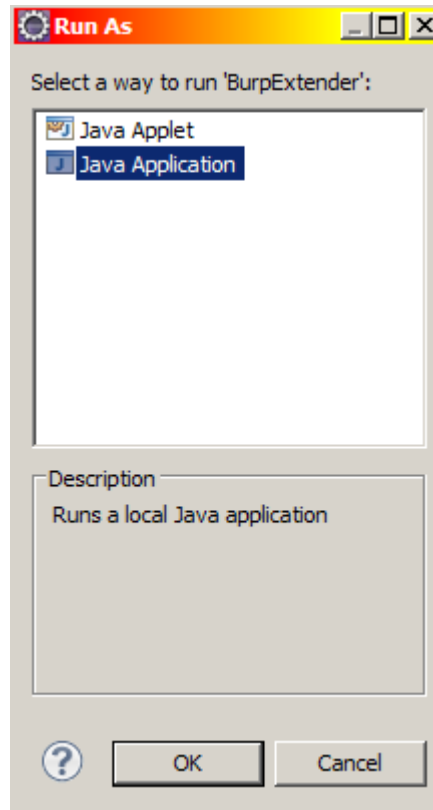
To write an extender we must implement an IBurpExtender and depends on that what we want to do another class, for my test the IProxyListener required.

To create an IProxyListener one must implement the processProxyMessage method, it will be called every time a message transferred by the burp proxy. The basic code, what is capable to run although it does not do anything useful yet looks like as:

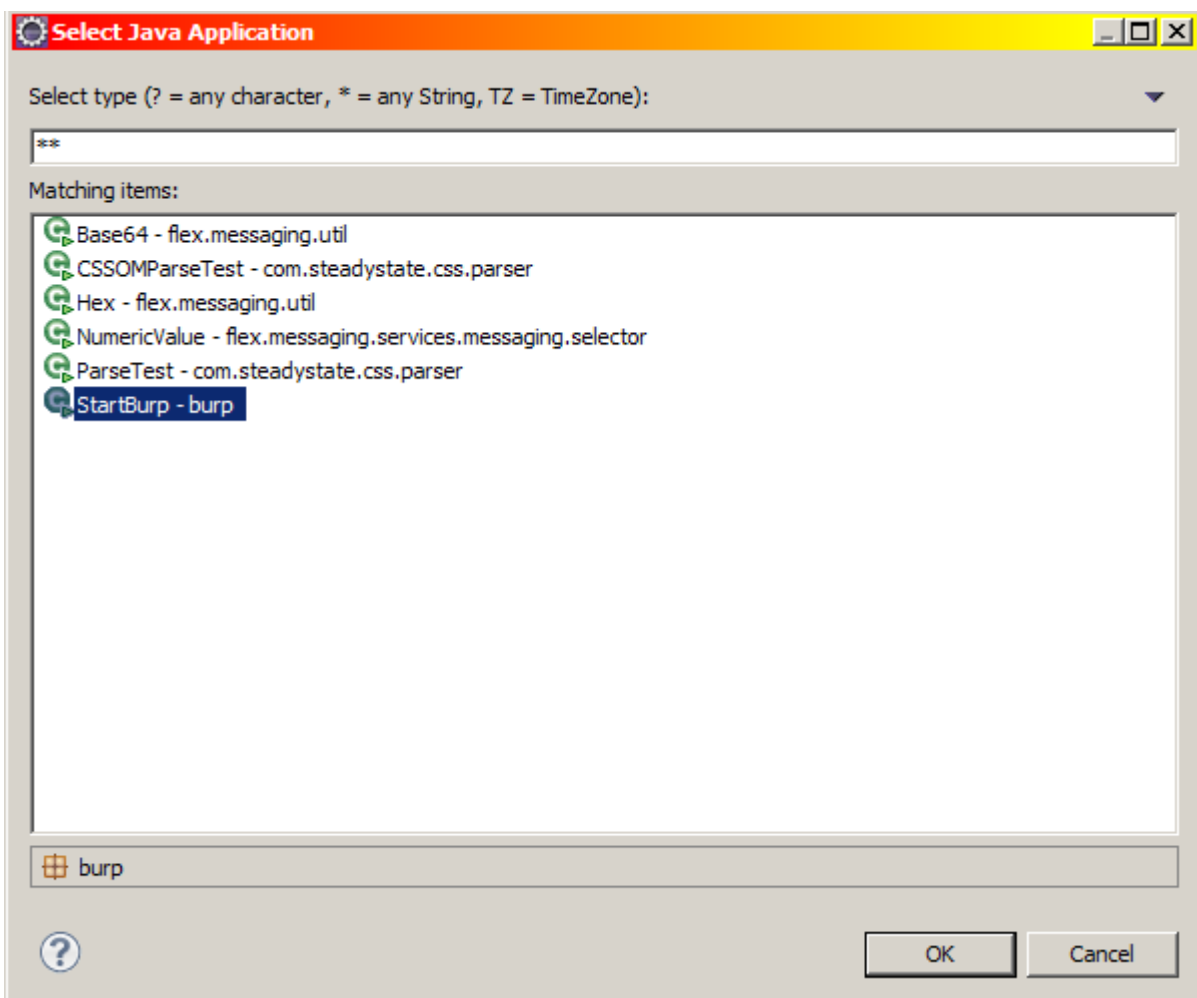
```
package burp;  
public class BurpExtender implements IBurpExtender, IProxyListener  
{  
    public void registerExtenderCallbacks (IBurpExtenderCallbacks  
callbacks)  
    {  
        callbacks.setExtensionName("test");  
        callbacks.registerProxyListener(this);  
    }  
    public void processProxyMessage (boolean arg0,  
IInterceptedProxyMessage arg1) {  
        // TODO Auto-generated method stub  
    }  
}
```



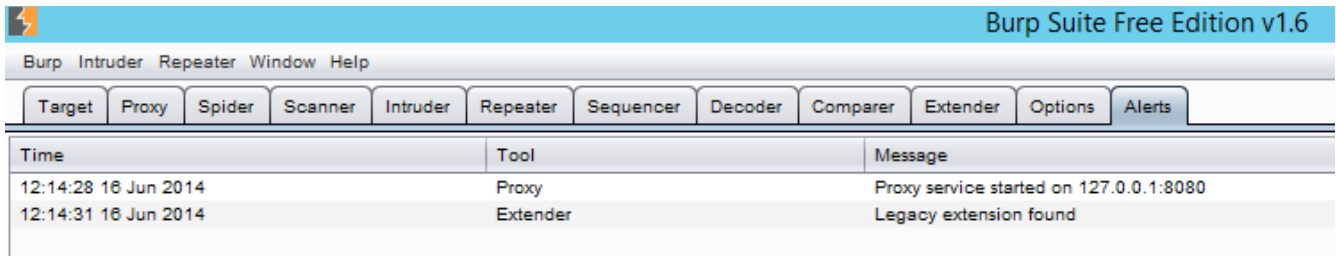
Save it, then run the package. When it asks you select to run as Java Application:



When it asks for which java application to start select the StartBurp – burp and click to the OK button.



When the burp proxy starts go to the alerts tab. You must see your Extender:



The screenshot shows the Burp Suite Free Edition v1.6 interface. The top menu bar includes 'Burp', 'Intruder', 'Repeater', 'Window', and 'Help'. Below the menu is a toolbar with buttons for 'Target', 'Proxy', 'Spider', 'Scanner', 'Intruder', 'Repeater', 'Sequencer', 'Decoder', 'Comparer', 'Extender', 'Options', and 'Alerts'. The 'Alerts' tab is selected, displaying a table of messages.

Time	Tool	Message
12:14:28 16 Jun 2014	Proxy	Proxy service started on 127.0.0.1:8080
12:14:31 16 Jun 2014	Extender	Legacy extension found

If it works close the burp proxy, and continue with the development of the extender

HelloWorld extender

Our purpose is to write an extender, which automatically corrects the checksum in every request so change the content of the processProxyMessage method, to print some message to the screen every time it see a request, to see if it works.

It can be done with the following code:

```
package burp;  
public class BurpExtender implements IBurpExtender, IProxyListener  
{  
    public void registerExtenderCallbacks (IBurpExtenderCallbacks  
callbacks)  
    {  
        callbacks.setExtensionName("test");  
        callbacks.registerProxyListener(this);  
    }  
    public void processProxyMessage (boolean messageIsRequest,  
                                     IInterceptedProxyMessage  
message) {  
        if (messageIsRequest) {  
            System.out.println("Request");  
        }  
    }  
}
```

```

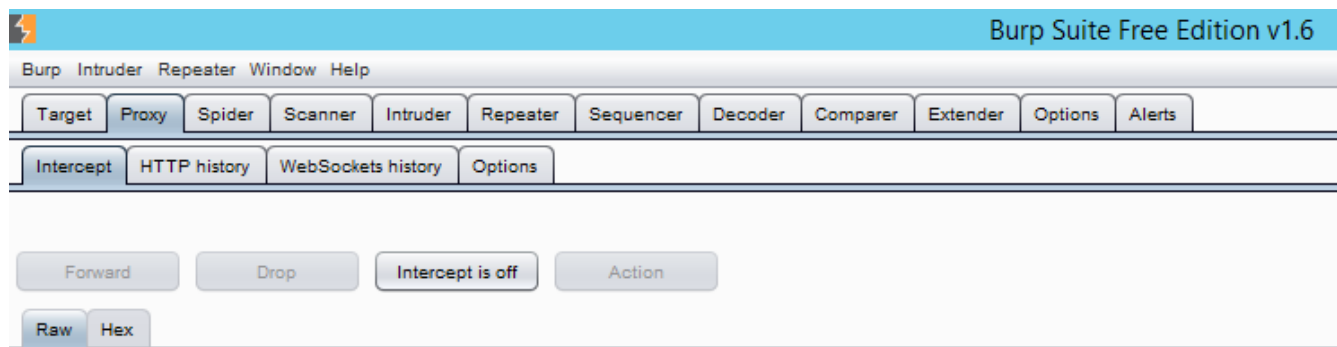
BurpExtender.java
package burp;

public class BurpExtender implements IBurpExtender, IProxyListener
{
    public void registerExtenderCallbacks(IBurpExtenderCallbacks callbacks)
    {
        callbacks.setExtensionName("test");
        callbacks.registerProxyListener(this);
    }

    public void processProxyMessage(boolean messageIsRequest,
                                    IInterceptedProxyMessage message) {
        if (messageIsRequest) {
            System.out.println("Request");
        }
    }
}

```

Then start again the Java Application. Go to the Proxy / Intercept tab and turn off the intercept. Then visit anything with your browser:



When you go back to the eclipse on the console window you must see some Request text appearing.

```

Problems  @ Javadoc  Declaration  Console
StartBurp [Java Application] C:\Program Files (x86)\Java\jre8\bin\javaw.exe (Jun 16, 2014, 12:21:01 PM)
Request
Request
|

```

Correct the checksum

Create a private helpers variable with IExtensionHelpers type. It will be used, to analyze the message.

To use the helper object in the registerExtenderCallbacks create give it a value with the callbacks.getHelpers() command.

```
package burp;  
  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.util.List;  
  
public class BurpExtender implements IBurpExtender, IProxyListener  
{  
    private IExtensionHelpers helpers;  
    public void registerExtenderCallbacks (IBurpExtenderCallbacks  
callbacks)  
    {  
        helpers = callbacks.getHelpers();  
        callbacks.setExtensionName("test");  
        callbacks.registerProxyListener(this);  
    }  
  
    public void processProxyMessage (boolean messageIsRequest,  
                                     IInterceptedProxyMessage  
message) {  
        if (messageIsRequest) {  
            IHttpRequestResponse msginfo =  
message.getMessageInfo();  
            byte[] msgarray = msginfo.getRequest();  
            IRequestInfo reqinfo =  
helpers.analyzeRequest(msginfo);  
            String req = new String(msgarray);  
            List<IParameter> params = reqinfo.getParameters();  
            String tohash = "";  
            String oldhash = "";  
            for (IParameter element : params)  
            {  
                if (element.getType() ==  
burp.IParameter.PARAM_BODY) {  
                    if (!element.getName().equals("checksum")) {  
                        tohash = tohash +  
element.getValue().toString();  
                    } // END IF (Name is NOT checksum)  
                } else{  
                    oldhash = element.getValue();  
                } // END ELSE (Name is checksum)  
            }  
        }  
    }  
}
```

```

        } // END IF parameter is PARAM_BODY
    } //END FOR
    String hash="";
    try {
        hash = MessageDigest.getInstance("SHA-
1").digest(tohash.getBytes()).toString();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    if (!hash.isEmpty() && !tohash.isEmpty())
    {
        req = req.replace(oldhash, hash);
        msginfo.setRequest(req.getBytes());
    } // END if hash and tohash not empty
    } // END if messageisrequest
} // END processProxyMessage
}

```

```

BurpExtender.java
package burp;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.List;

public class BurpExtender implements IBurpExtender, IProxyListener
{
    private IExtensionHelpers helpers;

    public void registerExtenderCallbacks (IBurpExtenderCallbacks callbacks)
    {
        helpers = callbacks.getHelpers();
        callbacks.setExtensionName ("test");
        callbacks.registerProxyListener (this);
    }

    public void processProxyMessage (boolean messageIsRequest,
                                     IInterceptedProxyMessage message) {
        if (messageIsRequest) {
            IHttpRequestResponse msginfo = message.getMessageInfo();
            byte[] msgarray = msginfo.getRequest();
            IRequestInfo reqinfo = helpers.analyzeRequest (msginfo);
            String req = new String (msgarray);
            List<IParameter> params = reqinfo.getParameters();
            String tohash = "";
            String oldhash = "";
            for (IParameter element : params)
            {
                if (element.getType() == burp.IParameter.PARAM_BODY) {
                    if (!element.getName().equals("checksum")) {
                        tohash = tohash + element.getValue().toString();
                    } // END IF (Name is NOT checksum)
                    else {
                        oldhash = element.getValue();
                    } // END ELSE (Name is checksum)
                } // END IF parameter is PARAM_BODY
            } //END FOR
            String hash="";
            try {
                hash = MessageDigest.getInstance("SHA-1").digest(tohash.getBytes()).toString();
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
            if (!hash.isEmpty() && !tohash.isEmpty())
            {
                req = req.replace(oldhash, hash);
                msginfo.setRequest(req.getBytes());
            } // END if hash and tohash not empty
        }
    }
}

```

Now try the automated tools. And you will see they are still not able to find the SQL injection

vulnerability.

Convert the hash to hex array

If you watch the requests in burp proxy you will see the checksum values are not good, because it sends a string, while the original application sends hex encoded strings. So we must add a string to hex encoder to our code.

```
package burp;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.List;

public class BurpExtender implements IBurpExtender, IProxyListener
{
    private IExtensionHelpers helpers;
    public void registerExtenderCallbacks (IBurpExtenderCallbacks
callbacks)
    {
        helpers = callbacks.getHelpers();
        callbacks.setExtensionName("test");
        callbacks.registerProxyListener(this);
    }

    final protected static char[] hexArray =
"0123456789ABCDEF".toCharArray();
    public static String bytesToHex(byte[] bytes)
    {
        char[] hexChars = new char[bytes.length * 2];
        for ( int j = 0; j < bytes.length; j++ )
        {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = hexArray[v >>> 4];
            hexChars[j * 2 + 1] = hexArray[v & 0x0F];
        }
        return new String(hexChars);
    }
    public void processProxyMessage (boolean messageIsRequest,
                                   IInterceptedProxyMessage
message) {
        if (messageIsRequest) {
            IHttpRequestResponse msginfo =
message.getMessageInfo();
            byte[] msgarray = msginfo.getRequest();
            IRequestInfo reqinfo =
helpers.analyzeRequest(msginfo);
            String req = new String(msgarray);
            List<IParameter> params = reqinfo.getParameters();
            String tohash = "";
        }
    }
}
```

```

String oldhash = "";
for(IParameter element : params)
{
    if (element.getType() ==
burp.IParameter.PARAM_BODY) {
        if (!element.getName().equals("checksum")) {
            tohash = tohash +
element.getValue().toString();
        } // END IF (Name is NOT checksum)
        else{
            oldhash = element.getValue();
        } // END ELSE (Name is checksum)
    } // END IF parameter is PARAM_BODY
} //END FOR
String hash="";
try {
    hash = bytesToHex(MessageDigest.getInstance("SHA-1").digest(tohash.getBytes()));
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
if (!hash.isEmpty() && !tohash.isEmpty())
{
    req = req.replace(oldhash, hash);
    msginfo.setRequest(req.getBytes());
} // END if hash and tohash not empty
} // END if messageisrequest
} // END processProxyMessage
}

```

```

        if (element.getType() == burp.IParameter.PARAM_BODY) {
            if (!element.getName().equals("checksum")) {
                tohash = tohash + helpers.urlDecode(element.getValue()).toString();
            } // END IF (Name is NOT checksum)
            else{
                oldhash = element.getValue();
            } // END ELSE (Name is checksum)
        } // END IF parameter is PARAM_BODY
    } //END FOR
    String hash="";
    try {
        hash = MessageDigest.getInstance("SHA-1").digest(tohash.getBytes()).toString();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    if (!hash.isEmpty() && !tohash.isEmpty())
    {
        req = req.replace(oldhash, hash);
        msginfo.setRequest(req.getBytes());
    } // END if hash and tohash not empty
}
}
}

```

Try to run this code, and use the automated tools, to find the SQL injection vulnerability. But it does not work. If you check the requests and responses you will find that, the checksum sometimes not correct.

URL decode the values

The problem why the code not works is that, the burp proxy will see the values in the requests as URL encoded (like %20 instead of space and so on) and because of it it will calculate the checksum of that string. While the PHP will calculate the checksum on the URL decoded string. The two values of course will be different sometimes. The solution is easy, it must URL decode the values before we add it to the hashable string.

```
package burp;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.List;

public class BurpExtender implements IBurpExtender, IProxyListener
{
    private IExtensionHelpers helpers;
    public void registerExtenderCallbacks (IBurpExtenderCallbacks
callbacks)
    {
        helpers = callbacks.getHelpers();
        callbacks.setExtensionName("test");
        callbacks.registerProxyListener(this);
    }

    final protected static char[] hexArray =
"0123456789ABCDEF".toCharArray();
    public static String bytesToHex(byte[] bytes)
    {
        char[] hexChars = new char[bytes.length * 2];
        for ( int j = 0; j < bytes.length; j++ )
        {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = hexArray[v >>> 4];
            hexChars[j * 2 + 1] = hexArray[v & 0x0F];
        }
        return new String(hexChars);
    }
    public void processProxyMessage(boolean messageIsRequest,
IInterceptedProxyMessage
message) {
        if (messageIsRequest) {
            IHttpRequestResponse msginfo =
message.getMessageInfo();
            byte[] msgarray = msginfo.getRequest();
            IRequestInfo reqinfo =
helpers.analyzeRequest(msginfo);
            String req = new String(msgarray);
```

```

List<IParameter> params = reqinfo.getParameters();
String tohash = "";
String oldhash = "";
for(IParameter element : params)
{
    if (element.getType() ==
burp.IParameter.PARAM_BODY) {
        if (!element.getName().equals("checksum")) {
            tohash = tohash +
helpers.urlDecode(element.getValue()).toString();
        } // END IF (Name is NOT checksum)
        else{
            oldhash = element.getValue();
        } // END ELSE (Name is checksum)
    } // END IF parameter is PARAM_BODY
} //END FOR
String hash="";
try {
    hash = bytesToHex(MessageDigest.getInstance("SHA-
1").digest(tohash.getBytes()));
} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
if (!hash.isEmpty() && !tohash.isEmpty())
{
    req = req.replace(oldhash, hash);
    msginfo.setRequest(req.getBytes());
} // END if hash and tohash not empty
} // END if messageisrequest
} // END processProxyMessage
}

```

```

BurpExtender.java

final protected static char[] hexArray = "0123456789ABCDEF".toCharArray();
public static String bytesToHex(byte[] bytes)
{
    char[] hexChars = new char[bytes.length * 2];
    for ( int j = 0; j < bytes.length; j++ )
    {
        int v = bytes[j] & 0xFF;
        hexChars[j * 2] = hexArray[v >>> 4];
        hexChars[j * 2 + 1] = hexArray[v & 0x0F];
    }
    return new String(hexChars);
}

public void processProxyMessage(boolean messageIsRequest,
                                IInterceptedProxyMessage message) {
    if (messageIsRequest) {
        IHttpRequestResponse msginfo = message.getMessageInfo();
        byte[] msgarray = msginfo.getRequest();
        IRequestInfo reqinfo = helpers.analyzeRequest(msginfo);
        String req = new String(msgarray);
        List<IParameter> params = reqinfo.getParameters();
        String tohash = "";
        String oldhash = "";
        for(IParameter element : params)
        {
            if (element.getType() == burp.IParameter.PARAM_BODY) {
                if (!element.getName().equals("checksum")) {
                    tohash = tohash + helpers.urlDecode(element.getValue()).toString();
                } // END IF (Name is NOT checksum)
            } else {
                oldhash = element.getValue();
            } // END ELSE (Name is checksum)
        } // END IF parameter is PARAM_BODY
    } //END FOR
    String hash="";
    try {
        hash = bytesToHex(MessageDigest.getInstance("SHA-1").digest(tohash.getBytes()));
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    if (!hash.isEmpty() && !tohash.isEmpty())
    {
        req = req.replace(oldhash, hash);
        msginfo.setRequest(req.getBytes());
    } // END if hash and tohash not empty
    } // END if messageisrequest
} // END processProxyMessage
}

```

Now if you run again the automated tool it should find the SQL injection vulnerability.