



1 Hack your own Web-App


Is your web application secure? How can you tell? This workshop gives you an opportunity to experience your web application from the perspective of a hacker. It starts out with importing and configuring the supplied virtual machines and some quick connectivity tests. Then it guides you across the most common vulnerabilities using manual and automated testing. After which you can apply the things you've learned on your own web application (or the supplied one with lots of holes!).

Duration	2 Hours
Audience	Web Application Developers of all levels.
Prerequisites	Laptop capable of running VirtualBox, Hacker.ova and HackMe.ova virtual machines.

- [Hack your own Web-App](#)
- [Hack Lab Preparation](#)
 - [Configuring Virtual Box](#)
 - [Instructions for Virtual Box 4.x](#)
 - [Instructions for Virtual Box 5.x](#)
 - [Importing the VirtualMachine](#)
 - [Testing Network Connectivity](#)
 - [Installing the HackMe Virtual Machine](#)
 - [Lab 1 : Finding the HackMe VM's IP Address.](#)
- [Hacking your Web Application](#)
 - [Owasp Top 10 - 2013](#)
 - [Lab 2 : Exploring the vulnerabilities in the Bodgeit store.](#)
 - [Missing Function Level Access Control \(A7\)](#)
 - [SQL Injection \(A1\)](#)
 - [Cross Site Scripting \(A3\)](#)
 - [Missing Server Side Validation](#)
 - [Insecure Direct Object References \(A4\)](#)
 - [Broken Authentication and Session Management \(A2\)](#)
 - [Cross-Site Request Forgery \(A8\)](#)
 - [Sensitive Data Exposure \(A6\)](#)
 - [Tools on the Hacker VM](#)
 - [Lab 3 : The Owasp Zed Attack Proxy](#)
 - [The Once Click Attack](#)
 - [Passive Scanning and Request manipulation](#)
 - [Force Browsing Directories](#)
 - [Lab 4 : Brute forcing Login Forms with THC Hydra](#)
 - [Lab 5 : Web server scanning with Nikto](#)
 - [Lab 6 : Web server fuzzing with Wfuzz](#)
 - [Lab 7 : Vulnerability Scanning with W3af](#)
 - [Lab 8 : Brute Forcing Hashes with John the Ripper](#)
- [The End](#)

2 Hack Lab Preparation

This workshop comes with two virtual machines. 'Hacker.ova' is an Ubuntu Desktop VM loaded with all the tools for this workshop. 'HackMe.ova' is an Ubuntu Server VM running two very vulnerable web applications. We will be using the first web application (the [bodgeit](#) store, by Psiion) to practice on before attacking the web application you brought with you. For those that didn't bring their own web-application, the second one called 'hackme' is yours and full of features to be exploited :-).

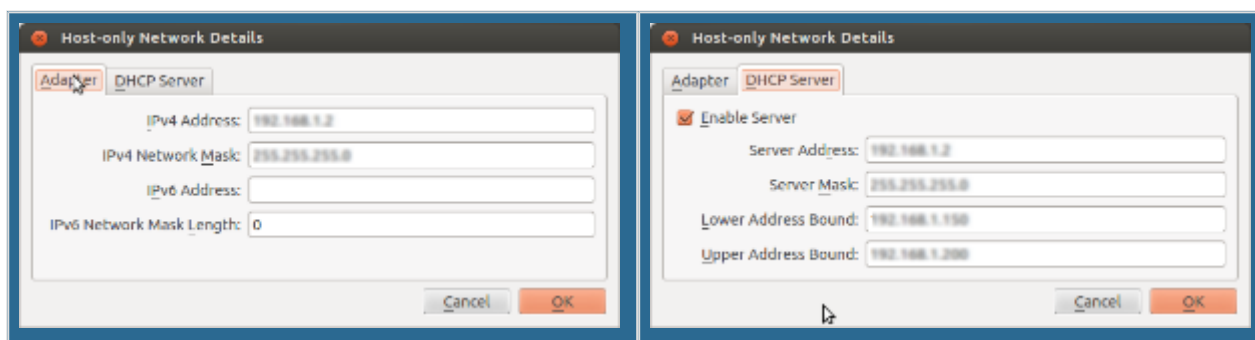
 Hacking stuff that you do not own is pretty much illegal. So we must make sure we cannot accidentally hack someone else. That is why we will be using VirtualBox 'Host Only Networking'. This creates a virtual network that consists only of the Host (your laptop) and the virtual machines that are running on it.

2.1 Configuring Virtual Box

First we'll need to configure the Host Only Networking settings:

2.1.1 Instructions for Virtual Box 4.x

- Start Virtual Box
- Goto File->Preferences and select the Network Tab
- Select the 'Host-only networks' tab if on Windows.
- Select the default entry 'VirtualBox Host Only Adapter' (windows) or 'vboxnet0' (Linux) and press the edit button on the right.

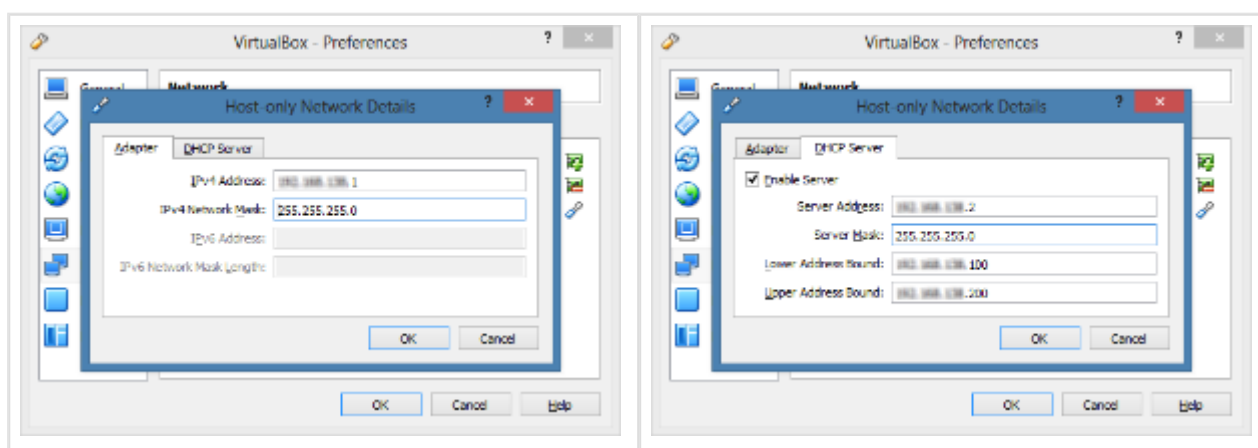


- Make a note of the default settings (IPv4 and DHCP lower and upper bound) but don't change them.
- Check that the DHCP server is enabled. If not enable it and press Ok.

2.1.2 Instructions for Virtual Box 5.x

- Start Virtual Box
- Goto File->Preferences and select the Network Tab.
- Select the 'Host-only networks' tab if on Windows.
- In Virtual Box 5.x the Host Only adapter is not always present by default, you may need to add it manually. If so do the following:
 - On the Host-only networks tab press the 'Add' button.
 - After a short while a Host only adapter will appear.

- Press the 'Edit' button.
 - The Adapter tab may come populated (192.168.x.y). In that case make a note of those settings but don't change them.
 - //the IPv4 address is four zero's you need to configure them yourself. Use 192.168.138.1 for IPv4 and 255.255.255.0 for the Mask.
- Change to the DHCP Server Tab.
- In Virtual Box 5.0 the DHCP server is not always configured by default. In that case you may need to configure it yourself:
 - The server Ip address must be identical of the IP address of the adapter except for the last digit (choose 2 for the last digit).
 - The network mask must be identical to the adapters network mask.
 - The lower address bound must be identical of the IP address of the adapter except for the last digit (choose 100 for the last digit)
 - The higher address bound must be identical of the IP address of the adapter except for the last digit (choose 200 for the last digit)
- Make a note of the settings (IPv4 and DHCP lower and upperbound).
- Check that the DHCP server is enabled. If not, enable it. Press Ok.



⚠ Depending on your local network configuration and Virtual Box version the actual DHCP ranges may differ.

⚠ If on Linux the vboxnet0 network is not present its possible a kernel update messed up the VirtualBox kernel modules. Read [this](#). And then execute `sudo /etc/init.d/vboxdrv setup`

2.2 Importing the VirtualMachine

The virtual machine for this workshop comes packaged as an appliance, which is basically the hard disk image of the VM and its configuration conveniently packaged into one file. It needs to be imported which goes like this:

- Select File->Import Appliance in the menu.
- In the dialog that opens select the 'Hacker.ova' file.
- Press Next and verify the settings.
- Press Import
- Be patient, this may take a while..

- After the boot sequence you should be greeted by a Login screen.
- The password is **I33t**

2.3 Testing Network Connectivity

Now we must make sure that the virtual machine can talk to your laptop and vice versa.

- Login as hacker into the Hacker VM and open a command prompt (`ctrl-alt-T`).
- Enter the command `ifconfig` and check the `eth0` inet addr.
- It should be somewhere in the configured DHCP range (check your notes!)
- Your laptop should be IP `192.168.x.y` (check your notes again!) so see if you can ping it:
- `ping 192.168.x.y`
- press `ctrl-C` to abort
- Other networks should be unreachable so, check using `42.nl`
- `ping 178.18.86.219`

```
hacker@hacker-VirtualBox -
hacker@hacker-VirtualBox:~$ ifconfig
eth0    Link encap:Ethernet  HWaddr 08:00:27:38:11:86
        inet addr:192.168.1.150  Bcast:192.168.1.255  Mask:255.255.255.0
        inet6 addr: fe80::a86:27ff:fe38:1186/64  Scope:link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:14  errors:0  dropped:0  overruns:0  frame:0
        TX packets:73  errors:0  dropped:0  overruns:0  carrier:0
        collisions:0  txqueuelen:1000
        RX bytes:2166 (2.1 KB)  TX bytes:11790 (11.7 KB)

lo      Link encap:local loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128  Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:172  errors:0  dropped:0  overruns:0  frame:0
        TX packets:172  errors:0  dropped:0  overruns:0  carrier:0
        collisions:0  txqueuelen:0
        RX bytes:11140 (11.1 KB)  TX bytes:11140 (11.1 KB)

hacker@hacker-VirtualBox:~$ ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data:
64 bytes from 192.168.1.2: icmp_req=1 ttl=64 time=1.18 ms
64 bytes from 192.168.1.2: icmp_req=2 ttl=64 time=0.291 ms
64 bytes from 192.168.1.2: icmp_req=3 ttl=64 time=0.278 ms
^C
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2802ms
rtt min/avg/max/mdev = 0.278/0.583/1.182/0.424 ms
hacker@hacker-VirtualBox:~$ ping 178.18.86.219
connect: Network is unreachable
hacker@hacker-VirtualBox:~$
```

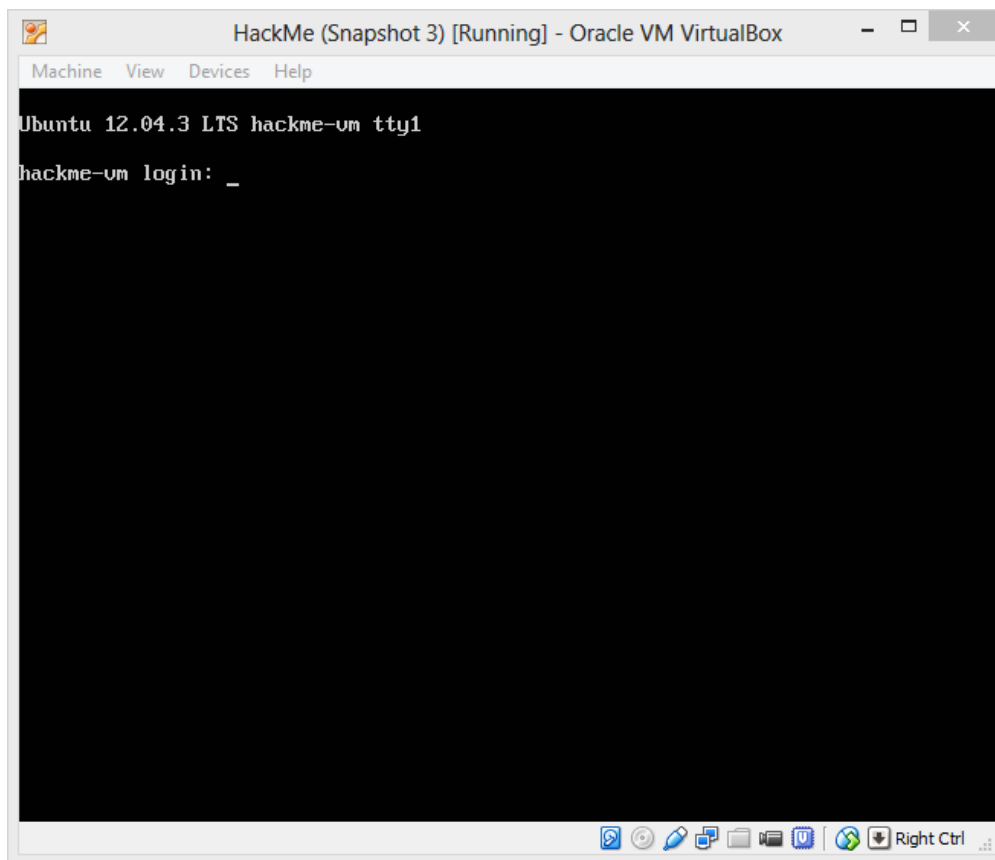
- Good. Now lets check if your laptop can see the virtual machine.
- Open a command prompt on your laptop and enter
- `ping 192.168.1.150` (or actually the IP you found when doing `ifconfig`).



For more convenience (screen resize, copy pasting) you can also install the VirtualBox Guest Addons in your VM. It is an ISO file that comes with VirtualBox.

2.4 Installing the HackMe Virtual Machine

The installation follows the same steps as the Hacker virtual machine. After importing check the network settings (should be Host only Adapter) and start the virtual machine.



After the boot sequence there will be a log-in prompt begging, but leave that for now. Instead we will try and reach the VM via the network:

2.5 Lab 1 : Finding the HackMe VM's IP Address.

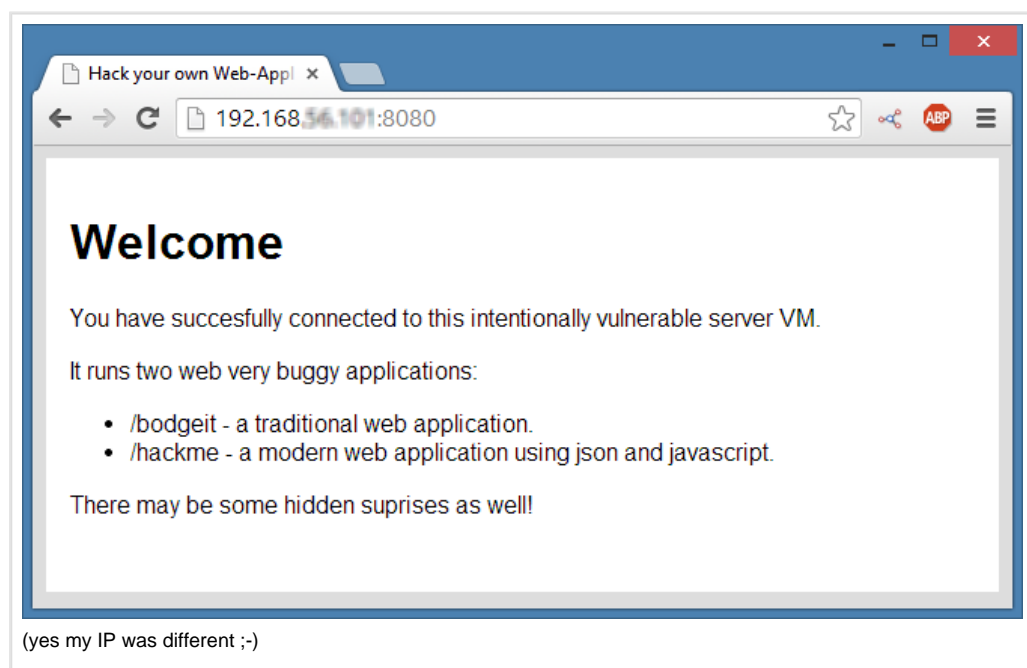
The HackMe VM has its IP Address assigned using DHCP so it must be somewhere in the DHCP range. Unless you want to try every IP by hand it is better to use a tool. We'll use `nmap` (a powerful port scanner) to do a ping scan from the Hacker VM:

- Open a command prompt on the Hacker VM (`ctrl-alt-T`).
- execute the following command (if your DHCP settings were different replace the first 3 digits of the ip address with the one you've written down).

```
> sudo nmap -n -sP 192.168.x.0/24
Starting Nmap 5.21 ( http://nmap.org ) at 2014-01-23 11:25 CET
Nmap scan report for 192.168.x.2
Host is up (0.00052s latency).
Nmap scan report for 192.168.x.150
Host is up (0.00071s latency).
Nmap scan report for 192.168.x.151
Host is up (0.00069s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.51 seconds
```

So the HackMe VM is on 192.168.x.151 - we could have guessed that :-)

Now open a browser on your laptop and enter `http://192.168.x.151:8080/` and you should be greeted by the following screen:



Nmap has many more features than just a ping scan. You can scan a host for services (`-sS`), let it identify the service (`-sV`) and probe for more details (`-sC`) for you. Let's scan the HackMe VM:

```
> sudo nmap -sS -sV -sC 192.168.x.y
```

Did you uncover anything interesting?

For more information on Nmap consult the man page (`man nmap`)



3 Hacking your Web Application

Little mistakes in the code of your web-application can give hackers an opportunity to obtain sensitive data, inject malicious scripts or steal sessions. Most of these mistakes are quite common, so common in fact that OWASP (Open Web Application Security Project) has compiled a top 10 list of the most common flaws in webapplications:

3.1 Owasp Top 10 - 2013

from owasp.org:

Name	Description	Example
A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.	' or '1'='1
A2-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.	not setting 'http-only' and 'secure' flags on cookies.
A3-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.	<script>alert(1);</script>
A4-Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.	hxxp://example.org?id=876 and change the value of id.
A5-Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.	Having the tomcat manager application deployed on your server with default credentials.
A6-Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.	Not encrypting sensitive data or allowing it to be cached.
A7-Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.	Not checking user roles on the service layer.



Name	Description	Example
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.	Not checking <code>referer</code> headers, not having a form nonce.
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.	Using Spring MVC 2.5.4 on a Servlet 3.0 container.
A10- Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.	<code>hxxp://example.com/redirect.jsp?url=evil.com</code>

There are many ways of preventing these mistakes, such as using best practices, static code analysis, code reviews and up-to-date libraries. Ultimately, however, the proof of the pudding is in the eating. So you'll have to check if these vulnerabilities don't exist in your web application. This is what the workshop is about!

3.2 Lab 2 : Exploring the vulnerabilities in the Bodgeit store.

Before attempting to hack your own web application it is a good idea to have some basic understanding on the kinds of vulnerabilities that exist and how to exploit them. For this we will use the Bodgeit web shop by Psiinon.

3.2.1 Missing Function Level Access Control (A7)

- Start the Firefox browser in the Hacker VM and navigate to the Bodgeit store. For me it is `http://192.168.56.101:8080/bodgeit/` but your IP may differ.
- Examine the source code of that page. You'll find a commented out link.
- Navigate to that page.
- You will find some very usable data: basket id's and usernames (no passwords though! :-)

This is a typical example of an A7 vulnerability. The server should check if you are authorized to access that page, but it doesn't, instead it relies on the fact that the user cannot see it or knows about it.

3.2.2 SQL Injection (A1)

Now that we know the user names, we may attempt to login. We could try and guess the password, but that may take a long time. First let's see if the login form is vulnerable to SQL Injection.

An SQL Injection occurs when user supplied values are placed unescaped into the database query. Take the following code for example:

```
String query = "SELECT * FROM users WHERE user='" + request.getParameter("username") + "' AND pass='" + request.getParameter("password") + "'";
```

When used normally, the user name and password are filled in and the query performs as it should. If the username or password contains a single quote (') the SQL statement will become unbalanced and break.



```
correct use : test welkom : SELECT * FROM users WHERE user='test' AND pass='welkom';  
query breaks: test' welkom : SELECT * FROM users WHERE user='test' AND pass='welkom';
```

- goto the login.jsp page
- enter one of the usernames and something as a password. Examine the response.
- enter one of the usernames with an additional ' and something as a password. Examine the response. What is different? (Hint: top left).
- figure out how to trick the query into accepting any username and password combination. (Hint: you need to `OR` with something that is always true and make sure the quotes are balanced).

More info [here](#). Prevention cheat sheet [here](#).

3.2.3 Cross Site Scripting (A3)

Cross Site Scripting or XSS is about inserting user provided data and rendering it unescaped into a HTML page. It comes in three flavors: stored (in the database), reflected (from the http request) and dom based (interpreted in the browser).

- Navigate to the `search` page.
- Search for `pindakaas` and examine the response.
- Now search for `<u>pindakaas</u>` and examine the response. Interesting, our search term is underlined. This means our input is rendered unescaped into the HTML.
- Now try `<script>alert('Reflected XSS!');</script>`



Chrome and Safari have built in XSS filters which block this attack. The FireFox browser in the Hacker VM doesn't.

Another page that is vulnerable to XSS is the Contact page. As its comments are stored in the database, this is an example of stored XSS.

- Navigate to the `contact` page (log out first).
- Try again with `<script>alert('Reflected XSS!');</script>`
- However it does not work. Perhaps the programmer did some (poorly implemented) escaping.
- See if you can find a way round.

More info [here](#). Prevention cheat sheet [here](#).

3.2.4 Missing Server Side Validation

The server should not blindly trust the data the `browser` sends it. In the introduction of the Lab the server didn't restrict access to the admin page. However form value restrictions must be checked on the server as well. In the bodgeit shop, the shopping cart suffers from this flaw.

- find some products and add them to your basket.
- navigate to the basket page.
- notice how you only can change the amount using the plus and minus buttons and no negative values are possible.
- updating sends posts a request with the updated values.

Installed on the Hacker VM's Firefox is a plugin called [TamperData](#) which allows you to intercept post requests and modify them. It can be accessed from the Tools menu.

- Select Tools, Tamper Data. A new window will open.



- In that window select Start Tamper.
- Now press the Update Basket button.
- A dialog opens asking you to tamper with the request. Press Tamper.
- In the dialog that opens, change the quantity parameters to negative values.
- Press the Stop Tamper button.
- Examine the basket. Good, the shop now owes us money :-)

3.2.5 Insecure Direct Object References (A4)

There is something else wrong with the basket. It stores the unique id of the basket in a session cookie. This can be easily changed using the a browser plugin such as [FireBug](#) which is installed in the Firefox on the Hacker VM.

- Logout
- Navigate to the basket page.
- Press the bug button in the top right corner of the browser.
- Refresh the page.
- In the panel that appears select the cookie tab.
- Aside from the session cookie you will see a cookie called `b_id` as well.
- Select it and from the context menu select `edit`.
- Change the value to 1.
- Refresh the page.
- The contents of the basket changes and you've successfully hijacked the basket of someone else!

3.2.6 Broken Authentication and Session Management (A2)

The Cookies used by the BodgeIt store are *not* marked `HttpOnly`. You can see this in the FireBug cookies tab which reserves a special column for this marker. It means that they can be read using JavaScript. For example using cross site scripting:

- Navigate to the `search` page.
- Enter `<script>alert(document.cookie);</script>` as a search term.
- Examine the resulting popup message and compare the contents with the cookie tab from FireBug.

While this may seem harmless it now is possible to craft a bit of javascript to send the visitors sessionid to some other website where it can be used to impersonate the visitor.

3.2.7 Cross-Site Request Forgery (A8)

Cross-Site Request Forgery (CSRF) uses an XSS flaw in an existing website to execute requests on a different website. For example you are logged in on twitter in one tab of the browser and visiting some compromised website on another tab. Using a bit of JavaScript on the compromised website the attacker can make your browser execute a request to twitter to post a Viagra spam tweet or something on your behalf. As the browser knows the session id for twitter it will attach it to the request and the twitter server will accept it.

We don't have two web applications but its quite possible to forge requests on behalf of some innocent visitor by combining two flaws in the BodgeIt store. The first is hidden in the change password form:

- Login using a new or hacked account.
- Click on the username
- Open FireBug and open and activate the `Net` tab.
- Change your password. Note the response.
- Examine the request and post data.
- Open the FireBug HTML tab and navigate to the form.
- Change the POST to GET.
- Change your password again. Note the response.



- Examine the request.

So the change password function accepts both POST and GET requests. GET requests are really easy to create, for example using an image tag.

- Copy the change password request URL from FireBug.
- Change your password to something else.
- Navigate to the `contact` page.
- Now formulate an `img` tag that uses the URL you've just copied as its `src`.
- Press submit and examine the broken image on resulting page.
- Logout.
- Try to login using your current password. This fails if the hack succeeded.
- Try to login using the password from the GET request.

More info [here](#). Prevention cheat sheet [here](#).

3.2.8 Sensitive Data Exposure (A6)

The BodgeIt Shop stores its passwords unencrypted. If we're able to execute queries with results using SQL Injection we can read them. For this hack we'll use the `advanced` search page which has quite an elaborate client side encoding scheme.

- Navigate to the `advanced` search page (accessible from the search page).
- Turn on debug mode by appending `?debug=true` to the url and refreshing the page. Yes. Its a hidden debug feature ;)
- Enter something in the form and press search.
- You can now read the query you're executing.
- Sadly injecting a quote doesn't do anything as they are escaped *client* side.
- Open FireBug again and activate the JavaScript tab. Refresh the page if needed.
- Scroll down to the `encryptForm` method in `advanced.jsp`.
- Note the client side escaping of HTML entities.
- Place a breakpoint just after that line (at: `if(params.length > 0)`).
- Enter something in the form and press search.
- When breakpoint triggers, replace value of the `params` variable in the Watch tab with `type:blah%' union select password,2,3,4,5 from users --+%'`
- Press the Play button.
- The resulting table holds the unencrypted passwords..

Well done! You've completed the first lab and gained (I hope :-)) some insight in the most common type of vulnerabilities and attacks. There are of course [many more](#) :-)



If firebug doesn't accept your changes when you've edited the `params` variable you have to put double quotes around it (")

3.3 Tools on the Hacker VM

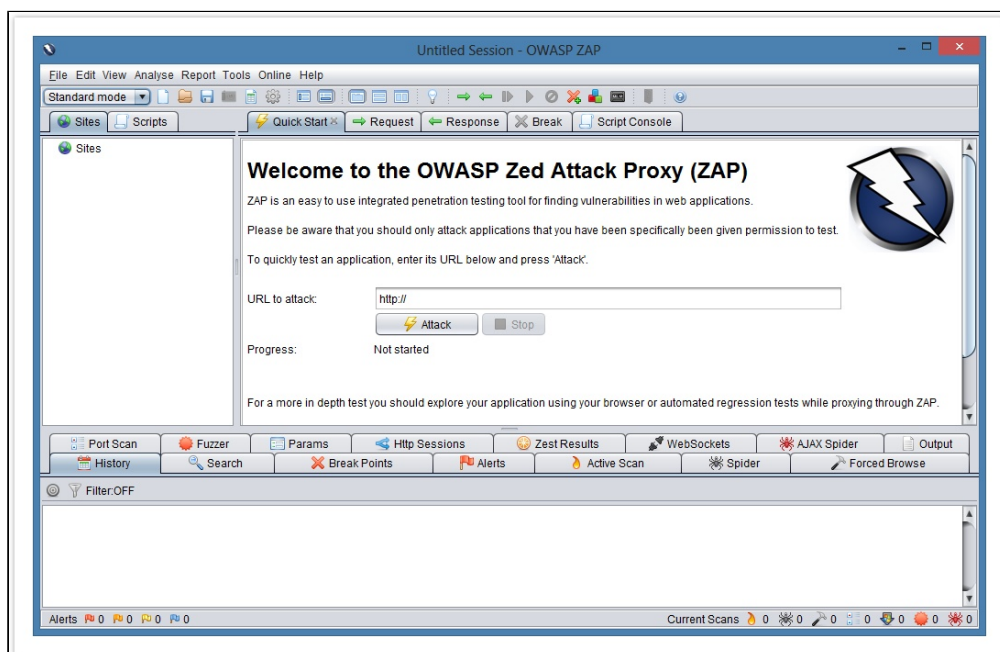
There are many tools available for hacking. There are even specialized [Linux distributions](#) for them. This is of course too much for a workshop. So I had to make a selection:

Tool	Description
Owasp Zed Attack Proxy (ZAP)	An easy to use integrated penetration testing tool for finding vulnerabilities in web applications.
W3af	w3af is a Web Application Attack and Audit Framework. Its goal is to create a framework to help you secure your web applications by finding and exploiting all web application vulnerabilities.
Wfuzz-2.0	Wfuzz is a tool designed for bruteforcing Web Applications.
Nikto	Nikto is an Open Source web server scanner.
THC Hydra	A very fast network logon cracker which support many different services.
John the Ripper	John the Ripper is a fast password cracker. Its primary purpose is to detect weak Unix passwords.
nmap	Nmap ("Network Mapper") is a free and open source utility for network discovery and security auditing.

3.4 Lab 3 : The Owasp Zed Attack Proxy

The Owasp Zed Attack Proxy (short name: ZAP) is a tool that allows inspection and modification of all HTTP traffic by sitting in between the browser and the server. Typically you instruct your browser to use ZAP as a proxy channeling all traffic through it. Then you can use ZAP to watch what URLs have been visited, set breakpoints on certain requests and modify them while they happen. Also there are a whole bunch of scanning options. Most of the Top 10 issues can be found using ZAP.

You can start ZAP using the blue/white lightning bolt icon on the side bar. After a brief wait the user interface will appear.



The user interface of ZAP has quite an intimidating number of tabs. Fortunately you will only use a few at the same time as most of the bottom tabs are dedicated to a specific tool embedded in ZAP. Let's enumerate:



- Top left
 - Sites : holds a tree that displays the URLs that were visited by ZAP and or the browser.
 - Scripts : allows you to create custom scripts to automate testing in various scripting languages.
- Center
 - Quick Start : allows single click testing and Firefox browser configuration.
 - Request : shows the current request.
 - Response : shows the current response.
 - Break : is used when a breakpoint has been triggered allowing you to modify the request and response.
 - Script Console : allows the running/testing of scripts.
- Bottom
 - Port Scan : a crude TCP based portscanner which is useful in environments where you can't use nmap.
 - Fuzzer : allows the fuzzing of request parameters (automatic injection of semi-random data).
 - Params : lists all request parameters encountered
 - Http Sessions: lists all http sessions encountered and allows to select one as the current active one.
 - Zest Results : script output
 - Web Sockets : active web sockets connections.
 - Spider : automated browsing of a web application using the HTTP response as source of links.
 - Ajax Spider : automated browsing of a web application using the browser as source of links.
 - Output: console output
 - History : URLs visited
 - Search: allows you to find a specific phrase somewhere.
 - Break Points : the set break points, ZAP allows you to modify request and response on URLs that have a break point.
 - Alerts : Lists possible problems with requests and responses. Also explains why.
 - Active Scan : attempts to find potential vulnerabilities by using known attacks against the selected target.
 - Forced Browse : uses a dictionary to find hidden services and URLs on the host.

Also there is the little combo box in the top left. It allows you to control the mode ZAP operates in:

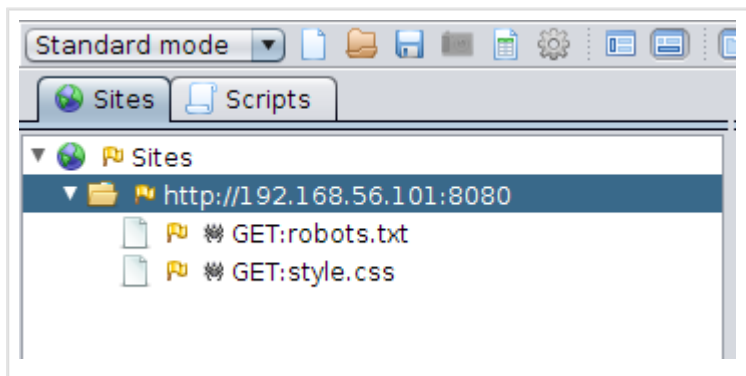
- Standard Mode : Anything goes, all safeties are off.
- Protected Mode : Only allow attacking websites that have been added to a context.
- Safe Mode : No attacks possible.

So much for the introduction, time of some action!

3.4.1 The Once Click Attack

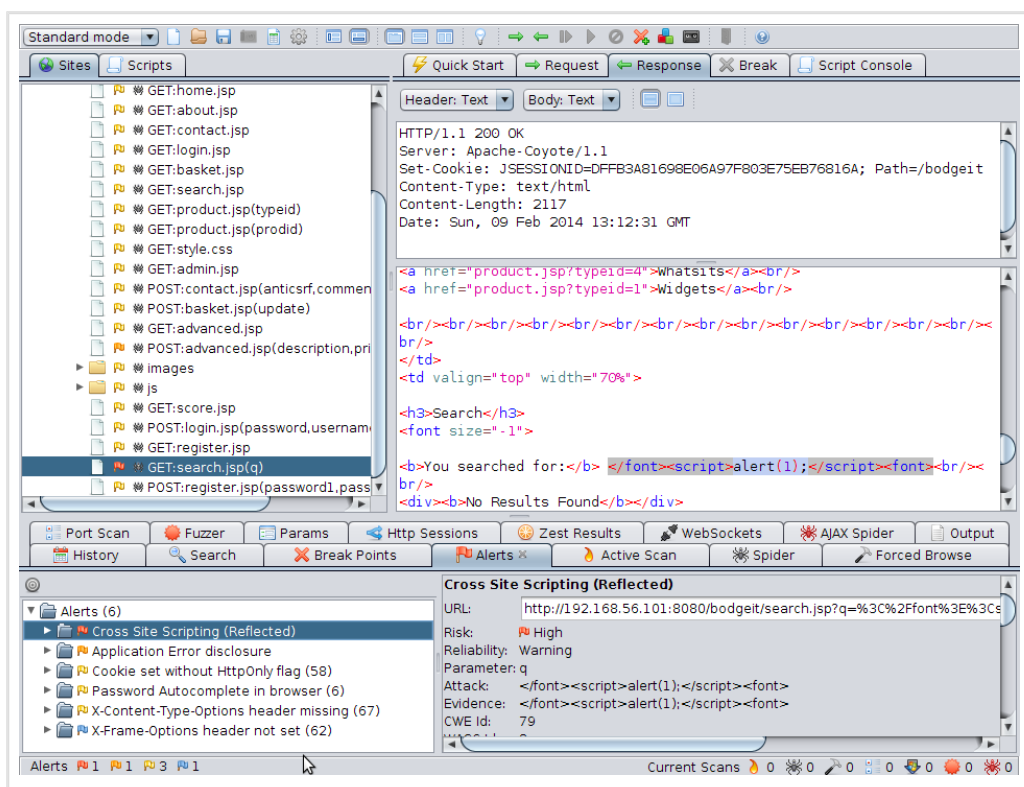
ZAP has a nice 'one click attack' feature that first scans a website for all accessible pages (this process is called spidering) and then uses the built in attack engine to find vulnerabilities.

- Check if ZAP is in Standard Mode.
- Enter the base URL of the Hackme VM (for me `http://192.168.56.101:8080/`) in the URL to attack textfield on the Quick Start tab.
- Press the Attack button.
- Watch what happens.



Well, that was not very impressive. It only attacked the home page and didn't find the two web applications. Maybe it needs a little help:

- Attack the bodgeit web shop by entering its url (for me <http://192.168.56.101:8080/bodgeit/>) and pressing attack.
- Watch what happens (it takes a bit longer though :-)



Good! ZAP found the XSS vulnerability on the search page, it managed to crash the advanced search page, concluded that the store is using cookies without the HttpOnly flag and found the hidden admin page. On the negative side, it missed the other vulnerabilities we already discovered.

The main reason for this is that the attack ran based on what the spider found and entered in the forms. It defaults to ZAP which may not match a lot of the applications input criteria. For example:

- scroll to the bottom of the sites tab and examine the request and response of the `register.jsp`

ZAP tried to register an account here but failed the input criteria. Any pages beyond this point (such as the change password page) were not found. There are ways around this, but first:



- Start your own web application.
- Select File->New Session from the menu and confirm.
- Enter the base URL of your web application into the URL to attack field (alternatively use the provided HackMe web application. Its base url for me is `http://192.168.56.101:8080/hackme/`)
- Press attack.

How far did it come? Did it find anything interesting?

3.4.2 Passive Scanning and Request manipulation

Passive scanning is a technique where you explore the web application through a web browser and ZAP records and examines all the requests and responses. It also records the values you have entered in the forms. Later on you can use these recorded values to attack. But first we will need to configure FireFox to use ZAP as a proxy:


- Start a New Session
- Copy the Plug'N'Hack URL from the Quick Start tab.
- Start FireFox and paste the URL in the address bar.
- Follow the instructions and install the plugin when asked.

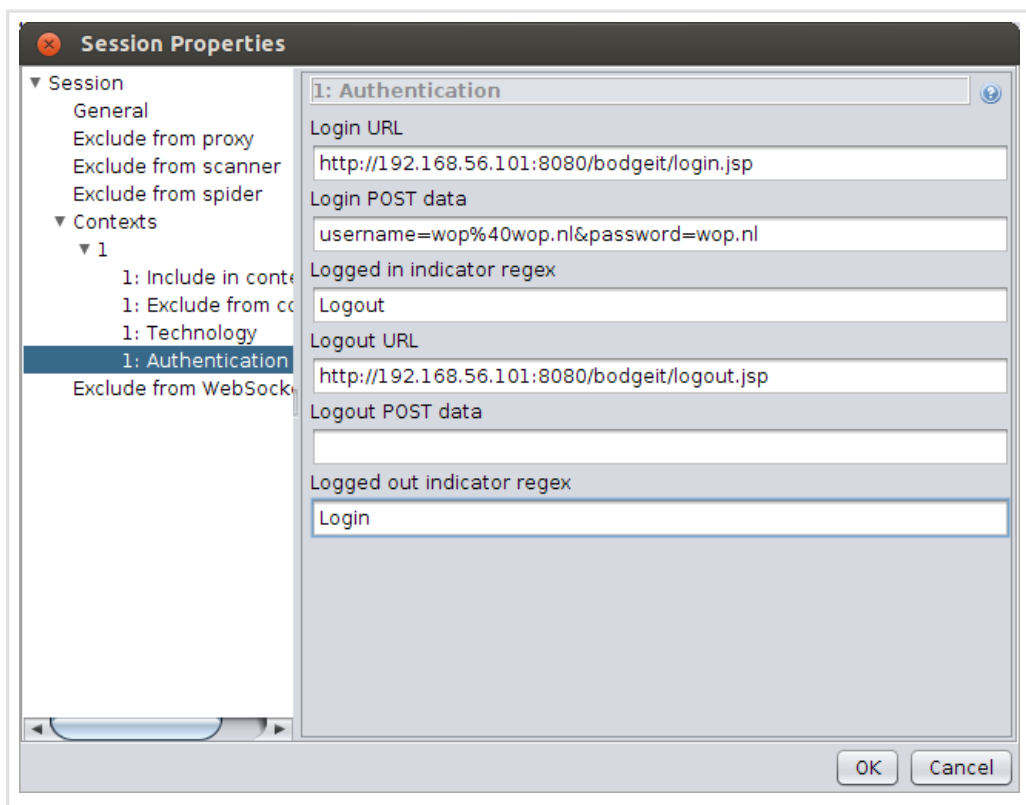
With ZAP configured as a proxy it will monitor all our requests and responses. Let's see what happens:

- Navigate to the bodgeit store.
- Create an account.
- Logout
- Login again
- Add some products to the basket
- Change the product amounts and refresh the basket
- Visit the other pages.
- Examine the results in ZAP.

Notice how just by listening it can already detect some issues with the web application as some alerts have appeared in the Alerts tab. Nothing big though :-)

With the Login and Logout requests captured it now becomes possible to teach ZAP about these requests so that it can Login by itself.

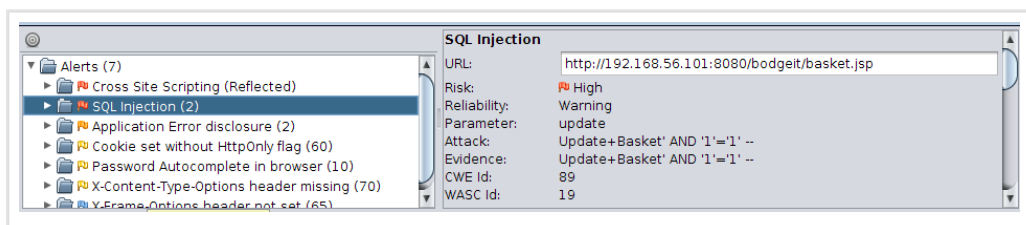
- Switch ZAP to Protected Mode
- Select the root node from the bodgeit store and select 'Include in Context' -> '1' from the popup menu on the node.
- Notice how the matching nodes become 'targeted'.
- Select the Login POST request and select 'Flag as Context' -> 'Login Request' from the popup menu on the node.
- Enter `Logout` as the 'Logged in indicator regex' and close the dialog.
- Select the Logout request and select 'Flag as Context' -> 'Logout Request' from the popup menu on the node.
- Enter `Login` as the 'Logged out indicator regex' and close the dialog.
- By completing this dialog, the 'automatic reauthentication' button () in the toolbar becomes enabled.
- Click it to have ZAP automatically Login when it needs to.



- ✔ If you're not using form based authentication, but HTTP basic authentication you can configure this in ZAP as well. Its in the Tools -> Options dialog under the Authentication tab.

Now with ZAP properly taught let's see if it can uncover some more vulnerabilities.

- Spider the bodgeit store again. This time use the context menu on a targeted URL.
- Attack the bodgeit store again. This time use the context menu on a targeted URL.
- Wait patiently for the attack to complete.



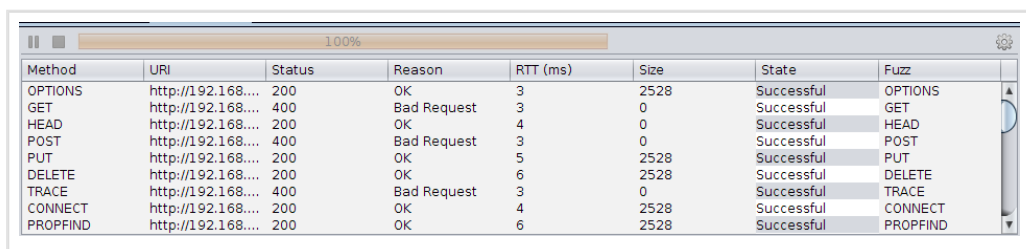
ZAP uncovered a SQL Injection vulnerability in an unexpected place, the `basket` page!

- Examine the request and response, can you find anything suspicious?
- If not it may be a false positive. However its better to be safe than sorry. So let's manually investigate.
- Find the `basket` POST request in the sites tree. There should be a number of different variants.
- Select the one that has `product_id` as parameter.
- From the context menu select `Resend`.
- Adjust the value of the `product_id` parameter to two single quotes (`' '`) and press `Send`.
- Examine the response. There is definitively an SQL Injection present here.

ZAP also allows the manual **fuzzing** of a single request parameter. This is sometimes useful because it allows us to see how the application responds to strange or very large values.

- Select the `password` request from the sites tree (where you can change the password).
- Open the request tab.
- Select `POST` from the HTTP request in the request tab.
- From the context menu select 'Fuzz..'
- In the dialog select for the fuzz category `Fuzz db 1.09 / attack-payloads / http-protocol` and subcategory `http-protocol-methods`.
- Press Fuzz

ZAP will now replace the `POST` with the all of the HTTP methods it knows and record the response of the server. It seems the server responds to a lot more methods than you would expect :-)



Method	URI	Status	Reason	RTT (ms)	Size	State	Fuzz
OPTIONS	http://192.168....	200	OK	3	2528	Successful	OPTIONS
GET	http://192.168....	400	Bad Request	3	0	Successful	GET
HEAD	http://192.168....	200	OK	4	0	Successful	HEAD
POST	http://192.168....	400	Bad Request	3	0	Successful	POST
PUT	http://192.168....	200	OK	5	2528	Successful	PUT
DELETE	http://192.168....	200	OK	6	2528	Successful	DELETE
TRACE	http://192.168....	400	Bad Request	3	0	Successful	TRACE
CONNECT	http://192.168....	200	OK	4	2528	Successful	CONNECT
PROPFIND	http://192.168....	200	OK	6	2528	Successful	PROPFIND

It is time to apply the things you've learned to your own web application!

- Start a New Session.
- Navigate Firefox to the start page of your web application.
- Browse through the application, filling forms as you go.
- When you're done include your web application to the Context.
- Setup the login and logout requests and automatic reauthentication if applicable.
- Find some vulnerabilities!

✔ If you're using the `/hackme` web-application to practice on and need some basic credentials for the login form you can guess them or ask an instructor.

3.4.3 Force Browsing Directories

Aside from spidering, another way of obtaining application URLs is take a huge list of possible names and try them all. ZAP comes with number of built in word lists and the Forced Browse module to perform this tedious job. Let's see what happens if we feed it the bodgeit webshop!

- Start a New Session
- In the Forced Browse tab select the `directory-list-lowercase-2.3-small.txt`
- In Firefox navigate to the Bodgeit home page.
- In ZAP select the node that holds the root of the bodgeit web shop.
- From the context menu of that node select Attack -> Force Browse Directory (and children)
- Watch the Force Browse module do its work.
- Did it find anything you didn't know? There might be a secret hidden somewhere :-)
- Try it on your own web application!

Site:	192.168.56.101:8080	List:	directory-list-lowercase-2.3-medium.txt	79%	Current Scans:1
http://192.168.56.101:8080/bodgeit/	200	OK			
http://192.168.56.101:8080/bodgeit/about.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/advanced.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/basket.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/contact.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/home.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/js/util.js	200	OK			
http://192.168.56.101:8080/bodgeit/login.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/product.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/register.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/score.jsp	200	OK			
http://192.168.56.101:8080/bodgeit/search.jsp	200	OK			

This is the end of the second lab. However ZAP has many more features, such as HTTP breakpoints, a Selenium based Ajax spider and Web Sockets support. Feel free to explore those on your own!

3.5 Lab 4 : Brute forcing Login Forms with THC Hydra

Hydra can be used to check common username and password combinations (I am sure that your organization also has a `welkom01` password somewhere in use). It is also very fast and supports a wide variety of protocols including SSH, HTTP and HTTPS forms.

The main lesson to be learned from this tool is: If you don't use a rate limiter (such as a captcha or a timed lockout) on your login forms you may be in trouble :-)

First let's try a form with a known username and password:

- Open a command prompt (`ctrl-alt-T`)
- `hydra` is on the path and ready to be used.
- The following example will attempt to try a single username and password against the `bodgeit` store.

```
hydra 192.168.x.101 http-form-post "/bodgeit/login.jsp:username=^USER^&password=^PASS^:invalid" -l
username -p password -s 8080 -t 1
```

- the quoted and colon separated string holds : the url to test against, the post string with placeholders for username and password, the string to detect a login failure.
- the `-t` limits the number of threads to one.
- the `-l` and `-p` specify the (one) username and password. If capitalized a file name is expected.
- use an existing or create a new login in the `bodgeit` store and see if you can get a successful login from `hydra`.



You can use `https-form-post` if you're running `https`.

Now point `hydra` against your own web application.

- Make up your own username list, or use a single known username.
- A rather large password file is stored in the folder `~/tools/password/`
- You need to unpack it before use: `bunzip2 rockyou.txt.bz2`
- Alternatively use the `-x` option to bruteforce through the alphabet.

For full details check the man page (`man hydra`).



When running `THC Hydra` your internet security solution may think you are being `DOSsed`. Use `-t` to reduce the number of threads.



3.6 Lab 5 : Web server scanning with Nikto

Nikto is a web server scanner, it scans for known vulnerabilities and configuration errors in a web server setup. While this is not directly relevant for testing a web application, most web applications use web servers as a platform to run on (PHP) or as a frontend for a web container (Apache httpd in front of Tomcat). Nikto uses the concept of a single click scan and as such is pretty easy to perform.

- Open a command prompt (`ctrl-alt-T`)
- Nikto is on the path and ready to be used.

```
> nikto -host 192.168.x.101 -port 8080 -Tuning x8
```

- The above instructs Nikto to do a scan on the given host, not testing any Remote Shell attacks.
- Run Nikto against the HackMe virtual machine and examine the results.

Nikto reports issues against the Open Sourced Vulnerability Database ([OSVDB](#)) which provides details on the issues found. You can look them up by number. Did you find anything interesting?

Note the `-Tuning x8` is important because otherwise Nikto will attempt Remote Shell attacks which may set off your antivirus solution.

3.7 Lab 6 : Web server fuzzing with Wfuzz

Wfuzz allows you to fuzz HTTP requests. It works more or less the same as the fuzzer in ZAP except that it has more word lists and other creative options such as the number ranges (useful for trying insecure object references), recursion, multiple fuzzed strings and different encodings (to circumvent buggy filters).

- Open a command prompt (`ctrl-alt-T`)
- Move to `~/tools/wfuzz-2.0`
- Read the `README` file or just enter `./wfuzz.py` for a shorter summary.
- As an example we will use the `tomcat vulnerabilities` file to see if there are any problems with the Tomcat on the HackMe VM:

```
./wfuzz.py -c -v -z file,wordlist/vulns/tomcat.txt --hc 404 http://192.168.x.101:8080/FUZZ
```

This will instruct wfuzz to:


- `-c` : scan with colors
- `-v` : be verbose
- `-z file,wordlist/vulns/tomcat.txt` : use the specified file as payload.
- `--hc 404` : ignore 404's
- `http://192.168.x.101:8080/FUZZ` : use this url to fuzz, replacing `FUZZ` with whatever there is in the file.


Are you curious of what will be found?

- Run the fuzzer against the HackMe VM!
- .. now what was that default username/password again ..



- Oh dear.
- Next, try enumerate the products in bodgeit store using the `-z range` option.
- And, of course, try it against your own web application!

 Some of the wordlists contain remote shells that may upset your antivirus solution.

 The WFuzz folder also contains the `wfuzzforpentester2011.pdf` where you can learn more about the advanced possibilities of WFuzz.

3.8 Lab 7 : Vulnerability Scanning with W3af

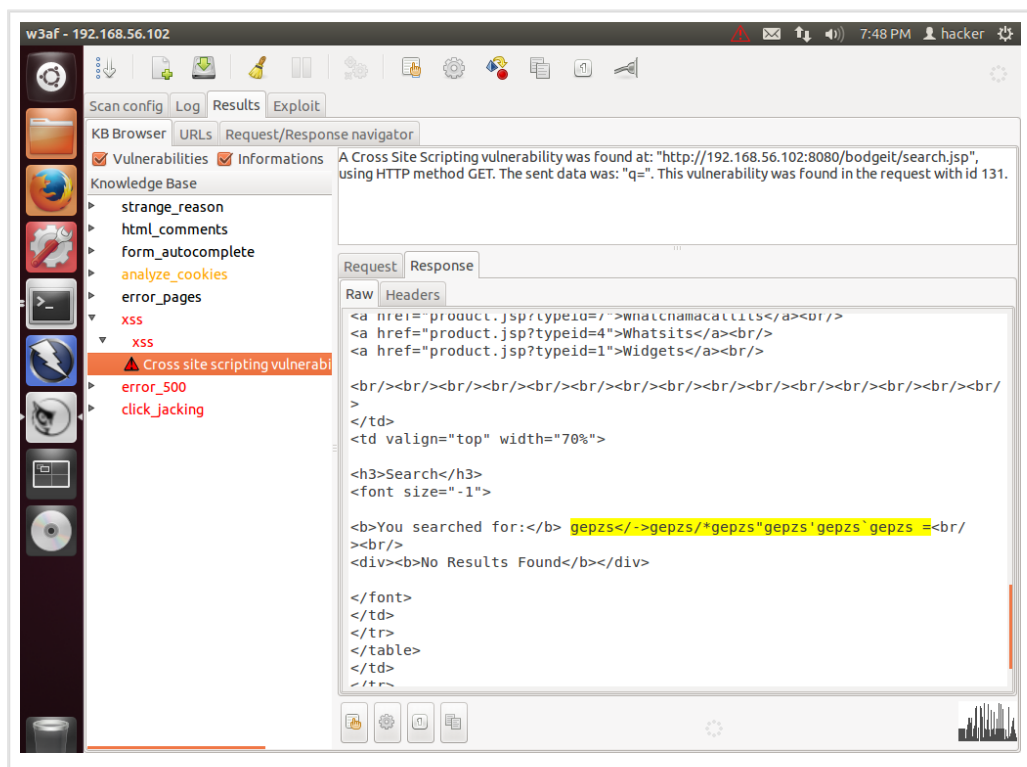
W3af (Web Application audit and attack framework) is a feature rich vulnerability scanner. It consists of a series of plugins that work together to produce a thorough scan. There are 9 categories of plugins:

```
infrastructure -----\  
    /-> bruteforce -\  
crawl --+> audit -----+> output  
    \-> grep -----/  
    \-> auth -----/  
        | |  
        evasion  
        mangle
```

- Infrastructure plugins try to find information about the web server infrastructure components (web server type and version, reverse proxy, etc)
- Crawl plugins use various techniques to find URLs to scan (like basic crawling, using search engines, etc)
- Evasion and Mangle plugins allow the modification of the requests sent in order to change something on the fly or evade intrusion detection systems.
- Audit scans for specific vulnerabilities (SQLi, XSS etc)
- Grep extracts potential useful information (such as email addresses)
- Bruteforce attempts to bruteforce any login forms encountered.
- Auth enters a preconfigured username and password when a login form is encountered.
- Finally, the output plugin generates the findings in a report like format.

To make things easier, plugin selection and configuration can be stored in so called profiles. Also they can be put in scripts. We're now going to run a fast scan on the `bodgeit` store.

- Open a command prompt (`ctrl-alt-T`)
- Navigate to `~/tools/w3af`
- Execute `./w3af_gui`
- Select the `fast_scan` profile.
- Examine the plugins that are activated by this profile.
- Enable the `output_text_file` plugin (bottom center pane)
- Enter the bodgeit url (for me `http://192.168.x.102:8080/bodgeit`) in the target text box and press Start.
- Wait for the scan to complete.
- Examine the results in the `Results` tab.



- Quit w3af and examine the output files in your home (~) folder.

While W3af is quite a powerful tool its limited by a buggy user interface. Modifying existing profiles does not really work well, so its often better to use the the empty profile and work from there.

- Restart W3af
- Create your own profile to attack your own web application.
- Do not use any of the infrastructure plugins, they tend to take very long and are very heavy.
- Also don't use any plugins that require the internet.
- Finally don't use any of the `audit_ssl` plugins if you're not running on https.

3.9 Lab 8 : Brute Forcing Hashes with John the Ripper

John the Ripper is a password cracker that allows you to test wordlists and alphabets against the results of various hashing algorithms (md5, sha1, etc) in order to find the password that generated the hash. John comes in three different versions, the Pro (paid), normal (Open source) and Jumbo (with community additions which makes it able to brute force passwords on many file formats). In this tutorial we'll use the normal version.

On Linux systems the usernames and passwords are stored in two files, `passwd` and `shadow`. `passwd` holds account details (and can be read by all users), `shadow` holds the actual password hashes (only readable by root). For this Lab I have managed to obtain the `passwd` and `shadow` files of the HackMe VM and we're going to use John to check if the passwords are any good.

- Open a command prompt (`ctrl-alt-T`)
- Navigate to the `john` folder.
- In it you will find the two files from HackMe.
- In order to use these files in John we need to combine them into one first using the `unshadow` utility.

```
unshadow passwd shadow > out.txt
```



- Examine the resulting out.txt file. How does it differ from passwd?
- Now we can use John to check if there are any easy to brute force passwords in the file.

```
john out.txt
```

- Ok that was pretty quick. I suppose that password was really easy :-)
- Next we'll make up a password using a weak hashing algorithm (crypt-MD5)

```
mkpasswd -m md5 boem > out.txt
```

- open your favorite editor (such as gedit or pico) and put `test:` before the hash and save (this is the format John wants its hashes in).
- Now try again and see if John can find the password for this hash.
- How long did it take? How many hashes per second did John do?
- Next we'll take a more secure hashing algorithm (sha-512):

```
mkpasswd -m sha-512 boem > out.txt
```

- Again, edit the file and place a fake user in front of the hash.
- Rerun John.
- How long did it take this time? How many hashes per second?

Choosing an computationally expensive hashing algorithm makes it much harder to brute force hashes and protects passwords longer.

Of course with the aid of graphics cards to do parallel hashing (using [Hashcat](#) for example) and cloud infrastructure that time becomes shorter as computing power increases.

More [information on hashing and salting](#) passwords.



4 The End

You have reached the end of this workshop. I hope you have gained some insight in the things that can go wrong with web-applications and how to test for them while having fun!

All workshop material compiled and written by [Erik Hooijmeijer](#). Thanks to [Robert Bor](#) and [Kees van Dieren](#) for their assistance and providing valuable feedback. Thanks as well to the participants of the trial runs of this workshop at [42 B.V.](#) and IOO Gemeente Rotterdam.

(c) 2014 E.Hooijmeijer