LAMPSecurity Project Capture the Flag

Web Application to Root Via Vulnerability Exploit

by Justin C. Klein Keane
justin@madirish.net

http://www.MadIrish.net

Table of Contents

About this Document

This document should accompany the CTF7 exercise of the LAMP Security Project, which is hosted on SourceForge.net (currently at the URL http://sourceforge.net/projects/lampsecurity). This document is copyrighted work, which means you cannot copy or redistribute it without permission, especially for profit. I do this so assholes don't put the contents of this document on their websites to drive traffic and sell advertisements.

This document is time sensitive. As years pass this document will likely show age and certain technical details may change (such as the virutalization software in use). I have tried my best to keep things current while trying to address this issue, but please be aware that things like URL's or availability of certain products may have changed since CTF7 was devised.

Credentials and Logging In

The number one most frequently asked question I get is "what is the username and password to log into the CTF?" The exercise is designed so that you don't know this information – you are supposed to break into the target and crack the login yourself. However, if you absolutely must log into the image (for instance if you want to change the networking configuration so you can deploy the image in your environment) you can find those details in the appendix or by reading through this document.

Warning!

The CTF7 image contains a number of serious vulnerabilities. **Do not** deploy this image in a production environment or in any way connect it to the internet! Doing so could likely result in the image being quickly compromised and becoming a pivot for attackers.

Before you Begin

The contents of this exercise assume that you are using a BackTrack Linux version 5 Release 3 virtual machine as the attack platform. BackTrack is a wonderful penetration testing Linux distribution that comes with a multitude of free security testing tools installed and configured. For more information about BackTrack and to download a bootable CD image, Vmware image, or other format see http://www.backtrack-linux.org/.

You'll need VMware's free player in order to run the image. You can download the CTF7 image from https://sourceforge.net/projects/lampsecurity/files. You can download the VMware player from http://www.vmware.com/download/player. Alternatively you may be able to get the image to work using Oracle's VirtualBox (https://www.virtualbox.org/) or other virtualization software.

Note for VirtualBox Users

Note that if you're using VirtualBox you may experience some issues. The CTF7 image is configured with a network adapter with the MAC address 00:0C:29:9D:12:A9 (you can find this value in the CentOS.vmx file). You may need to adjust the settings for the CTF image to use this MAC address. The image is also built to use NAT for networking. This should be changed to "host-only" networking in order to work in VirtualBox. To create the VirtualBox version simply create a new virtual machine with the Linux and Other 2.6 Kernel specifications, then when prompted for a hard disk, choose existing and select the CTF hard disk (CentOS.vmdk).

Conventions Used in this Document

Arrows are used to indicate progression between menus in a program. For instance, if you are being instructed to click on the File menu in a program, then select the Properties option this is denoted using:

File → Properties

All command line instructions are listed in courier fixed font. These will often include the prompt preceeding the command, such as:

```
$ ls -lah
```

It is not necessary to type the '$' as part of the command, it is merely listed for completeness.

Purpose

This exercise is intended to be an educational experience. In particular it is designed to demonstrate how vulnerabilities can be "chained" together to lead to a complete compromise. There is no system on the target that is immediately exploitable to become root, but there are problems that can be exploited in tandem to compromise the root account.

This exercise can also be used to benchmark automated testing tools. In particular this exercise seeks to expose participants to effective, free, open source security testing tools as well as to demonstrate many of the common weaknesses of such tools. Although the approach to this exercise is scripted, there are a number of unscripted vectors that can be used to exploit the target. I encourage you to try the exercise without this document, then refer to the document if you get stuck or don't know what to do next.

Step 1 – Finding the Target

The first step in the exercise is to get your virtual LAN set up and find the target.

Getting Started

The first step to carrying out the exercise is to start up the virtual machine image and locate the LAN segment upon which it resides. To do this open up VMWare Player and click the 'Open' menu and navigate the the CentOS.vmx image, then click the 'Play virtual machine' button.

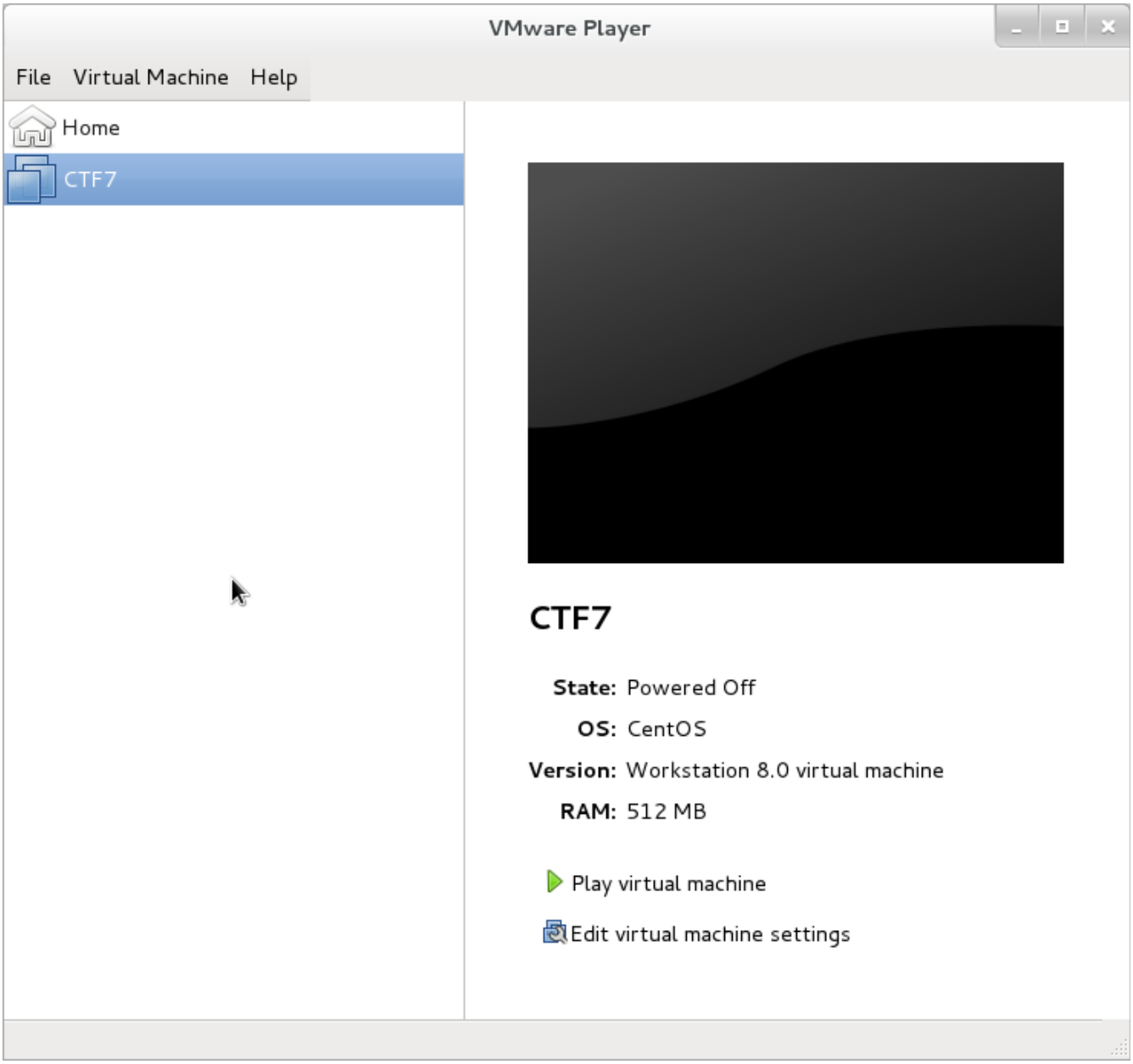


Illustration 1: VMWare Player opening the target

If prompted, you should select the 'I moved it' option to let VMWare know that you have changed the supporting hardware platform.

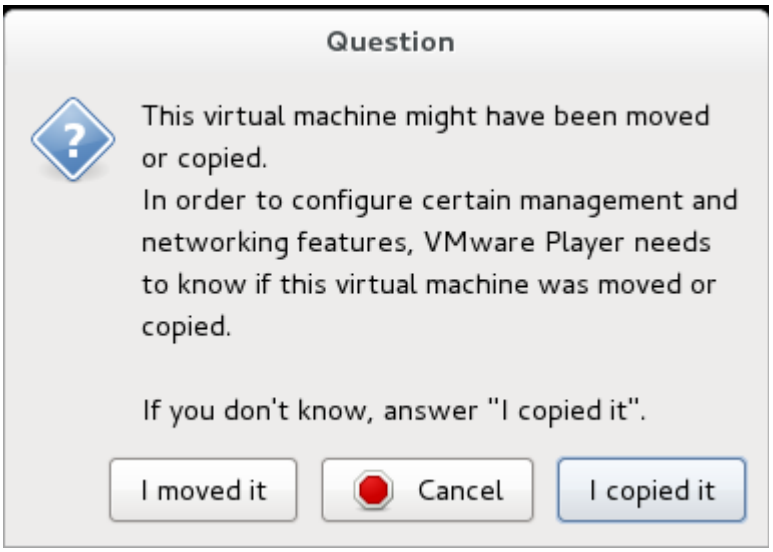


Illustration 2: VMWare may ask if the machine has been moved

Now that the target is up and running there's no easy way to determine the IP address of the target. One way to do this, however, is to start up another virtual machine on the same virtual LAN segment. Start up your BackTrack virtual machine, ensuring that it is assigned the same virtual networking settings in VMWare Player. For the purposes of this documentation I have created a new virtual machine with a 16 GB hard drive and booted from the BackTrack 5 R 3 32-bit Gnome ISO, with BackTrack installed to the hard drive. The advantage to this set up over using the bootable ISO, is responsiveness. You can also update all the tools and retain the updates over time.

Boot up the BackTrack image and ensure the networking settings are the same (Virtual Machine → Virtual Machine Settings → Network Adapter). After starting up the virtual machine log in with the default BackTrack credentials (username “root” with the password “toor”).

You can find the IP address of the virtual machine using the command:

```
# ifconfig eth0
```

The output will show the IP address after the “inet addr” entry.

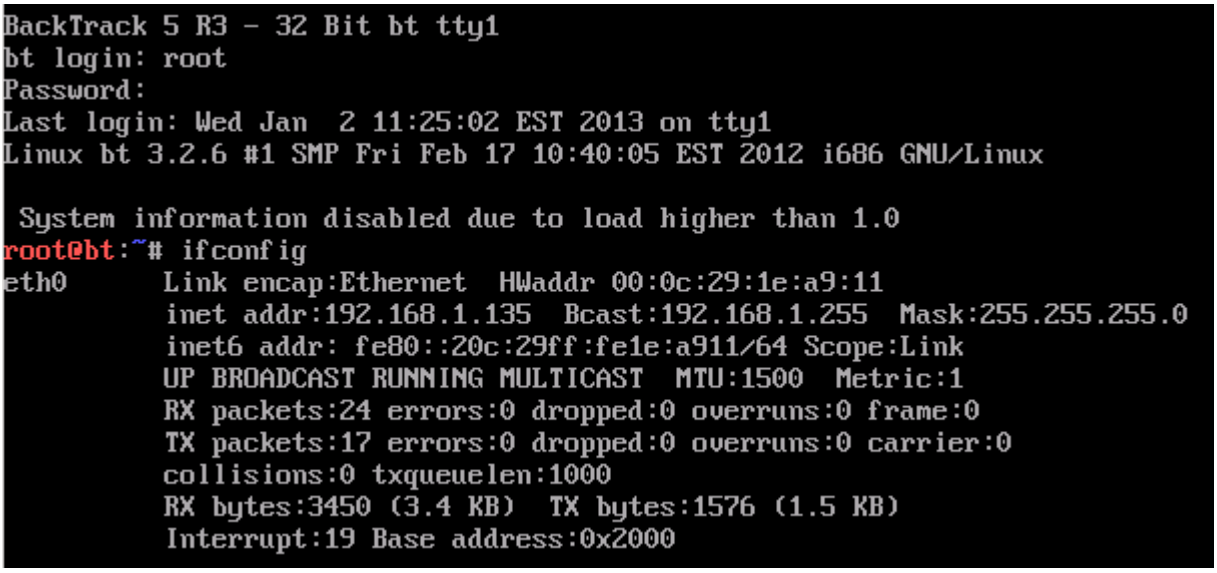


Illustration 3: Determining the BackTrack VM IP address

In the above example you can see the IP address is '192.168.1.135'. Now that we have the IP address we can easily search for the target using the network mapper NMAP. You can run a quick NMAP scan using the code:

```
# nmap 192.168.1.2-254
```

The code should run rather quickly and outline the target. Note that two results will appear, one for the BackTrack attack platform and one for the CTF target. You may even note an entry for your hardware host depending on your setup. The target is set up to use DHCP, so it likely has an address contiguous to that of the BackTrack image. For instance, if your BackTrack image has the IP 192.168.1.120 then the target is likely at 192.168.1.119 or 192.168.1.121. Look for the machine with lots of unknown ports as it is likely the target.

```
root@bt:~# nmap 192.168.1.2-254

Starting Nmap 6.25 ( http://nmap.org ) at 2013-01-03 14:27 EST
Nmap scan report for 192.168.1.2
Host is up (0.00049s latency).
All 1000 scanned ports on 192.168.1.2 are closed
MAC Address: 00:50:56:FA:17:10 (VMware)

Nmap scan report for 192.168.1.6
Host is up (0.00059s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
111/tcp  open  rpcbind
443/tcp  open  https
3306/tcp open  mysql
MAC Address: 00:0C:29:3C:5C:22 (VMware)

Nmap scan report for 192.168.1.135
Host is up (0.0000060s latency).
All 1000 scanned ports on 192.168.1.135 are closed

Nmap scan report for 192.168.1.136
Host is up (0.00049s latency).
Not shown: 992 filtered ports
PORT      STATE  SERVICE
22/tcp    open   ssh
80/tcp    open   http
139/tcp   open   netbios-ssn
901/tcp   open   samba-swat
3306/tcp  open   mysql
5900/tcp  closed vnc
8080/tcp  open   http-proxy
10000/tcp open   snet-sensor-mgmt
MAC Address: 00:0C:29:9D:12:A9 (VMware)

Nmap scan report for 192.168.1.254
Host is up (0.00011s latency).
All 1000 scanned ports on 192.168.1.254 are filtered
MAC Address: 00:50:56:E6:04:C4 (VMware)

Nmap done: 253 IP addresses (5 hosts up) scanned in 12.53 seconds
```

Illustration 4: NMAP discovering the CTF target

We can see in the results above that the target IP address is 192.168.1.136. Additionally it is interesting to note that there are quite a few different ports open. For this exercise we're only interested in web server ports, but even limiting our investigation to those services doesn't necessarily limit us to port 80. Using the NMAP service version detection flag (-sV) we can attempt to determine what programs are listening on the ports of the target.

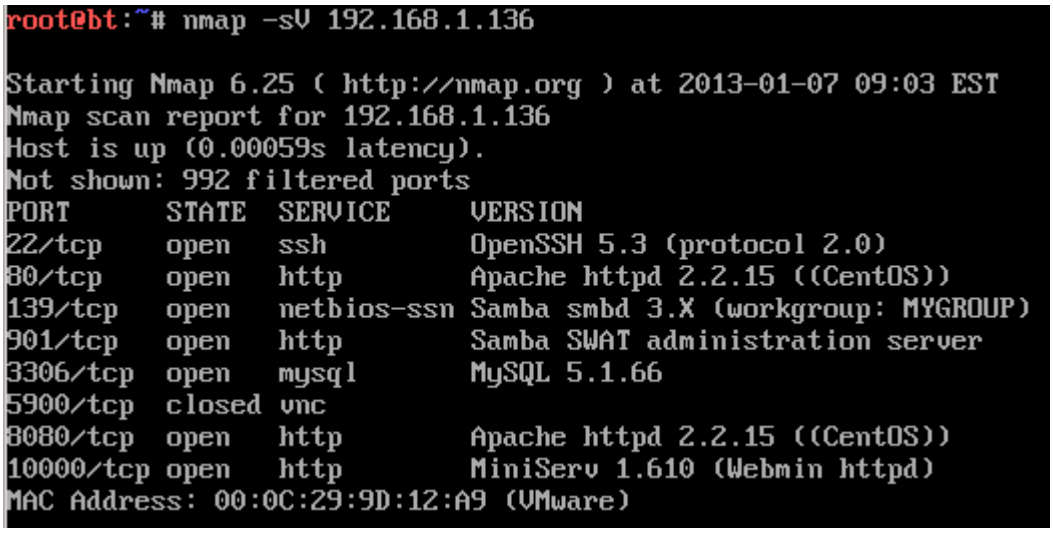


Illustration 5: NMAP performing service version detection

Looking at the version detection output we can clearly see that an Apache web server is running on both ports 80 and 8080. Interestingly, Webmin, is running on port 10000 and seems accessible remotely. Webmin is a web based server administration tool. Similarly, SWAT, the Samba Web Administration Tool, is available for configuring Windows shares via a web interface. At this point we can pull up a web browser and confirm.

If you haven't already, start the X server in BackTrack by typing in:

```
# startx
```

to get the graphical desktop so we can start the Firefox web browser. X is the windowing environment on Linux and Unix machines. Gnome is the actual desktop, but it is set as the default desktop manager in BackTrack, so starting X will fire up both X and Gnome so we can use the graphical interface to perform the remainder of our testing. Using the graphical interface has some usability advantages, but it is slightly slower and takes up more resources. You can still use the icon at the top of the screen to open command prompts, however, so the command line is only one click away. You can return to text only mode simply by logging out of BackTrack under the System menu. Once the desktop is started you'll find all the programs on the BackTrack image under the Applications menu, and most of the attack tools under Applications → BackTrack.

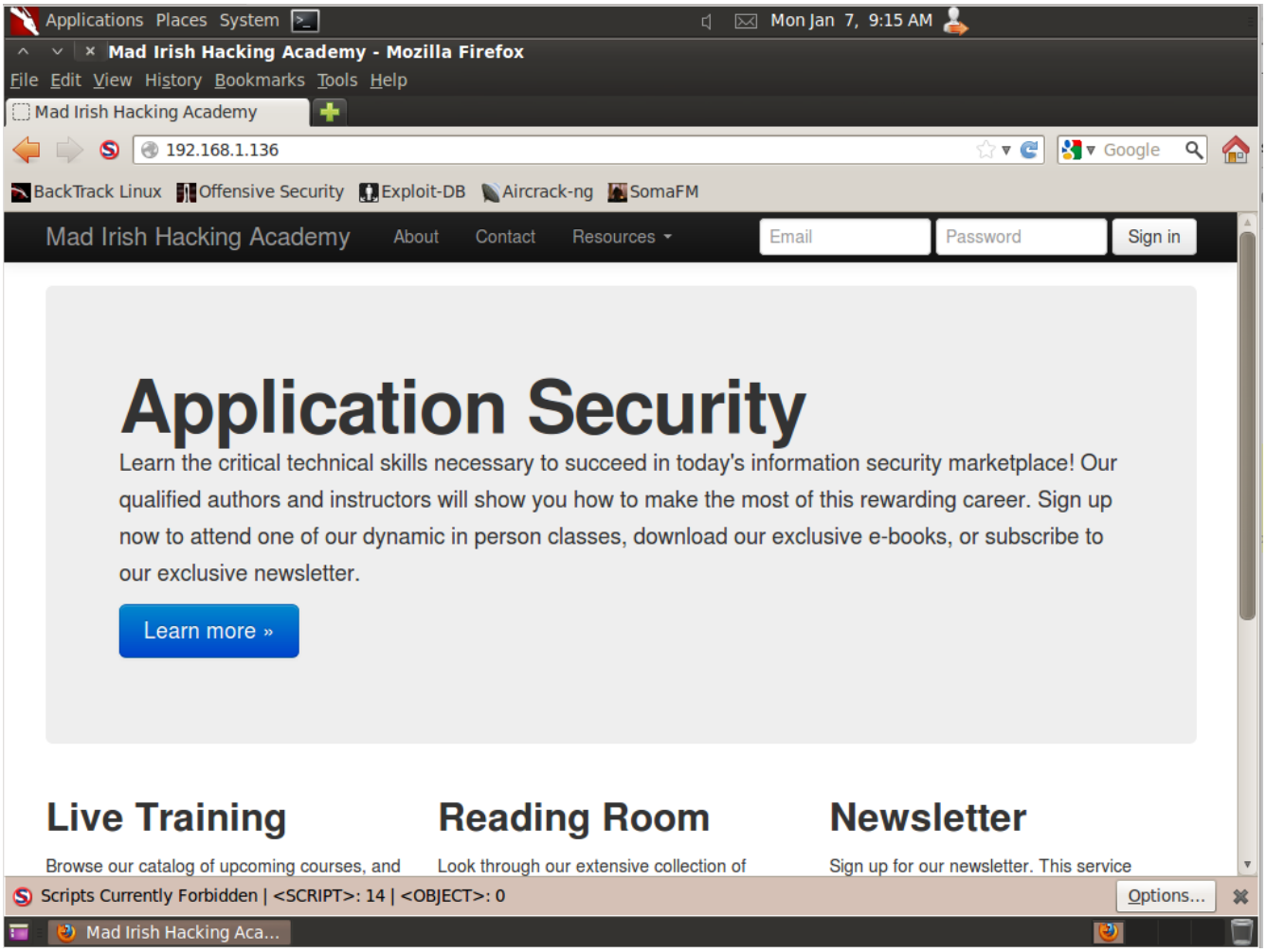


Illustration 6: Firefox running in the BackTrack VM

Poking around the target website will reveal that it is relatively complex. Auditing by hand, that is, going through and probing for vulnerabilities, is a time consuming and somewhat arduous task. There are a number of things to look for, and testing for vulnerabilities by hand is both tedious and error prone. In order to be thorough and complete it is helpful to follow a checklist (like the OWASP top 10) and search each page for vulnerability. In practice, this is exactly the type of boring, repetitive work that is ideally suited for a computer. Once issues are identified, as we'll see, the automated tools will begin to show their weaknesses and human testing will be superior, but at this point it is best to use programs to test the broad surface of the web application for potential issues.

It would be even more effective to review the actual source code of the application. This “white box” testing makes it much easier to test all of the application code and spot flaws. However, it is commonly the case that source code isn't available and “black box” testing, of the type in this exercise, is required.

Step 2 – Automated Testing

Now that we have identified the target it's time to use some automated tools to test for vulnerabilities and explore the target.

Automated Tools

There are a number of great web application testing tools included on the BackTrack CD. Each of these tools is available for download, although they are sometimes temperamental to install (thus the benefit of BackTrack). However, if you find yourself using these tools on a regular basis, it will be much more efficient to install them on an actual machine since virtual machines have greatly reduced performance due to the overhead of the virtualization software. The three tools we'll start with are:

- Nikto (http://www.cirt.net/nikto2)
- w3af (http://w3af.sourceforge.net/)
- OWASP Zed Attack Proxy (ZAP) (https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

All three of these tools can be found under the Applications → BackTrack → Vulnerability Assessment → Web Application Assessment → Web Vulnerability Scanners.

Nikto

Nikto is an open source vulnerability scanner written in Perl. It will run on any machine upon which Perl is installed, making it highly portable. Nikto uses a database of signatures to URL's that indicate vulnerable software is installed, which has advantages and disadvantages. Like many of the tools on BackTrack, Nikto is a command line tool.

Nikto's Strengths

Where Nikto truly shines is its ability to identify vulnerable versions of common software. For instance, if a vulnerability is identified in Foo CMS 1.4.6 then Nikto is a great tool for finding instances of Foo CMS that match this version in an environment. Nikto is a command line tool that can output in many different formats, which makes it ideal for use in an automated script or to populate results for use in another system. Nikto is also a passive scanner, which makes it very safe in a production environment. Nikto merely makes URL requests, rather than submitting forms or fuzzing input parameters, so the likelihood of unexpected problems with a testing target is low. Another great strength of Nikto is the fact that it is written in Perl, and the codebase is rather small, which means making modifications or adding custom tests is extremely easy. Nikto includes a user defined database of tests, so you can add tests specific to your environment quickly and easily.

Nikto's Weaknesses

Nikto's weakness is it's reliance on a white list of vulnerabilities and the fact that is only uses URL's to identify vulnerability. Nikto does notably poorly at identifying problems in custom web applications (since URL's for such applications aren't part of the Nikto database).

Running Nikto

Start up Nikto from the BackTrack menu. This will open a command prompt in the directory where Nikto is installed. The version of Nikto installed by default has a bug and outputs the error:

```
Undefined subroutine &main::get_ips called at
/pentest/web/nikto/plugins/nikto_headers.plugin line 72.
```

Download the latest version of Nikto, that has fixed this bug, into the folder /pentest/web/nikto2. To do this use the following commands:

```
# cd /pentest/web
# wget http://www.cirt.net/nikto/nikto-2.1.5.tar.gz
# tar -xvzf nikto-2.1.5.tar.gz
# mv nikto-2.1.5 nikto2
# cd nikto2
# chmod +x nikto.pl
```

The first step to running Nikto is to update the database. To do this type the following:

```
# ./nikto.pl -update
```

Once complete Nikto is ready to be run. To do this type:

```
# ./nikto.pl -host 192.168.1.136
```

This will start a Nikto scan of the web applications served on port 80. We can use the '-port' flag to change this behavior. To scan the applications at port 8080 we simply use:

```
# ./nikto.pl -host 192.168.1.136 -port 8080
```

Fire up Nikto and observe the output. Note how quickly Nikto runs, but also be aware that because Nikto is merely doing signature matching there may be some false positives (that is Nikto may report vulnerabilities that might not actually exist).

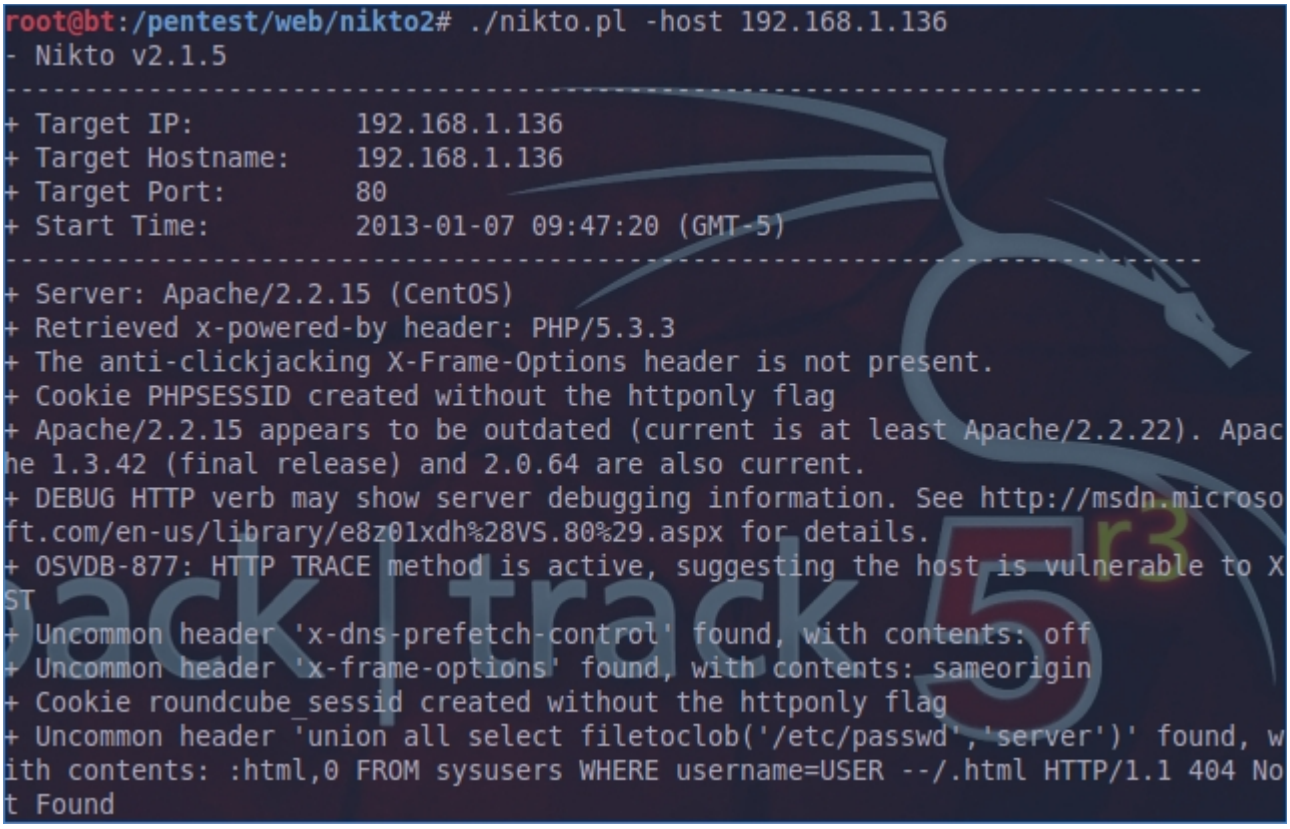


Illustration 7: Nikto running against the target

Note the great number of hits that Nikto identified as vulnerabilities! This is somewhat suspicious, and worth checking out in a web browser. Pull up one of the vulnerable URL's in Firefox (note that your output may differ slightly), such as http://192.168.1.136/mailman/listinfo/<script>ALERT Vulnerable</script>#54897390251357384 74. Any suspiciously detailed output about a system you suspect isn't running on the target (stuff like Cisco web managers or indicators of other commercial software is a false positive) will work for this test. The idea is to validate a false positive result in the Nikto output.

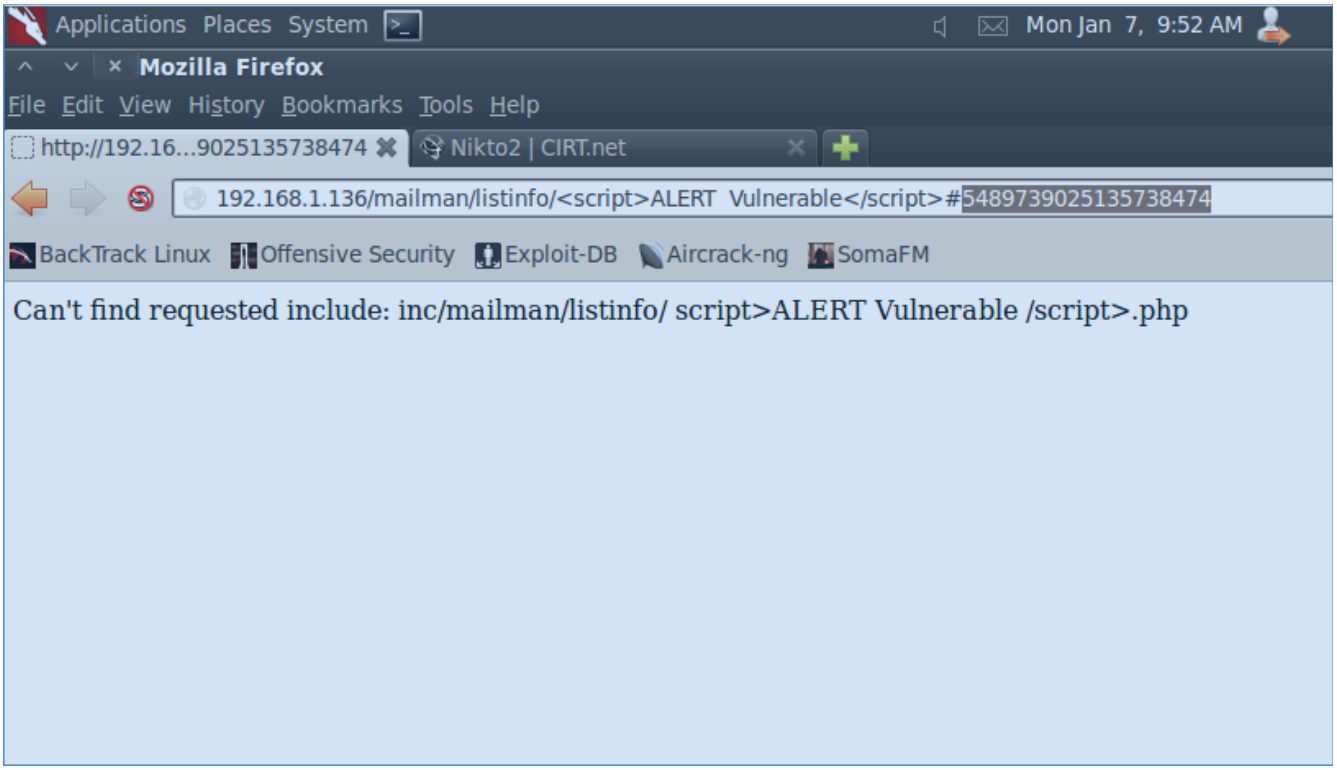


Illustration 8: Using Firefox to verify Nikto results

Note that is page is a 404 error, but like many web applications, the custom application on port 80 seems to be redirecting all requests to some sort of a handler. The Apache web server is actually using mod_rewrite to rewite the URL for all incoming requests to point them to a central controlling script (which is then responsible for returning the 404 error). Thus, any URL will result in some return, which may confuse Nikto since patterns will match the Nikto database.

One interesting finding is that it appears this page has a reflected cross site scripting (XSS) vulnerability. This vulnerability isn't actually identified by Nikto, again, because Nikto is merely doing pattern matching on URL's to determine if known vulnerable software is installed. This is again, one of Nikto's main weaknesses.

Note that despite the false positives and false negatives (that is, vulnerabilities Nikto does not report), Nikto still finds a host of useful information, such as the existence of subdirectories like /img, /webailizer, and the PHP hidden credits that can reveal version information at /index.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000.

W3af

w3af is the Web Application Attack and Audit Framework. The project, which is now part of Rapid7, aims to be the Metasploit (http://www.metasploit.com/) of web applications. W3af includes modules to discover, audit, analyze and exploit web applications. W3af includes a graphical front end, but according to the developers the command line utility is far superior. The GUI, however, is imminently more approachable, despite it's instability and lack of support for certain features.

W3af's Strengths

W3af has a number of strengths. Open source, and written in Python, w3af's greatest strength is its extensible code base. The sheer number of tests and exploits built into w3af is staggering. For instance, if you run w3af with all tests enabled it could take days to finish testing. This level of thoroughness is impressive, but can also be daunting. W3af also does a good job of intelligent testing and the code can discover a number of vulnerabilities using fuzzing techniques.

W3af's Weaknesses

w3af's greatest weakness is its complexity. The code base is extremely large and prone to obscure bugs. It is not unusual to run into some problems when running w3af, but online resources are fairly abundant for problem solving. W3af suffers from the same limitations of many of it's counterparts in that it will only test what it can find. This means that testing applications behind even simple authentication is problematic.

Another issue with w3af is that many of the tests it runs are active. This means that w3af will actually try to exploit the target web application. This can cause a host of problems in live application environments including cluttering up log files and databases as well as potentially inadvertently triggering alerts, sending e-mails, or inject XSS code. Great care should be given when choosing which modules in w3af to run in a production environment.

Running w3af

Running w3af against the target can take some time, so we will first and foremost want to limit our scans to test for only certain vulnerabilities. Also, as you may have already noticed, there are portions of the target web site that are only accessible if you log in. Creating an account is simple but w3af will not be able to complete this process so it is necessary to do so manually. To get started pull up the web site in a browser and click on the newletter link. This will show you the "You must be logged in" message and a link to sign up for an account (at /signup). Create a new account with whatever credentials you like.

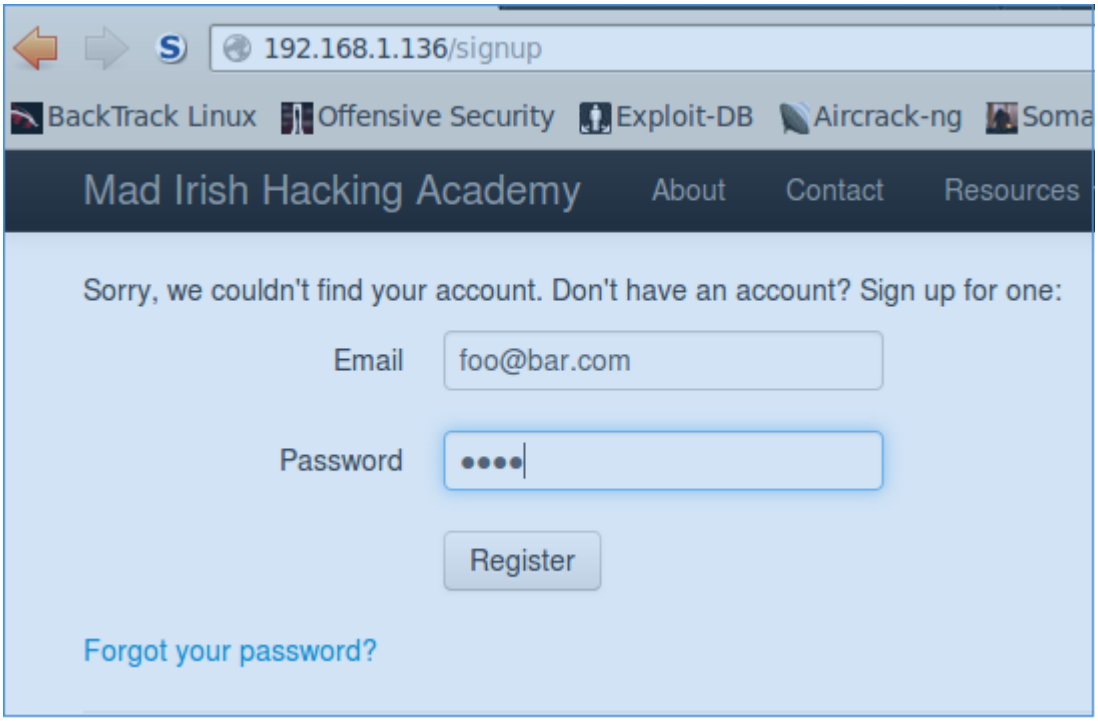


Illustration 9: Signing up for a new account to get an authentication token

You'll notice after this step that you're authenticated. Like most web applications, the target uses sessions managed by the application server to track things like who is logged in. This is generally done using cookies. In worst cases, variables are actually stored in cookies, but more often only a "token" is stored in a cookie that is used to match requests with known users.

Once authenticated, we need to copy the cookie that is used to track our session so we can share that with w3af for more complete testing. To do this right click on the page in Firefox and choose 'View Page Info' then click the 'Security' tab and finally the 'View Cookies' button.

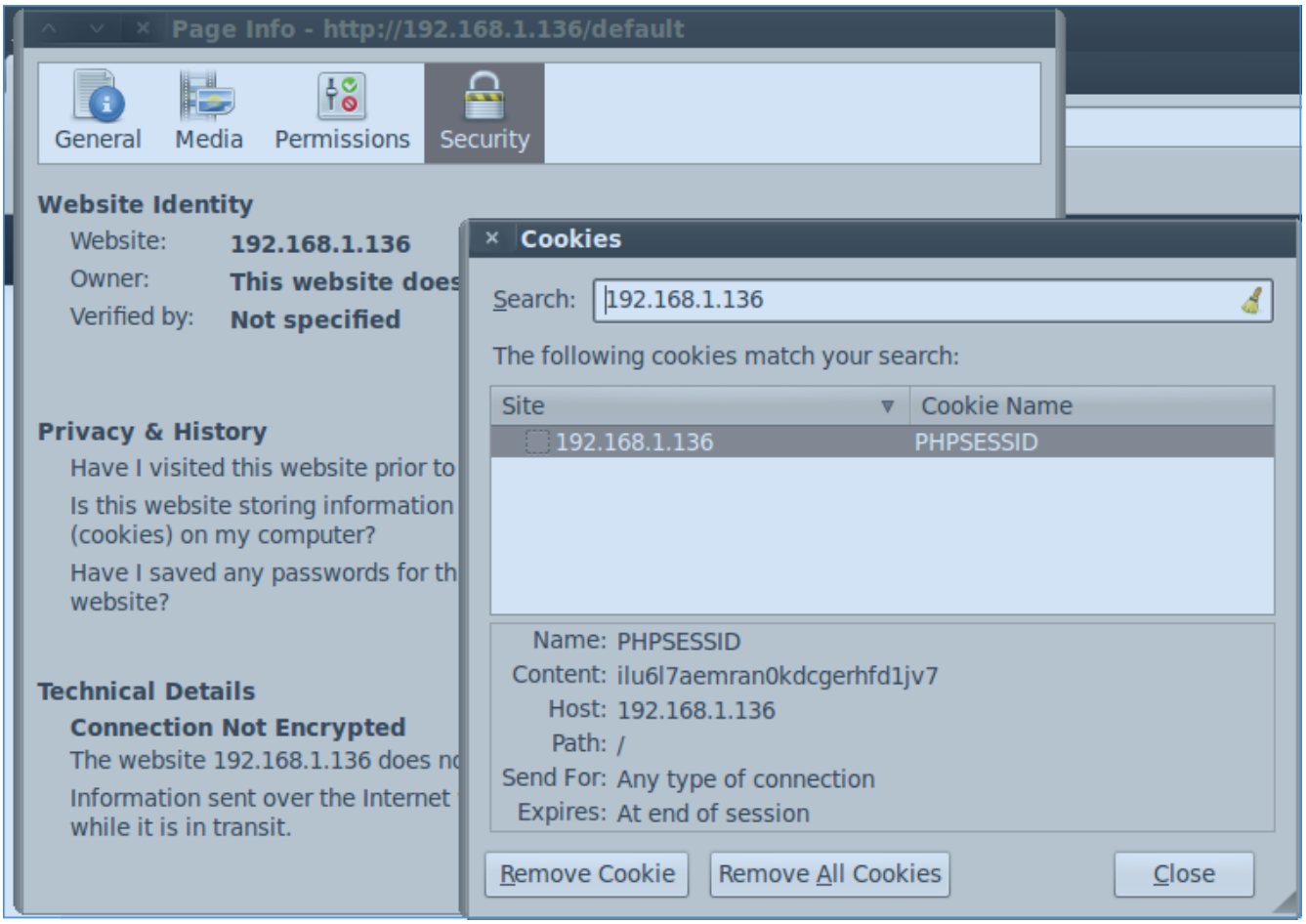


Illustration 10: Discovering the PHPSESSID token value

Next copy out the 'Content' value (in this case "ilu6l7aemr…"). We need to use this data to create a new cookie.txt file on the Desktop for w3af to use. Go ahead and open up a text editor (Applications → Accessories → gEdit Text Editor) and copy and paste the following text into the file, substituting the URL and cookie value you got when you logged in:

```
# Netscape HTTP Cookie File
.192.168.1.136   TRUE  /     FALSE 0     PHPSESSID  ilu6l7aemran0kdcgerhfd1jv7
```

Note that there are tab spaces between the values and there is only **one** line break (after the word "File"). W3af is very temperamental about the format of cookie files and any mistake will cause errors. In this respect, w3af is not unique. Many testing tools have great difficulty in utilizing custom cookies when making requests.

Once you have a text file with the cookie, start the w3af GUI from Applications → BackTrack → Vulnerability Assessment → Web Application Assessment → Web Vulnerability Scanners → w3af gui. When w3af starts you're presented with a list of plugins to use. To start with let's select a very minimal set of tests, in order to keep the scan time to a minimum. From the 'audit' menu select 'localFileInclude', 'sqli', and 'xss' options and from 'discovery' choose the 'web spider' option. Next set up the cookie file. Select from the Configuration menu Configuration → HTTP Config, then click the 'Cookies' tab. Next fill in the path to our cookie.txt file (/root/Desktop/cookie.txt).

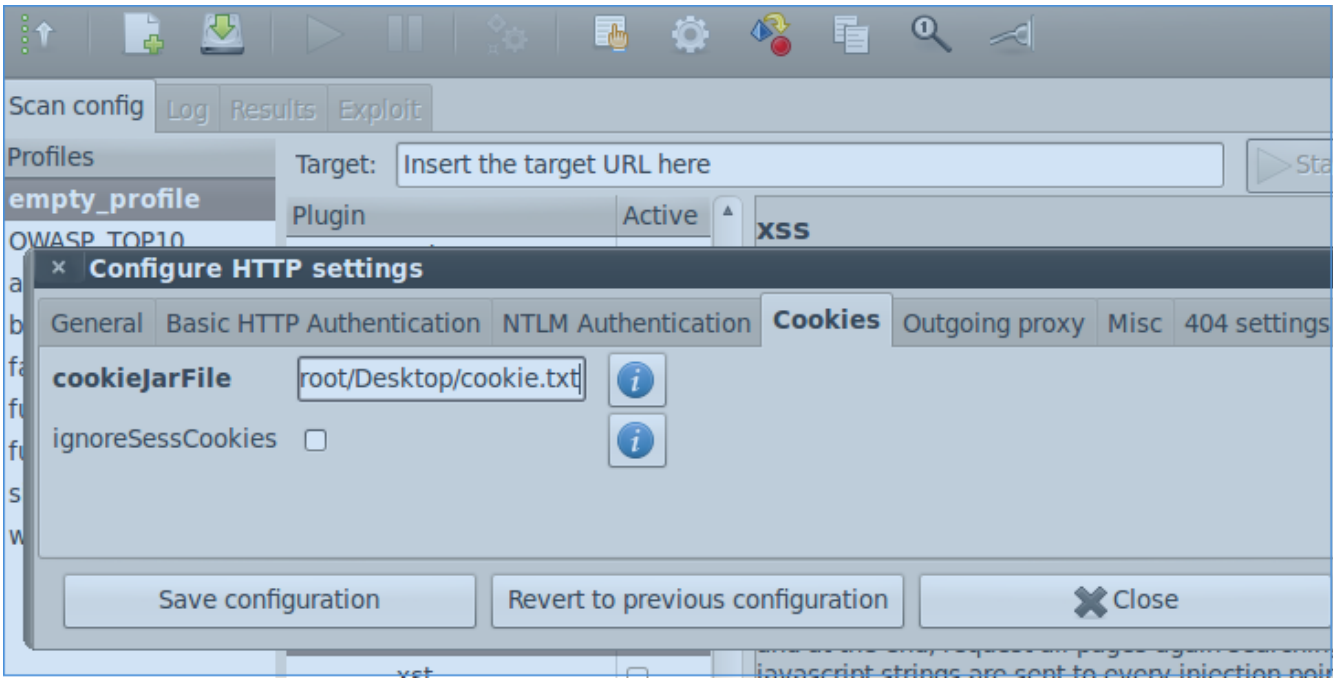


Illustration 11: Setting a cookie store in w3af

Click 'Save configuration' and if you don't get any errors then everything is set. Once the cookie file is saved, fill in the 'Target' and click the start button.

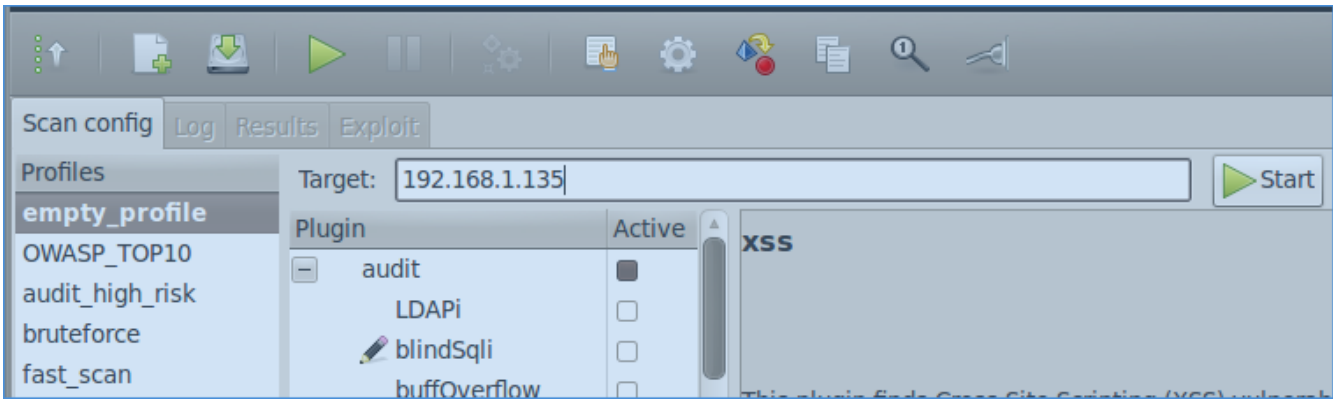


Illustration 12: Specifying the w3af target in the GUI

Once the scan starts w3af will switch to the 'Log' tab. Scanning can take some time, so remain patient while the results begin to populate. You can click the 'Results' tab at any time to view the vulnerabilities identified by w3af during the scan, but clicking around in the GUI may cause some instability in the tool. Once complete the 'Results' tab will contain a number of extremely interesting results. Clicking on the 'Response' tab will generally be more informative since it will display the data returned from the server (whereas the 'Request' tab shows the data submitted by w3af).

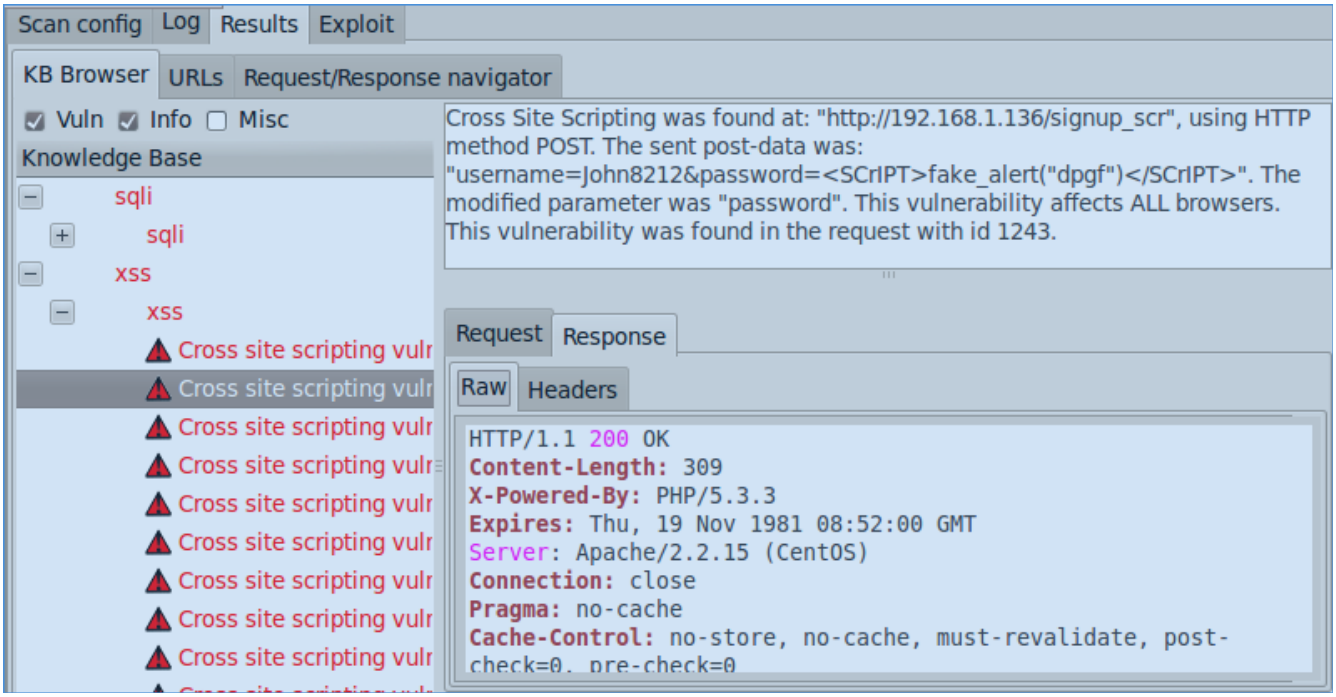


Illustration 13: w3af scan results

You will note that in the 'Results' tab you can even view a site map of the target under the URL's tab. This can be handy in reviewing coverage and for checking to ensure that w3af successfully accessed portions of the target that require an authentication cookie.

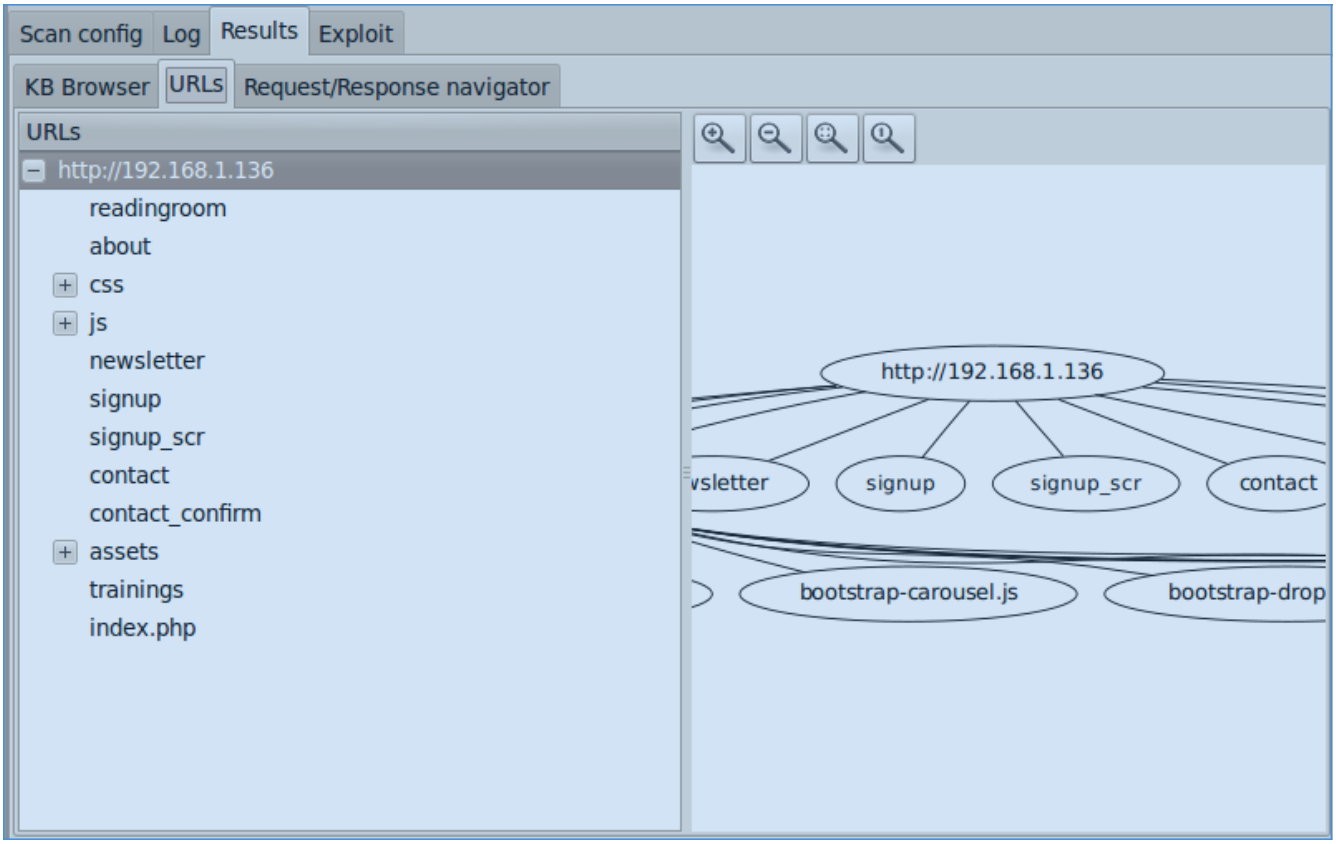


Illustration 14: w3af sitemap

Again, cookie support is somewhat troublesome and you may notice that w3af hasn't found some obvious SQL injection vulnerabilities behind the authenticated portions of the site.

ZAP

The OWASP Zed Attack Proxy is a fork of the popular, and venerable, testing proxy Paros. ZAP is written in Java so it should run pretty much anywhere. Although ZAP is primarily a proxy, it also has some extremely useful automated tools that can be extremely helpful.

ZAP's Strengths

Because ZAP runs as a proxy, it is designed to enhance manual testing rather than replace it. This means that ZAP quietly listens and records as the user manually browses a site, but allows a tester to do easy things like authenticate to the application and store session cookies as well as manipulate requests and examine responses in minute detail. ZAP draws from the well respected and solid Paros proxy but the new features are welcome additions. Unlike its best known commercial rival, Burp Suite, ZAP is open source and completely free. ZAP can be used by a skilled tester to extremely accurately identify, and exploit application vulnerabilities in intelligent ways unrivaled by the fully automated tools.

ZAP's Weaknesses

ZAP is a proxy, which means it is designed to be run by a human tester. This means that ZAP is not quite as useful in a scripted scenario. Although ZAP has automation features, these must be triggered via the GUI so using ZAP in a headless mode is not an option at this time.

Nota Bene

ZAP includes an 'Active Scan' feature that is extremely powerful. Be aware that this functionality will also do extremely effective fuzzing of form and other variable inputs. This can expose things like SQL injection vulnerabilities, but it could also exploit such vulnerabilities. In testing ZAP's active scan against earlier versions of the target the tool reset every password in the user table by triggering a SQL injection flaw. This could have been disastrous in a production environment, so be extremely careful when using ZAP for testing.

Running ZAP

To run ZAP simply start it from the Applications → BackTrack → Vulnerability Assessment → Web Application Assessment → Web Vulnerability Scanners → owasp-zap menu. This will start up the ZAP gui. If prompted you can generate a CA Certificate but it isn't necessary for this exercise.

By default ZAP listens on port 8080, so the first thing we need to do is adjust our Proxy settings in Firefox, found under Edit → Preferences → Advanced → Network Tab → Settings. Once the Connection Settings dialog opens, click the radio button next to 'Manual' and set the port to 8080.

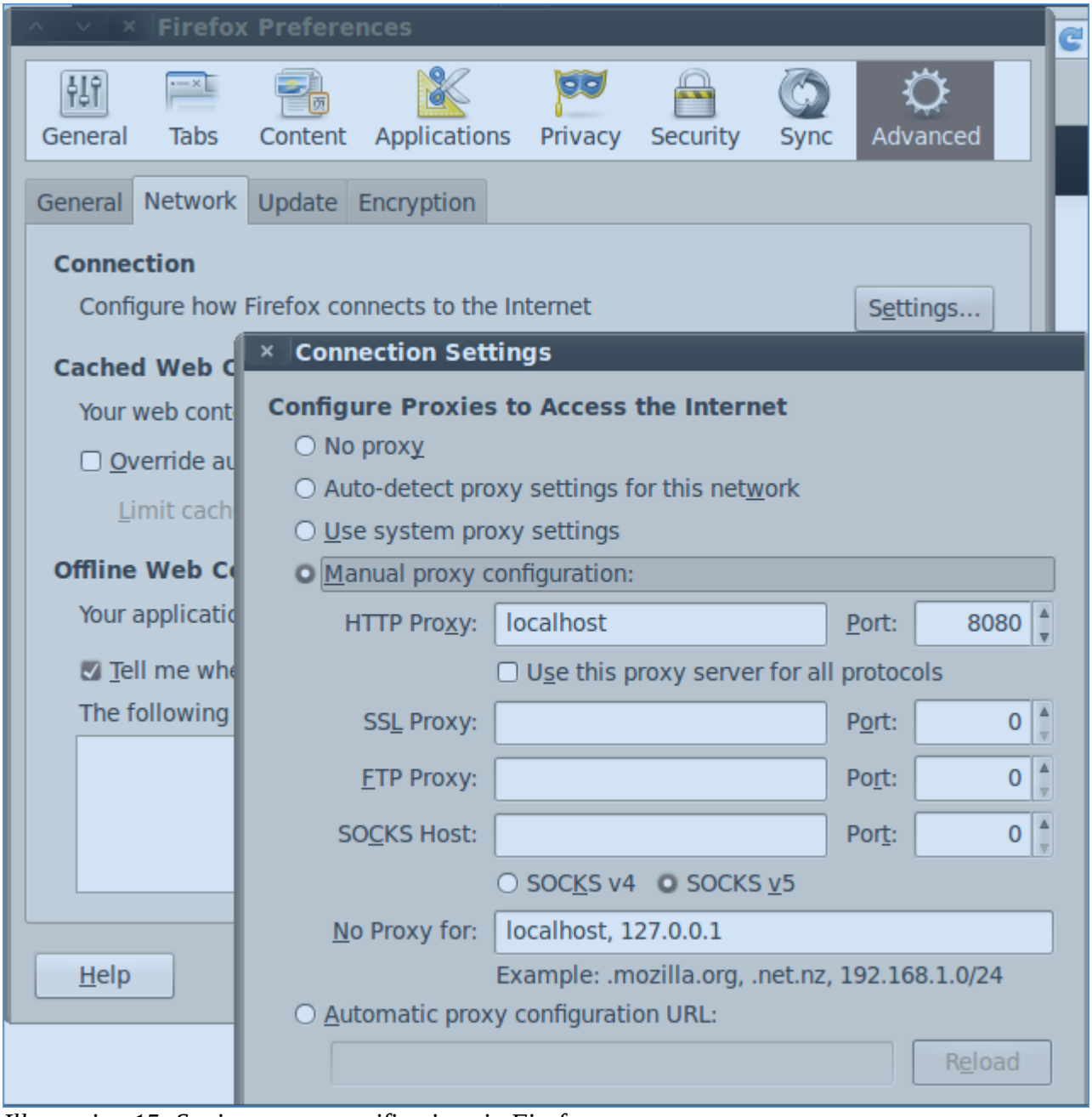


Illustration 15: Setting proxy specifications in Firefox

Click “OK” then close the settings. Hit refresh in your browser and look at the ZAP interface, which should now be recording your session.

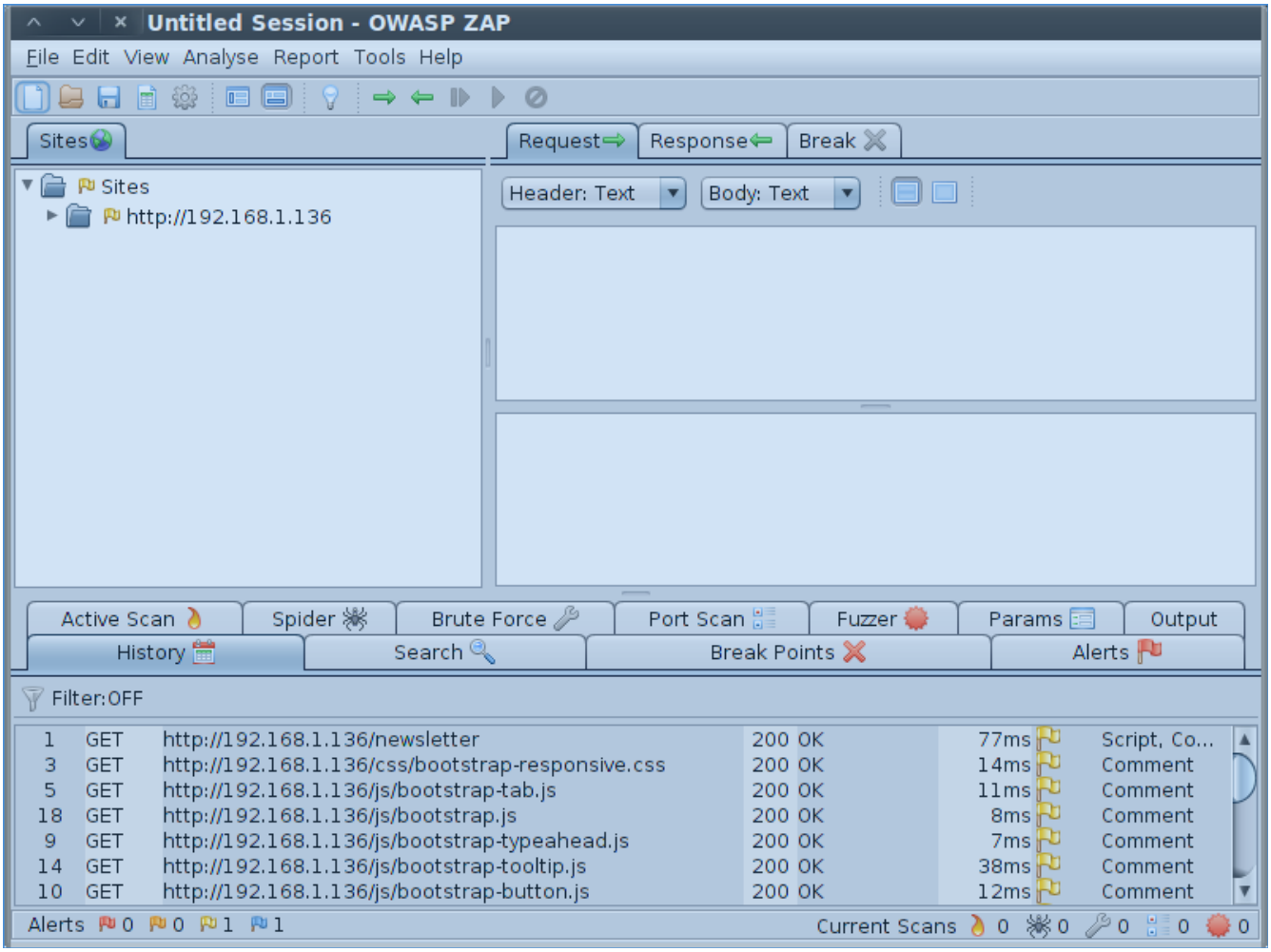


Illustration 16: ZAP records browsing

The first thing to test is the 'Spider' functionality. To do this click the 'Spider' tab and select the appropriate site from the drop down then click the 'Play' button. This will populate the 'Sites' folder in the upper left pane of the ZAP interface. Of particular note is the fact that the spider found the /newsletter&id=1 URL, which is access restricted, meaning ZAP has properly utilized our session token for the spider. This is one of the primary conveniences of such a proxy, it can listen to a session guided by a human tester, then use cookies and other session variables initiated by the human tester to perform further automated tests.

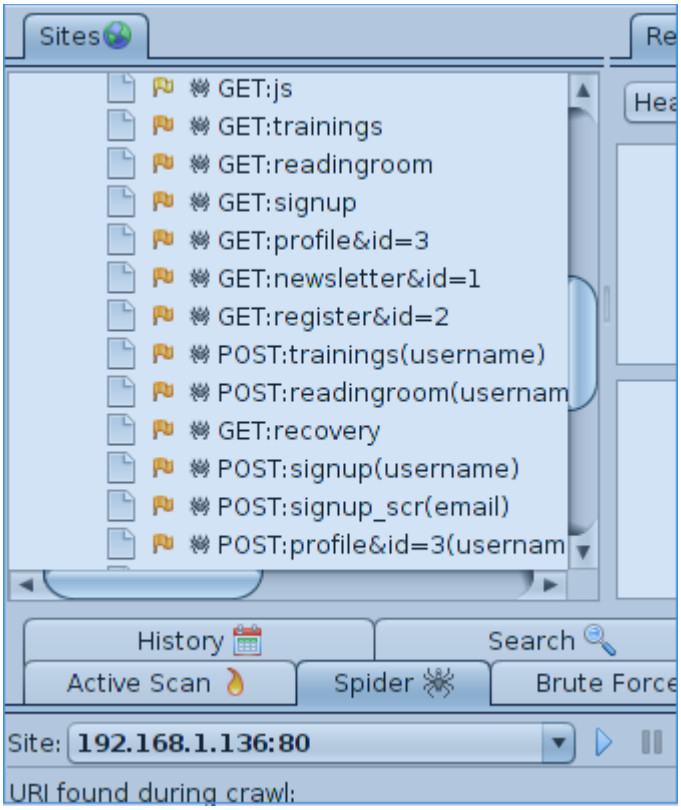


Illustration 17: ZAP includes a powerful web spider

To test out the automated vulnerability discovery capabilities of ZAP click on the 'Active Scan' tab (again, be aware this could cause problems in a production environment). Next select the appropriate site from the drop down and click the start button. As the scan runs, alerts will show up in the bottom left hand corner of the ZAP interface as counts next to little flags.

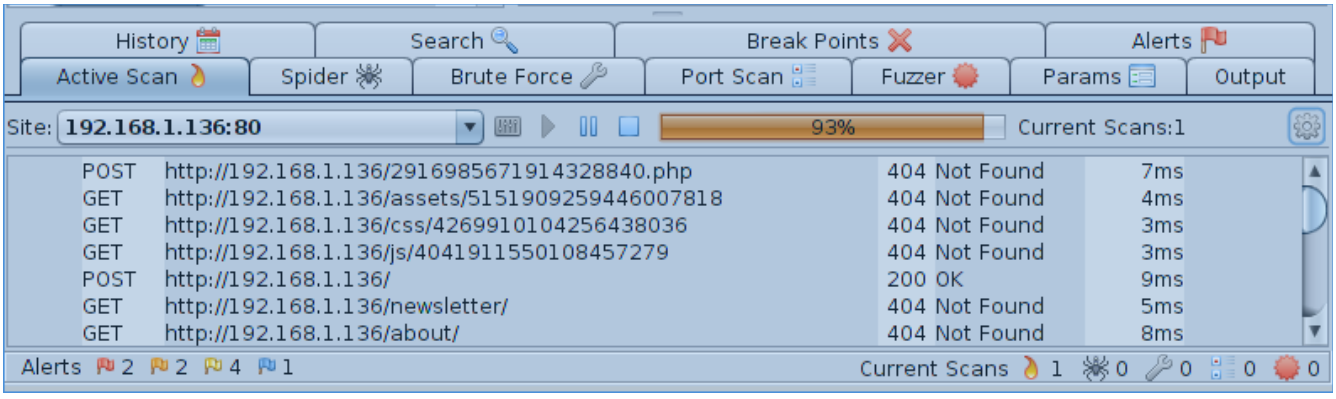


Illustration 18: ZAP's active scan capabilities find vulnerabilities quickly

If you click on the 'Alerts' tab you can see that ZAP collects quite a few vulnerabilities very quickly. These can serve as great starting points for exploiting the target application.

Step 3 – Exploitation

Once vulnerabilities have been identified it's time to exploit them to gain access to the application.

Exploitation

In earlier testing you should have noticed a number of SQL injection vulnerabilities in the application. SQL injection vulnerabilities are some of the most devastating vulnerabilities because they can expose sensitive data such as user credentials, payment details, or sensitive customer information. Exploiting SQL injection takes some skill, however, since SQL is superficially a fairly straightforward language, but with PHP/MySQL applications where query stacking isn't supported, crafting injection strings takes knowledge of more obscure SQL functionality such as sub-queries, UNIONs, JOINs, and other features. Automated tools can make this somewhat easier, but often times are of limited usefulness.

One extremely popular SQL injection utility is SQLMap (http://sqlmap.org/). SQLMap is written in Python so is fairly portable. SQLMap includes a huge array of functions for exploitation and can work with a wide variety of databases. Like the other tools used in this example, SQLMap supports cookies so we can use it to test vulnerabilities behind authentication.

One vulnerability we've already identified is the /newsletter&id=1 URL that displays one of the newsletters. We can easily see that it's exploitable if we change the value of the parameters to "&id='1" which will cause a full MySQL error to display. This functionality is common in many web applications (the display of full database errors) but is extremely problematic. Although this output is helpful to a developer, it is meaningless to an end user, and worse, it gives valuable information to attackers.

Let's point SQLMap at this URL and see if we can retrieve any valuable information. We have already fingerprinted the database as MySQL so we can tune SQLMap to run better by pinning the target database type. Open up SQLMap from Applications → BackTrack → Exploitation Tools → Database Exploitation Tools → MySQL Exploitation Tools → sqlmap. This will open a command prompt at the installation root for SQLMap and print out the usage details. To attack the target with SQLMap type (all on one line):

```
# ./sqlmap.py -u "http://192.168.1.136/newsletter&id=1"
--cookie="PHPSESSID=ilu6l7aemran0kdcgerhfd1jv7" --dbms="MySQL" -v 1 --dbs
```

Be sure to substitute the proper target URL and authenticated cookie. What we're telling SQLMap to do is to test the URL using a cookie and attempting to list all the databases in the MySQL back end with verbosity level 1 (from 1-6). SQLMap will complain that no parameters were provided and when it asks "do you want to try URI injections in the target url itself?" respond "y." You should see that SQLMap properly identifies all the databases in the target:

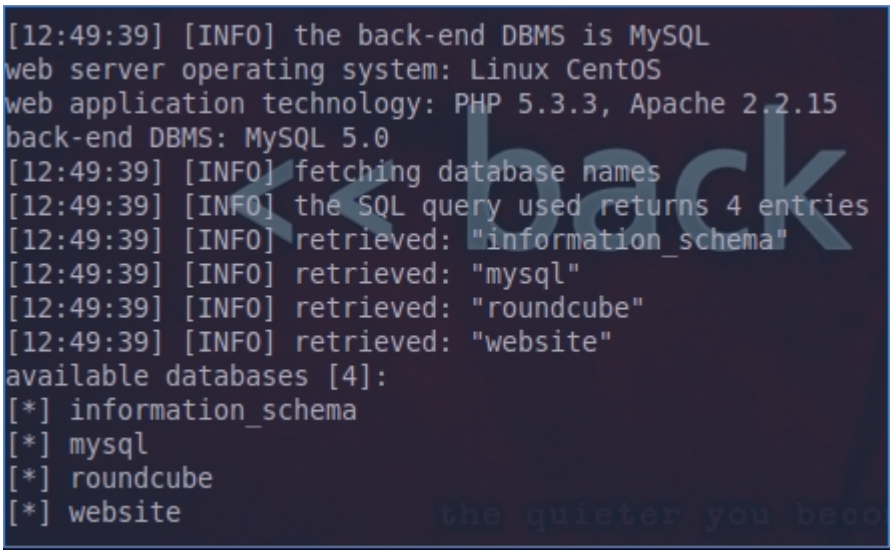


Illustration 19: SQLMap identifying tables

Next we can attempt to have SQLMap dump all of the data in the tables it can find. Again, because of PHP/MySQL's lack of query stacking this can take some time. To do this simply change the "--dbs" flag in the previous command (which lists the databases) to "--dump" to dump all data. Answering "yes" or suggested values to the resulting questions should be sufficient:

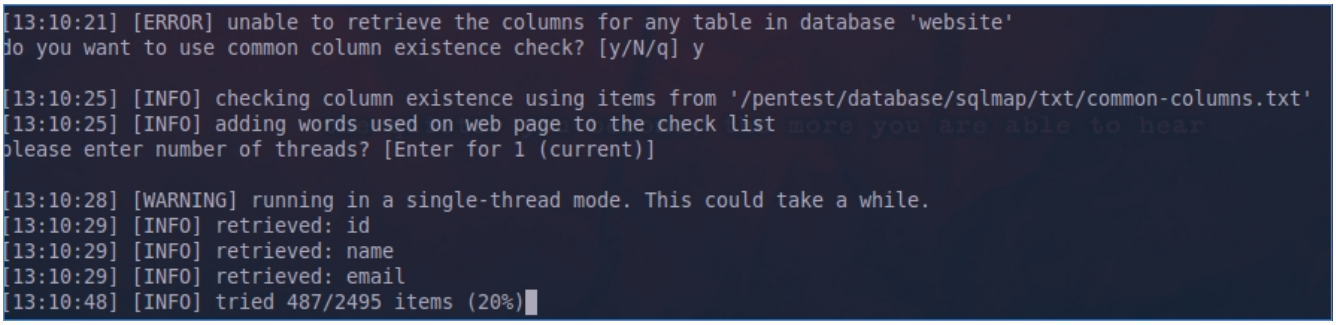


Illustration 20: SQLMap dumping data

Note that SQLMap must resort to brute force to figure out the contents of the database. Again, this is a result of the limitations of SQL injection with PHP/MySQL. This is because although a parameter might be injectable, the base of the SQL query cannot be changed, thus the statement:

```
SELECT * FROM table WHERE id=x
```

Might be injectable at the position "X", the prefix of the query (that is the part that reads "SELECT * FROM table WHERE id"), cannot be changed. The query must always begin with the prefix. This means we can't change the query to read "DESC table" with any amount of SQL back flips.

SQLMap is notoriously bad at performing injection against queries that use parenthesis surrounding the injectable parameter. For instance, the query:

```
SELECT user_id FROM users WHERE username = $_POST['username'] AND
password = MD5($_POST['password']);
```

is trivial for a human to bypass using SQL injection, but SQLMap will fail to exploit this vulnerability. This is a great example of the fact that despite SQLMap's power, it only functions optimally in a restricted set of circumstances. This is why most scans using the tool will only look for classic injection strings (such as ?id=X in a URL string where X is numeric).

Despite these limitations. SQLMap will still do a reasonable job of figuring out the contents of tables using a dictionary list of potential column names (but it won't, for instance, find the 'dtstamp' column in the “contact” table).

Note that SQLMap will store the results from any table it dumps to a text file. Sometimes you might want to start a new SQLMap session and rid yourself of previous data. To do this use the command:

```
# ./sqlmap.py –purge-output
```

To review any of the data stored by SQLMap you can look in /pentest/database/sqlmap/output in any of the subdirectories. Note that SQLMap did an excellent job of identifying the contents of the log table, which seem to include a plain text password:

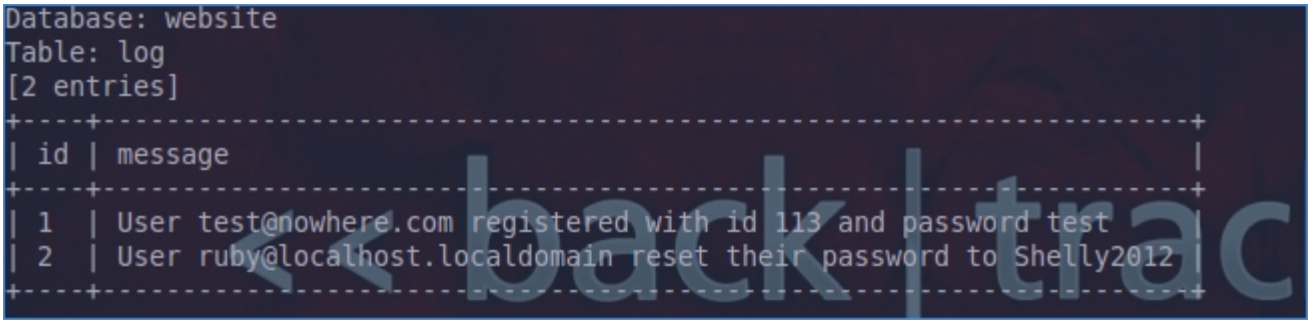


Illustration 21: SQLMap finds application log data

This is a perfect example of a non-technical vulnerability (i.e. a business logic flaw). Designed as an internal auditing function in the application (probably to help user support, for instance being able to look up a password immediately after a user changes it to help them reset their passwords) it's a security problem. Even though the application might store user passwords as hashes in the users table, the fact that the “log” table stores any updated passwords as plain text values is a vulnerability.

Assuming the password for “Ruby” is correct, we can now test to see if password reuse is allowed. We know that the site has an administrative function on port 8080. Let's try to log into that site using the username “ruby@localhost.localdomain” and the password “Shelly2012”.

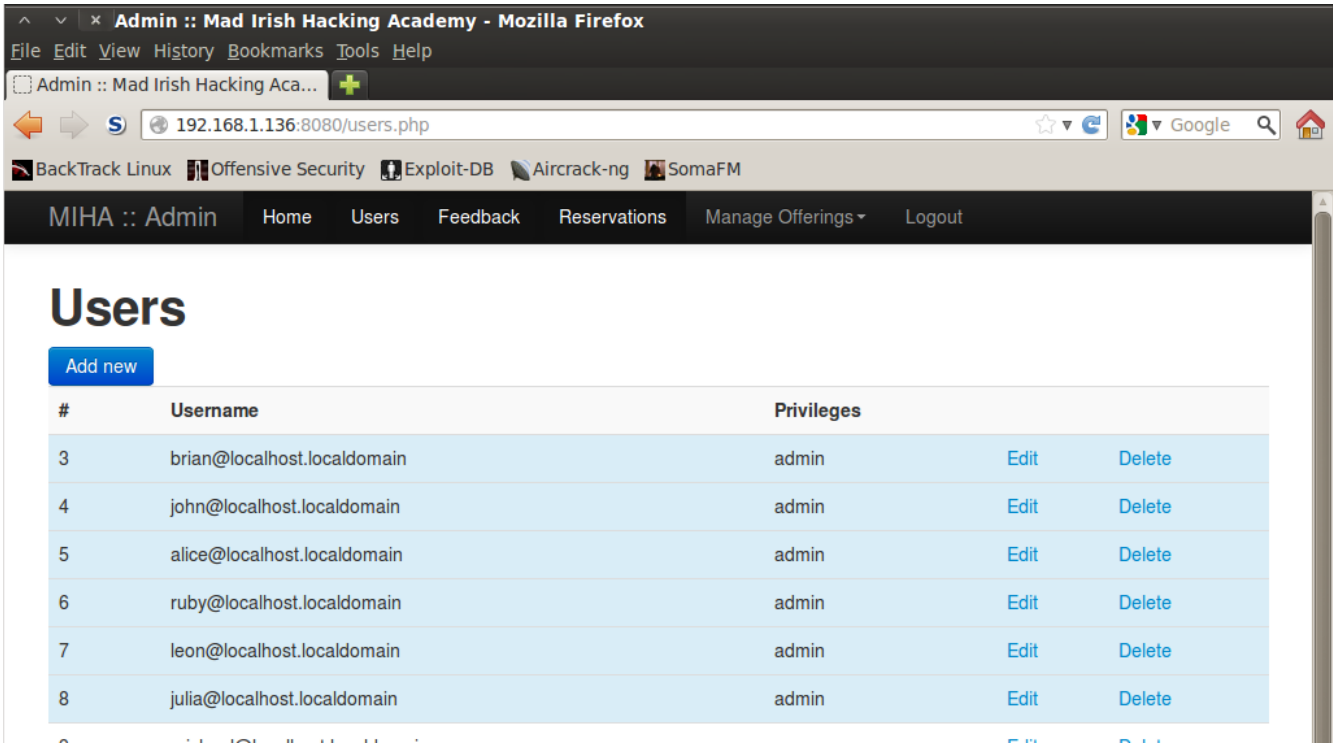


Illustration 22: Exposed passwords allow access to the admin area

It turns out this password is valid! Now that we have access to the admin portion of the application we can test it using w3af, SQLMap, or ZAP.

Step 4 – Cracking Passwords

Now that we have a foothold in the system it's time to attempt to escalate privilege.

Password Problems

Password reuse is a widespread problem that plagues many sites and servers on the internet. Because good passwords are hard to remember, users commonly have the same password on many different services. Out of curiosity, let's see if the “ruby” username and password will let us in via SSH:

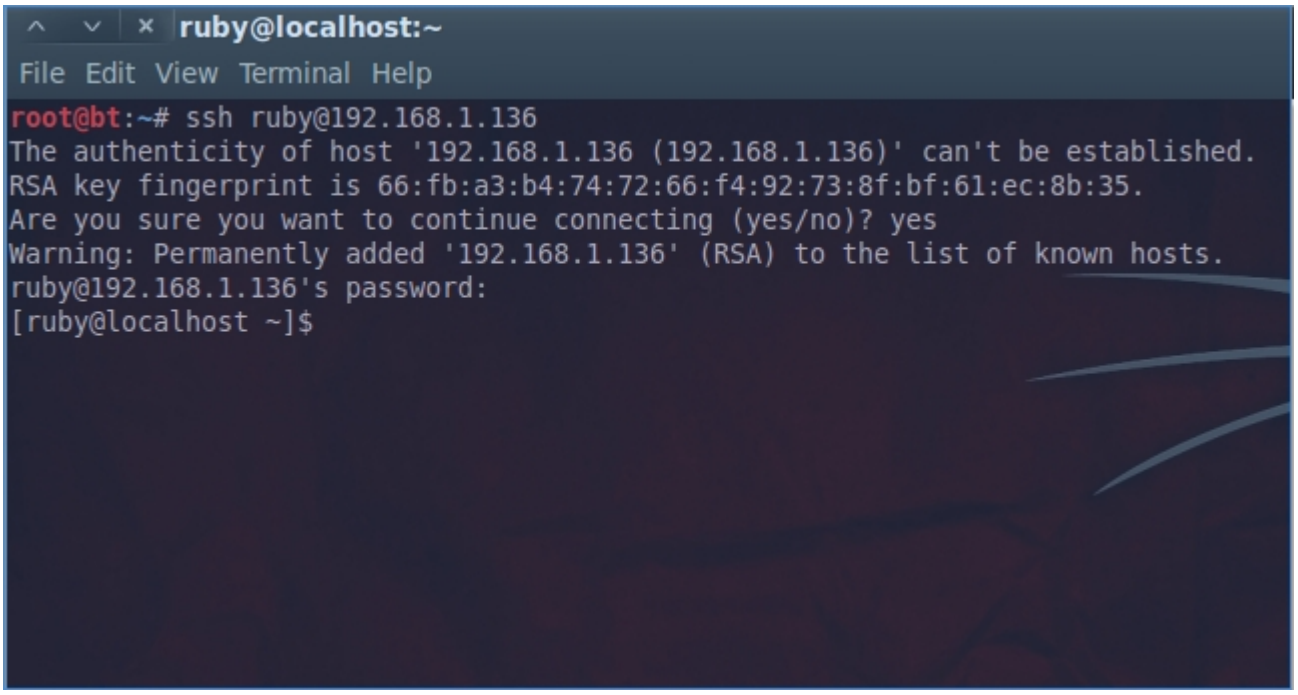


Illustration 23: Password re-use allows us to SSH to the target

And it turns out that we can! Now we have a foothold on the target server. The next step is to try and get access to the root account. We can do this in a number of ways. The simplest way is to check and see if we can use “sudo” to gain access to the root account by typing:

```
$ sudo su
```

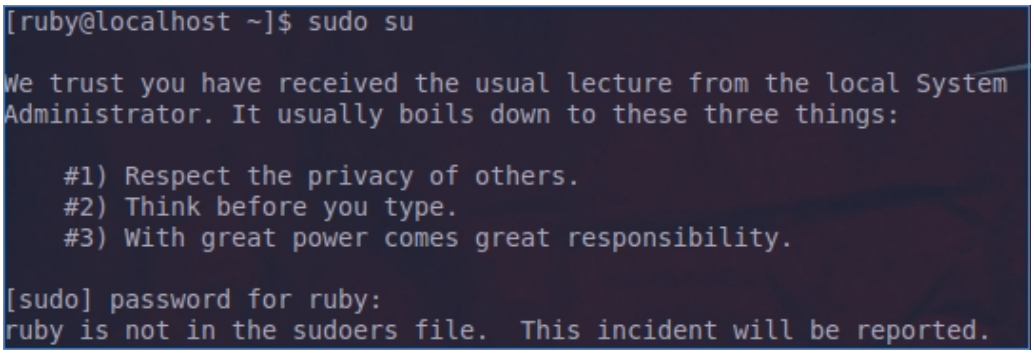


Illustration 24: This account can't sudo to root

It turns out that the “ruby” account doesn't have access to the “sudo” command. Another way we can try to do this is to crack other local user account passwords and see if they have sudo access. This is difficult since the system stores passwords in a shadow file that is only readable by the root account. There might be other “password” files on the system, however. Even though SQLMap wasn't able to get a list of passwords from the users table in the website database, we may be able to access those details now that we have an account on the system if we can find credentials to log into the MySQL database directly from the command line.

One way to look for MySQL credentials is to search for them in the PHP files that power the web application. Because the application must have access to the credentials, these details will be stored somewhere in plain text on the filesystem. Even though this data may not be visible via the web, it is likely available to a user with filesystem access. Do a directory listing on the default web root using:

```
$ ls /var/www/html
```

using the “ruby” account and see if you can spot a file that might contain this data. You can use the “cat” command to concatenate the contents of files out onto the screen.

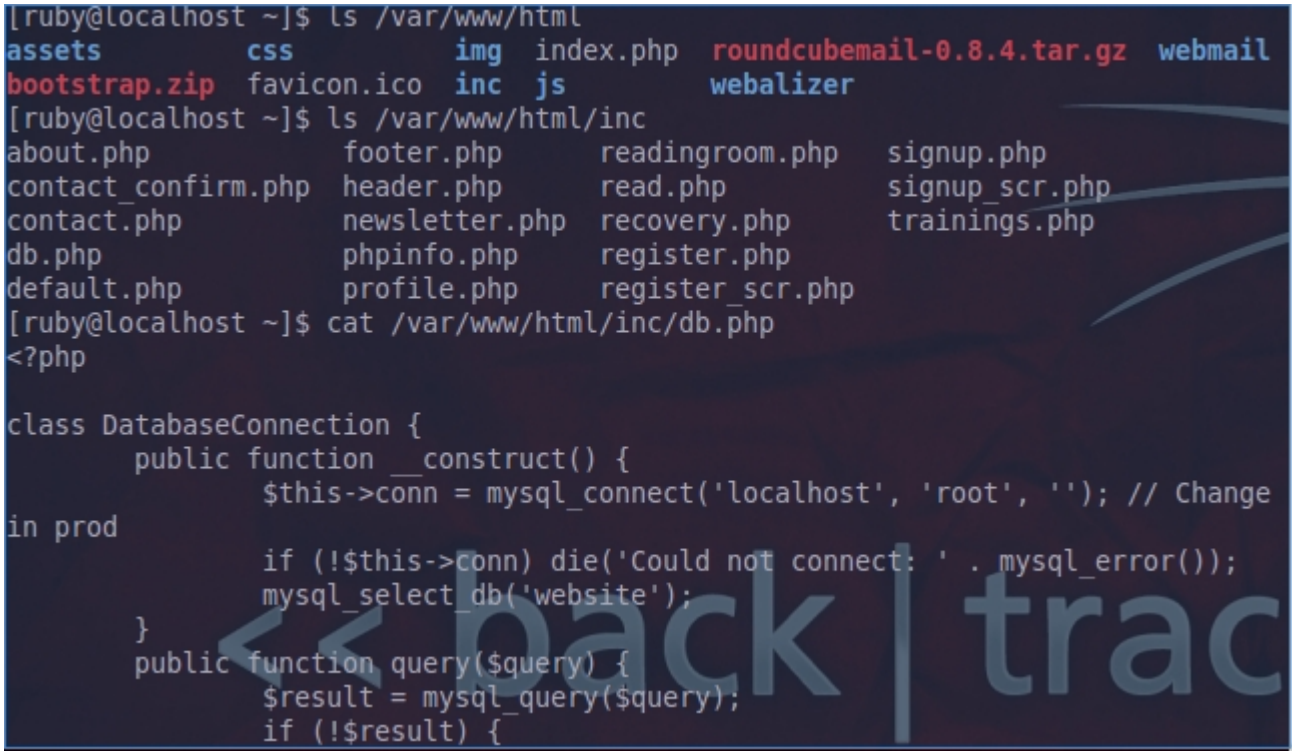


Illustration 25: Finding MySQL credentials on the filesystem

Looking at the results illuminates two disastrous configuration errors. The first is that the web application is running as the MySQL “root” user, an account with unlimited power in the database. Secondly, the system seems to be using a blank password for this account.

We can test these credentials using the MySQL command line utility like so:

```
$ mysql -u root website
```

This command will attempt to access the "website" database in the local MySQL server as the root user without any password.

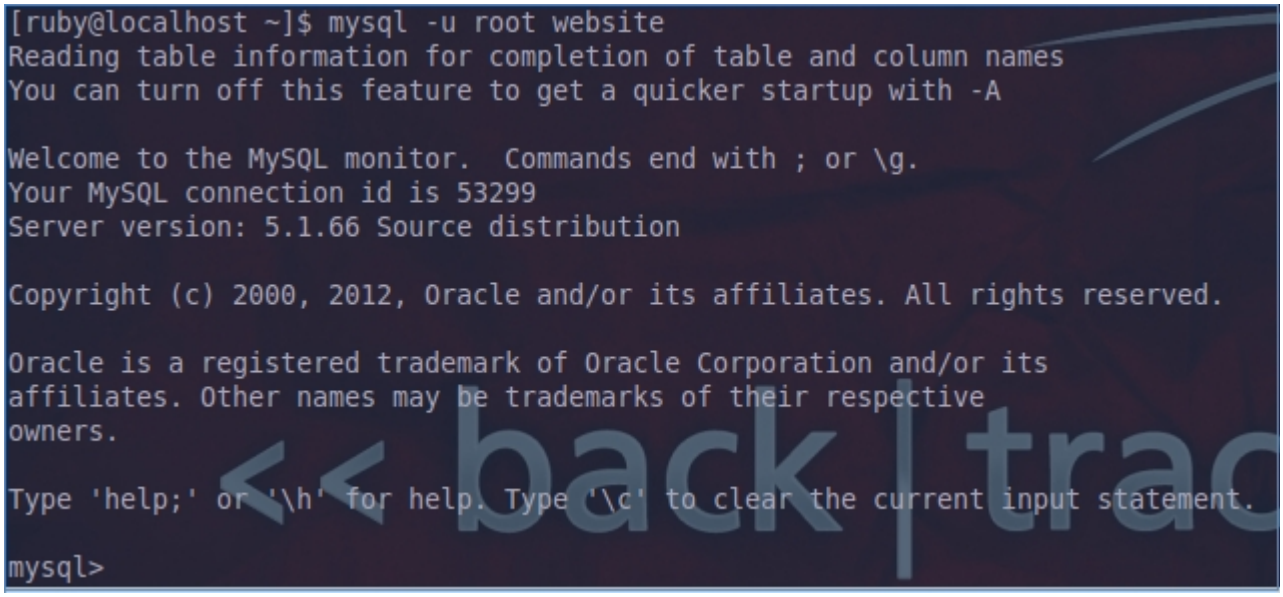
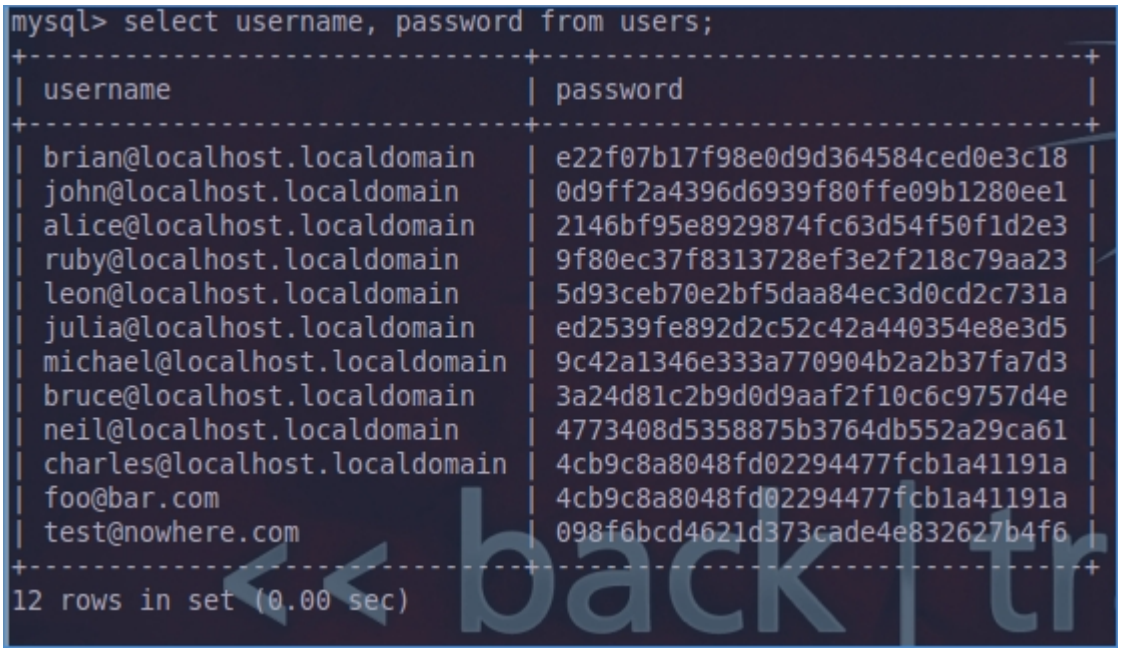


Illustration 26: Accessing the MySQL database

It worked! Now that we have access to the database we can dump out the passwords with the command:

```
mysql> select username, password from users;
```



```
mysql> select username, password from users;
+-------------------------------+----------------------------------+
| username                      | password                         |
+-------------------------------+----------------------------------+
| brian@localhost.localdomain   | e22f07b17f98e0d9d364584ced0e3c18 |
| john@localhost.localdomain    | 0d9ff2a4396d6939f80ffe09b1280ee1 |
| alice@localhost.localdomain   | 2146bf95e8929874fc63d54f50f1d2e3 |
| ruby@localhost.localdomain    | 9f80ec37f8313728ef3e2f218c79aa23 |
| leon@localhost.localdomain    | 5d93ceb70e2bf5daa84ec3d0cd2c731a |
| julia@localhost.localdomain   | ed2539fe892d2c52c42a440354e8e3d5 |
| michael@localhost.localdomain | 9c42a1346e333a770904b2a2b37fa7d3 |
| bruce@localhost.localdomain   | 3a24d81c2b9d0d9aaf2f10c6c9757d4e |
| neil@localhost.localdomain    | 4773408d5358875b3764db552a29ca61 |
| charles@localhost.localdomain | 4cb9c8a8048fd02294477fcb1a41191a |
| foo@bar.com                   | 4cb9c8a8048fd02294477fcb1a41191a |
| test@nowhere.com              | 098f6bcd4621d373cade4e832627b4f6 |
+-------------------------------+----------------------------------+
12 rows in set (0.00 sec)
```

Illustration 27: Exposing password hashes in the users table

Of course, these are the password hashes, that we'll have to crack. You can attempt this using Google by simply searching for the hash, or by using a password cracker like John the Ripper (http://www.openwall.com/john/).
John the Ripper is installed on BackTrack, but may take a while to run due to the CPU intensive nature of doing hashes to look up values. For this reason it's best to install John on a physical computer with a fairly beefy processor.

To begin working with John, first copy and paste the user table from the MySQL window into a text file using gEdit. Next, use the Search → Replace menu to replace all pipes with no character.

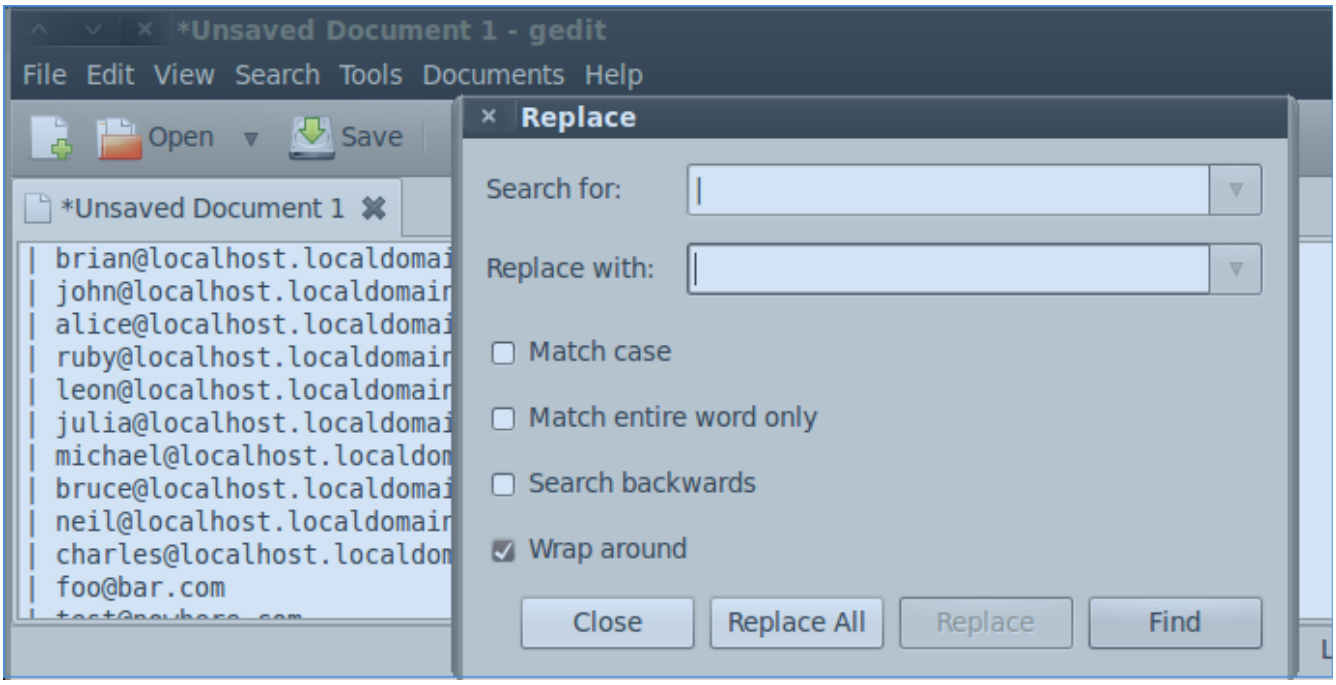


Illustration 28: Formatting the password file for John the Ripper

This will allow John to recognize the contents of the file more easily. Next you'll have to place a colon (:) between the username and the password, and ensure all spaces are removed from the file. Then save this file as "passhashes.txt" in the /root directory.

Next start John from Applications → BackTrack → Privilege Escalation → Password Attacks → Offline Attacks → john the ripper. At the command line type in:

```
# ./john -format=raw-md5 /root/passhashes.txt
```

If you get an error "No password hashes loaded (see FAQ)" it means the format of your file is incorrect, and likely you have extra spaces at the start and/or end of the lines.

You'll see that John spits out a few matches right away, but it will take a lot longer to crack the more difficult passwords:

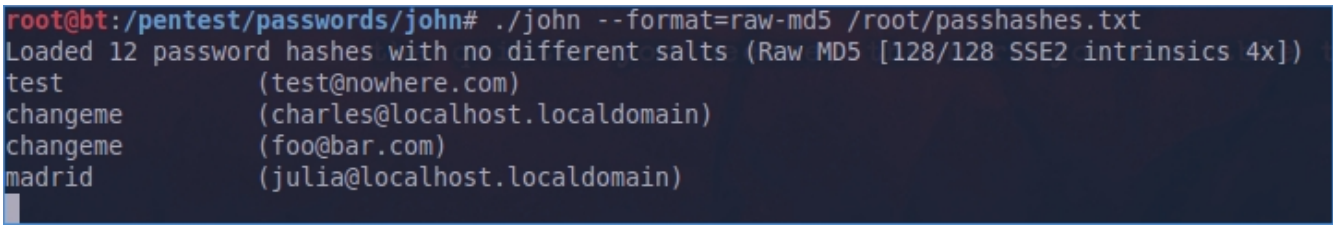


Illustration 29: John cracking password hashes

Note however that we only need one weak password on an account with superuser privileges to gain control of the host. Let's try switching to the 'julia' account using the su command, like so, from our terminal where we have SSH'ed to the host as 'ruby':

```
$ su julia
```

And then enter julia's password (madrid) when prompted. Next, try to switch users (su) to the root account (with the “sudo su” command) and re-enter Julia's password when prompted.

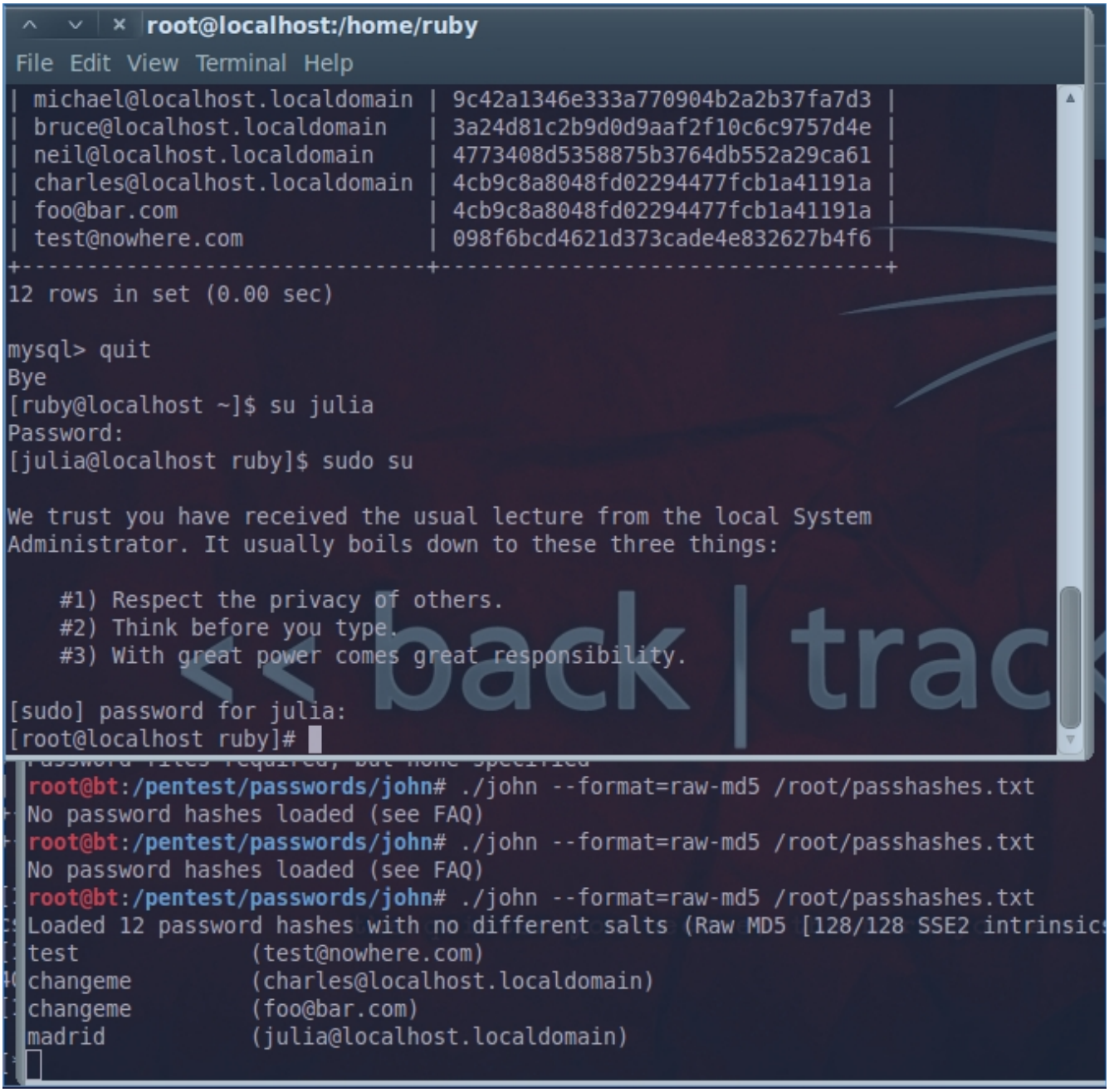


Illustration 30: Gaining root via the julia account

Viola, we have the root account! Now we can grab a copy of the /etc/shadow file and run it through John the Ripper and hopefully crack the root password.

Appendix I – Conclusions

This target virtual machine suffers from a number of problems. As demonstrated in this exercise, the most damaging issue is the fact that the system's users re-use passwords on the web interface and the shell. This means that any password exposure will lead to a local shell account. Enforcing a password policy is difficult, especially when users are allowed to create their own passwords. Even if the application had a check to ensure that the application password was different from the system password it would be prone to issues (for instance, a compromised web account could be used to determine the shell password by attempting password resets and seeing which ones failed).

The custom web applications running on port 80 and 8080 suffer from a number of problems, but the fact that they use a few tricks in presentation (such as requiring login to view certain pages and Apache redirects for all requests) greatly reduces the effectiveness of automated tools. Although the automated tools quickly find several flaws, they do not seem to be able to identify many of the most damaging (such as the fact that login forms are vulnerable to SQL injection that allow for authentication bypass). This observation is critical when evaluating the use of any automated solution.

The widespread issue of greatest impact affecting the custom application code is SQL injection vulnerabilities. These can lead to information disclosure and file system access for attackers. Every SQL query should run through some sort of input sanitizing routine. The most practical approach would be to use prepared statements with the MySQLi interface. Furthermore, each application should be run with it's own non-root account with privileges limited only to those necessary to carry out the function of the application. This would prevent unauthorized changes to the database beyond the scope of the application.

Following SQL injection in terms of quantity, the applications contain a number of XSS and XSRF vulnerabilities. These are somewhat more difficult to demonstrate in a capture the flag exercise since they involve interaction from legitimate application users, and therefore were not highlighted in this document.

There are also a number of local file include vulnerabilities that could expose raw PHP code and other sensitive files on the filesystem. All user input that is utilized to open local system files should be checked against a whitelist of available resources and file access should be limited to specific directories.

A number of resources on the target should be access restricted. This includes access to many of the services (such as SWAT and Webmin) as well as directories in the web root (such as webalizer). Limiting access to these resources using the built in iptables firewall is straightforward. At the very least .htaccess passwords should be in use to protect certain web based resources.

Appendix II – Vulnerabilities

The following is a non-comprehensive list of vulnerabilities that appear in the target, that can be used to gauge the effectiveness of automated testing.

Information Disclosure

- The /webalizer directory is under the web root, making it discoverable. The reports in this directory contain configuration details about the system.
- The 404 error page contains information about the system type (OS and HTTPD version)
- The MySQL errors contain details about the database as well as full queries.
- /phpinfo reveals the output of the phpinfo() command.
- The /profile page allows for account enumeration.

Local File Include

- The index page, called on every request, contains a local file include vulnerability in the URL "action" variable.
- The /read page contains a LFI since the URL variable "file" is not checked before it is included or scope limited to a downloads directory.

SQL Injection

- The login form contains a SQL injection as the "username" and "password" variables are not sanitized.
- The /newsletter page contains a SQL injection vulnerability as the "id" URL parameter is not sanitized
- The /profile page contains a SQL injection vulnerability as form inputs aren't sanitized.
- The /recovery page contains a SQL injection vulnerability in the "email" POST variable
- The /register_scr page contains multiple SQL injection vulnerabilities stemming from unsanitized POST variables.
- The /signup_scr page contains multiple SQL injection vulnerabilities.

XSS

- The /profile page contains multiple XSS vulnerabilities including a reflected XSS in the URL "id" parameter as well as stored XSS resulting from data in the database not being sanitized before display.
- The /register page contains a reflected XSS vulnerability from the URL "id" parameter

Logic Flaws

- The /recovery page allows any account to have its password reset to "changeme" without proper validation.

Appendix IV – Afterword

I would like to acknowledge the support of the University of Pennsylvania, School of Arts & Sciences computing for making my work on this exercise possible. This exercise was first delivered at the Philadelphia chapter meeting of the Open Web Application Security Project (OWASP). For more details about OWASP, and the Philadelphia chapter, see https://www.owasp.org/index.php/Philadelphia. I would also like to thank my Drexel co-op, Matt Bucci, for his help in testing the target and reviewing this documentation.

I welcome any feedback, questions, or comments. Feel free to e-mail me at justin@madirish.net.

I'd like to dedicate this work to the memory of my son, Tristan.