Top 10 Security Vulnerabilities

Susam Pal

Based on OWASP Top Ten Project (2006)
http://www.owasp.org/index.php/OWASP_Top_Ten_Project

Agenda

- Unvalidated Input
- Broken Access Control
- Broken Authentication
- Cross Site Scripting
- Buffer Overflows
- Injection Flaws
- Improper Error Handling
- Insecure Storage
- Denial of Service
- Insecure Configuration

Unvalidated Input

All input is evil ….

…. until proven otherwise.

Types of Input:

- HTTP Requests
- Commands
- Network packets (TCP/IP, UDP, etc.)

Unvalidated Input

Common Input Tampering Attacks:

- Forced Browsing
- Command Insertion
- Cross Site Scripting
- Buffer Overflows
- SQL Injection
- Hidden Field Manipulation

Unvalidated Input

Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability (BID 1806)

Web Root: C:\Inetpub\wwwroot\

Unsafe Procedure:

```
map [URL] to [path of the resource] on the hard disk;
if [path of the resource] starts with [C:\Inetpub\wwwroot]
then
        allow access;
else
        deny access;
```

Unvalidated Input

Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability (BID 1806)

Exploit:
http://www.example.com/../../windows/system32/cmd.exe

URL Mapped to Resource on Hard Disk:
C:\Inetpub\wwwroot\..\..\windows\system32\cmd.exe

What IIS/PWS forgot to check:
C:\Inetpub\wwwroot\..\..\windows\system32\cmd.exe
= C:\Inetpub\..\windows\system32\cmd.exe
= C:\windows\system32\cmd.exe

Unvalidated Input

Microsoft IIS and PWS Extended Unicode Directory Traversal Vulnerability (BID 1806)

Safe Procedure:

```
map [URL] to [path of the resource] on the hard disk;
canonicalize [path of the resource];
if [path of the resource] starts with [C:\Inetpub\wwwroot]
then
        allow access;
else
        deny access;
```

Reference:

http://www.securityfocus.com/bid/1806

Unvalidated Input

Prevention:

- Canonicalization: Input should be converted to the simplest form before they are validated.
- Server Side Validation: It is must. First code Server Side Validation, then code Client Side Validation.
- Validate against a positive (not negative) specification.
- Detailed code review for tainted parameters.

Broken Access Control

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others.

Features:

- Access control is closely associated with the content and functions of the application.
- Users are generally managed by putting them under various groups/roles with various priveleges.

Broken Access Control

Common Mistakes:

- Administrative interfaces that allow site administrators to manage a site over the Internet.
- Insertion of access control rules in various locations all over the code on an ad hoc basis.

Common Attacks:

- Exploiting Insecure Id's
- Forced Browsing Past Access Control Checks
- Path traversals
- Exploiting browser cache

Broken Access Control

How to disable Browser Cache:

HTML <meta> Tag

```
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Expires" content="-1">
<meta http-equiv="Cache-control" content="no-cache">
<meta http-equiv="Cache" content="no-cache">
<meta http-equiv="Expires" content="thu, 01 Jan 1980
   12:00:00 GMT">
```

ASP

```
<% Response.AddHeader "Pragma", "no-cache" %>
<% Response.Expires = -1 %>
<% Response.CacheControl="private" %>
<% Response.CacheControl = "no-cache" %>
<% Response.CacheControl = "no-store" %>
```

Broken Access Control

How to disable Browser Cache:

JSP

```
<% response.setHeader("Pragma","no-cache"); %>
<% response.setHeader("Cache-Control","no-cache"); %>
<% response.setDateHeader ("Expires", 0); %>
<% response.setHeader("Cache-Control","no-store"); %>
```

PHP

```
<?php
Header("Pragma: no-cache");
Header("Cache-control: private, no-cache, no-store");
Header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
?>
```

Broken Authentication

Types of Authentication:

- User ID and Password
- Software & Hardware based cryptographic tokens
- Biometrics

Session Management:

- Web applications must establish sessions to keep track of the stream of requests from each user.
- HTTP doesn’t provide this capability, so web applications must create it themselves.
- Generally developers create session tokens to manage sessions.

Broken Authentication

Common Attacks:

- “Walk by” Attacks
- Session Hijacking

Prevention:

- Re-authentication for critical functionalities.
- Use of SSL to protect authentication credentials and session identifiers.
- Code review and penetration testing
- POST method to submit authentication & session data.
- Use autocomplete=“off” with <form> and <input> tags.
- Avoid implicit trust between components.

Cross Site Scripting

XSS occurs when an attacker uses a web application to send malicious code, generally in the form of a script, to a different end user.

Types of XSS attacks:

- Stored - Injected code is permanently stored on the target servers, such as in a database, forum, visitor log, comment field, etc.
- Reflected – Injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request.

Cross Site Scripting

Common Attacks:

- Annoyance to the end user.
- Redirecting the user to some other page or site
- Disclosure of the user's session cookie.
- Session hijacking.

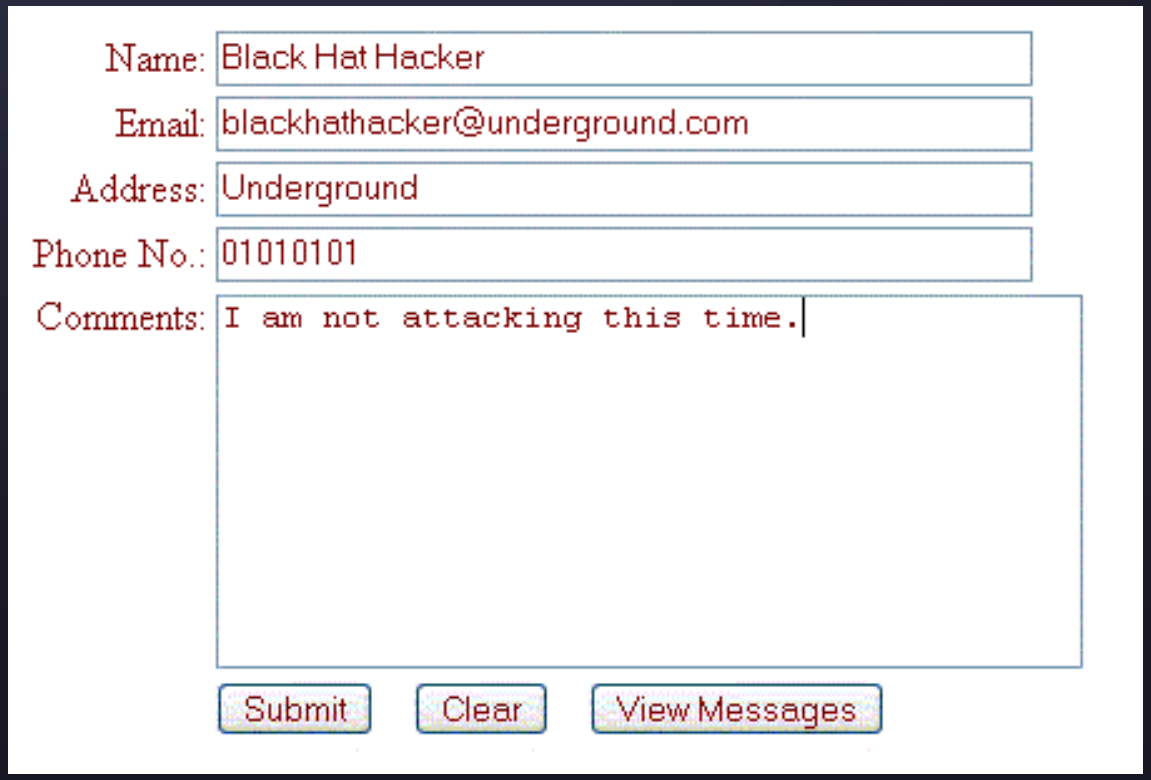
Prevention:

- Code review of all places where input from an HTTP request could possibly make its way into the HTML output.
- Validate all headers, cookies, query strings, form fields and all such inputs against a rigorous positive specification.

Cross Site Scripting

Community Architect Guestbook XSS (SA19742)

When the attacker is in a good mood:



Cross Site Scripting

Community Architect Guestbook XSS (SA19742)

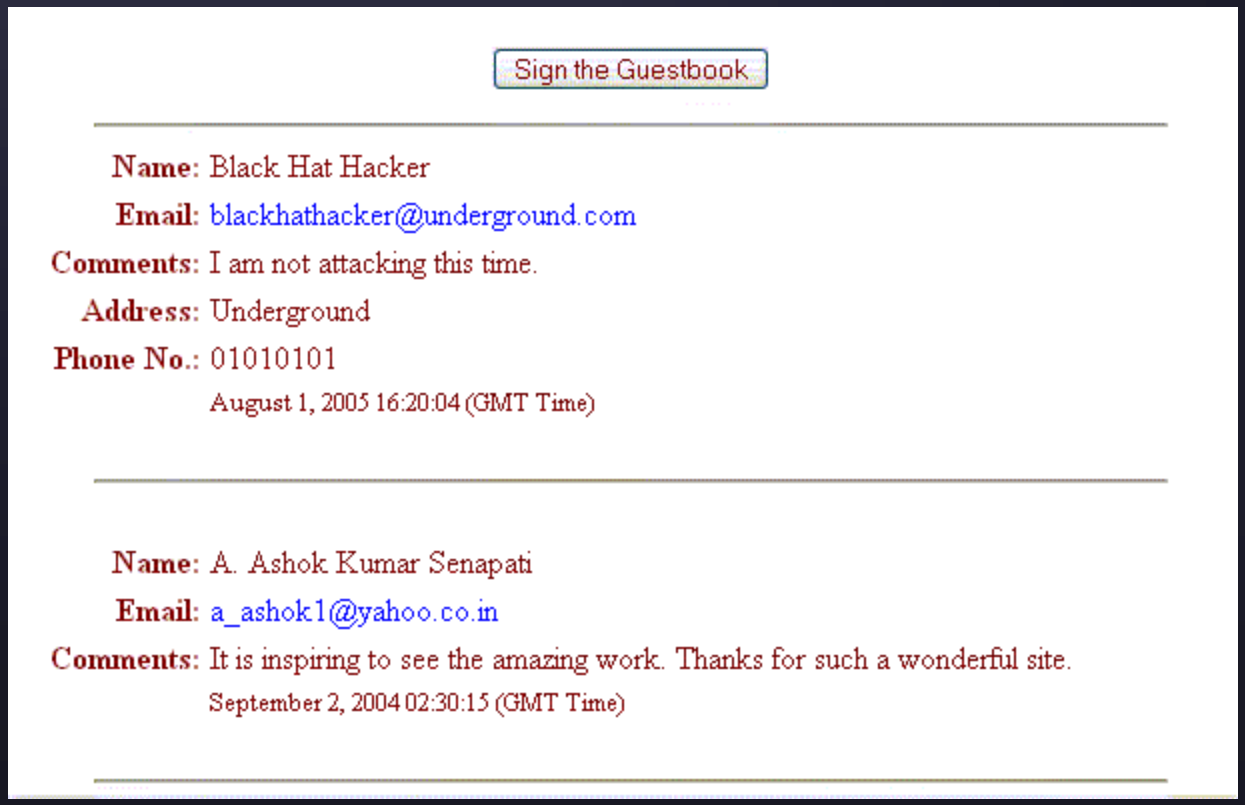
Where the input goes:

```
<!-- Guestbook Insertion Point -->
<table border="0" cellpadding="2" cellspacing="0">
<tbody><tr><td align="right"><b>Name:</b></td><td>Black Hat Hacker</td></tr>
<tr><td align="right"><b>Email:</b></td><td><a
href="mailto:blackhathacker@underground.com">blackhathacker@underground.com</a></
<tr><td align="right" valign="top"><b>Comments:</b></td><td width="500">I am
not attacking this time</td></tr>
<tr><td align="right" valign="top"><b>Address:
</b></td><td valign="top">Underground</td></tr>
<tr><td align="right" valign="top"><b>Phone No.:
</b></td><td valign="top">01010101</td></tr>
<tr><td></td><td><font size="2">August 1, 2005  16:20:04 (GMT
Time)</font></td></tr>
</tbody></table>
<br>
<hr size="2" width="80%">
<br>
```

Cross Site Scripting

Community Architect Guestbook XSS (SA19742)

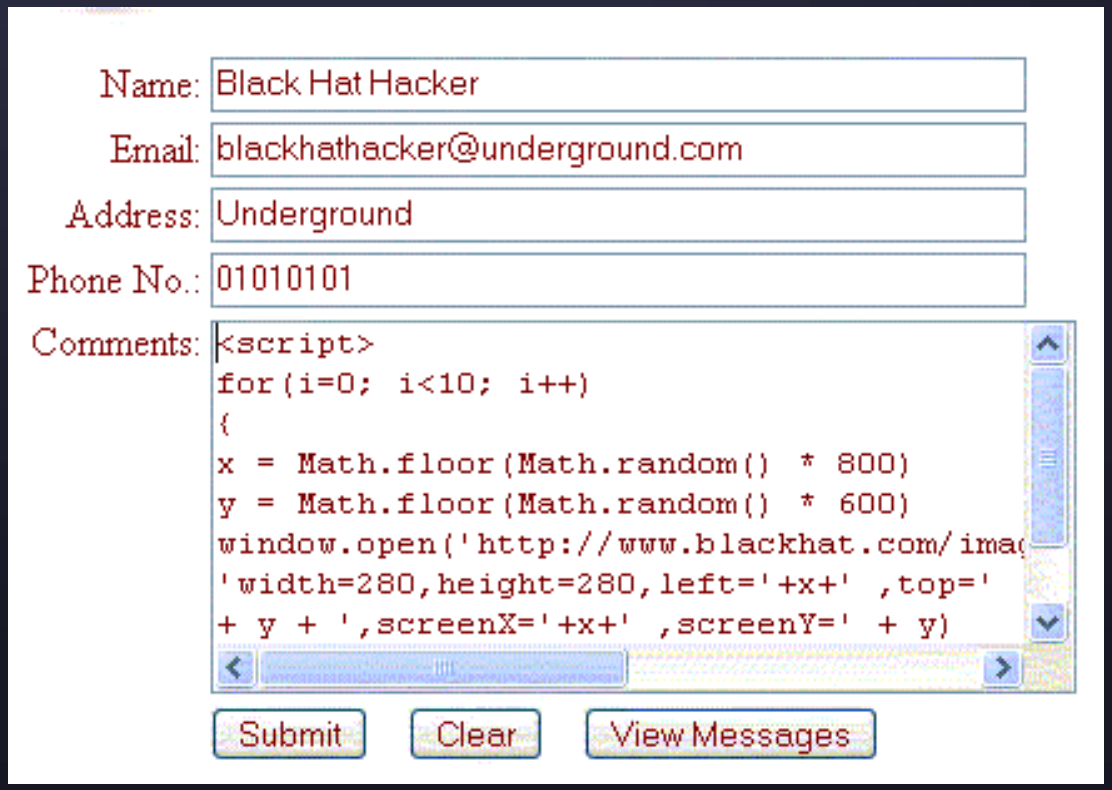
The Output:



Cross Site Scripting

Community Architect Guestbook XSS (SA19742)

If the attacker attacks:



Cross Site Scripting

Community Architect Guestbook XSS (SA19742)

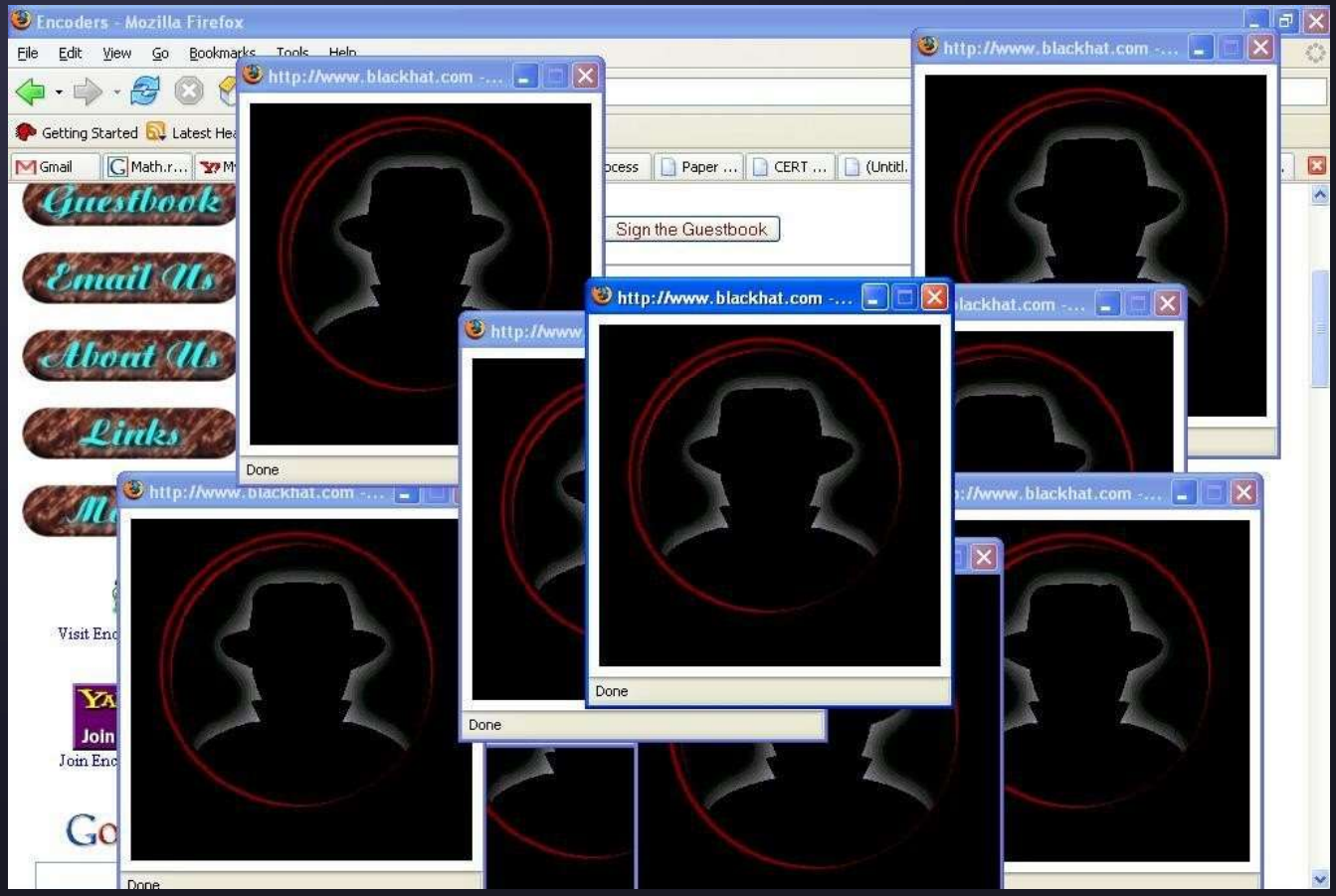
Where the input goes:

```
<!-- Guestbook Insertion Point -->
<table border="0" cellpadding="2" cellspacing="0">
<tbody><tr><td align="right"><b>Name:</b></td><td>Black Hat Hacker</td></tr>
<tr><td align="right"><b>Email:</b></td><td><a
href="mailto:blackhathacker@underground.com">blackhathacker@underground.com</a></
<tr><td align="right" valign="top"><b>Comments:</b></td><td width="500">
<script>
for(i=0; i<10; i++)
{
x = Math.floor(Math.random() * 800)
y = Math.floor(Math.random() * 600)
window.open('http://www.blackhat.com/images/bh-splash/bhcircle2.gif',x,
'width=280,height=280,left='+x+' ,top=' + y + ',screenX='+x+' ,screenY=' + y)
}
</script>
</td></tr>
<tr><td align="right" valign="top"><b>Address:
```

Cross Site Scripting

Community Architect Guestbook XSS (SA19742)

The Output:



Reference:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2003

Cross Site Scripting

Potential Harmful Tags:

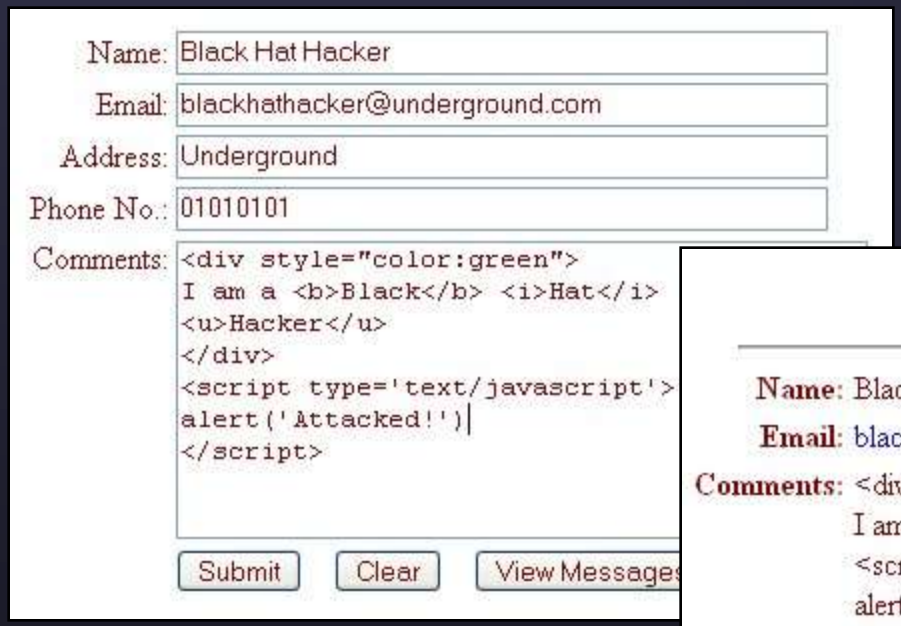
<script>
<object>
<embed>
<applet>
<form>

Decision to be taken for XSS prevention:

- Whether tags should appear in the output as a tags.
- Whether tags should not appear in the output at all.
- Whether text formatting is to be allowed.

Cross Site Scripting

If tags should appear in the output.



Convert special characters to HTML entities

Cross Site Scripting

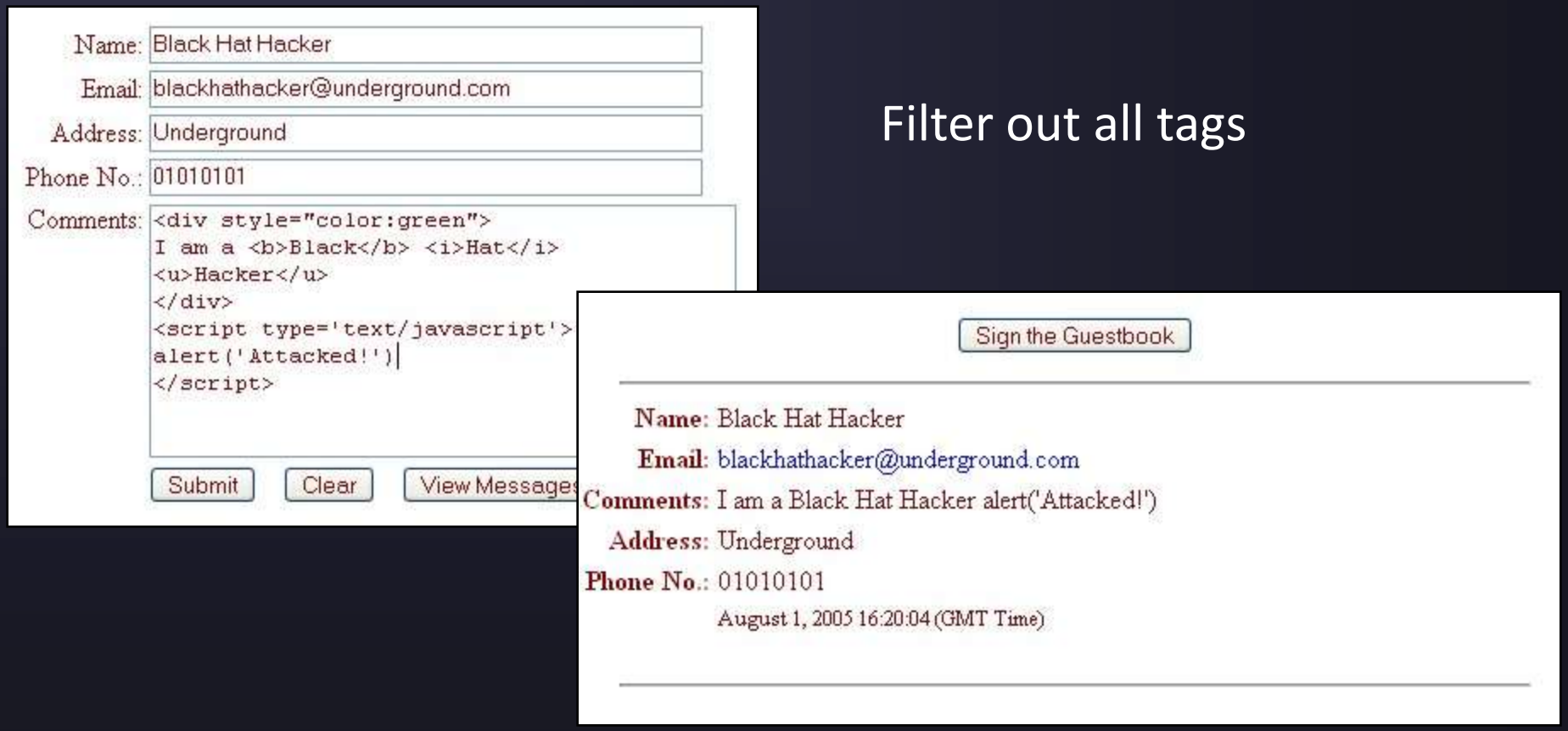
If tags should appear in the output.

Convert special characters to HTML entities:

Special Character	HTML Entity
<	<
>	&rt;
"	"
&	&
(	(
)	)
#	#

Cross Site Scripting

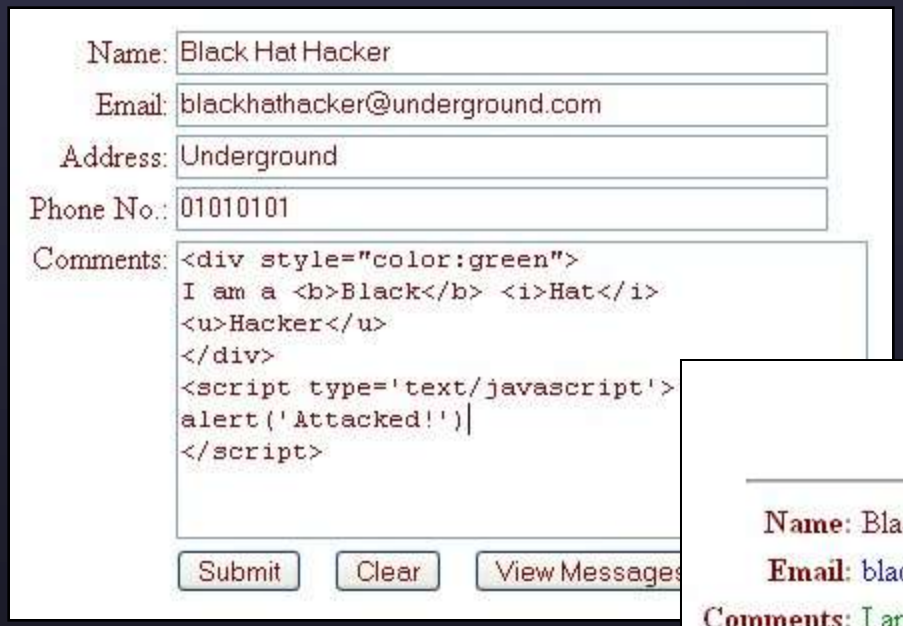
If tags should not appear in the output.



Filter out all tags

Cross Site Scripting

If text formatting should be allowed



Allow necessary tags only.

Cross Site Scripting

If text formatting should be allowed.

Tags that can be allowed:

<p>
<b>
<u>
<i>
<strike>
<h1>, <h2>, <h3>, <h4>, <h5>, <h6>

Buffer Overflows

Attackers use buffer overflows to corrupt the execution stack of an application.

Buffer overflows are easy to avoid, difficult to discover and extremely difficult to exploit.

Types of Attacks:

- Attackers send carefully crafted input to a web application, an attacker can cause the application to execute arbitrary code.
- When difficult to exploit, the attackers can crash the application thereby leading to DoS (Denial of Service).

Buffer Overflows

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Hi!";
  function(string);
}
```

Return Address

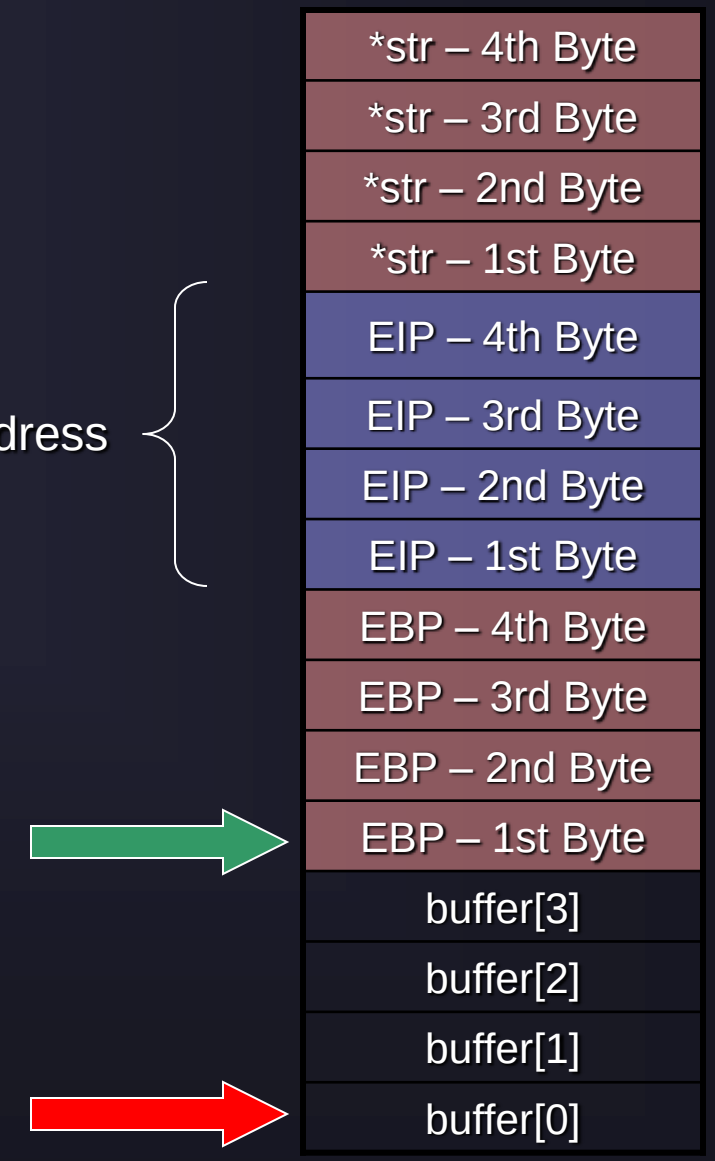
*str – 4th Byte
*str – 3rd Byte
*str – 2nd Byte
*str – 1st Byte
EIP – 4th Byte
EIP – 3rd Byte
EIP – 2nd Byte
EIP – 1st Byte
EBP – 4th Byte
EBP – 3rd Byte
EBP – 2nd Byte
EBP – 1st Byte

indicates what ESP is pointing to

indicates what EBP is pointing to

Buffer Overflows

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Hi!";
  function(string);
}
```

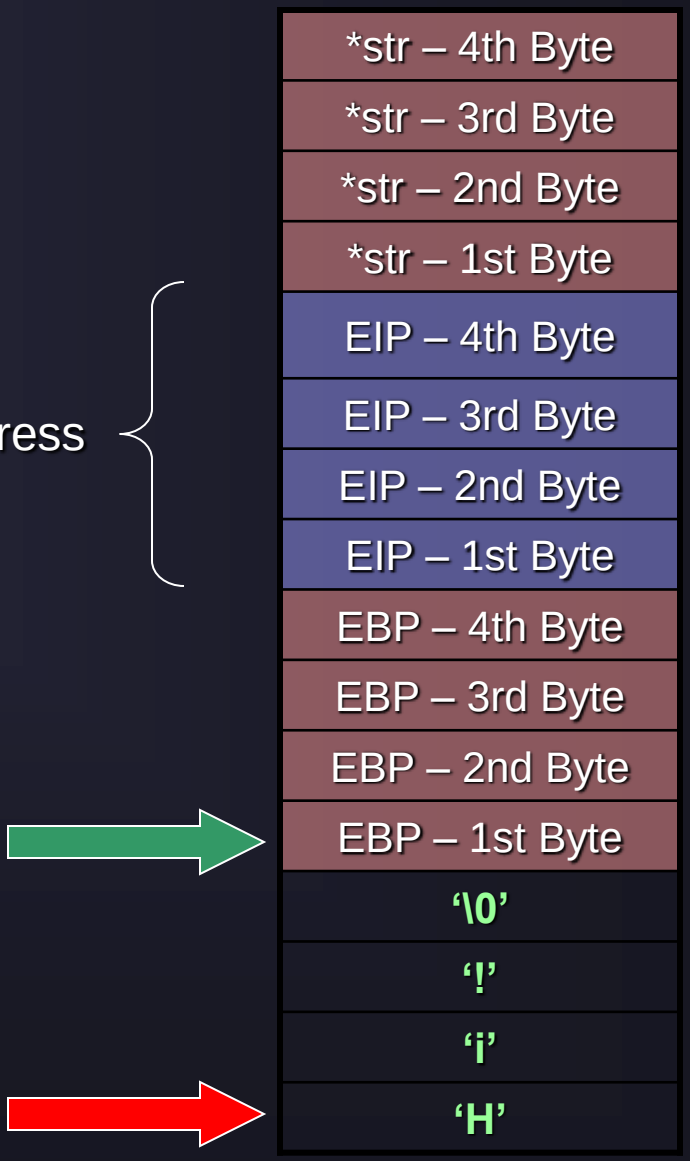


indicates what ESP is pointing to

indicates what EBP is pointing to

Buffer Overflows

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Hi!";
  function(string);
}
```

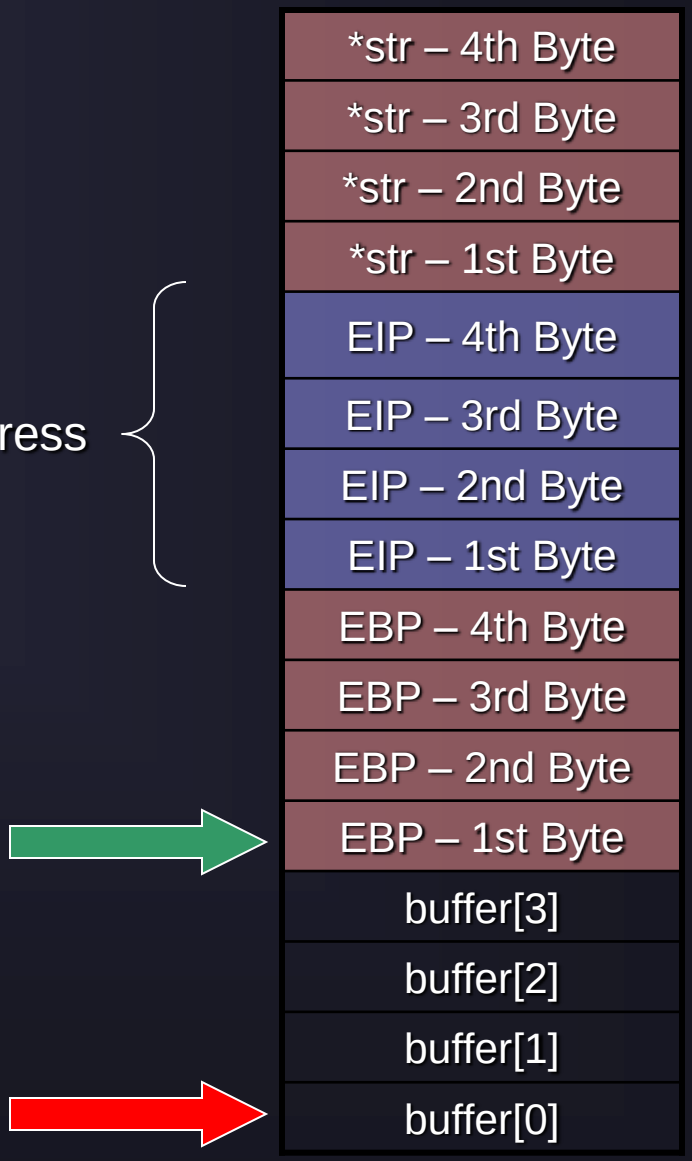


indicates what ESP is pointing to

indicates what EBP is pointing to

Buffer Overflows

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Good Morning!";
  function(string);
}
```



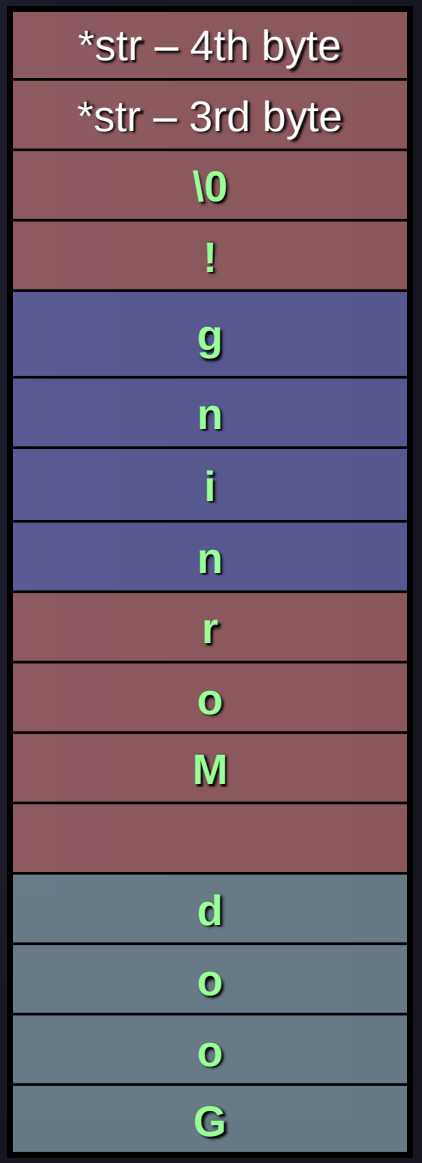
indicates what ESP is pointing to

indicates what EBP is pointing to

Buffer Overflows

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Good Morning!";
  function(string);
}
```

Return Address



indicates what ESP is pointing to

indicates what EBP is pointing to

Buffer Overflows

```
#include <string.h>

void function(char *str)
{
    char buffer[4];
    strcpy(buffer, str);
}

main()
{
  char string[]="Good Morning!";
  function(string);
}
```

Return Address

Returns to offset 676E696EH

*str -	4th byte
*str -	3rd byte
\0	00H
!	21H
g	67H
n	6EH
i	69H
n	6EH
r	72H
o	6FH
M	4DH
	20H
d	64H
o	6FH
o	6FH
G	47H

indicates what ESP is pointing to

indicates what EBP is pointing to

Buffer Overflows

Major Attacks:

Code Red Virus

Attacked a buffer overflow weakness in the Microsoft IIS Server API on July 19, 2001.

Damage: $2.6 billion

Sasser Worm

Attacked the Microsoft LSAS buffer overflow vulnerability on April 30, 2004.

Damage: $3.5 billion

Buffer Overflows

Prevention:

- C Programmers now refrain from using strcpy(), strcat() in their programs. They use strncpy(), strncat() instead which perform bound checking. Never use gets().
- Programmers now write their applications in languages like Java, Visual Basic which has better stack and memory management features.
- Sun Microsystems is trying to make their Sparc processor immune to stack overflow attack by introducing new feature to protect the return address during a subroutine call.

Injection Flaws

Injection flaw lets you inject executable code at places where it is not allowed in a secure system which might lead to execution of the code and compromise of the system.

Common Attacks:

- Relay malicious through a web application to another system.
- Calls to operating system via system calls.
- Use of external programs via shell commands.
- Calls to backend databases via SQL.
- Complete script injection into web applications.

Injection Flaws

SQL Injection:

- SQL injection is particularly widespread and dangerous form of injection.
- SQL Injection is possible if the value of any parameter is used in SQL queries and the parameter is not sanitised properly.
- The attacker can retrieve, modify, corrupt or destroy database contents.

Injection Flaws

Bloggage SQL Injection:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC Microsoft Access Driver] Syntax error in string in query expression 'Account_Name='blackhathacker'' AND Password=''.

/bloggage/check_login.asp, line 24

SELECT <some_column> FROM <some_table> WHERE Account_Name = '<user>' AND Password ='<password>'

SELECT <some_column> FROM <some_table> WHERE Account_Name ='blackhathacker'' AND Password =''

Injection Flaws

Bloggage SQL Injection:

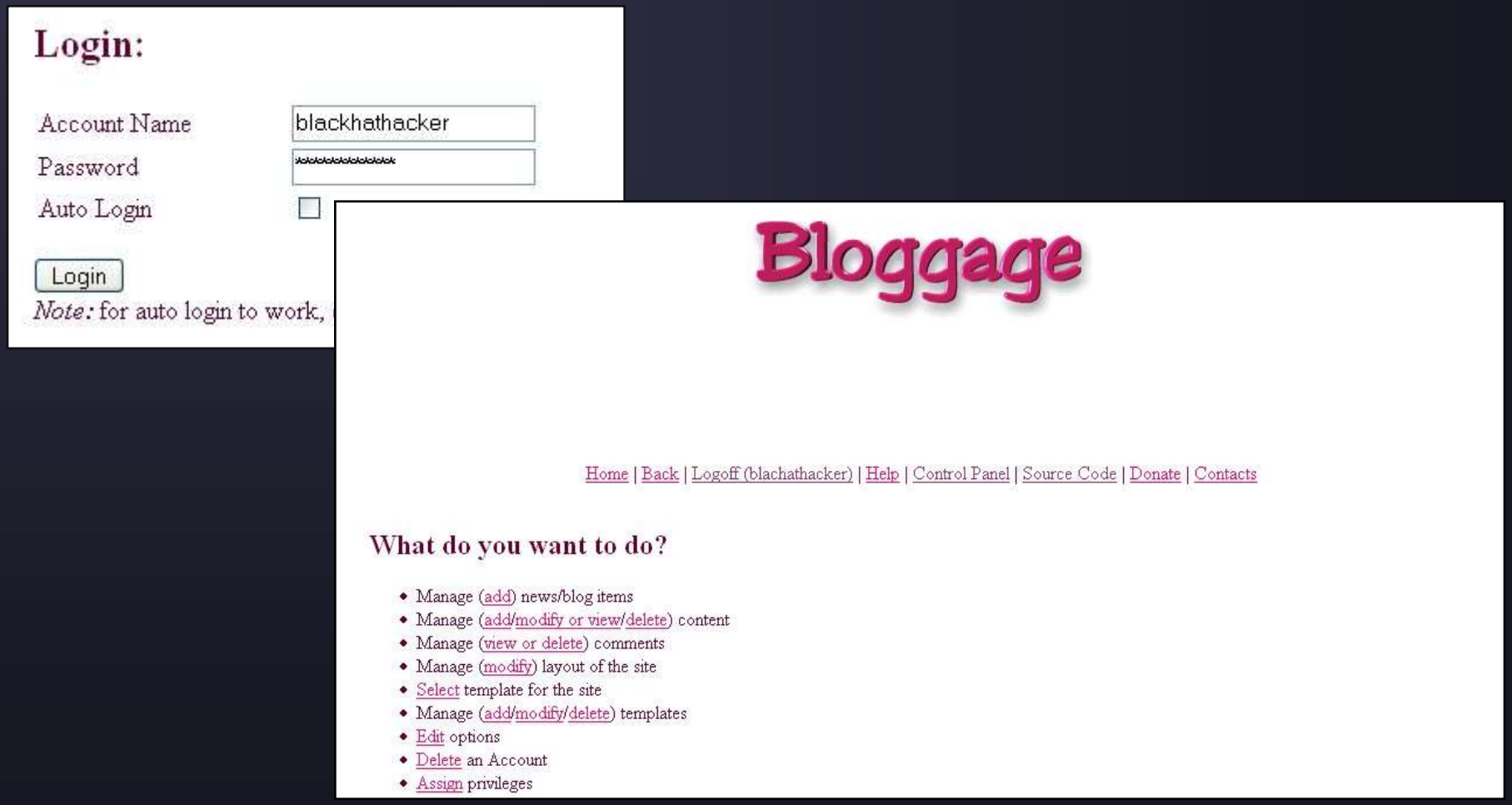
```
SELECT <some_column> FROM <some_table> WHERE
    Account_Name = '<user>' AND Password ='<password>'

SELECT <some_column> FROM <some_table> WHERE
    Account_Name = 'blackhathacker' AND
    Password = 'a' OR 'a' = 'a'
```

(Always a true condition)

Account Name: blackhathacker
Password: a' OR 'a' = 'a

Injection Flaws

Bloggage SQL Injection:



Injection Flaws

Prevention:

- Avoid accessing external interpreters wherever possible.
- For calls to backend databases, carefully validate the data provided to ensure that it does not contain any malicious content.
- Structure requests in a manner that ensures that all supplied parameters are treated as data, rather than potentially executable content.
- Convert special characters to the correct form.
- Run the application with minimum priveleges required.

Improper Error Handling

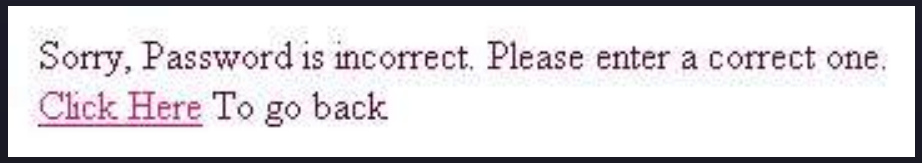
Improper handling of errors can introduce a variety of security problems.

Common Problems:

- Inconsistencies in error messages which can reveal important clues on how the system works.
- Stack traces, database dumps and error codes are displayed on error.
- Syntax errors are displayed along with portions of the codes.

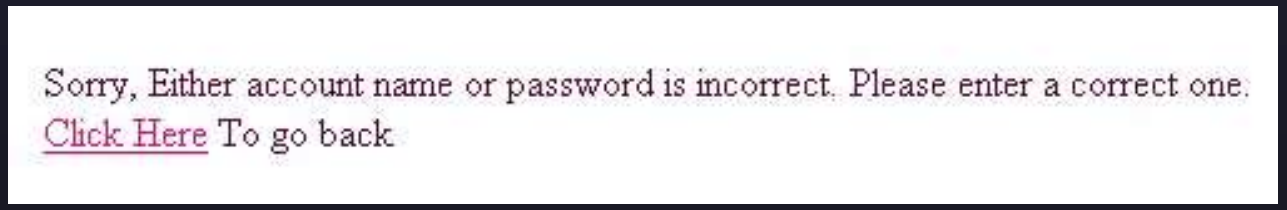
Improper Error Handling

A Bad Error Message



Improper Error Handling

An Ideal Error Message



Improper Error Handling

IncredibleIndia.org SQL Injection (WHID 2006-27)

A valid page:
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828

Trial 1:
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828'

Error:
Unclosed quotation mark before the character string ' and mncpage.mnccategoryid = mnccategory.mnccategoryid'.

Conclusion:
Direct SQL Injection is possible. There are 2 tables, 'mncpage' and 'mnccategory'. Both of them have a column called 'mnccategoryid'.

Improper Error Handling

IncredibleIndia.org SQL Injection (WHID 2006-27)

Trial 2:
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828 order by 1--
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828 order by 2--
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828 order by 3--

No error

Improper Error Handling

IncredibleIndia.org SQL Injection (WHID 2006-27)

Trial 3:
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828 order by 4--

Error:
The ORDER BY position number 4 is out of range of the number of items in the select list.

Conclusion:
The table being used by the query selects 3 columns and one of them is an integer.

Improper Error Handling

IncredibleIndia.org SQL Injection (WHID 2006-27)

Trial 4:
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828 union select 'varchar1', 'varchar2', 'varchar3' from mncpage --

Error:
Syntax error converting the varchar value 'varchar1' to a column of data type int.

Conclusion:
The 1st column in the select query is an integer.

Improper Error Handling

IncredibleIndia.org SQL Injection (WHID 2006-27)

Trial 5:
http://www.incredibleindia.org/newsite/cms_Page.asp?PageID=828 union select mnccategoryid, 'varchar2', 'varchar3' from mncpage--

No Error

Conclusion:
The column 'mnccategory' is of integer type.

Reference:
http://www.webappsec.org/projects/whid/list_id_2006-27.shtml

Insecure Storage

Most applications need to store sensitive information, either in a database or on a file system.

Types of Sensitive Information:

- Passwords
- Credit Card Numbers
- Account Records
- Proprietary Information

Insecure Storage

Common Mistakes:

- Failure to encrypt critical data
- Insecure storage of keys and passwords
- Improper storage of secrets in memory
- Poor sources of randomness
- Poor choice of algorithm
- Attempting to invent a new encryption algorithm

Types of Attacks:

- Examining tokens, session IDs, cookies, etc. to see if they are obviously not random.
- Finding cryptographic flaws.

Insecure Storage

Prevention:

- Minimize encryption. Use it to keep information that is absolutely necessary.
- Instead of storing encrypted passwords, use a one-way function, such as a SHA-1, to has the passwords
- If cryptography must be done, choose a library that has been exposed to public scrutiny and makes sure that there are no open vulnerabilities.
- Clear the clear-text sensitive information from the memory as soon as possible.

Denial of Service

Denial of Service (DoS) attack is an attempt to make a computer service unavailable to its intended users.

Common Attacks:

- Consume bandwidth, database connections, disk storage, CPU, memory, threads, or application specific resources.
- Locking out a legitimate user by sending invalid credentials until the system locks out the account.
- Exploiting buffer overflows & injection flaws to inject code/commands to shut down or stall importand service.

Denial of Service

0verkill recv_packet() UDP Handling Overflow DoS

The vulnerability was an integer underflow error in a Linux based gaming daemon called 0verkill. It could be exploited to launch DoS attacks by sending UDP packets of size less than 12 bytes to the 0verkill daemon.

Prevention:

- Load balancing makes these attacks more difficult.
- Analyse each resource and check if there is a way to exhaust it.
- Limit resources to any user to bare minimum.
- Check error handling scheme and ensure that an error can not affect the overall operation of the system.

Insecure Configuration

Insecure configuration of a system can itself be a potential vulnerability.

Common Configuration Problems:

- Unpatched security flaws in a software.
- Server software flaws or misconfigurations that permit directory listing and directory traversal attacks
- Improper file and directory permissions
- Unnecessary services enabled
- Default accounts with default passwords
- Administrative or debugging functions that are enabled
- Improper authentication with external systems

Insecure Configuration

Apache URL Validation Vulnerability (BID 19447)

```
# Sample Safe Configuration for Unix/Linux
DocumentRoot "/home/webmaster/site/docroot/"
ScriptAlias /cgi-bin/ "/home/webmaster/site/docroot/cgi-bin"

# Sample Unsafe Configuration for Windows
DocumentRoot "C:/webmaster/site/docroot"
ScriptAlias /cgi-bin/ "C:/webmaster/site/docroot/cgi-bin/"
```

Accessing a script in the cgi-bin directory:
http://www.example.com/cgi-bin/somescript

Exploit:
http://www.example.com/CGI-BIN/somescript

Insecure Configuration

Apache URL Validation Vulnerability (BID 19447)

```
# Sample Safe Configuration for Windows
DocumentRoot "C:/webmaster/site/docroot"
ScriptAlias /cgi-bin/ "C:/webmaster/site/cgi-bin/"
```

Accessing a script in the cgi-bin directory:
http://www.example.com/cgi-bin/somescript

Exploit URL is now invalid:
http://www.example.com/CGI-BIN/somescript

Reference:
http://www.securityfocus.com/bid/19447

Insecure Configuration

Prevention:

- Configure all security mechanisms
- Turn off all unused services
- Set up roles, permissions and accounts
- Disable all default accounts or change their passwords
- Enable logging and alerts
- Monitor the latest security vulnerabilities published
- Apply the latest security patches
- Update the security configuration guidelines
- Scan for vulnerabilities with a vulnerability scanner regulary

Thank You!