

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Secure Programming Guidelines

This document is aimed towards presenting secure programming guidelines for web site and application developers. The document brings about an understanding of OWASP vulnerabilities encountered in web sites along with illustrations. This document is not supposed to replace the OWASPTop10 document.

A large number of sites have been audited for application security vulnerabilities in the past two years. Although each site has it's own unique features and it may have been developed/hosted in different environment and each may require detailed study involving source code and design review depending on the sensitivity of the data/application, a black box method can be deployed for assessing the vulnerabilities in these sites commonly encountered in web sites as identified by OWASP.

It has emerged over this period of time that sites can be broadly categorized into

- (i) Static Web sites**
- (ii) Dynamic Web sites**

(i) **Static web sites** can be further categorized into

- a. Pure static web sites where all the pages are purely static and hosting pages such as .pdf, .html, .htm, .doc, .xls, .ppt
- b. Static web sites where the contents are rendered through static content generating .asp or .php pages. There are no interactive elements.
- c. Static web site having a feedback form. The form in this case submits the form contents to an application running on another URL.

(ii) **Dynamic web sites** can be further categorized into

- a. Primarily Static web sites with one feedback page or application
- b. Primarily Static web site with one application for a query module
- c. Sites hosting a large number of applications and open to all
- d. Sites hosting applications for a closed user group

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

- i. Only authenticated users can access the functional modules
- ii. Only authenticated users having the privilege can access the modules for which they are authorised

(i).a Pure Static Web Sites : For the sites in the category of **(i).a** above the sites can be safely hosted on the production server with a simple Read only permission.

(i).b Static Web sites with static content generating script Files : For the sites in **(i).b** above the sites can be hosted on the production server with Read and Execute permission.

(i).c Static Web sites with feedback form (handling program remotely hosted): For the sites in **(i).c** above the site can be hosted with Read only permissions. But the form handling program/application on the foreign URL remains to be audited for security vulnerabilities.

For example: this may be vulnerable as in the case of http://indiaimage.nic.in/mail_feed3.cgi. This program is used for submitting forms POST contents and is used from several web sites, which are primarily static.

(ii).a Primarily Static site with one Feedback application: For sites in the category of **(ii).a.** above, the sites having one feedback / Grievance/ Guestbook form which are filled up by site visitors. The form handling program is typically as .asp, .pl, .php, .aspx, .jsp or a java servlet etc. based application. This application accepts the form elements passed from the form and processes them before committing to a database for later retrieval or sends them in a mail to the intended recipient of the feedback.

This program is typically prone to

- (a) Input Validations vulnerabilities
- (b) Cross site scripting vulnerabilities
- (c) Buffer overflow
- (d) Error Handling vulnerabilities
- (e) Denial of Service vulnerabilities

(ii).b Primarily Static web site with one application for a query module: For sites in the category of **(ii).b** above, the site hosts an application such as a query module. This module presents an interface for accepting a query input for example: a search string or a employeecode. The handling program accepts the form input and processes it for example to

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

look up a database and present a suitable response as per the program logic. This program is typically prone to

- (a) Input validation vulnerabilities
- (b) Cross site scripting vulnerabilities
- (c) SQL injection vulnerabilities
- (d) Buffer overflow
- (e) Error Handling vulnerabilities
- (f) Denial of service vulnerabilities

(ii).c Sites hosting a large number of applications and open to all: For sites in the category (ii).c above, the site may be host to applications which may accept input for committing to a database or for querying a database. These applications are accessible to all site visitors. These applications may be prone to

- (g) Input validation vulnerabilities
- (h) Buffer overflow
- (i) Cross site scripting vulnerabilities
- (j) SQL injection vulnerabilities
- (k) Error Handling vulnerabilities
- (l)** Denial of service vulnerabilities

(ii).d.i Sites hosting applications for a closed user group Only authenticated users can access the functional modules

For sites in this category, the site is host to a set of applications which can be accessed by users who have authenticated successfully. That is there is a common set of functionality which can be accessed by any one who authenticates successfully. These applications are prone to

- (a) Input validations
- (b) Buffer overflow
- (c) SQL injection
- (d) Cross site scripting injections
- (e) Error Handling vulnerabilities
- (f) Denial of service vulnerability
- (g) Broken access control
- (h) Broken account and session management
- (i) Insecure storage

(ii).d.ii Sites hosting applications for a closed user group Only authenticated users having the privilege can access the modules for which they are authorised

For sites in this category, the site is host to a set of applications which can be accessed by users who have authenticated successfully. That is there is a set of functionality which can be accessed by one who authenticates successfully and who is authorized. There may be one user who authenticates successfully but may not have access to same functionality accessible to

Name of the Document	Secure Programming Guidelines		
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

another user. This is a variation of the earlier kind. Here however, privilege infringement may occur and so access control checks are more rigorously checked. These applications are prone to

- (j) Input validations
- (k) Buffer overflow
- (l) SQL injection
- (m) Cross site scripting injections
- (n) Error Handling vulnerabilities
- (o) Denial of service vulnerability
- (p) Broken access control
- (q) Broken account and session management
- (r) Insecure storage

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

1. Input validation Vulnerabilities

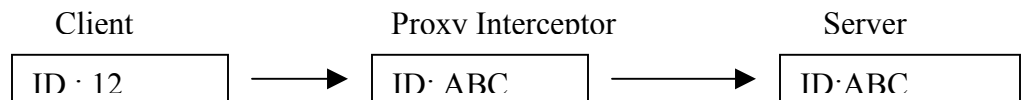
Unvalidated Parameters

Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backside components through a web application.

Web sites are host to applications which accept input in URL strings, form fields, hidden form fields etc.

Example:

- (a) A parameter in the URL accepting a numeric input also can be used to pass alphabetic input or junk data. The application processing this input may not function in it's desired way upon receiving input in the alphabetic or alphanumeric format or junk data.
- (b) Many applications do incorporate input validations to address the above kind of requirements. However, they only do so on the client side of the application. Sophisticated proxying or intercepting tools are available which make the client side checks inadequate. For example: A program has incorporated javascript code on the client side to perform check of the input submitted. Then a proxying tool can be used to modify the data from form POST. This leads to the situation that what is submitted from the client is not the same as what is ultimately submitted to the server side application.



The above depiction illustrates that there may be client side input validation checks imposed, but this have been inadequate as the proxy interceptor tool was used to manipulate the input before forwarding to the server. The server side program may not behave in the intended ways as designed by the developer.

(c) The third kind of issue with input validation vulnerabilities indicates that even after input validation checks have been applied to take care of the vulnerabilities encountered, some form of input may still be able to make it past the validation check/controls. These are those form of input which are also called as encoded form. There are various forms of encoding. Hexadecimal values can be used to display non-standard letters and characters in browsers and plug-ins.

For example, a validation check may be in place to filter out single quote('), however a single quote (') may be inserted in the URL encoded form such as %27.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Solution:

Perform Input Validations. Refer to **Input Validations in Appendix.**

2. Broken Access Control

Web sites allow certain users to access or perform certain functions while disallowing others access to these functions. This is termed as Access Control or Authorisation. These checks are performed after authentication.

Some sites are host to administrative interfaces that allow for managing site related functions.

Broken Access Control cases:

1. Forced Browsing past access control checks
2. Client side caching
3. Insecure Ids
4. File permission

1. Forced Browsing past Access Control checks. Many sites require users to pass certain checks before being granted access to certain URLs that are typically 'deeper' down in the site. These checks must not be bypassable by a user that simply skips over the page with the security check.

Example : An unauthorised user can add, update, delete data

One of the web site is host to directly invoked URL pages that are meant to be the privileged functions of an authenticated and authorised user. Any user can invoke the pages (previously browsed) without authentication.

This insecurity presents a high risk to the application, as it would enable the attacker to bypass the authentication mechanism and gain the privileged access of an authorized user of the application/site.

This is a very simple attack. The information about the previously browsed page can be seen from a browser's history. So the attacker needs to have an access to a machine previously used for browsing by an authorized user of the site.

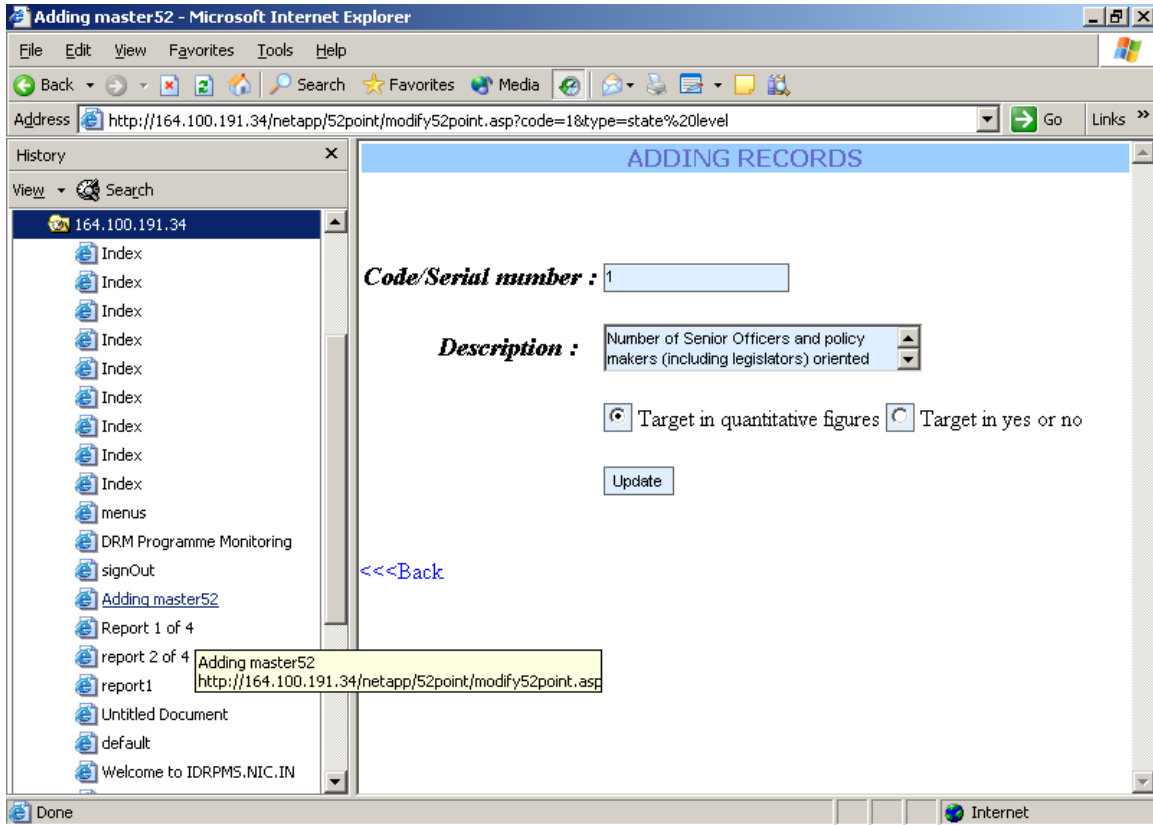
By bypassing the authentication control mechanism the attacker can perform some of the privileged functions of an authorized user. He can View/Add / Modify / Delete Application related information on the site. This include all/some of the privileged functions

This vulnerability is of high impact as it damages the integrity of the site.

Pasting a URL on the browser address bar

It was possible to navigate to the page which was otherwise accessible after authentication and visiting the page.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	



It was seen that form saving was successful.

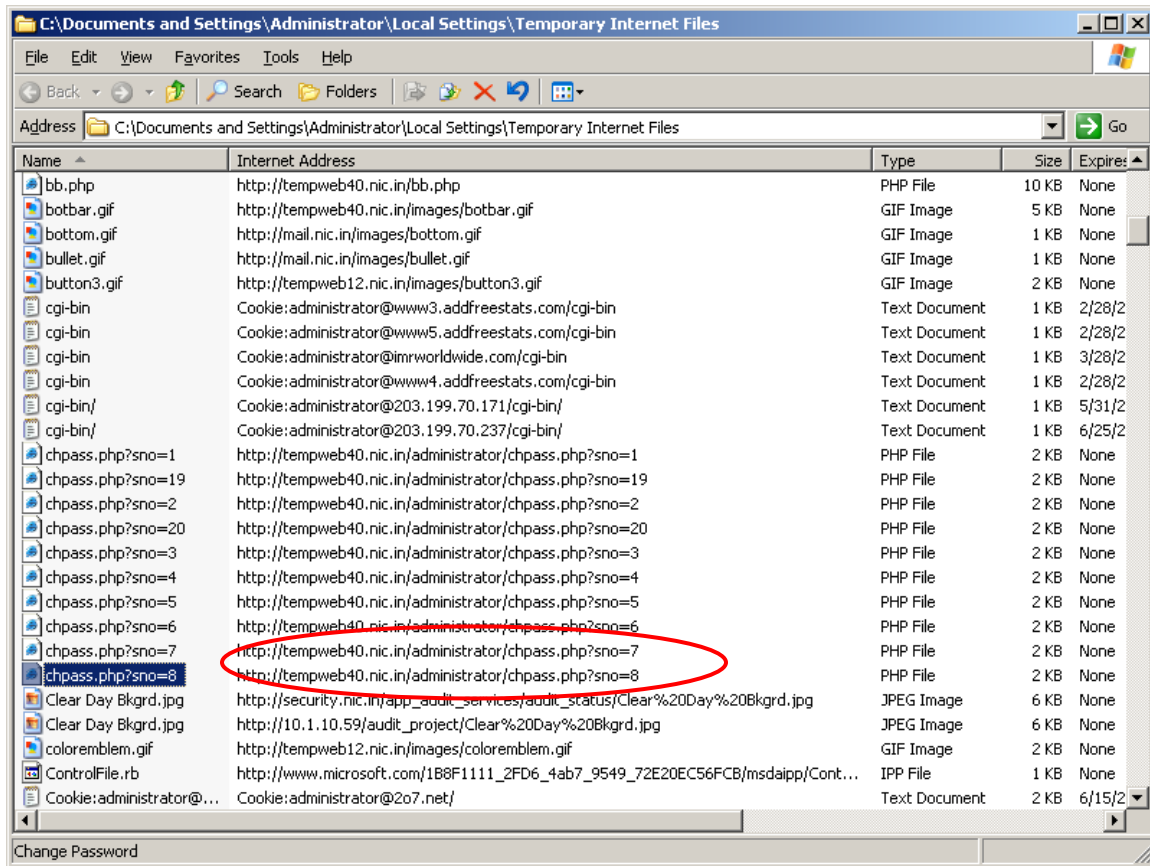
2. Client Side Caching : Many sites are accessed from shared computers in libraries and kiosks. The visited pages get cached in the browser. So a second person using the same computer to visit a site can access pages visited by the previous user even if he has not visited these pages or has no knowledge of these pages.

Example : Malicious user can by-pass authentication to change password of Zone Ids which is a privileged function of Administrator

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

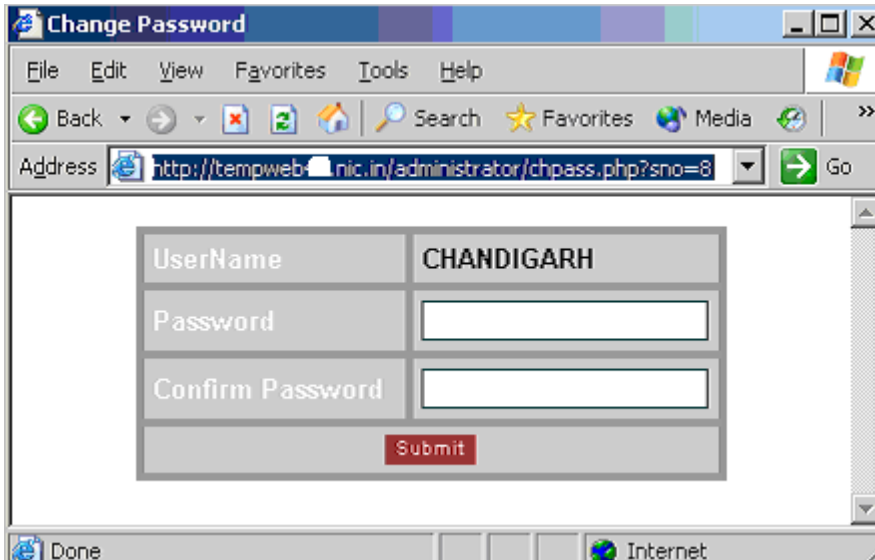
A web site had a Change password module, which was supposed to be accessed after authentication. This page allowed change of password for various geographically dispersed zonal administrators. It was seen from tests that it was possible to view the pages by directly going to the "Temporary Internet Files" folder using the windows explorer or from the browser history. This was possible for pages that were previously viewed by an authorized user. This is a case of broken access control. The page was supposed to be accessed after authentication by those who are authorized to access it. But by simply accessing it from browser history directly the module could be invoked and authentication could be bypassed.

(1) From The temporary Internet Files folder the following could be seen



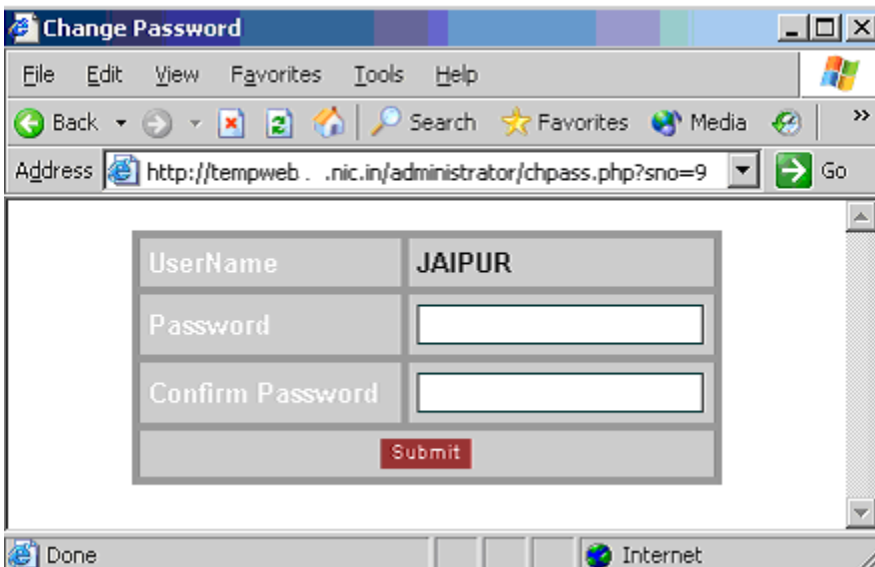
The URL as seen from above was invoked in a browser session.

Name of the Document	Secure Programming Guidelines		
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	



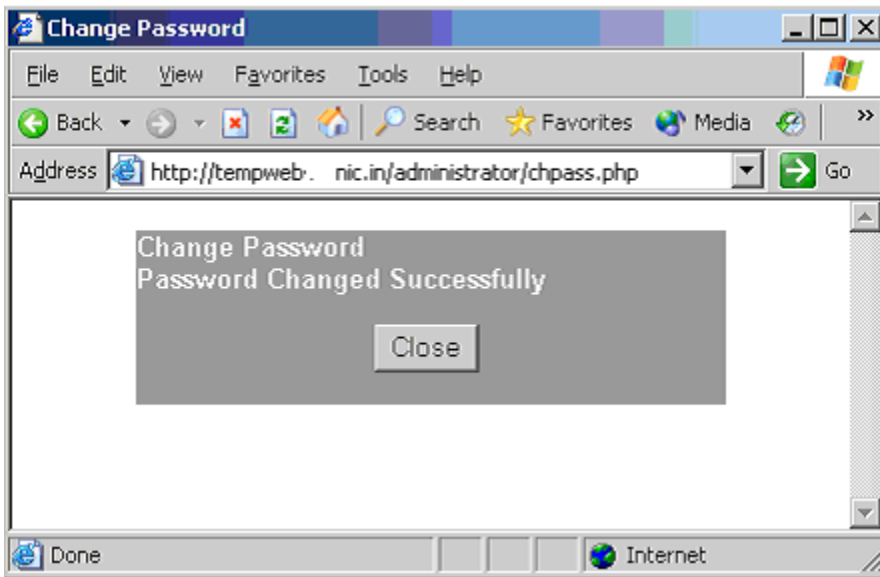
The above snap shot reveals the page, which should have been accessible after authentication. As seen above, the username is 'Chandigarh' and is the interface to change the password of the Chandigarh zone.

By varying the parameter sno to 9 the change password interface for the jaipur zone was made available. Similarly by varying this parameter all the zone administrator accounts password could be changed.



The following shot shows a message for successful change of password in the above manner.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	



These password could be used to gain access to the system later.

Solution

Apply strong authentication and session management logic in addition to access control on each of these pages. So that a user needs to be authenticated first before being able to access a page he is authorized to access.

The application's access control requirements are to be identified and a web application security policy must be documented. An access control matrix may be used to define the access control rules. The policy should document what types of users can access the system, and what functions and content each of these types of users should be allowed to access.

This policy must be strongly enforced in the application logic. Sessions must be enforced in all these pages to be accessed after authentication.

The access control mechanism should be extensively tested to be sure that there is no way to bypass it. This testing requires a variety of accounts and extensive attempts to access unauthorized contents or functions.

3. Insecure Ids : Many applications use ids, key etc to represent user, role, content,object or function. If an attacker can guess these ids and the program does not perform sufficient check to see that the id is not supposed to be accessed from the current user, the attacker can exercise the access control scheme to see what he can exploit.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Example : An authenticated user can Add/Update data of another user.

One of the website has an application that is accessible to authenticated and authorized users only. The application enables the authorized users to view , add and update functional data such as Monthly Progress output, yearly targets for states monitored by them. An unauthorized user can view/add/update these data without logging in as the concerned user.

This is a case of Broken Access control. Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other user's accounts or use unauthorized functions.

The impact of this is that the data in the database will be fake or modified leading to inconsistent information and cannot be relied upon.

The module has two users aparna1 and abha1. aparna1 when logged in can monitor the details for Delhi state and abha1 when logged in can monitor the details for West Bengal state.

Logging in as user Aparna1

The screenshot shows a web application interface. At the top, it says "Welcome: Aparna kanda". Below this is a navigation bar with "Monthly Progress of Outputs" and links for "Main Menu" and "Sign Out". There are two dropdown menus: "For Year" set to "2004" and "For Month" set to "NOVEMBER". Below these is a red instruction: "Click on Modify to Add / Edit values for a Level." A table follows with columns: S.No, Level, Monthly Progress of Outputs OK, and Action. The table has 6 rows with levels: State, District, Block, GP, Village, and ULB. Each row has a "Modify" link in the Action column. At the bottom, there are three buttons: "Run Checklist", "Final Submit", and "Show Monthly progress".

S.No	Level	Monthly Progress of Outputs OK	Action
1	State	NO	Modify
2	District	NO	Modify
3	Block	NO	Modify
4	GP	YES	Modify
5	Village	NO	Modify
6	ULB	YES	Modify

Clicking the option Modify for State Level, the following url gets invoked.
<http://tempweb1.nic.in/getMonthlyDataTemp.aspx?stateid=3&levelid=1&Monthid=11&yearid=2004&userid=130&userLevel=2>

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Monthly Progressive Outputs Main Menu Entry screen Sign Out

Note :- Values above the input boxes are the last month's figures. Current values should be greater or equal to last month's figures.

Year	2004	TOTAL U P D A T E	
Month	11		
State	Delhi		
Level	State		
Outputs			
Senior Officers and policy makers (including legislators) oriented	<input type="text" value="200"/>		
State Disaster Management Policy approved (yes = 1, no = 0)	<input type="text" value="1"/>		
State Disaster Management Act enacted (yes = 1, no = 0)	<input type="text" value="0"/>		
State Disaster Management Authority set up (yes = 1, no = 0)	<input type="text" value="1"/>		
Redesignation of nodal department as department for DM (yes = 1, no = 0)	<input type="text" value="0"/>		
State disaster Management plan finalized (yes = 1, no = 0)	<input type="text" value="0"/>		

Display Targets Save Data

This page allows display of a set of data for monthly progressive output for Delhi state

The user can vary the state id to 12 to invoke the same URL as follows:
<http://tempweb1.nic.in/getMonthlyDataTemp.aspx?stateid=12&levelid=1&Monthid=11&yearid=2004&userid=130&userLevel=2>

Monthly Progressive Outputs Main Menu Entry screen Sign Out

Note :- Values above the input boxes are the last month's figures. Current values should be greater or equal to last month's figures.

Year	2004	TOTAL U P D A T E	
Month	11		
State	West Bengal		
Level	State		
Outputs			
Senior Officers and policy makers (including legislators) oriented	<input type="text" value="80"/>		
State Disaster Management Policy approved (yes = 1, no = 0)	<input type="text" value="0"/>		
State Disaster Management Act enacted (yes = 1, no = 0)	<input type="text" value="0"/>		
State Disaster Management Authority set up (yes = 1, no = 0)	<input type="text" value="0"/>		
Redesignation of nodal department as department for DM (yes = 1, no = 0)	<input type="text" value="0"/>		
State disaster Management plan finalized (yes = 1, no = 0)	<input type="text" value="0"/>		

Display Targets Save Data

This screen displays the monthly progressive output for the West Bengal State.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Modifying the data on the screen as follows:

Clicking Save Data.

The saving is successful.

As can be seen from above, the user logged in as Aparna1. He could update the monthly progressive output for west Bengal state, which are details, monitored by user Abha1.

This implies that the user can also modify/save data for another user. This is a case of privilege infringement.

Solution: Care should be taken to see that users/functions are not represented as insecure Ids which can be guessed or manipulated (even by an authenticated user)

4. File Permissions

Web sites and the hosted applications are hosted with ACL of the file system of the underlying platform. The ACLs should be configured just rightly so as to make information available. Often however, configuration files, scripts files etc. are left with such ACLs that this leads to significant and sensitive information being made available to all of the site visitors. For example, the screen shot below shows the contents of a configuration file used by the applications in the site to connect to backend database.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

```

Address http://www.templeweb40.nic.in/common.inc
Google Search PageRank 27 blocked Check All OFF Options

<?php
//common.inc

function db_connect() // To connect to database
{
    $link_id = mysql_connect('164.100.9.207','nes','kga138jwq1');
    mysql_select_db("nes");
    return $link_id;
}

function error_message($msg) //To display a javascript base error message
{
    echo "<Script>alert(\"Error:$msg\");history.go(-1)</Script>";
    exit();
}

?>

```

This was possible because Read permission was given on the file. Only files that are specifically marked to be presented to site visitors should be marked readable. Script files should be marked executable.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

3. Broken Authentication and Session Management

Often sites have applications designed for closed user group access. Only authenticated users can have access to resources and functions. These applications suffer from certain issues such as the session life cycle being not managed or being faulty. Also the credentials used for the purposes of authentication are not properly managed with the result that the credentials can be exploited by a malicious user to gain privileged access.

HTTP is a stateless protocol. Web servers treat each client request as one independent request and do not link them as coming from same source. Multiple requests originating from a client are linked to each other by applying a state mechanism scheme such as a 'Session'. Identifying each session to comprise a user action separately and uniquely is critical for web application security. The role of the developer is very important here in implementing a session management scheme. This involves managing the session life cycle by generating a new session just as when a user authenticates and logs in to the system and invalidating the session when the user session terminates or ends.

Session Time out : Session token which do not expire on the HTTP or web server can enable an attacker to guess or bruteforce a valid authenticated session.

Example:

Session replay possible due to improper termination of session on the server

Example 1:

1. From an active session in one of the web applications Logout option was activated. The request on the proxy (traveling from the client towards server) looks as follows :

```
GET http://164.100.9.34/drmreport/signOut.aspx HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
Referer: http://164.100.9.34/drmreport/menus.aspx
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR 1.1.4322)
Host: 164.100.191.34
Cookie: ASP.NET_SessionId=sp4eh545c1avao55yyu23545;
ASPSESSIONIDAQCSBTQA=PMNPGKNAOOFBLBBMGLICKDDD;
Login=CBE6FAFD6339A03DBD599D6EB990AA0B63CF0CAE2D8B16426C8D5DC2EDA59494385AA539
F31CF070B43424F641465AB3CC0C83EA9170412F0B45076B23619E6A4350E17E324E9606
Proxy-Connection: Keep-Alive
```

The user is logged out of the session.
Note the cookie value marked above.

2. On the proxy, the request (which was captured from the earlier session) was pasted and submitted.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

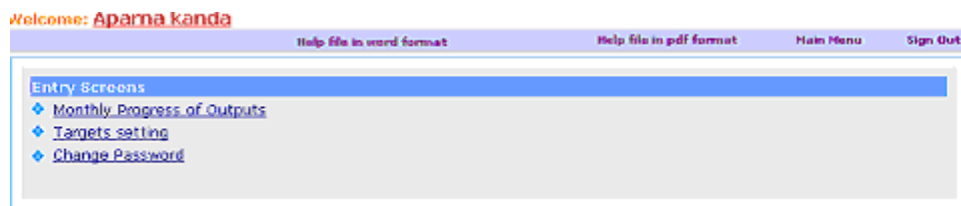
```

GET http://164.100.9.34/drmreport/menus.aspx HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
Referer: http://164.100.9.34/drmreport/index.aspx
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR 1.1.4322)
Host: 164.100.191.34
Pragma: no-cache
Cookie: ASP.NET_SessionId=sp4eh545c1avao55yyu23545;
ASPSESSIONIDAQCSBTQA=PMNPGKNAOOFBLBBMGLICKDDD;
Login=CBE6FAFD6339A03DBD599D6EB990AA0B63CF0CAE2D8B16426C8D5DC2EDA59494385AA539
F31CF070B43424F641465AB3CC0C83EA9170412F0B45076B23619E6A4350E17E324E9606
Proxy-Connection: Keep-Alive

```

Note the cookie in the request on the proxy in the two cases (1. and 2.) above. They are the same.

Login is successful. The following screen is displayed.



⇒ The earlier session is not terminated on the server, even when Logout option was selected.

Example 2

An active session was Logged out from and Browser closed. And a new browser session was launched. The following URL was Invoked <http://164.100.9.34/drmreport/menus.aspx>

(this url is normally reached after authentication. However, here it is directly pasted on the browser).

On the proxy the following cookie (recorded from the earlier session) was pasted along with the request

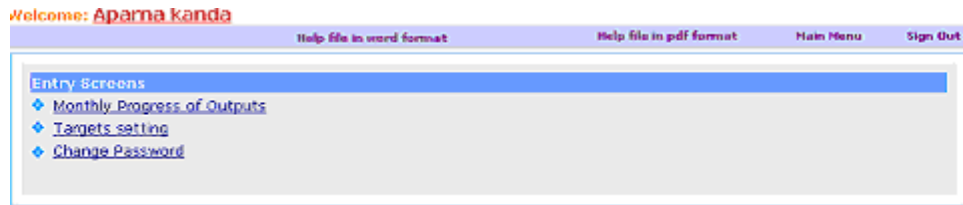
```

Cookie: ASP.NET_SessionId=3qmo2q55bs2jyem1qpia0vup;
Login=56156CC508F5EE27FD195F5E2C92ACE6C0F7DA921E58DB182EDE176DD647C2E2D2689B6C4B
22498F447E3AF5B4781451ADC1CE23146DD4A93BBD9FA18643C842D0FDF3BA2FFFD9CD

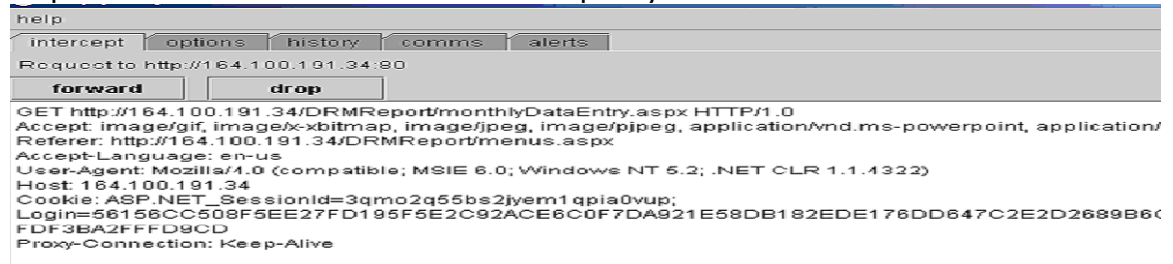
```

The following screen is displayed.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	



Clicking the link monthly progress of output, replace the cookie in the request with the above cookie on the proxy.



Similarly on the next proxy request, replace the cookie with the above cookie.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

burp proxy v1.22

help

intercept options history comms alerts

Request to http://164.100.191.34:80

forward drop

POST http://164.100.191.34/DRMReport/monthlyDataEntry.aspx HTTP/1.0
 Accept: image/gif, image/x-bitmap, image/jpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, *
 Referer: http://164.100.191.34/DRMReport/monthlyDataEntry.aspx
 Accept-Language: en-us
 Content-Type: application/x-www-form-urlencoded
 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR 1.1.4322)
 Host: 164.100.191.34
 Pragma: no-cache
 Content-Length: 2929
 Cookie: ASP.NET_SessionId=3qmo2q55bs2jyem1qpia0vup;
 Login=56156CC508F5EE27FD195F5E2C92ACE6C0F7DA921E58DB182EDE176DD647C2E2D2689B6C4B22498F447E3AF5B4781451ADC1CE23146DD4A93BBD9FA186A
 FDF3BA2FFFD9CD]
 Proxy-Connection: Keep-Alive

__EVENTTARGET=dataLevel%3A_ctl%3AInkModify&__EVENTARGUMENT=&__VIEWSTATE=dDwtMTI3NjkzMjUyODt0PHA8bDxMb2dpbkIE00xY2F0aW9uSUQ7TG9naW5MZA
 BO2w8YXBhcm5hMTtpPDM%2BOzI7Pj47bDxpPDE%2BOz47bDx0PDtsPGk8MT47aTw3PjtPDK%2BO2k8MTE%2BO2k8MTM%2BO2k8MTU%2BO2k8MTk%2BO2k8MJE%2BO
 BOz47bDx0PHA8cDxsPFRIeHQ7PjtsPEFwYXJuYSBRyW5kYTs%2BPjs%2BOzs%2BO3Q8cDxwPGw8VGv4dDs%2BO2w8XGU7Pj47Pjs7Pjt0PHQ8cDxwPGw8RW5hYmxlZDs%
 zxmPjs%2BPjs%2BOztsPGk8Mj47Pj47Oz47dDx0PHA8cDxsPEVUyWJszWQ7PjtsPG88Zj47Pj47Pjs7bDxpPDExPjs%2BPjs7Pjt0PHA8cDxsPFZpc2libGU7PjtsPG88Zj47Pj47Pjs
 8cDxwPGw8RGF0YVRleHRGaWVsZDIeYXRhVmfSdWVGaWVsZDIeFibGvK01Zpc2libGU7PjtsPFNUQvRFx05BTUu7U1RBVEVfSUQ7bzxmPjtvPGY%2BOz4%2BOz47dDxp
 PC1TZWY3Qto0Fzc2Fio0JpaGFyO0RlbgGhp00d1amFyYXQ7TWfOYXJhc2h0cmE7TWVnaGFsYXlh009yaXNzYTTaWtraW07VGfTaWwTmFkdTvdHRhcbGcmFkZkXNoO1V0d
 GFsO1dlc3QgQmVuz2Fs01RyaXB1cmE7TWfuaXB1cjtNaXpvcmtF00FydW5hY2hhbCBGcmFkZkXNoO05hZ2FsyWw5kOz47QDwtMTsxOzI7Mzs0OzU7Njs3Ozg7OTsxMDsxMTsxM
 DsxNTsxNjxNzs%2BPjsPGk8Mz47Pj47Oz47dDxwPDtwPGw8b25jbGjzjs%2BO2w8cmV0dXJlGNoZWNIvMfSdWUoKvW7Oz4%2BPjs7Pjt0PEAwPHA8cDxsPFbHz2VDb3Vu
 W1Db3VudDtlURhdGFTb3VyY2VjdGtQ291bnQ7RGF0YUtleXM7PjtsPGk8MT47aTw2PjtPDY%2BO2w8Pjs%2BPjs%2BOzs%2BOz4%2BOz47bDxpPDA%2BOz47bDx0PDtsPG
 wyPjtPDM%2BO2k8ND47aTw1PjtPDY%2BO2k8Nz47PjtsPHQ8O2w8aTwwPjtPDE%2BO2k8Mj47aTwzPjs%2BO2w8dDxwPHA8bDxUZxh0Oz47bDwXOz4%2BOz47Oz47dD
 DA%2BOz47bDx0PEA8U3RhdGU7Pjs7Pjs%2BPjt0PDtsPGk8MD47PjtsPHQ8QDxOTzs%2BOzs%2BOz4%2BO3Q8O2w8aTwwPjs%2BO2w8dDxwPHA8bDxUZxh0Oz47bDwXO
 7Oz47Pj47Pj47dDw7bDxpPDA%2BO2k8MT47aTwPjtPDM%2BOz47bDx0PHA8cDxsPFRIeHQ7PjtsPDI7Pj47Pjs7Pjt0PDtsPGk8MD47PjtsPHQ8QDxEaXN0cmliDds%2BOzs%
 BO3Q8O2w8aTwwPjs%2BO2w8dDxAPE5Poz47Oz47Pj47dDw7bDxpPDM%2BOz47bDx0PHA8cDxsPFRIeHQ7PjtsPDI7Pj47Pjs7Pjs%2BPjs%2BPjt0PDtsPGk8MD47aTwPjtP
 2k8Mz47PjtsPHQ8cDxwPGw8VGv4dDs%2BO2w8Mzs%2BPjs%2BOzs%2BO3Q8O2w8aTwwPjs%2BO2w8dDxAPEJsb2NrOz47Oz47Pj47dDw7bDxpPDA%2BOz47bDx0PEA8T
 s%2BPjt0PDtsPGk8Mz47PjtsPHQ8cDxwPGw8VGv4dDs%2BO2w8Mzs%2BPjs%2BOzs%2BOz4%2BOz4%2BO3Q8O2w8aTwwPjtPDE%2BO2k8Mj47aTwzPjs%2BO2w8dDw
 UZxh0Oz47bDw0Oz4%2BOz47Oz47dDw7bDxpPDA%2BOz47bDx0PEA8R1A7Pjs7Pjs%2BPjt0PDtsPGk8MD47PjtsPHQ8QDxZRVM7Pjs7Pjs%2BPjt0PDtsPGk8Mz47PjtsPHQ8
 8VGv4dDs%2BO2w8NDs%2BPjs%2BOzs%2BOz4%2BOz4%2BO3Q8O2w8aTwwPjtPDE%2BO2k8Mj47aTwzPjs%2BO2w8dDxwPHA8bDxUZxh0Oz47bDw1Oz4%2BOz47Oz
 DxpPDA%2BOz47bDx0PEA8VmlsbGFnZTs%2BOzs%2BOz4%2BO3Q8O2w8aTwwPjs%2BO2w8dDxAPE5Poz47Oz47Pj47dDw7bDxpPDM%2BOz47bDx0PHA8cDxsPFRIeHQ7
 Pj47Pjs7Pjs%2BPjs%2BPjt0PDtsPGk8MD47aTwPjtPDI%2BO2k8Mz47PjtsPHQ8cDxwPGw8VGv4dDs%2BO2w8Njs%2BPjs%2BOzs%2BO3Q8O2w8aTwwPjs%2BO2w8d
 s%2BOzs%2BOz4%2BO3Q8O2w8aTwwPjs%2BO2w8dDxAFFIUz%2BOzs%2BOz4%2BO3Q8O2w8aTwwPjs%2BO2w8dDxwPHA8bDxUZxh0Oz47bDw2Oz4%2BOz47Oz47P
 w7bDxpPDA%2BOz47bDx0PHA8cDxsPFRIeHQ7PjtsPDA7Pj47Pjs7Pjs%2BPjs%2BPjs%2BPjt0PHA8cDxsPEVUyWJszWQ7PjtsPG88Zj47Pj47Pjs7Pjs%2BPjs%2BPjs%2Bp
 CF0mUkdWFHsDM%3D

The following is displayed.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Welcome: [Aparna kanda](#) [Main Menu](#) [Sign Out](#)

Monthly Progress of Outputs

For Year: *
For Month: *

Click on **Modify** to Add / Edit values for a Level.

S.No	Level	Monthly Progress of Outputs OK	Action
1	State	NO	Modify
2	District	NO	Modify
3	Block	NO	Modify
4	GP	YES	Modify
5	Village	NO	Modify
6	ULB	YES	Modify

* Use <Run Checklist> button to display errors in output's progresses entered .
* Use <Show Monthly progress> button to display & check all your entries before finally submit this month's progress .
* Use <Final Submit> button to Submit this month's progress.

DRM Monitoring System Developed by UNDP

Clicking Modify for the first entry in the above list and replacing the cookie with the above cookie in the request on the proxy

```

help
intercept options history comms alerts
Request to http://164.100.191.34:80
forward drop 18x
GET http://164.100.191.34/DRMReport/getMonthlyDataTemp.aspx?stateid=3&levelid=1&Monthid=11&yearid=2004&userid=130&userLevel=2 HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
Referer: http://164.100.191.34/DRMReport/monthlyDataEntry.aspx
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR 1.1.4322)
Host: 164.100.191.34
Pragma: no-cache
Cookie: ASP.NET_SessionId=3qmo2q55bs2jyem1qpia0vup;
Login=5b156cc508f5ee27fd195f5e2c92ace6c0f7da921e58db182ede176d847c2e2d2689b6c4b22498f447e3af5b4781451adc1ce23146dd4a93bbd9fa18643c
fdf3ba2fffd9cd
Proxy-Connection: Keep-Alive

```

The form is displayed as follows.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Monthly Progressive Outputs

Note :- Values above the input boxes are the last month's figures. Current values should be greater or equal to last month's figures.

Year	2004	T O T A L O U T P U T D A T A	
Month	11		
State	Delhi		
Level	State		

Outputs

Senior Officers and policy makers (including legislators) oriented	2004	100
State Disaster Management Policy approved (yes = 1, no = 0)		1
State Disaster Management Act enacted (yes = 1, no = 0)		0
State Disaster Management Authority set up (yes = 1, no = 0)		1
Redesignation of nodal department as department for DM (yes = 1, no = 0)		0
State disaster Management plan finalized (yes = 1, no = 0)		0

Display Targets Save Data

Changed data to 100 in the last entry and clicking Save and replacing the session cookie in the request.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Monthly Progressive Outputs

Note :- Values above the input boxes are the last month's figures. Current values should be greater or equal to last month's figures.

Year	2004	T O T A L U P T O D A T E	
Month	11		
State	Delhi		
Level	State		
Outputs			
l, no = 0)	<input type="text" value="0"/>		
Strengthening of a state training institute in DM (yes = 1, no = 0)	<input type="text" value="0"/>		
Master trainers trained at State level	<input type="text" value="180"/>		
Professional bodies and corporate sector sensitized	<input type="text" value="8"/>		
Manuals and SOP finalized and approved for multiple hazards	<input type="text" value="0"/>		
Massive awareness campaigns conducted (TV, Radio, newspaper)	<input type="text" value="1"/>		
	<input type="text" value="100"/>		

Display Targets Save Data

Monthly Progressive Outputs [Main Menu](#) [Entry screen](#) [Sign Out](#)

Thanks for Saving the Data. Now you can Add Data for different Levels

[Go Back to Entry screen](#)

Project Supported By UNDP

The data is saved.

⇒ **The session is not invalidated when Sign Out option is clicked by the user. This is due to Improper session handling.**

What has been done in the above steps is for the requests generated, the session cookie is replaced with a session cookie which is still valid on the server. As a result, the server treated each request as a genuine request of that of an authenticated user.

Solution :

1. Generate a unique session id for each login.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

2. Invalidate the session on the server application when Sign Out option is clicked by the user. The session may also be invalidated after a certain time of inactivity.

Password Strength: Certain web application were found to have user credentials hardcoded in the client side code of the login page. Any one visiting the authentication page and examining the client side page source code could see the user credentials. This defeats the purpose of the authentication of the users. The site must have a password policy well laid out.

1. Password should be enforced to be of a minimum length and comprising of mix of alphabets, numbers and characters.
2. Users should be required to change their passwords periodically
3. Avoid reuse of previous passwords.

Password Use: A certain critical web application was found to have enforced restricted access to functions and resources through authentication and access control. However, there was no proof of site visitors having attempted password guessing or failed login attempts.

1. System to include audit logs to monitor system access and to detect failed login attempts.
2. Failed login attempts must be logged and a minimum number of failed login attempts must be allowed in a unit of time.
3. System should not tell whether it was the user name or password that failed.
4. The password should not be logged (as anyone gaining access to the logs can gain access to the user password).
5. A user must be informed of last time of successful login and no of failed login attempts since then.

Password Change Control

1. Whenever a password change is required, both the old and new password have to be required to be given.
2. If a forgotten password is required to be mailed to a user, then any change of email address (any account management function) should necessitate a re-authentication. This is so as to prevent walkby attacks where for a logged in session, a casual user walks in and performs profile change function or changes the e-mail address. If a request for mailing the forgotten password is made then the password would be sent to the changed email address.
3. A single password change mechanism must be used wherever it is used.

Name of the Document	Secure Programming Guidelines		
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Password Storage:

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

4. Cross-Site Scripting (XSS) Flaws

The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine or spoof content to fool the user.

Example:

A malicious user causes inconvenience to the intended recipient of the feedback

One of the website has an application for 'Write to Governor' that receives feedback from the user. The feedback form performs no input validation. A malicious user can make use of this flaw to insert malicious scripts in the application. For instance, a script could be written that may take the viewer of the feedback data to another website or flash unwanted Pop up messages.

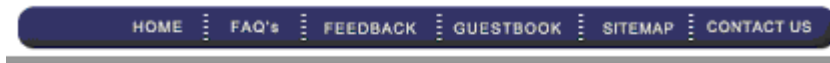
The data posted from the feedback form, when viewed, may execute scripts that were inserted by the malicious user. For example, the script injected may redirect to an unwanted web site. The pre-requisite for the scripts to execute is that scripting must be enabled on the mail client used to view the user's feedback.

The sample screen dump illustrates one of the sites where a script could be injected into a guestbook entry.

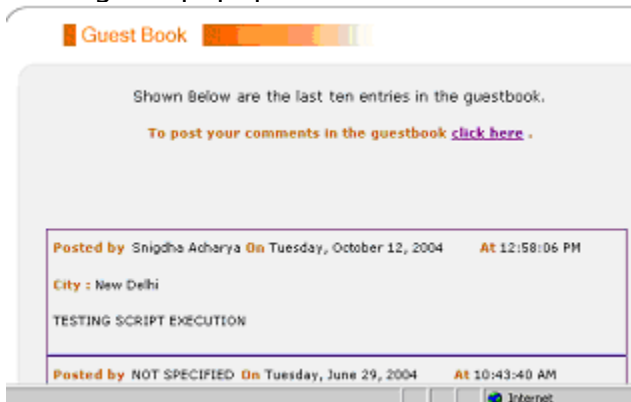
The screenshot shows a web form titled "Guest Book". It contains several input fields: "First Name" (Snigdha), "Last Name" (Acharya), "E Mail" (snigdha.acharya@nic.in), and "City" (New Delhi). Below these is a "Your Message" section with a text area containing the following code: `<script>alert("HELLO WORLD");</script> TESTING SCRIPT EXECUTION`. At the bottom of the form are two buttons: "Submit My Message" and "Clear Form".

On visiting the guestbook to view the guestbook entries the following was encountered.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	



Closing the popup then showed the guestbook entries as shown.



This popup was the result of the previous script injection which now executed in the browser context of the viewer. A variation of this attack would be to present multiple popups and cause inconvenience to visitors visiting this page.

Instead of flashing a popup, a script to redirect to an unwanted site could have been injected. As a result the guestbook could not have been available for viewing.

Another possible impact of this attack causes unwanted/junk mails to be sent to the user viewing the feedback. In an event of script execution, it can cause unwarranted actions on behalf of the user. Due to this legitimate feedback takes time to be acted upon and is of nuisance value.

Solution: Apply strong input validation to be applied on the variables. The application must provide strong Input/Output validation checks across all the fields in the feedback form page. It should specifically disallow all special characters in the "Name", "Tel" and "Feedback" fields. For the "Tel" field it must only allow numeric input to be accepted - telephone number is normally

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

numeric. In the "Feedback" field it must limit the number of characters to be taken as feedback to a minimum number. All these validations have to be performed on the server side. Refer to **Input validations** in **Appendix**.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

5. Buffer Overflow

In these kind of attacks, the attacker sends an arbitrarily large input to the application. By sending carefully crafted input in the web application, the attacker can execute arbitrary code and corrupt the execution stack of the application and causing the application to execute the code that was inserted in the input- effectively taking over the machine.

These kinds of attacks effect all kind of environments. These are specifically seen in web applications using C for development. For example, when calls to C statement like 'gets' etc are made that becomes a potential for this kinds of attack. This requires attackers to have expertise in understanding the application or product and then exploiting the flaw. Buffer overflow can be present in both the web server and application server products. These vulnerabilities for widely used server products can rapidly be known and pose significant risks to user of these products.

Solution: It is recommended to apply latest patches for the products. For applications, review the code which accept input thru HTTP requests. Size checking on all such inputs is required.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

6. Injection Flaws

This kind of vulnerability allows an attacker to attack backend components interfaced to by web applications. Examples included Databases, calls to OS commands and external programs like send mail. The web application is used to transport the attack to the backend SQL server or the external program. This attack assumes a common form as SQL injection.

SQL injection is a technique using which an attacker can inject a string such that the embedded SQL statement for the application gets modified. The string so injected, modifies the SQL statement being executed and/or causes extra SQL statements of the attacker's choice to execute.

Example:

1. A malicious user can log in without a valid account

One of the web site has an application which is accessible to different users after authentication. However, it was observed that this authentication could be bypassed to access the application's privileged functions. This application was vulnerable to SQL injection attack. The user could bypass the authentication control mechanism of the application and login to the Privileged section of the site using SQL injection. It was possible for attackers to exploit this flaw to get access to the application and to gather more information and perform unauthorized functions.

Logging in without any credentials by simply injecting the following patterns.

Enter your Username and password to sign in

Username :

Password :

Please Use Internet Explorer 5.5 or above

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

The user is taken to the subsequent page. This was possible because the query for looking up the user name and password as bound in the authenticating application was modified by the string pattern. When the modified query was executed, the entire tables records were returned and from this recordset the first user's details were made available as per the application logic.



Solution

Perform input validations for user input and perform roles-user- authorisation checks to allow access to the application. Refer to Recommendations section for guidelines on Input Validation.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

7. Error Handling Problems

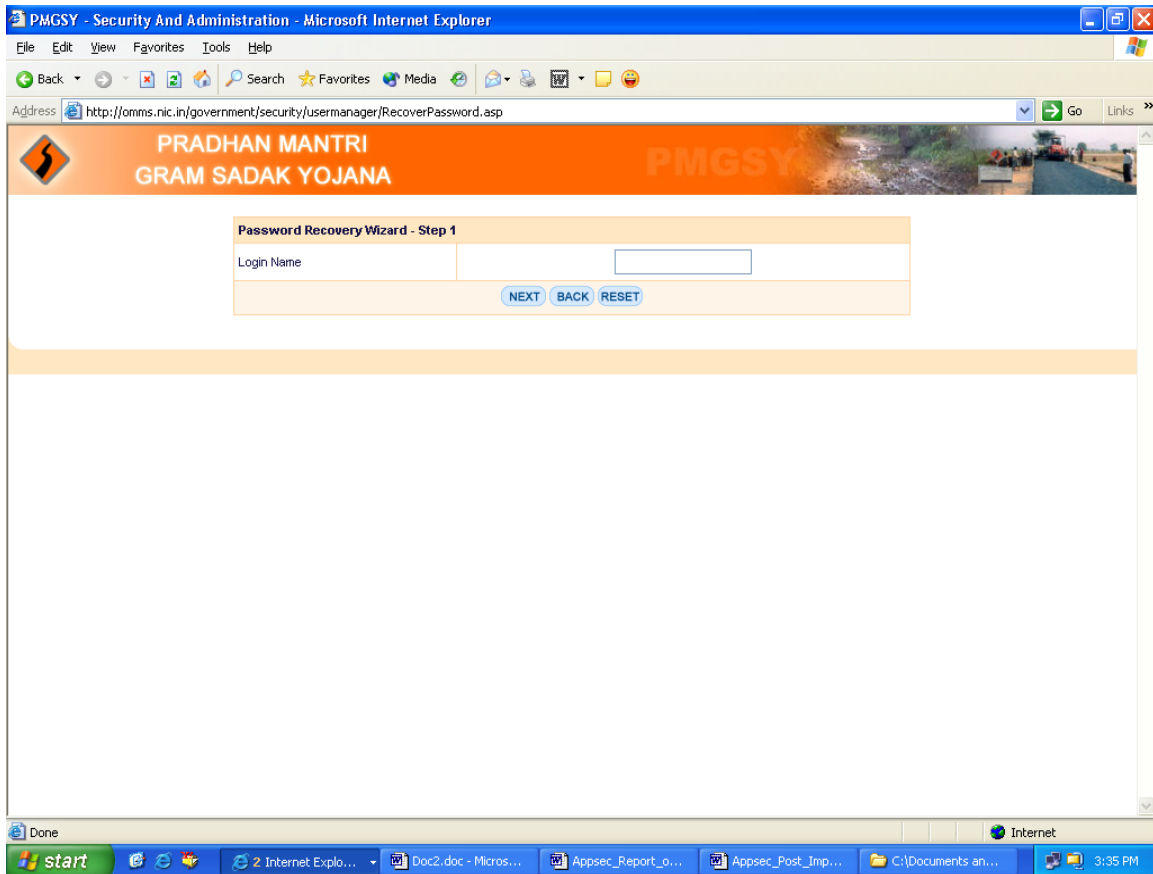
Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, and cause security mechanisms to fail, or crash the server.

Example: Malicious users can enumeration of backend database information.

It was observed from the sites audited that it was possible for a malicious user to enumerate backend information from the application. This was achieved by sending well-crafted strings to the server as part of the input to the server. Taking advantage of the detailed error messages generated from the database server, backend information could be obtained. The malicious user can use the fact that the application did no error handling on the server side and threw back all the detailed errors to the user's browser. Moreover, due to no input validation, the application passed all the queries of the malicious user to the backend database that eventually resulted in enumeration of the column names, table names and the SQL queries. This

Name of the Document	Secure Programming Guidelines		
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

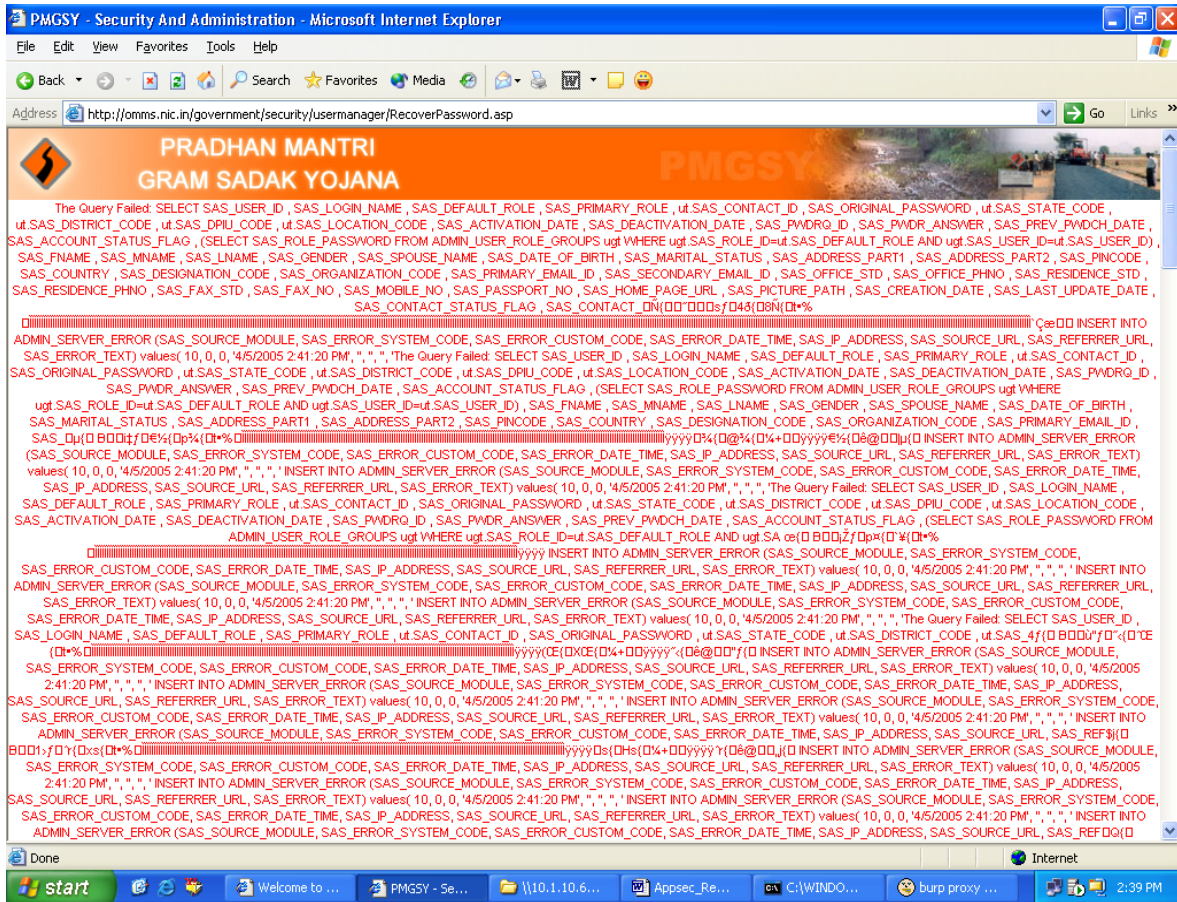
information may be useful for the attacker to carry out further attacks.



Injecting unwanted data in the login name field.

The following snap shot shows the output of the previous step due to lack of error handling mechanism and improper input validation. It displays entire SQL query that is being used by the application with table names and column names.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	



Solution

1. Apply strong input validation. Refer to **Appendix for Input Validation**.
2. Error handling must be performed through well thought out scheme and the following must be taken care of:
 - a. All exceptions that can be possibly generated by the application must be taken care of and handled in the application.
 - b. Custom error messages must be built for each kind of exception.
 - c. Errors generated by the application, should provide necessary message to user, information to developers and absolutely nothing to attackers.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

8. Insecure Storage

This vulnerability deals with improper protection of sensitive information like password, or any other sensitive information.

This includes failure to encrypt critical data, insecure storage of secrets in memory, insecure storage of passwords, keys, certificates etc.

Example: A malicious user can steal user credentials

The username and password of a user can be stolen by launching memory-editing tools. Whenever a user authenticates using a pair of username/password, the credentials are stored in the physical memory (RAM) used by the browser. If the user has not closed the browser window then another user with malicious intent can launch a memory editing/viewing tool and grab the username/password of the last logged in user from the physical memory (RAM).

The following screen dump shows a memory editing tool used to view the credentials from an instance of the browser from which the user had provided authentication credentials.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
001B4960	41	42	67	42	49	41	41	41	41	44	67	41	4F	41	45	34	ABgBIAAAADgAOAE4	
001B4970	41	41	41	41	55	41	42	51	41	58	41	41	41	41	41	41	AAAAUABQAXAAAAAA	
001B4980	41	41	41	43	67	41	41	41	41	42	59	4B	49	6F	67	55	AAACgAAAABYKIogU	
001B4990	43	7A	67	34	41	41	41	41	50	62	67	42	70	41	47	4D	Czg4AAAAFPbgBpAG	
001B49A0	41	62	67	42	70	41	47	4D	41	4D	77	41	77	41	44	63	AbgBpAGMAMwAwADc	
001B49B0	41	4E	67	42	44	41	46	4D	41	52	77	42	51	41	45	55	ANgBDAFMARwBQAEU	
001B49C0	41	54	67	42	55	41	45	55	41	55	77	42	55	41	43	4D	ATgBUAEUAUwBUAC	
001B49D0	6E	58	58	33	6A	46	48	4D	36	41	41	41	41	41	41	41	nXX3jFHM6AAAAAAAE	
001B49E0	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	45	73	AAAAAAAAAAAAAAAAAE
001B49F0	35	4E	51	47	2B	4F	66	69	54	4A	5A	37	4D	31	53	45	5NQG+OfiTJZ7M1SE	
001B4A00	65	69	77	69	69	2B	38	61	69	76	79	33	66	64	70	3D	eiwii+8aivy3fdp=	
001B4A10	3D	0D	0A	43	6F	6F	6B	69	65	3A	20	50	48	50	53	45	=..Cookie: PHPSE	
001B4A20	53	53	49	44	3D	63	63	39	30	31	33	35	30	65	62	33	SSID=cc901350eb3	
001B4A30	36	33	66	61	66	65	32	30	65	30	33	64	36	31	61	65	63fafa20e03d61ae	
001B4A40	64	38	32	36	39	3B	20	63	73	64	3D	31	35	0D	0A	0D	d8269; csd=15...	
001B4A50	0A	73	74	61	74	75	73	3D	63	68	65	63	6B	26	75	73	.status=check&us	
001B4A60	65	72	6E	61	6D	65	3D	64	65	6C	68	69	26	70	61	73	ername=delhi&pas	
001B4A70	73	77	6F	72	64	3D	64	65	6C	68	69	26	53	75	62	6D	sword=delhi&Subr	
001B4A80	69	74	3D	4C	6F	67	69	6E	05	00	70	00	97	01	08	00	it=Login..p.!	
001B4A90	00	00	00	00	EC	DD	03	00	78	45	1E	00	98	B8	1A	00iY..xE..!	
001B4AA0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Solution

1. The solution to the above is to implement the salted MD5 technique. Refer to Appendix for details.

9. Denial of Service

Web applications are susceptible to denial of service attacks. Once an attacker can consume all or some of the required resources, they can

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

prevent legitimate users from accessing the system. Attackers can cause the user accounts to be locked or even the entire application to fail.

A malicious user may cause a denial of service attack by automated postings. The 'write to governor' application is vulnerable to automated posting attack. This may lead to a denial of service on the associated E-Mail infrastructure by causing flooding of unwanted mails to a particular email id as bound in the application. A DOS attack may cause non-availability of the email infrastructure, as the associated mail servers are busy processing This may lead to a situation where a genuine feedback may not be received. The malicious user needs to run a script which would send arbitrary requests to the application continuously. Since all the requests are sent onto an email id, this may cause a heavy load on the e-mail infrastructure. This attack requires knowledge of programming and the understanding the workflow of the application.

A script can be run that will keep entering inputs into the form. This will keep the email infrastructure busy and as such it will not be able to process the genuine feedbacks. (Similar tests have been performed in the lab).

Solution : Implement a challenge/response mechanism in the feedback form as a protection against automated postings.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

10. Insecure Configuration Management

Configuration of web server and application server is critical to the security of the hosted web sites and application. In addition many services are offered by application servers, which are used by web applications. Very often the application owner are separate from the group running the site. Web application security concerns often span this gap and require members from both side to properly ensure security of a site's applications.

Server configuration problems include

- 1.Unpatched flaws in the server s/w
- 2.Server s/w flaws or misconfiguration that allow directory listing or directory traversal
- 3.Improper file and directory permission
- 4.Unnecessary services enabled including remote administration
- 5.Default a/cs with default passwords

Example : A malicious user can upload malicious contents.

In one of the sites audited the following was observed:

1. Tools deployed had successfully been able to upload files on the hosting web server. A malicious user could deface the web site with such uploads. This is due to permissions anomaly on the web server hosting the application.
2. The application also provided a facility for uploading files. However, the applications for this facility could be directly invoked without authenticating. So the user could upload any document of his choice. He could also execute scripts of his choice. The folder had combination of permission for write as well as execute which was a harmful combination.

Test 1: Tool based Hacks : The files were uploaded by the tool could be invoked from the web site as a URL.



Notice:

This is an HTML file that was loaded as a result of exploiting an existing vulnerability in the Web Application.

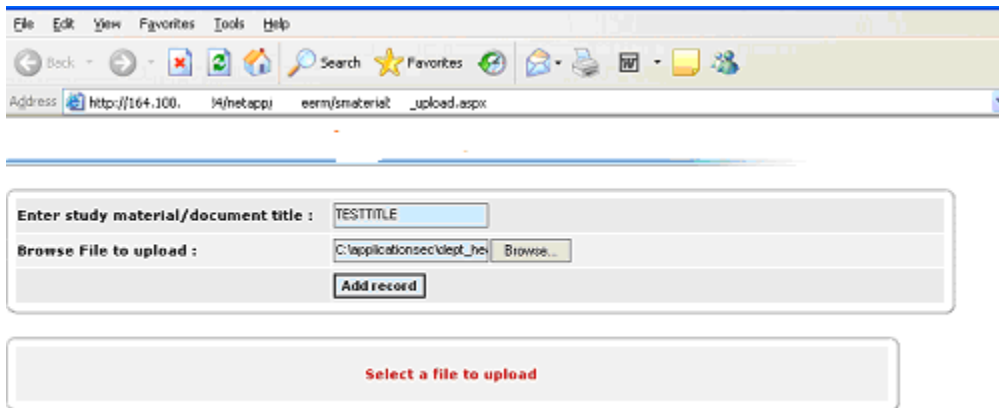
In other words, this site was:

**SUCCESSFULLY
HACKED!**

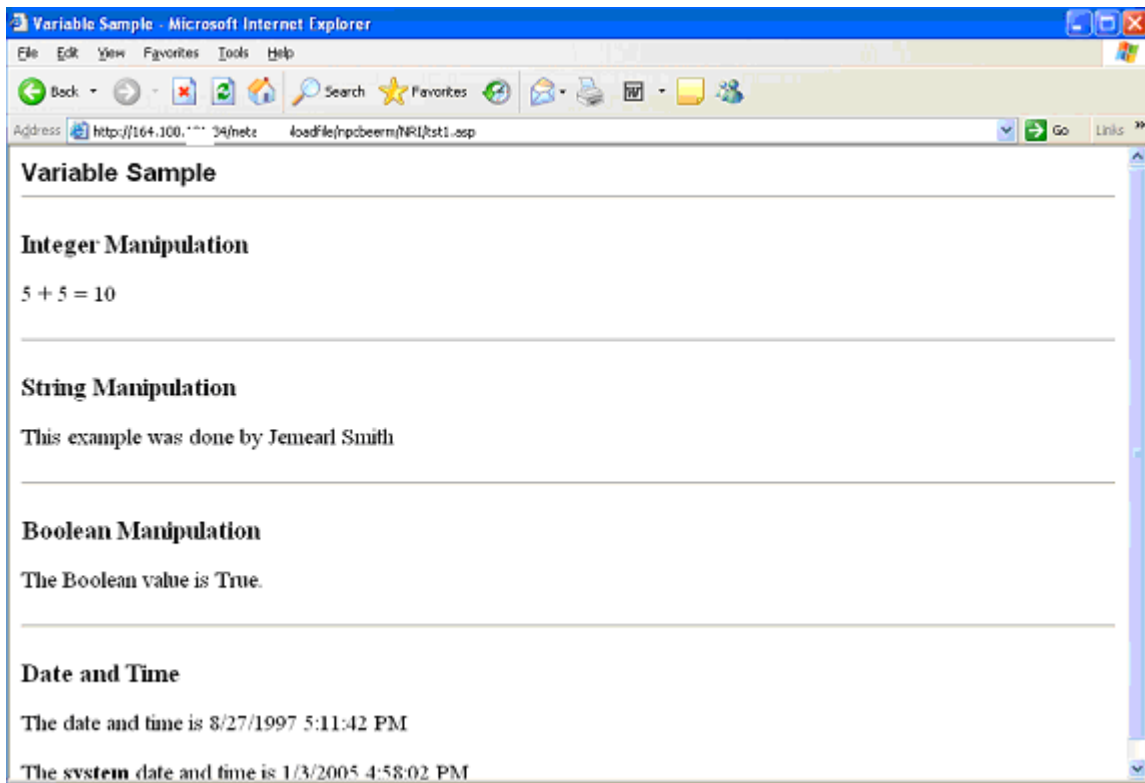
Test 2:. Application based upload:

From the browser history the URL for file upload could be found.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	



A file filestst1.asp was uploaded. Which was later invoked as shown



File uploaded could be executed.

This means a malicious content based file also can be uploaded and executed.

Solution :

Disable write and Delete permission for the folders on the web server hosting the application.

Assess the requirement of the application for file upload facility. In this case, the directory used for file repository must not be in the direct path of the web application. The permissions on this folder must only be having write privilege for the

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

account used by the application. Note: A combination of write and execute permissions on web hosting folders is strictly forbidden.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

Appendix

A. Input Validations

5. Perform input validations both on client as well as server side.
6. Perform input validations mandatorily on server side
7. Perform type checking of parameter/input field data. Ex: String, Numeric, alphabetic etc. Check for integer type if the input expected is integer type or else reject it

Set data type

Reinforce the previous step by explicitly setting the type.

```
Eid = Cint(request("eid"))
```

```
Nm = Cstr(request("name"))
```

4. Perform length checking of data

5. Accept Only Known Valid Data : Restrict input to an allowed character set. A character set may be defined for each field where input from the user is accepted. E.g. "A-Z, a-z, @, ., 0-9, _" is a character set for a field that accepts user email.. Example server side code snippet in java to illustrate restricting input in an address field from a form to the allowed character set.

```
try
{
    int i, j;
    char ch;
    String val;
    String check = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
1234567890. "
    val=(String)req.getParameter("Address");
    if(val.length()==0)
        throw new UnsupportedOperationException("demo");

    for (i = 0; i < val.length(); i++)
    {
        ch = val.charAt(i);
        for (j = 0; j < check.length(); j++)
        { if (ch == check.charAt(j))
            break;}
        if (j == check.length())
            throw new IndexOutOfBoundsException("demo");
    }
}
try
{ if(val.length() > 50)
    throw new IllegalArgumentException("demo");
}
catch(IllegalArgumentException e)
{ handleError (e , res, msg1, url1);
  return;
}
```

Note : handleError above is a customized function to display an error message.

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

6. Reject Known Bad Data

A character set of bad data may be defined for the site that has to be rejected. E.g. "CREATE, DROP, OR"

7. Check whether NULL values are allowed

8. Check whether duplicates are allowed

9. Check numeric range. If a parameter accepts numeric input it makes sense to check that the numeric input is in an allowed range.

For example: A popular web site had categories of inventory represented in numeric form. So any item falling in a category was represented with the number representing the category. However, it was found that it was possible to enter a fictitious item entry with a large number representing the category. The database had no category represented by this number. This implies that the data in the database is not validated and is not accurate.

10. Validate data

Make sure the data is validated against a positive specification that define specific patterns(or regular expressions). That is a set of regular patterns may be enumerated. Data may be validated to be using these regular patterns only. Any other patterns may not be accepted.

11. Sanitize Known Bad Data

A character set of bad data is defined and any input field that has such a character is modified. E.g. "If there is a single quote (') in the data, it is replaced with two single quotes.

Remove apostrophes, or replace them with double quotes.

```
Function quotehandle(data)
  Quotehandle=replace(data,"'","''")
End function
```

Remove semi colons and double dashes

```
Function sqldash(data)
  Sqldash=replace(data,"-",",")
End function
```

```
Function sqlcolon(data)
  Sqldash=replace(data,";",",")
End function
```

12. It is recommended to use the strategy of "Accept only known data".

Further all the allowed input/output data must be sanitized on the server side

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

by replacing scripts tags, sent as part of user input/output, with appropriate representations.

. For example

"<" by <

">" by >

"(" by (

This would avoid scripts from being executed on the client side.

13. Client side input must also be checked for URL encoded data. URL encoding sometimes referred to as percent encoding, is the accepted method of representing characters within a URI that may need special syntax handling to be correctly interpreted. This is achieved by encoding the character to be interpreted with a sequence of three characters. This triplet sequence consists of the percentage character "%" followed by the two hexadecimal digits representing the octet code of the original character. For example, the US-ASCII character set represents a space with octet code 32, or hexadecimal 20. Thus its URL-encoded representation is %20.

Other common characters that can be used for malicious purposes and their URL encoded representations are: -

#	Character	URL encoded
1.	'	%27
2.	"	%22
3.	;	%3b
4.	<	%3c
5.	=	%3d
6.	>	%3e
7.)	%29
8.	(%28

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

All input validation checks should be completed after the data has been decoded and validated as acceptable content (e.g. maximum and minimum lengths, correct data type, does not contain any encoded data, textual data only contains the characters a-z and A-Z etc.)

14. A one-time check on the database is to be made for invalid malicious data. This would enable removal of input that has not been validated in earlier sessions. As otherwise the invalid data may cause script execution on the users browser.

B. Challenge Response Mechanism

Automated registrations can cause a DOS attack. To avoid such an attack, an interactive component such as a challenge-response mechanism should be incorporated in the form where such an attack is possible. This ensures that a user has to manually enter the data.

The challenge is a randomly generated unique string superimposed on an image. This string is displayed on the form where user input is required. The user has to then enter this string along with the data he would like to fill in the form. This data received from the user must be processed only if the string that the user enters matches with the one that was displayed to him.

A mechanism to maintain the association of the text displayed by the server and the text entered by the user should be established.

C. Salted MD5

MD5 hash is a cryptographic technique in which an input when subjected to a hash function, a fixed length data pattern called the MD5 hash is generated. The actual value can never be recovered from this hash. The hash is a one way function. Varying a bit in the input when hashed, results in a varied hash output.

For implementation of salted MD5, the pre-requisite is that the backend database stores a MD5 hash of the password. Here is how the salted MD5 technique works:

Name of the Document		Secure Programming Guidelines	
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

When a client requests for the login page, the server generates a random number, the salt, and sends it to the client along with the page. A JavaScript code on the client computes the MD5 hash of the password entered by the user. It then concatenates the salt to the hash and re-computes the MD5 hash. This result is then sent to the server. The server picks the hash of the password from its database, concatenates the salt and computes the MD5 hash. If the user entered the correct password these two hashes should match. The server compares the two and if they match, the user is authenticated.

Name of the Document	Secure Programming Guidelines		
Classification	<i>Restricted</i>	Audience	<i>NIC web application developers</i>
Version	<i>1.0</i>	Data of last change	<i>May 4 2005</i>
Status	<i>Draft</i>	Document No.	

References:

1. <http://owasp.org>
Owasptop10.pdf
2. Audited web Site reports