Web Application Report

This report includes important security information about your web application.

Security Report

This report was created by HCL AppScan Standard 10.0.6
Scan started: 1/10/2022 5:24:54 PM

Table of Contents

Introduction

This report contains the results of a web application security scan performed by HCL AppScan Standard.

High severity issues:	5
Medium severity issues:	57
Total security issues included in the report:	62
Total security issues discovered in the scan:	114

General Information

Scan file name:	survey_f_line_modified_10-01-2022
Scan started:	1/10/2022 5:24:54 PM
Test policy:	Application-Only
Test optimization level:	Fast

Host	10.163.30.226
Port	8088
Operating system:	Unknown
Web server:	Unknown
Application server:	JavaAppServer

Login Settings

Login method:	Recorded login
Concurrent logins:	Enabled
In-session detection:	Enabled
In-session pattern:	`null`
Tracked or session ID cookies:	`JSESSIONID` `JSESSIONID`

Tracked or session ID parameters:	`jsoncallback`
	`jsoncallback`
	`jsoncallback`
	`_`
	`jsoncallback`
	`jsoncallback`
	`jsoncallback`
	`jsoncallback`

Login sequence:

http://10.163.30.226:8088/survey_f_line_audit/
http://10.163.30.226:8088/survey_f_line_audit/CaptchaTest
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/?jsoncallback=jQuery21102899024832830397_1641815324531
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/?jsoncallback=jQuery21102899024832830397_1641815324531
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21102899024832830397_1641815324531&_=1641815324532
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21102899024832830397_1641815324533&_=1641815324534
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21102899024832830397_1641815324535&_=1641815324536
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21102899024832830397_1641815324537&_=1641815324538
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21102899024832830397_1641815324539&_=1641815324540
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21102899024832830397_1641815324541&_=1641815324542
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21102899024832830397_1641815324543&_=1641815324544
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/swservice/getDistrictUniCodeDetails

https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/sw
service/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/get_firka_name/15/04/03?
jsoncallback=jQuery21102899024832830397_1641815324545&_=164181532454
6
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/sw
service/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/sw
service/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/get_firka_name/15/04/03?
jsoncallback=jQuery21102899024832830397_1641815324547&_=164181532454
8
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/sw
service/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/sw
service/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/get_firka_name/15/04/03?
jsoncallback=jQuery21102899024832830397_1641815324549&_=164181532455
0
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/sw
service/getDistrictUniCodeDetails
https://edistricts.tn.gov.in:8443/survey_f_line_service/resources/sw
service/getTalukUnicode/15
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/get_firka_name/15/04/03?
jsoncallback=jQuery21102899024832830397_1641815324551&_=164181532455
2
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/getLoginRole/?
jsoncallback=jQuery21102899024832830397_1641815324553
http://10.163.30.226:8088/survey_f_line_audit/firka_surveyor_wf.jsp
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/getApp?jsoncallback=jQuery21107474938346486029_1641815341696
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/getApp_count/no/?
jsoncallback=jQuery21107474938346486029_1641815341696
http://10.163.30.226:8088/survey_f_line_audit/firka_surveyor_survey_
wise.jsp
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_
det/getAppDet_surveywise/?
jsoncallback=jQuery21104092514116324690 3_1641815402224

Summary

Issue Types 4

TOC

	Issue Type	Number of Issues
H	Application Error	5
M	Link Injection (facilitates Cross-Site Request Forgery)	19
M	Phishing Through Frames	19
M	Reflected Cross Site Scripting	19

Vulnerable URLs 19

TOC

	URL	Number of Issues
H	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App	6
H	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/	5
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_DIS_App	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_TSLR_AppDet	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getDISRemarks	3
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks	3

M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks_returned	3	
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getTSLR_Remarks	3	
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_Report	3	
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_attachments	3	
M	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_ins	3	

Fix Recommendations 2

TOC

	Remediation Task	Number of Issues
H	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions	5
M	Review possible solutions for hazardous character injection	57

Security Risks 4

TOC

	Risk	Number of Issues
H	It is possible to gather sensitive debugging information	5
M	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.	38
M	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user	38
M	It is possible to upload, modify or delete web pages, scripts and files on the web server	19

Causes 7

TOC

	Cause	Number of Issues
H	Proper bounds checking were not performed on incoming parameter values	5
H	No validation was done in order to make sure that user input matches the data type expected	5
M	Sanitation of hazardous characters was not performed correctly on user input	38
M	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser.	19
M	In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data.	19
M	In reflected attacks, an attacker tricks an end user into sending request containing	19

	malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser.		
M	The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.	19	

WASC Threat Classification

TOC

Threat	Number of Issues
Content Spoofing	38
Cross-site Scripting	19
Information Leakage	5

Issues Sorted by Issue Type

H Application Error 5 TOC

Issue 1 of 5 TOC

Application Error

Severity:	High
CVSS Score:	0.0
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/
Entity:	->"pass" (Parameter)
Risk:	It is possible to gather sensitive debugging information
Cause:	Proper bounds checking were not performed on incoming parameter values No validation was done in order to make sure that user input matches the data type expected
Fix:	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

Difference: **Parameter** `->"pass"` manipulated from: `e5631953ffda425f2a62283ad1a27d06b126337455ca14124cad7dcd24f3f711` to: `%00`

Reasoning: The application has responded with an error message, indicating an undefined state that may expose sensitive information.

Raw Test Response:

```
...

Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "user_name": "**CONFIDENTIAL 0**",
 "pass": "\u0000"
}

HTTP/1.1 500
Content-Length: 3397
Location: /survey_f_line_service_audit/eror.jsp
Content-Language: en
Set-Cookie: JSESSIONID=0918DA4F1D08731AB91DF4751275A894; Path=/survey_f_line_service_audit; HttpOnly
Connection: close
Date: Tue, 11 Jan 2022 05:49:28 GMT
Content-Type: text/html;charset=utf-8

<!doctype html><html lang="en"><head><title>HTTP Status 500 - Internal Server Error</title><style type="text/css">body
{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-colo
...
```

Issue 2 of 5

TOC

Application Error

Severity:	High
CVSS Score:	0.0
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/
Entity:	->"user_name" (Parameter)
Risk:	It is possible to gather sensitive debugging information
Cause:	Proper bounds checking were not performed on incoming parameter values No validation was done in order to make sure that user input matches the data type expected
Fix:	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

Difference: **Parameter** `->"user_name"` manipulated from: `**CONFIDENTIAL 0**` to: `%00`

Reasoning: The application has responded with an error message, indicating an undefined state that may expose sensitive information.

Raw Test Response:

```
...

Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "user_name": "\u0000",
 "pass": "e5631953ffda425f2a62283ad1a27d06b126337455ca14124cad7dcd24f3f711"
}

HTTP/1.1 500
Content-Length: 3397
Location: /survey_f_line_service_audit/eror.jsp
Content-Language: en
Set-Cookie: JSESSIONID=68F61C98F2FF1A13C290C83D4DF6D71E; Path=/survey_f_line_service_audit; HttpOnly
Connection: close
Date: Tue, 11 Jan 2022 05:49:23 GMT
Content-Type: text/html;charset=utf-8

<!doctype html><html lang="en"><head><title>HTTP Status 500 - Internal Server Error</title><style type="text/css">body
{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-colo
...
```

Issue 3 of 5

TOC

Application Error

Severity:	High
CVSS Score:	0.0
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App
Entity:	->"frmDate" (Parameter)
Risk:	It is possible to gather sensitive debugging information
Cause:	Proper bounds checking were not performed on incoming parameter values No validation was done in order to make sure that user input matches the data type expected
Fix:	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

Difference: **Parameter** `->"frmDate"` manipulated from: `01/08/2021` to: `%00`

Reasoning: The application has responded with an error message, indicating an undefined state that may expose sensitive information.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "frmDate": "\u0000",
 "toDate": "10/01/2022",
 "search_code": "DATE"
}

HTTP/1.1 500
Connection: close
Content-Length: 2828
Content-Language: en
Set-Cookie: JSESSIONID=1DDEA8CE4B0C847F6A47117E6463E90C; Path=/survey_f_line_service_audit; HttpOnly
Date: Tue, 11 Jan 2022 06:17:42 GMT
Content-Type: text/html;charset=utf-8

<!doctype html><html lang="en"><head><title>HTTP Status 500 - Internal Server Error</title><style type="text/css">body
{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-
size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-
color:#525D76;border:none;}</style></head><body><h1>HTTP Sta
...
```

Issue 4 of 5

TOC

Application Error

Severity:	High
CVSS Score:	0.0
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App
Entity:	->"toDate" (Parameter)
Risk:	It is possible to gather sensitive debugging information
Cause:	Proper bounds checking were not performed on incoming parameter values No validation was done in order to make sure that user input matches the data type expected
Fix:	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

Difference: **Parameter** `->"toDate"` manipulated from: `10/01/2022` to: `%00`

Reasoning: The application has responded with an error message, indicating an undefined state that may expose sensitive information.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "frmDate": "01/08/2021",
 "toDate": "\u0000",
 "search_code": "DATE"
}

HTTP/1.1 500
Connection: close
Content-Length: 2828
Content-Language: en
Set-Cookie: JSESSIONID=3A64CC6A1DA426E1247E8C7680961B5F; Path=/survey_f_line_service_audit; HttpOnly
Date: Tue, 11 Jan 2022 06:18:53 GMT
Content-Type: text/html;charset=utf-8

<!doctype html><html lang="en"><head><title>HTTP Status 500 - Internal Server Error</title><style type="text/css">body
{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-
size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-
color:#525D76;border:none;}</style></head><body><h1>HTTP Sta
...
```

Issue 5 of 5

TOC

Application Error

Severity:	High
CVSS Score:	0.0
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App
Entity:	->"search_code" (Parameter)
Risk:	It is possible to gather sensitive debugging information
Cause:	Proper bounds checking were not performed on incoming parameter values No validation was done in order to make sure that user input matches the data type expected
Fix:	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

Difference: **Parameter** `->"search_code"` manipulated from: `->"search_code"` to: `__ORIG_VAL__.`

Reasoning: The application has responded with an error message, indicating an undefined state that may expose sensitive information.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "frmDate": "01/08/2021",
 "toDate": "10/01/2022",
 "search_code.": "DATE"
```

```
}

HTTP/1.1 500
Connection: close
Content-Length: 2690
Content-Language: en
Set-Cookie: JSESSIONID=43D30DB67CDB419405F73D478783D88A; Path=/survey_f_line_service_audit; HttpOnly
Date: Tue, 11 Jan 2022 06:19:01 GMT
Content-Type: text/html;charset=utf-8

<!doctype html><html lang="en"><head><title>HTTP Status 500 – Internal Server Error</title><style type="text/css">body
{font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-color:#525D76;} h1 {font-size:22px;} h2 {font-
size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {height:1px;background-
color:#525D76;border:none;}</style></head><body><h1>HTTP Sta
...
```

M Link Injection (facilitates Cross-Site Request Forgery) 19 TOC

Issue 1 of 19 TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery211057065805348576 77_1641881499581` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF1430.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 2 of 19 TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF2572.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 3 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF1592.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 4 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF1664.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 5 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF1922.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 6 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_ins
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF1834.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 7 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_attachments
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `%22%27%3E%3CA+HREF%3D%22%2FWF_XSRF2337.html%22%3EInjected+Link%3C%2FA%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 8 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_Report
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF2158.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 9 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF2806.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 10 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`%22%27%3E%3CA+HREF%3D%22%2FWF_XSRF3313.html%22%3EInjected+Link%3C%2FA%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 11 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery211057065805348576 77_1641881499581` to:
`%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF3025.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 12 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getTSLR_Remarks
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery211037178639015719 05_1641885146482` to:
`%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF243.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 13 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks_returned
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF3946.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 14 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getDISRemarks
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`%22%27%3E%3CA+HREF%3D%22%2FWF_XSRF3556.html%22%3EInjected+Link%3C%2FA%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 15 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF4946.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 16 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF4366.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 17 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_DIS_App
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF5239.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 18 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to: `%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF4725.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

Issue 19 of 19

TOC

Link Injection (facilitates Cross-Site Request Forgery)

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_TSLR_AppDet
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc. It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user It is possible to upload, modify or delete web pages, scripts and files on the web server
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`%22%27%3E%3CIMG+SRC%3D%22%2FWF_XSRF5400.html%22%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a link to the file "WF_XSRF.html".

M Phishing Through Frames 19

TOC

Issue 1 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D1431+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E`

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 2 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D1670+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 3 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D1594+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 4 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_ins
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D1837+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 5 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_Report
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D2160+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 6 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D1926+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 7 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_attachments
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D2342+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 8 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D2578+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 9 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D2807+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 10 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D3029+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 11 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D3318+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 12 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks_returned
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D3947+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 13 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getTSLR_Remarks
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:

```
jQuery21103717863901571905_1641885146482%27%22%3E%3Ciframe+id%3D245+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 14 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getDISRemarks
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:

```
jQuery21105706580534857677_1641881499581%27%22%3E%3Ciframe+id%3D3555+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 15 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:

```
jQuery21103717863901571905_1641885146482%27%22%3E%3Ciframe+id%3D4727+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 16 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:

```
jQuery21103717863901571905_1641885146482%27%22%3E%3Ciframe+id%3D4368+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 17 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:

```
jQuery21103717863901571905_1641885146482%27%22%3E%3Ciframe+id%3D4948+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 18 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_TSLR_AppDet
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:

```
jQuery21103717863901571905_1641885146482%27%22%3E%3Ciframe+id%3D5402+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

Issue 19 of 19

TOC

Phishing Through Frames

Severity:	Medium
CVSS Score:	6.4
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_DIS_App
Entity:	jsoncallback (Parameter)
Risk:	It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Cause:	Sanitation of hazardous characters was not performed correctly on user input
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:

```
jQuery21103717863901571905_1641885146482%27%22%3E%3Ciframe+id%3D5241+src%3Dhttp%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E
```

Reasoning: The test result seems to indicate a vulnerability because the test response contained a frame/iframe to URL "http://demo.testfire.net/phishing.html".

M Reflected Cross Site Scripting 19

TOC

Issue 1 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to: `jQuery21105706580534857677_1641881499581%3Cimg+src%3Djavascript%3Aalert%281412%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "user_name": "**CONFIDENTIAL 0**",
 "pass": "e5631953ffda425f2a62283ad1a27d06b126337455ca14124cad7dcd24f3f711"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Set-Cookie: JSESSIONID=5BB48F28C7809F3D295CF4B4A4FA26F9; Path=/survey_f_line_service_audit; HttpOnly
Date: Tue, 11 Jan 2022 05:49:25 GMT
Content-Type: text/plain

jQuery21105706580534857677_1641881499581<img src=javascript:alert(1412)>({"Status_Code":1,"arr_success":
[{"role_id":39,"count":1,"id":402}]})
...
```

Issue 2 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%281563%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Referer: http://10.163.30.226:8088/survey_f_line_audit/
Cookie: JSESSIONID=821E84BD69B91BD6C7A5FF8718D4CCC0
Connection: keep-alive
Host: 10.163.30.226:8088
Accept-Encoding: gzip
Accept: */*
Accept-Language: en-US


HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Cache-Control: private
Date: Tue, 11 Jan 2022 05:49:47 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/plain

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(1563)>({"Status_Code":1,"arr_success":
[{"firka_desc":"T.Pet"}]})
...
```

Issue 3 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_ins
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%281805%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

signature: 52dsfsd
Content-Type: application/json
emp_value: sss
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

%7B%22regn_no%22%3A%222021%2F16%2F000044%22%2C%22dist_code%22%3A%2215%22%2C%22taluk_code%22%3A%2204%22%2C%22village_code%22
%3A%22041%22%2C%22sur_no%22%3A%22667%22%2C%22sub_no%22%3A%225%22%2C%22inspec_date%22%3A%2216%2F01%2F2022%22%2C%22sms_text%2
2%3A%22ts%22%7D

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 05:50:47 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(1805)>({"Status":"Saved
Successfully","Status_Code":"1"})
...
```

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery211057065805348576 77_1641881499581` to:
`jQuery211057065805348576 77_1641881499581%3Ciframe+src%3Djavascript%3Aalert%281895%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "regn_no": "2021/16/000003",
 "sur_no": "50",
 "sub_no": "2"
}

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 05:51:01 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(1895)>({"arr_cnt":[],"Status_Code":1,"arr_success":
[{"email_id":"","inspect_det2":"","door":"18","inspect_det1":"","patta_attachment":"-
","doc_no":"","mobile_no":"8667383374","fh_name":"siva","gender_code":null,"sro_district_name":null,"applicant_name_t":null
,"district_name":"Perambalur","power_of_attorney_attachment":"-","street":"-","fh_relation":"Fa... Thurai","state_code":"-
","power_agent":"2"}]})
...
```

Issue 5 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%281666%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Origin: http://10.163.30.226:8088
Referer: http://10.163.30.226:8088/survey_f_line_audit/firka_surveyor_wf.jsp
Accept: text/javascript, application/javascript, application/ecmascript, application/x-ecmascript, */*; q=0.01
signature: 12wrerwe
Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==


HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 05:50:14 GMT
Content-Type: text/plain

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(1666)>({"Status_Code":1,"arr_success":
[{"village_code":"Vettankulam","regn_no":"2021\/16\/000032","update_...

...
```

Issue 6 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%282778%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

X-Requested-With: XMLHttpRequest
Connection: keep-alive
Referer: http://10.163.30.226:8088/survey_f_line_audit/view_report.jsp
Accept: text/javascript, application/javascript, application/ecmascript, application/x-ecmascript, */*; q=0.01
Content-Type: application/json
emp_value: null
Accept-Language: en-US


HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Cache-Control: private
Date: Tue, 11 Jan 2022 06:18:53 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/plain

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(2778)> ( [{"rName":"not qualified","rId":"1"},
{"rName":"not recommended","rId":"2"}] )
...
```

Issue 7 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%282573%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "frmDate": "01/08/2021",
 "toDate": "10/01/2022",
 "search_code": "DATE"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Set-Cookie: JSESSIONID=F71575239DC9F5BAA3260918DE7D7464; Path=/survey_f_line_service_audit; HttpOnly
Date: Tue, 11 Jan 2022 06:17:57 GMT
Content-Type: text/plain

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(2573)>({"Status_Code":1,"arr_success":
[{"village_code":"Vettankulam","search_code":"DATE","regn_no":"2021\...

...
```

Issue 8 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_Report
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Cimg+src%3Djavascript%3Aalert%282137%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

emp_value: sss
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

%7B%22regn_no%22%3A%222021%2F16%2F000003%22%2C%22dist_code%22%3A%2215%22%2C%22taluk_code%22%3A%2204%22%2C%22village_code%22
%3A%22041%22%2C%22remarks%22%3A%22test%22%2C%22rej_rea%22%3A%22%22%2C%22inspect_date1%22%3A%22%22%2C%22inspect_date2%22%3A%
22%20%22%2C%22inspect_date3%22%3A%22%20%22%2C%22inspec_det1%22%3A%22%22%2C%22inspec_det2%22%3A%22%20%22%2C%22inspec_det3%22
%3A%22%22%2C%22add_fee%22%3...

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 05:51:50 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21105706580534857677_1641881499581<img src=javascript:alert(2137)>({"Status":"Some data problem","Status_Code":2})
...
```

Issue 9 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_attachments
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%282271%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

 "land_parcel_type": "image/jpeg",
 "rej_doc": "",
 "rej_doc_fc": "",
 "rej_doc_name": "",
 "rej_doc_file": "",
 "rej_doc_type": "",
 "roleId": "39"
}

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 06:14:34 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(2271)>({"Status":"Some data problem","Status_Code":2})
...
```

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getTSLR_Remarks
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`jQuery21103717863901571905_1641885146482%27%22%2F%3E%3Ciframe+src%3Djavascript%3Aalert%28218%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "regn_no": "2021/16/000001",
 "sur_no": "50",
 "sub_no": "1"
}

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 06:25:17 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21103717863901571905_1641885146482'"/><iframe src=javascript:alert(218)>({"arr_cnt":[],"Status_Code":1,"arr_success":
[{"tslr_remarks":"dfs","tslr_recommend":"2","rej_reason":null}]})
...
```

Issue 11 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%282999%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "reg_no": "2021/16/000001",
 "sur_no": "50",
 "sub_no": "1"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 06:19:31 GMT
Content-Type: text/plain

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(2999)>({"Status_Code":1,"arr_success":
[{"village_code":"041","extent":"175.5","owner":"வினோத்குமார்","patta_no":"NA","sub_division_no":"1","assessment_per_acre":"4.34","
district_code":"15","survey_no":"50","update_dt":"18-08-2021","taluk_code":"04","classification":"2"}]})
...
```

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getDISRemarks
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%283520%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "regn_no": "2021/16/000001",
 "sur_no": "50",
 "sub_no": "1"
}

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 06:21:23 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(3520)>({"arr_cnt":[],"Status_Code":1,"arr_success":
[{"dis_remarks":"fg","dis_recommend":"1","rej_reason":null}]})
...
```

Issue 13 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Ciframe+src%3Djavascript%3Aalert%283295%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "regn_no": "2021/16/000001",
 "sur_no": "50",
 "sub_no": "1"
}

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 06:20:34 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21105706580534857677_1641881499581<iframe src=javascript:alert(3295)>({"arr_cnt":[],"Status_Code":1,"arr_success":
[{"bank_details":"","inpection_detail1":"approved","inpection_detail2":"","recommend":"1","challan_no":"","challan_date":nu
ll,"addl_paid":"","add_fee_flag":"N","inpection_detail3":"","inpection_date3":null,"inpection_date2":null,"remarks":"approv
ed","inpection_date":"30-08-2021","measurement_satisfy":"...
...
```

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks_returned
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21105706580534857677_1641881499581` to:
`jQuery21105706580534857677_1641881499581%3Cimg+src%3Dx+onerror%3Dalert%283935%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "regn_no": "2021/16/000001",
 "sur_no": "50",
 "sub_no": "1"
}

HTTP/1.1 200
Access-Control-Allow-Headers: X-Requested-With,Host,User-Agent,Accept,Accept-Language,Accept-Encoding,Accept-Charset,Keep-
Alive,Connection,Referer,Origin
Access-Control-Max-Age: 3600
Connection: keep-alive
Allow-Control-Allow-Methods: POST,GET,OPTIONS
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 06:22:59 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Access-Control-Allow-Credentials: true

jQuery21105706580534857677_1641881499581<img src=x onerror=alert(3935)>({"arr_cnt":[],"Status_Code":1,"arr_success":
[{"bank_details":"","inpection_detail1":"approved","inpection_detail2":"-
","recommend":"1","challan_no":"","challan_date":null,"addl_paid":"","add_fee_flag":"N","inpection_date1":"29-10-
2021","inpection_detail3":"-","inpection_date3":null,"inpection_date2":null,"remarks":",m.,","inpection_date":"29-10-
2021","...
...
```

Issue 15 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`jQuery21103717863901571905_1641885146482%3Ciframe+src%3Djavascript%3Aalert%284341%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "ins_date": "returned",
 "login_districtCode": "15",
 "login_taluk_code": "04",
 "login_firka_code": "03"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 07:03:18 GMT
Content-Type: text/plain

jQuery21103717863901571905_1641885146482<iframe src=javascript:alert(4341)>({"Issue":"Data  Not Available.
","Status_Code":2})
...
```

Issue 16 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`jQuery21103717863901571905_1641885146482%3Cimg+src%3Dx+onerror%3Dalert%284929%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "reg_no": "2021/16/000049",
 "page_code": "2"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 07:11:19 GMT
Content-Type: text/plain

jQuery21103717863901571905_1641885146482<img src=x onerror=alert(4929)>({"Status_Code":1,"arr_success":
[{"village_code":"Vettankulam","sub_division_no":"1","survey_no":"50","regn_no":"2021\/16\/000049","update_dt_str":"08-10-
2021","applicant_name":"ganesh"}]})
...
```

Issue 17 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`jQuery21103717863901571905_1641885146482%3Ciframe+src%3Djavascript%3Aalert%284700%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "user_name": "**CONFIDENTIAL 0**",
 "role_id": "39"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 07:08:57 GMT
Content-Type: text/plain

jQuery21103717863901571905_1641885146482<iframe src=javascript:alert(4700)>({"Status_Code":1,"arr_success":
[{"districtCode":"15","firka_code":"03","taluk_code":"04"},{"districtCode":"15","firka_code":"03","taluk_code":"04"},
{"districtCode":"15","firka_code":"03","taluk_code":"04"},{"districtCode":"15","firka_code":"03","taluk_code":"04"},
{"districtCode":"15","firka_code":"03","taluk_code":"04"},{"districtCode":"15","fir...
...
```

Issue 18 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_DIS_App
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`jQuery21103717863901571905_1641885146482%3Ciframe+src%3Djavascript%3Aalert%285214%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "login_districtCode": "15",
 "login_taluk_code": "04"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 07:14:51 GMT
Content-Type: text/plain

jQuery21103717863901571905_1641885146482<iframe src=javascript:alert(5214)>({"Status_Code":1,"arr_success":
[{"village_code":"Vettankulam","regn_no":"2021\/16\/000006","update_...

...
```

Issue 19 of 19

TOC

Reflected Cross Site Scripting

Severity:	Medium
CVSS Score:	7.5
URL:	http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_TSLR_AppDet
Entity:	jsoncallback (Parameter)
Risk:	It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
Cause:	Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser. In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data. In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser. The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.
Fix:	Review possible solutions for hazardous character injection

Difference: **Parameter** `jsoncallback` manipulated from: `jQuery21103717863901571905_1641885146482` to:
`jQuery21103717863901571905_1641885146482%3Ciframe+src%3Djavascript%3Aalert%285375%29%3E`

Reasoning: The test result seems to indicate a vulnerability because Appscan successfully embedded a script in the response, which will be executed when the page loads in the user's browser.

Raw Test Response:

```
...

Content-Type: application/json
Accept-Language: en-US
hmac: webform@n!c:7Ak3NN578krKrzAYTGnQVg==

{
 "login_districtCode": "15",
 "login_taluk_code": "04"
}

HTTP/1.1 200
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Date: Tue, 11 Jan 2022 07:16:53 GMT
Content-Type: text/plain

jQuery21103717863901571905_1641885146482<iframe src=javascript:alert(5375)>({"Status_Code":1,"arr_success":
[{"village_code":"Vettankulam","regn_no":"2021\/16\/000027","update_dt_str":"27-08-2021","applicant_name":"divya"},
{"village_code":"Vettankulam","regn_no":"2021\/16\/000033","update_dt_str":"27-08-2021","applicant_name":"ganesh"}]})
...
```

How to Fix

Application Error

TOC

Cause:

- Proper bounds checking were not performed on incoming parameter values
- No validation was done in order to make sure that user input matches the data type expected

Risk:

It is possible to gather sensitive debugging information
If an attacker probes the application by forging a request that contains parameters or parameter values other than the ones expected by the application (examples are listed below), the application may enter an undefined state that makes it vulnerable to attack. The attacker can gain useful information from the application's response to this request, which information may be exploited to locate application weaknesses.
For example, if the parameter field should be an apostrophe-quoted string (e.g. in an ASP script or SQL query), the injected apostrophe symbol will prematurely terminate the string stream, thus changing the normal flow/syntax of the script.
Another cause of vital information being revealed in error messages, is when the scripting engine, web server, or database are misconfigured.
Here are some different variants:
[1] Remove parameter
[2] Remove parameter value
[3] Set parameter value to null
[4] Set parameter value to a numeric overflow (+/- 99999999)
[5] Set parameter value to hazardous characters, such as ' " \' \") ;
[6] Append some string to a numeric parameter value
[7] Append "." (dot) or "[]" (angle brackets) to the parameter name

Affected Products:

This issue may affect different types of products.

Fix Recommendation:

General

[1] Check incoming requests for the presence of all expected parameters and values. When a parameter is missing, issue a proper error message or use default values.
[2] The application should verify that its input consists of valid characters (after decoding). For example, an input value containing the null byte (encoded as %00), apostrophe, quotes, etc. should be rejected.
[3] Enforce values in their expected ranges and types. If your application expects a certain parameter to have a value from a certain set, then the application should ensure that the value it receives indeed belongs to the set. For example, if your application expects a value in the range 10..99, then it should make sure that the value is indeed numeric, and that its value is in 10..99.
[4] Verify that the data belongs to the set offered to the client.
[5] Do not output debugging error messages and exceptions in a production environment.
In order to disable debugging in ASP.NET, edit your web.config file to contain the following:
<compilation
debug="false"
/>
For more information, see "HOW TO: Disable Debugging for ASP.NET Applications" in:
http://support.microsoft.com/default.aspx?scid=kb;en-us;815157
You can add input validation to Web Forms pages by using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validation (for example, testing for valid dates or values within a range), plus ways to provide custom-written validation. In addition, validation controls allow you to completely customize how error information is displayed to the user. Validation controls can be used with any controls that are processed in a Web Forms page's class file, including both HTML and Web server controls.
To make sure that all the required parameters exist in a request, use the "RequiredFieldValidator" validation control. This control ensures that the user does not skip an entry in the web form.
To make sure user input contains only valid values, you can use one of the following validation controls:
[1] "RangeValidator": checks that a user's entry (value) is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, and dates.
[2] "RegularExpressionValidator": checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.
Important note: validation controls do not block user input or change the flow of page processing; they only set an error state, and produce error messages. It is the programmer's responsibility to test the state of the controls in the code before performing further application-specific actions. There are two ways to check for user input validity:
1. Test for a general error state:
In your code, test the page's IsValid property. This property rolls up the values of the IsValid properties of all the validation controls on the page (using a logical AND). If one of the validation controls is set to invalid, the page's property will return false.
2. Test for the error state of individual controls:
Loop through the page's Validators collection, which contains references to all the validation controls. You can then examine the IsValid property of each validation control.
** Input Data Validation:
While data validations may be provided as a user convenience on the client-tier, data validation must be performed on the server-tier using Servlets. Client-side validations are inherently insecure because they can be easily bypassed, e.g. by disabling Javascript.
A good design usually requires the web application framework to provide server-side utility routines to validate the following:
[1] Required field
[2] Field data type (all HTTP request parameters are Strings by default)
[3] Field length
[4] Field range
[5] Field options
[6] Field pattern
[7] Cookie values
[8] HTTP Response
A good practice is to implement the above routine as static methods in a "Validator" utility class. The following sections describe an example validator class.
[1] Required field
Always check that the field is not null and its length is greater than zero, excluding leading and trailing white spaces.
Example of how to validate required fields:
// Java example to validate required fields public Class Validator { ... public static boolean validateRequired(String value) { boolean isFieldValid = false; if (value != null && value.trim().length() > 0) { isFieldValid = true; } return isFieldValid; } ... } ... String fieldValue = request.getParameter("fieldName"); if (Validator.validateRequired(fieldValue)) { // fieldValue is valid, continue processing request ... }
[2] Field data type
In web applications, input parameters are poorly typed. For example, all HTTP request parameters or cookie values are of type String. The developer is responsible for verifying the input is of the correct data type. Use the Java primitive wrapper classes to check if the field value can be safely converted to the desired primitive data type.
Example of how to validate a numeric field (type int):

// Java example to validate that a field is an int number public Class Validator { ... public static boolean validateInt(String value) { boolean isFieldValid = false; try { Integer.parseInt(value); isFieldValid = true; } catch (Exception e) { isFieldValid = false; } return isFieldValid; } ... } ... // check if the HTTP request parameter is of type int String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // fieldValue is valid, continue processing request ... }

A good practice is to convert all HTTP request parameters to their respective data types. For example, store the "integerValue" of a request parameter in a request attribute and use it as shown in the following example:

// Example to convert the HTTP request parameter to a primitive wrapper data type // and store this value in a request attribute for further processing String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // convert fieldValue to an Integer Integer integerValue = Integer.getInteger(fieldValue); // store integerValue in a request attribute request.setAttribute("fieldName", integerValue); } ... // Use the request attribute for further processing Integer integerValue = (Integer)request.getAttribute("fieldName"); ...

The primary Java data types that the application should handle:

- Byte
- Short
- Integer
- Long
- Float
- Double
- Date

[3] Field length

Always ensure that the input parameter (whether HTTP request parameter or cookie value) is bounded by a minimum length and/or a maximum length.

Example to validate that the length of the userName field is between 8 and 20 characters:

// Example to validate the field length public Class Validator { ... public static boolean validateLength(String value, int minLength, int maxLength) { String validatedValue = value; if (!validateRequired(value)) { validatedValue = ""; } return (validatedValue.length() >= minLength && validatedValue.length() <= maxLength); } ... } ... String userName = request.getParameter("userName"); if (Validator.validateRequired(userName)) { if (Validator.validateLength(userName, 8, 20)) { // userName is valid, continue further processing ... } }

[4] Field range

Always ensure that the input parameter is within a range as defined by the functional requirements.

Example to validate that the input numberOfChoices is between 10 and 20:

// Example to validate the field range public Class Validator { ... public static boolean validateRange(int value, int min, int max) { return (value >= min && value <= max); } ... } ... String fieldValue = request.getParameter("numberOfChoices"); if (Validator.validateRequired(fieldValue)) { if (Validator.validateInt(fieldValue)) { int numberOfChoices = Integer.parseInt(fieldValue); if (Validator.validateRange(numberOfChoices, 10, 20)) { // numberOfChoices is valid, continue processing request ... } } }

[5] Field options

Often, the web application presents the user with a set of options to choose from, e.g. using the SELECT HTML tag, but fails to perform server-side validation to ensure that the selected value is one of the allowed options. Remember that a malicious user can easily modify any option value. Always validate the selected user value against the allowed options as defined by the functional requirements.

Example to validate the user selection against a list of allowed options:

// Example to validate user selection against a list of options public Class Validator { ... public static boolean validateOption(Object[] options, Object value) { boolean isValidValue = false; try { List list = Arrays.asList(options); if (list != null) { isValidValue = list.contains(value); } } catch (Exception e) { } return isValidValue; } ... } ... // Allowed options String[] options = {"option1", "option2", "option3"); // Verify that the user selection is one of the allowed options String userSelection = request.getParameter("userSelection"); if (Validator.validateOption(options, userSelection)) { // valid user selection, continue processing request ... }

[6] Field pattern

Always check that the user input matches a pattern as defined by the functionality requirements. For example, if the userName field should only allow alpha-numeric characters, case insensitive, then use the following regular expression:

^[a-zA-Z0-9]*$

Java 1.3 or earlier versions do not include any regular expression packages. Apache Regular Expression Package (see Resources below) is recommended for use with Java 1.3 to resolve this lack of support.

Example to perform regular expression validation:

// Example to validate that a given value matches a specified pattern // using the Apache regular expression package import org.apache.regexp.RE; import org.apache.regexp.RESyntaxException; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { RE r = new RE(expression); match = r.match(value); } return match; } ... } ... // Verify that the userName request parameter is alpha-numeric String userName = request.getParameter("userName"); if (Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) { // userName is valid, continue processing request ... }

Java 1.4 introduced a new regular expression package (java.util.regex). Here is a modified version of Validator.matchPattern using the new Java 1.4 regular expression package:

// Example to validate that a given value matches a specified pattern // using the Java 1.4 regular expression package import java.util.regex.Pattern; import java.util.regexe.Matcher; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { match = Pattern.matches(expression, value); } return match; } ... }

[7] Cookie value

Use the javax.servlet.http.Cookie object to validate the cookie value. The same validation rules (described above) apply to cookie values depending on the application requirements, e.g. validate a required value, validate length, etc.

Example to validate a required cookie value:

// Example to validate a required cookie value // First retrieve all available cookies submitted in the HTTP request Cookie[] cookies = request.getCookies(); if (cookies != null) { // find the "user" cookie for (int i=0; i<cookies.length; ++i) { if (cookies[i].getName().equals("user")) { //

validate the cookie value if (Validator.validateRequired(cookies[i].getValue()) { // valid cookie value, continue processing request ... } } } }
[8] HTTP Response
[8-1] Filter user input
To guard the application against cross-site scripting, sanitize HTML by converting sensitive characters to their corresponding character entities. These are the HTML sensitive characters:
< > " ' % ;) (& +
Example to filter a specified string by converting sensitive characters to their corresponding character entities:
// Example to filter sensitive data to prevent cross-site scripting public Class Validator { ... public static String filter(String value) { if (value == null) { return null; } StringBuffer result = new StringBuffer(value.length()); for (int i=0; i<value.length(); ++i) { switch (value.charAt(i)) { case '<': result.append("<"); break; case '>': result.append(">"); break; case '"': result.append("""); break; case '\'': result.append("'"); break; case '%': result.append("%"); break; case ';': result.append(";"); break; case '(': result.append("("); break; case ')': result.append(")"); break; case '&': result.append("&"); break; case '+': result.append("+"); break; default: result.append(value.charAt(i)); break; } return result; } ... } ... // Filter the HTTP response using Validator.filter PrintWriter out = response.getWriter(); // set output response out.write(Validator.filter(response)); out.close();
The Java Servlet API 2.3 introduced Filters, which supports the interception and transformation of HTTP requests or responses.
Example of using a Servlet Filter to sanitize the response using Validator.filter:
// Example to filter all sensitive characters in the HTTP response using a Java Filter. // This example is for illustration purposes since it will filter all content in the response, including HTML tags! public class SensitiveCharsFilter implements Filter { ... public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException { PrintWriter out = response.getWriter(); ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response); chain.doFilter(request, wrapper); CharArrayWriter caw = new CharArrayWriter(); caw.write(Validator.filter(wrapper.toString())); response.setContentType("text/html"); response.setContentLength(caw.toString().length()); out.write(caw.toString()); out.close(); } ... public class CharResponseWrapper extends HttpServletResponseWrapper { private CharArrayWriter output; public String toString() { return output.toString(); } public CharResponseWrapper(HttpServletResponse response){ super(response); output = new CharArrayWriter(); } public PrintWriter getWriter(){ return new PrintWriter(output); } } } }
[8-2] Secure the cookie
When storing sensitive data in a cookie, make sure to set the secure flag of the cookie in the HTTP response, using Cookie.setSecure(boolean flag) to instruct the browser to send the cookie using a secure protocol, such as HTTPS or SSL.
Example to secure the "user" cookie:
// Example to secure a cookie, i.e. instruct the browser to // send the cookie using a secure protocol Cookie cookie = new Cookie("user", "sensitive"); cookie.setSecure(true); response.addCookie(cookie);
RECOMMENDED JAVA TOOLS
The two main Java frameworks for server-side validation are:
[1] Jakarta Commons Validator (integrated with Struts 1.1)
The Jakarta Commons Validator is a powerful framework that implements all the above data validation requirements. These rules are configured in an XML file that defines input validation rules for form fields. Struts supports output filtering of dangerous characters in the [8] HTTP Response by default on all data written using the Struts 'bean:write' tag. This filtering may be disabled by setting the 'filter=false' flag.
Struts defines the following basic input validators, but custom validators may also be defined:
required: succeeds if the field contains any characters other than white space.
mask: succeeds if the value matches the regular expression given by the mask attribute.
range: succeeds if the value is within the values given by the min and max attributes ((value >= min) & (value <= max)).
maxLength: succeeds if the field is length is less than or equal to the max attribute.
minLength: succeeds if the field is length is greater than or equal to the min attribute.
byte, short, integer, long, float, double: succeeds if the value can be converted to the corresponding primitive.
date: succeeds if the value represents a valid date. A date pattern may be provided.
creditCard: succeeds if the value could be a valid credit card number.
e-mail: succeeds if the value could be a valid e-mail address.
Example to validate the userName field of a loginForm using Struts Validator:
<form-validation> <global> ... <validator name="required" classname="org.apache.struts.validator.FieldChecks" method="validateRequired" msg="errors.required"> </validator> <validator name="mask" classname="org.apache.struts.validator.FieldChecks" method="validateMask" msg="errors.invalid"> </validator> ... </global> <formset> <form name="loginForm"> <!-- userName is required and is alpha-numeric case insensitive --> <field property="userName" depends="required,mask"> <!-- message resource key to display if validation fails --> <msg name="mask" key="login.userName.maskmsg"/> <arg0 key="login.userName.displayname"/> <var> <var-name>mask</var-name> <var-value>^[a-zA-Z0-9]*$</var-value> </var> </field> ... </form> ... </formset> </form-validation>
[2] JavaServer Faces Technology
JavaServer Faces Technology is a set of Java APIs (JSR 127) to represent UI components, manage their state, handle events and input validation.
The JavaServer Faces API implements the following basic validators, but custom validators may be defined:
validate_doublerange: registers a DoubleRangeValidator on a component
validate_length: registers a LengthValidator on a component
validate_longrange: registers a LongRangeValidator on a component
validate_required: registers a RequiredValidator on a component
validate_stringrange: registers a StringRangeValidator on a component
validator: registers a custom Validator on a component
The JavaServer Faces API defines the following UIInput and UIOutput Renderers (Tags):
input_date: accepts a java.util.Date formatted with a java.text.Date instance

output_date: displays a java.util.Date formatted with a java.text.Date instance
input_datetime: accepts a java.util.Date formatted with a java.text.DateTime instance
output_datetime: displays a java.util.Date formatted with a java.text.DateTime instance
input_number: displays a numeric data type (java.lang.Number or primitive), formatted with a java.text.NumberFormat
output_number: displays a numeric data type (java.lang.Number or primitive), formatted with a java.text.NumberFormat
input_text: accepts a text string of one line.
output_text: displays a text string of one line.
input_time: accepts a java.util.Date, formatted with a java.text.DateFormat time instance
output_time: displays a java.util.Date, formatted with a java.text.DateFormat time instance
input_hidden: allows a page author to include a hidden variable in a page
input_secret: accepts one line of text with no spaces and displays it as a set of asterisks as it is typed
input_textarea: accepts multiple lines of text
output_errors: displays error messages for an entire page or error messages associated with a specified client identifier
output_label: displays a nested component as a label for a specified input field
output_message: displays a localized message
Example to validate the userName field of a loginForm using JavaServer Faces:
<%@ taglib uri="https://docs.oracle.com/javaee/6/tutorial/doc/glxce.html" prefix="h" %> <%@ taglib uri="http://mrbool.com/how-to-create-a-login-validation-with-jsf-java-server-faces/27046" prefix="f" %> ... <jsp:useBean id="UserBean" class="myApplication.UserBean" scope="session" /> <f:use_faces> <h:form formName="loginForm" > <h:input_text id="userName" size="20" modelReference="UserBean.userName"> <f:validate_required/> <f:validate_length minimum="8" maximum="20"/> </h:input_text> <!-- display errors if present --> <h:output_errors id="loginErrors" clientId="userName"/> <h:command_button id="submit" label="Submit" commandName="submit" /><p> </h:form> </f:use_faces>
REFERENCES
Java API 1.3 -
https://www.oracle.com/java/technologies/java-archive-13docs-downloads.html
Java API 1.4 -
https://www.oracle.com/java/technologies/java-archive-142docs-downloads.html
Java Servlet API 2.3 -
https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://www.javaserverfaces.org/
** Error Handling:
Many J2EE web application architectures follow the Model View Controller (MVC) pattern. In this pattern a Servlet acts as a Controller. A Servlet delegates the application processing to a JavaBean such as an EJB Session Bean (the Model). The Servlet then forwards the request to a JSP (View) to render the processing results. Servlets should check all input, output, return codes, error codes and known exceptions to ensure that the expected processing actually occurred.
While data validation protects applications against malicious data tampering, a sound error handling strategy is necessary to prevent the application from inadvertently disclosing internal error messages such as exception stack traces. A good error handling strategy addresses the following items:
[1] Defining Errors
[2] Reporting Errors
[3] Rendering Errors
[4] Error Mapping
[1] Defining Errors
Hard-coded error messages in the application layer (e.g. Servlets) should be avoided. Instead, the application should use error keys that map to known application failures. A good practice is to define error keys that map to validation rules for HTML form fields or other bean properties. For example, if the "user_name" field is required, is alphanumeric, and must be unique in the database, then the following error keys should be defined:
(a) ERROR_USERNAME_REQUIRED: this error key is used to display a message notifying the user that the "user_name" field is required;
(b) ERROR_USERNAME_ALPHANUMERIC: this error key is used to display a message notifying the user that the "user_name" field should be alphanumeric;
(c) ERROR_USERNAME_DUPLICATE: this error key is used to display a message notifying the user that the "user_name" value is a duplicate in the database;
(d) ERROR_USERNAME_INVALID: this error key is used to display a generic message notifying the user that the "user_name" value is invalid;
A good practice is to define the following framework Java classes which are used to store and report application errors:
- ErrorKeys: defines all error keys
// Example: ErrorKeys defining the following error keys: // - ERROR_USERNAME_REQUIRED // - ERROR_USERNAME_ALPHANUMERIC // - ERROR_USERNAME_DUPLICATE // - ERROR_USERNAME_INVALID // ... public Class ErrorKeys { public static final String ERROR_USERNAME_REQUIRED = "error.username.required"; public static final String ERROR_USERNAME_ALPHANUMERIC = "error.username.alphanumeric"; public static final String ERROR_USERNAME_DUPLICATE = "error.username.duplicate"; public static final String ERROR_USERNAME_INVALID = "error.username.invalid"; ... }
- Error: encapsulates an individual error

// Example: Error encapsulates an error key. // Error is serializable to support code executing in multiple JVMs. public Class Error implements Serializable { // Constructor given a specified error key public Error(String key) { this(key, null); } // Constructor given a specified error key and array of placeholder objects public Error(String key, Object[] values) { this.key = key; this.values = values; } // Returns the error key public String getKey() { return this.key; } // Returns the placeholder values public Object[] getValues() { return this.values; } private String key = null; private Object[] values = null; }
- Errors: encapsulates a Collection of errors
// Example: Errors encapsulates the Error objects being reported to the presentation layer. // Errors are stored in a HashMap where the key is the bean property name and value is an // ArrayList of Error objects. public Class Errors implements Serializable { // Adds an Error object to the Collection of errors for the specified bean property. public void addError(String property, Error error) { ArrayList propertyErrors = (ArrayList)errors.get(property); if (propertyErrors == null) { propertyErrors = new ArrayList(); errors.put(property, propertyErrors); } propertyErrors.put(error); } // Returns true if there are any errors public boolean hasErrors() { return (errors.size > 0); } // Returns the Errors for the specified property public ArrayList getErrors(String property) { return (ArrayList)errors.get(property); } private HashMap errors = new HashMap(); }
Using the above framework classes, here is an example to process validation errors of the "user_name" field:
// Example to process validation errors of the "user_name" field. Errors errors = new Errors(); String userName = request.getParameter("user_name"); // (a) Required validation rule if (!Validator.validateRequired(userName)) { errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_REQUIRED)); } // (b) Alpha-numeric validation rule else if (!Validator.matchPattern(userName, "^[a-zA-Z0-9]*$")) { errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_ALPHANUMERIC)); } else { // (c) Duplicate check validation rule // We assume that there is an existing UserValidationEJB session bean that implements // a checkIfDuplicate() method to verify if the user already exists in the database. try { ... if (UserValidationEJB.checkIfDuplicate(userName)) { errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE)); } } catch (RemoteException e) { // log the error logger.error("Could not validate user for specified userName: " + userName); errors.addError("user_name", new Error(ErrorKeys.ERROR_USERNAME_DUPLICATE); } } // set the errors object in a request attribute called "errors" request.setAttribute("errors", errors); ...
[2] Reporting Errors
There are two ways to report web-tier application errors:
(a) Servlet Error Mechanism
(b) JSP Error Mechanism
[2-a] Servlet Error Mechanism
A Servlet may report errors by:
- forwarding to the input JSP (having already stored the errors in a request attribute), OR
- calling response.sendError with an HTTP error code argument, OR
- throwing an exception
It is good practice to process all known application errors (as described in section [1]), store them in a request attribute, and forward to the input JSP. The input JSP should display the error messages and prompt the user to re-enter the data. The following example illustrates how to forward to an input JSP (userInput.jsp):
// Example to forward to the userInput.jsp following user validation errors RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp"); if (rd != null) { rd.forward(request, response); }
If the Servlet cannot forward to a known JSP page, the second option is to report an error using the response.sendError method with HttpServletResponse.SC_INTERNAL_SERVER_ERROR (status code 500) as argument. Refer to the javadoc of javax.servlet.http.HttpServletResponse for more details on the various HTTP status codes.
Example to return a HTTP error:
// Example to return a HTTP error code RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp"); if (rd == null) { // messages is a resource bundle with all message keys and values response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, messages.getMessage(ErrorKeys.ERROR_USERNAME_INVALID)); }
As a last resort, Servlets can throw an exception, which must be a subclass of one of the following classes:
- RuntimeException
- ServletException
- IOException
[2-b] JSP Error Mechanism
JSP pages provide a mechanism to handle runtime exceptions by defining an errorPage directive as shown in the following example:
<%@ page errorPage="/errors/userValidation.jsp" %>
Uncaught JSP exceptions are forwarded to the specified errorPage, and the original exception is set in a request parameter called javax.servlet.jsp.jspException. The error page must include a isErrorPage directive as shown below:
<%@ page isErrorPage="true" %>
The isErrorPage directive causes the "exception" variable to be initialized to the exception object being thrown.
[3] Rendering Errors
The J2SE Internationalization APIs provide utility classes for externalizing application resources and formatting messages including:
(a) Resource Bundles
(b) Message Formatting
[3-a] Resource Bundles
Resource bundles support internationalization by separating localized data from the source code that uses it. Each resource bundle stores a map of key/value pairs for a specific locale.
It is common to use or extend java.util.PropertyResourceBundle, which stores the content in an external properties file as shown in the following example:
ErrorMessages.properties

required user name error message error.username.required=User name field is required # invalid user name format error.username.alphanumeric=User name must be alphanumeric # duplicate user name error message error.username.duplicate=User name {0} already exists, please choose another one ...
Multiple resources can be defined to support different locales (hence the name resource bundle). For example, ErrorMessages_fr.properties can be defined to support the French member of the bundle family. If the resource member of the requested locale does not exist, the default member is used. In the above example, the default resource is ErrorMessages.properties. Depending on the user's locale, the application (JSP or Servlet) retrieves content from the appropriate resource.
[3-b] Message Formatting
The J2SE standard class java.util.MessageFormat provides a generic way to create messages with replacement placeholders. A MessageFormat object contains a pattern string with embedded format specifiers as shown below:
// Example to show how to format a message using placeholder parameters String pattern = "User name {0} already exists, please choose another one"; String userName = request.getParameter("user_name"); Object[] args = new Object[1]; args[0] = userName; String message = MessageFormat.format(pattern, args);
Here is a more comprehensive example to render error messages using ResourceBundle and MessageFormat:
// Example to render an error message from a localized ErrorMessages resource (properties file) // Utility class to retrieve locale-specific error messages public Class ErrorMessageResource { // Returns the error message for the specified error key in the environment locale public String getErrorMessage(String errorKey) { return getErrorMessage(errorKey, defaultLocale); } // Returns the error message for the specified error key in the specified locale public String getErrorMessage(String errorKey, Locale locale) { return getErrorMessage(errorKey, null, locale); } // Returns a formatted error message for the specified error key in the specified locale public String getErrorMessage(String errorKey, Object[] args, Locale locale) { // Get localized ErrorMessageResource ResourceBundle errorMessageResource = ResourceBundle.getBundle("ErrorMessages", locale); // Get localized error message String errorMessage = errorMessageResource.getString(errorKey); if (args != null) { // Format the message using the specified placeholders args return MessageFormat.format(errorMessage, args); } else { return errorMessage; } } // default environment locale private Locale defaultLocale = Locale.getDefaultLocale(); } ... // Get the user's locale Locale userLocale = request.getLocale(); // Check if there were any validation errors Errors errors = (Errors)request.getAttribute("errors"); if (errors != null && errors.hasErrors()) { // iterate through errors and output error messages corresponding to the "user_name" property ArrayList userNameErrors = errors.getErrors("user_name"); ListIterator iterator = userNameErrors.iterator(); while (iterator.hasNext()) { // Get the next error object Error error = (Error)iterator.next(); String errorMessage = ErrorMessageResource.getErrorMessage(error.getKey(), userLocale); output.write(errorMessage + "\r\n"); } }
It is recommended to define a custom JSP tag, e.g. displayErrors, to iterate through and render error messages as shown in the above example.
[4] Error Mapping
Normally, the Servlet Container will return a default error page corresponding to either the response status code or the exception. A mapping between the status code or the exception and a web resource may be specified using custom error pages. It is a good practice to develop static error pages that do not disclose internal error states (by default, most Servlet containers will report internal error messages). This mapping is configured in the Web Deployment Descriptor (web.xml) as specified in the following example:
<!-- Mapping of HTTP error codes and application exceptions to error pages --> <error-page> <exception-type>UserValidationException</exception-type> <location>/errors/validationError.html</error-page> </error-page> <error-page> <error-code>500</exception-type> <location>/errors/internalError.html</error-page> </error-page> <error-page> ... </error-page> ...
RECOMMENDED JAVA TOOLS
The two main Java frameworks for server-side validation are:
[1] Jakarta Commons Validator (integrated with Struts 1.1)
The Jakarta Commons Validator is a Java framework that defines the error handling mechanism as described above. Validation rules are configured in an XML file that defines input validation rules for form fields and the corresponding validation error keys. Struts provides internationalization support to build localized applications using resource bundles and message formatting.
Example to validate the userName field of a loginForm using Struts Validator:
<form-validation> <global> ... <validator name="required" classname="org.apache.struts.validator.FieldChecks" method="validateRequired" msg="errors.required"> </validator> <validator name="mask" classname="org.apache.struts.validator.FieldChecks" method="validateMask" msg="errors.invalid"> </validator> ... </global> <formset> <form name="loginForm"> <!-- userName is required and is alpha-numeric case insensitive --> <field property="userName" depends="required,mask"> <!-- message resource key to display if validation fails --> <msg name="mask" key="login.userName.maskmsg"/> <arg0 key="login.userName.displayname"/> <var> <var-name>mask</var-name> <var-value>^[a-zA-Z0-9]*$</var-value> </var> </field> ... </form> ... </formset> </form-validation>
The Struts JSP tag library defines the "errors" tag that conditionally displays a set of accumulated error messages as shown in the following example:
<%@ page language="java" %> <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %> <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %> <html:html> <head> <body> <html:form action="/logon.do"> <table border="0" width="100%"> <tr> <th align="right"> <html:errors property="username"/> <bean:message key="prompt.username"/> </th> <td align="left"> <html:text property="username" size="16"/> </td> </tr> <tr> <td align="right"> <html:submit><bean:message key="button.submit"/></html:submit> </td> <td align="right"> <html:reset><bean:message key="button.reset"/></html:reset> </td> </tr> </table> </html:form> </body> </html:html>
[2] JavaServer Faces Technology
JavaServer Faces Technology is a set of Java APIs (JSR 127) to represent UI components, manage their state, handle events, validate input, and support internationalization.
The JavaServer Faces API defines the "output_errors" UIOutput Renderer, which displays error messages for an entire page or error messages associated with a specified client identifier.
Example to validate the userName field of a loginForm using JavaServer Faces:
<%@ taglib uri="https://docs.oracle.com/javaee/6/tutorial/doc/glxce.html" prefix="h" %> <%@ taglib uri="http://mrbool.com/how-to-create-a-login-validation-with-jsf-java-server-faces/27046" prefix="f" %> ... <jsp:useBean id="UserBean" class="myApplication.UserBean" scope="session" /> <f:use_faces> <h:form formName="loginForm" > <h:input_text id="userName" size="20"

modelReference="UserBean.userName"> <f:validate_required/> <f:validate_length minimum="8" maximum="20"/> </h:input_text> <!-- display errors if present --> <h:output_errors id="loginErrors" clientId="userName"/> <h:command_button id="submit" label="Submit" commandName="submit" /><p> </h:form> </f:use_faces>
REFERENCES
Java API 1.3 -
https://www.oracle.com/java/technologies/java-archive-13docs-downloads.html
Java API 1.4 -
https://www.oracle.com/java/technologies/java-archive-142docs-downloads.html
Java Servlet API 2.3 -
https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api
Java Regular Expression Package -
http://jakarta.apache.org/regexp/
Jakarta Validator -
http://jakarta.apache.org/commons/validator/
JavaServer Faces Technology -
http://www.javaserverfaces.org/
** Input Data Validation:
While data validations may be provided as a user convenience on the client-tier, data validation must always be performed on the server-tier. Client-side validations are inherently insecure because they can be easily bypassed, e.g. by disabling Javascript.
A good design usually requires the web application framework to provide server-side utility routines to validate the following:
[1] Required field
[2] Field data type (all HTTP request parameters are Strings by default)
[3] Field length
[4] Field range
[5] Field options
[6] Field pattern
[7] Cookie values
[8] HTTP Response
A good practice is to implement a function or functions that validates each application parameter. The following sections describe some example checking.
[1] Required field
Always check that the field is not null and its length is greater than zero, excluding leading and trailing white spaces.
Example of how to validate required fields:
// PHP example to validate required fields function validateRequired($input) { ... $pass = false; if (strlen(trim($input))>0){ $pass = true; } return $pass; ... } ... if (validateRequired($fieldName)) { // fieldName is valid, continue processing request ... }
[2] Field data type
In web applications, input parameters are poorly typed. For example, all HTTP request parameters or cookie values are of type String. The developer is responsible for verifying the input is of the correct data type.
[3] Field length
Always ensure that the input parameter (whether HTTP request parameter or cookie value) is bounded by a minimum length and/or a maximum length.
[4] Field range
Always ensure that the input parameter is within a range as defined by the functional requirements.
[5] Field options
Often, the web application presents the user with a set of options to choose from, e.g. using the SELECT HTML tag, but fails to perform server-side validation to ensure that the selected value is one of the allowed options. Remember that a malicious user can easily modify any option value. Always validate the selected user value against the allowed options as defined by the functional requirements.
[6] Field pattern
Always check that user input matches a pattern as defined by the functionality requirements. For example, if the userName field should only allow alpha-numeric characters, case insensitive, then use the following regular expression:
^[a-zA-Z0-9]+$
[7] Cookie value
The same validation rules (described above) apply to cookie values depending on the application requirements, e.g. validate a required value, validate length, etc.
[8] HTTP Response
[8-1] Filter user input
To guard the application against cross-site scripting, the developer should sanitize HTML by converting sensitive characters to their corresponding character entities. These are the HTML sensitive characters:
< > " ' % ;) (& +
PHP includes some automatic sanitization utility functions, such as htmlentities():
$input = htmlentities($input, ENT_QUOTES, 'UTF-8');
In addition, in order to avoid UTF-7 variants of Cross-site Scripting, you should explicitly define the Content-Type header of the response, for example:
<?php header('Content-Type: text/html; charset=UTF-8'); ?>
[8-2] Secure the cookie
When storing sensitive data in a cookie and transporting it over SSL, make sure that you first set the secure flag of the cookie in the HTTP

response. This will instruct the browser to only use that cookie over SSL connections.
You can use the following code example, for securing the cookie:
<$php $value = "some_value"; $time = time()+3600; $path = "/application/"; $domain = ".example.com"; $secure = 1; setcookie("CookieName", $value, $time, $path, $domain, $secure, TRUE); ?>
In addition, we recommend that you use the HttpOnly flag. When the HttpOnly flag is set to TRUE the cookie will be made accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. This setting can effectively help to reduce identity theft through XSS attacks (although it is not supported by all browsers).
The HttpOnly flag was Added in PHP 5.2.0.
REFERENCES
[1] Mitigating Cross-site Scripting With HTTP-only Cookies:
http://msdn2.microsoft.com/en-us/library/ms533046.aspx
[2] PHP Security Consortium:
http://phpsec.org/
[3] PHP & Web Application Security Blog (Chris Shiflett):
http://shiflett.org/

CWE:

550

External References:

An example for using apostrophe to hack a site can be found in "How I hacked PacketStorm (by Rain Forest Puppy), RFP's site"
"Web Application Disassembly with ODBC Error Messages" (By David Litchfield)
CERT Advisory (CA-1997-25): Sanitizing user-supplied data in CGI scripts

Link Injection (facilitates Cross-Site Request Forgery)

TOC

Cause:

Sanitation of hazardous characters was not performed correctly on user input

Risk:

It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
It may be possible to steal or manipulate customer session and cookies, which might be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user
It is possible to upload, modify or delete web pages, scripts and files on the web server
The software constructs all or part of a command, data structure, or record using externally-influenced input, but fails to neutralize elements that could modify how it is parsed or interpreted.
Link Injection is the modifying of the content of a site by embedding in it a URL to an external site, or to a script in the vulnerable site. After embedding the URL in the vulnerable site, an attacker is able to use it as a platform to launch attacks against other sites, as well as against the vulnerable site itself.
Some of these possible attacks require the user to be logged in to the site during the attack. By launching these attacks from the vulnerable site itself, the attacker increases the chances of success, because the user is more likely to be logged in.
The Link Injection vulnerability is a result of insufficient user input sanitization, the input being later returned to the user in the site response. The resulting ability to inject hazardous characters into the response makes it possible for attackers to embed URLs, among other possible content modifications.
Below is an example for a Link Injection (We will assume that site "www.vulnerable.com" has a parameter called "name", which is used to greet users).
The following request:
HTTP://www.vulnerable.com/greet.asp?name=John Smith
Will yield the following response:
<HTML> <BODY> Hello, John Smith. </BODY> </HTML>
However, a malicious user may send the following request:
HTTP://www.vulnerable.com/greet.asp?name=<IMG SRC="http://www.ANY-SITE.com/ANY-SCRIPT.asp">
This will return the following response:
<HTML> <BODY> Hello, <IMG SRC="http://www.ANY-SITE.com/ANY-SCRIPT.asp">. </BODY> </HTML>
As this example shows, it is possible to cause a user's browser to issue automatic requests to virtually any site the attacker desires. As a result, Link Injection vulnerability can be used to launch several types of attack:
[-] Cross-Site Request Forgery
[-] Cross-Site Scripting
[-] Phishing

Affected Products:

This issue may affect different types of products.

Fix Recommendation:

General

There are several mitigation techniques:
[1] Strategy: Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur, or provides constructs that make it easier to avoid.
Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.
[2] Understand the context in which your data will be used, and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.
For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.
Parts of the same output document may require different encodings, which will vary depending on whether the output is in the:
[-] HTML body
[-] Element attributes (such as src="XYZ")
[-] URIs
[-] JavaScript sections
[-] Cascading Style Sheets and style property
Note that HTML Entity Encoding is only appropriate for the HTML body.
Consult the XSS Prevention Cheat Sheet
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
for more details on the types of encoding and escaping that are needed.
[3] Strategy: Identify and Reduce Attack Surface
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.
[4] Strategy: Output Encoding
For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing the web page encoding. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.
[5] Strategy: Identify and Reduce Attack Surface
To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.
[6] Strategy: Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy: a whitelist of acceptable inputs that strictly conform to specifications. Reject input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on a blacklist of malicious or malformed inputs. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."
When dynamically constructing web pages, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. All input should be validated and cleansed: not only parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so on. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. It is common to see data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.
Note that proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent XSS, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, in a chat application, the heart emoticon ("<3") would likely pass the validation step, since it is commonly used. However, it cannot be directly inserted into the web page because it contains the "<" character, which would need to be escaped or otherwise handled. In this case, stripping the "<" might reduce the risk of XSS, but it would produce incorrect behavior because the emoticon would not be recorded. This might seem to be a minor inconvenience, but it would be more important in a mathematical forum that wants to represent inequalities.
Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.
Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.

CWE:

74

External References:

OWASP Article
The Cross-Site Request Forgery FAQ

Phishing Through Frames

TOC

Cause:

Sanitation of hazardous characters was not performed correctly on user input

Risk:

It is possible to persuade a naive user to supply sensitive information such as username, password, credit card number, social security number etc.
Phishing is a social engineering technique where an attacker masquerades as a legitimate entity with which the victim might do business in order to prompt the user to reveal some confidential information (very frequently authentication credentials) that can later be used by an attacker. Phishing is essentially a form of information gathering or "fishing" for information.
It is possible for an attacker to inject a frame or an iframe tag with malicious content. An incautious user may browse it and not realize that he is leaving the original site and surfing to a malicious site. The attacker may then lure the user to login again, thus acquiring his login credentials.
The fact that the fake site is embedded in the original site helps the attacker by giving his phishing attempts a more reliable appearance.

Affected Products:

This issue may affect different types of products.

Fix Recommendation:

General

There are several mitigation techniques:
[1] Strategy: Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur, or provides constructs that make it easier to avoid.
Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.
[2] Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.
For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.
Parts of the same output document may require different encodings, which will vary depending on whether the output is in the:
[-] HTML body
[-] Element attributes (such as src="XYZ")
[-] URIs
[-] JavaScript sections
[-] Cascading Style Sheets and style property
Note that HTML Entity Encoding is only appropriate for the HTML body.
Consult the XSS Prevention Cheat Sheet
http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
for more details on the types of encoding and escaping that are needed.
[3] Strategy: Identify and Reduce Attack Surface
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.
[4] Strategy: Output Encoding
For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.
[5] Strategy: Identify and Reduce Attack Surface
To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.
[6] Strategy: Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy: a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on a blacklist of malicious or malformed inputs. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."
When dynamically constructing web pages, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. All input should be validated and cleansed, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. It is common to see data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.
Note that proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent XSS, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, in a chat application, the heart emoticon ("<3") would likely pass the validation step, since it is commonly used. However, it cannot be directly inserted into the web page because it contains the "<" character, which would need to be escaped or otherwise handled. In this case, stripping the "<" might reduce the risk of XSS, but it would produce incorrect behavior because the emoticon would not be recorded. This might seem to be a minor inconvenience, but it would be more important in a mathematical forum that wants to represent inequalities.
Even if you make a mistake in your validation (such as forgetting one of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.
Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a

component is reused or moved elsewhere.

CWE:

79

External References:

FTC Consumer Alert - "How Not to Get Hooked by a 'Phishing' Scam"

Reflected Cross Site Scripting

TOC

Cause:

- Cross-site scripting (XSS) vulnerabilities arise when an attacker sends malicious code to the victim's browser, mostly using JavaScript. A vulnerable web application might embed untrusted data in the output, without filtering or encoding it. In this way, an attacker can inject a malicious script to the application, and the script will be returned in the response. This will then run on the victim's browser.
- In particular, sanitization of hazardous characters was not performed correctly on user input or untrusted data.
- In reflected attacks, an attacker tricks an end user into sending request containing malicious code to a vulnerable Web server, which then reflects the attack back to the end user's browser.
- The server receives the malicious data directly from the HTTP request and reflects it back in the HTTP response. The most common method of sending malicious content is adding it as a parameter in a URL that is posted publicly or e-mailed directly to the victim. URLs that contain the malicious script constitute the core of many phishing schemes, whereby the convinced victim visits a URL that refers to a vulnerable site. The site then reflects the malicious content back to the victim, and then the content is executed by the victim's browser.

Risk:

XSS attacks can expose the user's session cookie, allowing the attacker to hijack the user's session and gain access to the user's account, which could lead to impersonation of users.
An attacker could modify and view the users' records and perform transactions as those users. The attacker may be able to perform privileged operations on behalf of the user, or gain access to any sensitive data belonging to the user. This would be especially dangerous if the user has administrator permissions.
The attacker could even run a malicious script on the victim's browser which would redirect the user to other pages or sites, modify content presentation, or even make it possible to run malicious software or a crypto miner.

Exploit Example:

The following example shows a script that returns a parameter value in the response.
The parameter value is sent to the script using a GET request, and then returned in the response embedded in the HTML.

```
[REQUEST]
GET /index.aspx?name=JSmith HTTP/1.1
```

```
[RESPONSE]
HTTP/1.1 200 OK
Server: SomeServer
Date: Sun, 01 Jan 2002 00:31:19 GMT
Content-Type: text/html
Accept-Ranges: bytes
Content-Length: 27

<HTML>
Hello JSmith
</HTML>
```

An attacker might leverage the attack like this. In this case, the JavaScript code will be executed by the browser.

```
[REQUEST]
GET /index.aspx?name=>"'><script>alert('XSS')</script> HTTP/1.1
```

```
[RESPONSE]
HTTP/1.1 200 OK
Server: SomeServer
Date: Sun, 01 Jan 2002 00:31:19 GMT
Content-Type: text/html
Accept-Ranges: bytes
Content-Length: 83

<HTML>
Hello >"'><script>alert('XSS')</script>
</HTML>
```

Fix Recommendation:

General

Fully encode all dynamic data from an untrusted source that is inserted into the webpage, to ensure it is treated as literal text and not as a script that could be executed or markup that could be rendered.
Consider the context in which your data will be used, and contextually encode the data as close as possible to the actual output: e.g. HTML encoding for HTML content; HTML Attribute encoding for data output to attribute values; JavaScript encoding for dynamically generated JavaScript. For example, when HTML encoding non-alphanumeric characters into HTML entities, `<` and `>` would become `<` and `>`.
As an extra defensive measure, validate all external input on the server, regardless of source. Carefully check each input parameter against a rigorous positive specification (allowlist) defining data type; size; range; format; and acceptable values. Regular expressions or framework controls may be useful in some cases, though this is not a replacement for output encoding.
Output encoding and data validation must be done on all untrusted data, wherever it comes from: e.g. form fields, URL parameters, web service arguments, cookies, any data from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files and filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.
For every web page that is returned by the server, explicitly set the `Content-Type` HTTP response header. This header value should define a specific character encoding (charset), such as `ISO-8859-1` or `UTF-8`. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page, which would allow a potential attacker to bypass XSS protections.
Additionally, set the `httpOnly` flag on the session cookie, to prevent any XSS exploits from stealing a user's cookie.
Prefer using a framework or standard library that prevents this vulnerability by automatically encoding all dynamic output based on context, or at least that provides constructs that make it easier to avoid.
For every web page that is returned by the server, explicitly set the `Content-Security-Policy` HTTP response header, In order to make it significantly more difficult for the attacker to actually exploit the XSS attack.

CWE:

79

External References:

Cross-site Scripting (XSS)
OWASP XSS Cheat Sheet

Application Data

Visited URLs 155

TOC

URL
http://10.163.30.226:8088/survey_f_line_audit/CaptchaTest
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/?jsoncallback=jQuery21109417827936253953_1641815545330
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/?jsoncallback=jQuery21109417827936253953_1641815545330
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545330&_=1641815545331
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545332&_=1641815545333
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545334&_=1641815545335
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545336&_=1641815545337
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545338&_=1641815545339
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545340&_=1641815545341
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545342&_=1641815545343
http://10.163.30.226:8088/survey_f_line_audit/
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545344&_=1641815545345
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545346&_=1641815545347
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545348&_=1641815545349
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21109417827936253953_1641815545350&_=1641815545351
http://10.163.30.226:8088/survey_f_line_audit/firka_surveyor_wf.jsp
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21106754165392689226_1641815562259
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/?jsoncallback=jQuery21106754165392689226_1641815562259
http://10.163.30.226:8088/survey_f_line_audit/firka_surveyor_survey_wise.jsp
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211001098179449395 6087_1641815566534
http://10.163.30.226:8088/survey_f_line_audit/f_line_report.jsp
http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/?jsoncallback=jQuery21104672346449820 4725_1641815568585&_=1641815568586

http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise?jsoncallback=jQuery211046723464498204725_1641815568585

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report?jsoncallback=jQuery211046723464498204725_1641815568585

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_ins?jsoncallback=jQuery211046723464498204725_1641815568585

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21102890680393544227_1641815581691

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/?jsoncallback=jQuery21102890680393544227_1641815581691

http://10.163.30.226:8088/survey_f_line_audit/firka_surveyor_scheduled.jsp

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery211026767536530084257_1641815587455

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211031820618992229_1641815592549

http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/?jsoncallback=jQuery2110146332119191541_1641815594875&_=1641815594876

http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise?jsoncallback=jQuery2110146332119191541_1641815594875

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report?jsoncallback=jQuery2110146332119191541_1641815594875

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery211014904499440757268_1641815599036

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery211015977376173197055_1641815603018

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211008001124743538535_1641815606387

http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/?jsoncallback=jQuery211014591377617423462_1641815608034&_=1641815608035

http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise?jsoncallback=jQuery211014591377617423462_1641815608034

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report?jsoncallback=jQuery211014591377617423462_1641815608034

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_Report?jsoncallback=jQuery211014591377617423462_1641815608034

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/save_attachments?jsoncallback=jQuery211014591377617423462_1641815608036

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21105406775930404588_1641815644874

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/?jsoncallback=jQuery21105406775930404588_1641815644874

http://10.163.30.226:8088/survey_f_line_audit/officer_report_menu.jsp

http://10.163.30.226:8088/survey_f_line_audit/viewsummary.jsp

http://10.163.30.226:8088/survey_f_line_audit/view_report.jsp

http://10.163.30.226:8088/survey_f_line_service_audit/resources/det/getRejectedReason/?jsoncallback=jQuery21105513174258813793_1641815660608&_=1641815660609

http://10.163.30.226:8088/survey_f_line_service_audit/resources/Land/getAppDet_surveywise?jsoncallback=jQuery21105513174258813793_1641815660608

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/f_line_report?jsoncallback=jQuery21105513174258813793_1641815660608

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks?jsoncallback=jQuery21105513174258813793_1641815660608

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getDISRemarks?jsoncallback=jQuery21105513174258813793_1641815660608

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getFirkaRemarks_returned?jsoncallback=jQuery21105513174258813793_1641815660608

http://10.163.30.226:8088/survey_f_line_service_audit/resources/report/getTSLR_Remarks?jsoncallback=jQuery21105513174258813793_1641815660608

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_summary_App?jsoncallback=jQuery211013125545377596648_1641815653414

http://10.163.30.226:8088/survey_f_line_audit/view_report_surveyWise.jsp

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211062384093183127 68_1641815658305

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21105018571783927821_1641815681943

http://10.163.30.226:8088/survey_f_line_audit/js/sweetalert.min.js

http://10.163.30.226:8088/survey_f_line_audit/js/SHA1.js

http://10.163.30.226:8088/survey_f_line_audit/js/SHA256.js

http://10.163.30.226:8088/survey_f_line_audit/js/hmac-sha256.js

http://10.163.30.226:8088/survey_f_line_audit/resources/scripts/sha1.js

http://10.163.30.226:8088/survey_f_line_audit/resources/js/login.js

http://10.163.30.226:8088/survey_f_line_audit/js/modernizr.js

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/?jsoncallback=jQuery21103241729050346722_1641815799564

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/?jsoncallback=jQuery21103241729050346722_1641815799564

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103241729050346722_1641815799564&_=1641815799565

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21108858933608819208_1641815808605

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/?jsoncallback=jQuery21108858933608819208_1641815808605

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211072354368399431 39_1641815812612

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21107507759266118663_1641815855659

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211079121796243739 74_1641815898035

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21105816477746315216_1641815899695

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211026853445504308 97_1641815923056

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_DIS_App?jsoncallback=jQuery21101925987644364009_1641815944004

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_TSLR_AppDet?jsoncallback=jQuery211026128542355802_1641815957853

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/?jsoncallback=jQuery21104942975392205593_1641815967636

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/?jsoncallback=jQuery21103309822644855029 4_1641816279420

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/?jsoncallback=jQuery21103309822644855029 4_1641816279420

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279420&_=1641816279421

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279422&_=1641816279423

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279424&_=1641816279425

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279426&_=1641816279427

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855

0294_16418162794 28&_=1641816279429

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279430&_=1641816279431

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279432&_=1641816279433

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279434&_=1641816279435

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279436&_=1641816279437

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279438&_=1641816279439

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21103309822644855 0294_1641816279440&_=1641816279441

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery211035828906874429434_164181629 0153

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/?jsoncallback=jQuery211035828906874429434_1 641816290153

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211037219847282012 53_1641816293009

http://10.163.30.226:8088/survey_f_line_audit/js/sweetalert.min.js

http://10.163.30.226:8088/survey_f_line_audit/js/modernizr.js

http://10.163.30.226:8088/survey_f_line_audit/js/jquery.min.js

http://10.163.30.226:8088/survey_f_line_audit/js/SHA1.js

http://10.163.30.226:8088/survey_f_line_audit/js/SHA256.js

http://10.163.30.226:8088/survey_f_line_audit/js/esapi.js

http://10.163.30.226:8088/survey_f_line_audit/js/hmac-sha256.js

http://10.163.30.226:8088/survey_f_line_audit/js/jquery-ui.js

http://10.163.30.226:8088/survey_f_line_audit/resources/scripts/sha1.js

http://10.163.30.226:8088/survey_f_line_audit/js/ESAPI_Standard_en_US.properties.js

http://10.163.30.226:8088/survey_f_line_audit/js/Base.esapi.properties.js

http://10.163.30.226:8088/survey_f_line_audit/resources/js/service.js

http://10.163.30.226:8088/survey_f_line_audit/resources/js/login.js

http://10.163.30.226:8088/survey_f_line_audit/error.jsp

http://10.163.30.226:8088/survey_f_line_audit/js/jquery.min.js

http://10.163.30.226:8088/survey_f_line_audit/logout.jsp

http://10.163.30.226:8088/survey_f_line_audit/DIS_app_display.jsp

http://10.163.30.226:8088/survey_f_line_audit/Tahsildhar_app_display.jsp

http://10.163.30.226:8088/survey_f_line_audit/js/esapi.js

http://10.163.30.226:8088/survey_f_line_audit/js/ESAPI_Standard_en_US.properties.js

http://10.163.30.226:8088/survey_f_line_audit/js/Base.esapi.properties.js

http://10.163.30.226:8088/survey_f_line_audit/js/jquery.dataTables.min.js

http://10.163.30.226:8088/survey_f_line_audit/js/dataTables.bootstrap.min.js

http://10.163.30.226:8088/survey_f_line_audit/resources/js/service.js

http://10.163.30.226:8088/survey_f_line_audit/resources/js/form-36.js

http://10.163.30.226:8088/survey_f_line_audit/resources/js/on-load-form-36.js

http://10.163.30.226:8088/survey_f_line_audit/resources/js/app_det.js

http://10.163.30.226:8088/survey_f_line_audit/js/master.js

http://10.163.30.226:8088/survey_f_line_audit/js/jquery.js

http://10.163.30.226:8088/survey_f_line_audit/js/jquery.session.js

http://10.163.30.226:8088/survey_f_line_audit/js/bootstrap.min.js

http://10.163.30.226:8088/survey_f_line_audit/js/editor_plugin.js

http://10.163.30.226:8088/survey_f_line_audit/session_logout.jsp
http://10.163.30.226:8088/survey_f_line_audit/js/jquery.jqGrid.min.js
http://10.163.30.226:8088/survey_f_line_audit/js/jquery-ui.js
http://10.163.30.226:8088/survey_f_line_audit/js/jquery.blockUI.js
http://10.163.30.226:8088/survey_f_line_audit/js/grid.locale-en.js
http://10.163.30.226:8088/survey_f_line_audit/resources/js/f_line_report.js
http://10.163.30.226:8088/survey_f_line_audit/resources/js/f_line_workflow.js
http://10.163.30.226:8088/survey_f_line_service_audit/
http://10.163.30.226:8088/survey_f_line_audit/js/jquery.dataTables.min.js
http://10.163.30.226:8088/survey_f_line_audit/js/dataTables.bootstrap.min.js
http://10.163.30.226:8088/survey_f_line_audit/resources/js/form-36.js
http://10.163.30.226:8088/survey_f_line_audit/resources/js/on-load-form-36.js
http://10.163.30.226:8088/survey_f_line_audit/js/master.js
http://10.163.30.226:8088/survey_f_line_audit/resources/js/app_det.js
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/?jsoncallback=jQuery21107175137559710869_1641816384070
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_login_details/?jsoncallback=jQuery21107175137559710869_1641816384070
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384070&_=1641816384071
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384072&_=1641816384073
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384074&_=1641816384075
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384076&_=1641816384077
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384078&_=1641816384079
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384080&_=1641816384081
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384082&_=1641816384083
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384084&_=1641816384085
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384086&_=1641816384087
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384088&_=1641816384089
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/get_firka_name/15/04/03?jsoncallback=jQuery21107175137559710869_1641816384090&_=1641816384091
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp?jsoncallback=jQuery21106218749613529222_1641816393698
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getApp_count/no/?jsoncallback=jQuery21106218749613529222_1641816393698
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getAppDet_surveywise/?jsoncallback=jQuery211005638949848822 9606_1641816396898

Failed Requests 51

TOC

URL	Reason

http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getLoginRole/?jsoncallback=jQuery21109417827936253953_1641815545352	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/bootstrap.min.js	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/footer.html	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getLoginRole/?jsoncallback=jQuery21103241729050346722_1641815799586	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/login_validation/?jsoncallback=jQuery21104942975392205593_1641815967636	Response Status '500' - Internal Server Error
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getLoginRole/?jsoncallback=jQuery21103309822644855029 4_1641816279442	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/Pending_Workflow_RDO.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/Pending_Workflow_RDO.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/bootstrap.min.js	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/header.html	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/footer.html	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/RDO_Menu.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/Pending_Workflow_RDO.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/fs_menu.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/firka_surveyor_wf.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/Pending_Workflow_DRO.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/DIS_app_display.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/dis_menu.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/Tahsildhar_app_display.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/Pending_Workflow_CLR.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/tah_menu.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/sislist.html	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/error.html	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/Error500.html	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/DIS_survey_wise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/TSLR_survey_wise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/firka_surveyor_survey_wise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/f_line_report.jsp	Response Status '404' - Not Found

http://10.163.30.226:8088/survey_f_line_audit/resources/js/TSLR_report.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/DIS_report.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/Error500.html	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/session_logout.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/DIS_survey_wise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/TSLR_survey_wise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/firka_surveyor_survey_wise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/f_line_report.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/firka_returned_application.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/view_report.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/view_report_surveyWise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/TSLR_report.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/DIS_report.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/firka_returned_application.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/view_report.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/view_report_surveyWise.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/fs_menu.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/firka_surveyor_wf.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/dis_menu.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/DIS_app_display.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/tah_menu.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_audit/resources/js/Tahsildhar_app_display.jsp	Response Status '404' - Not Found
http://10.163.30.226:8088/survey_f_line_service_audit/resources/app_det/getLoginRole/?jsoncallback=jQuery21107175137559710869_1641816384092	Response Status '404' - Not Found