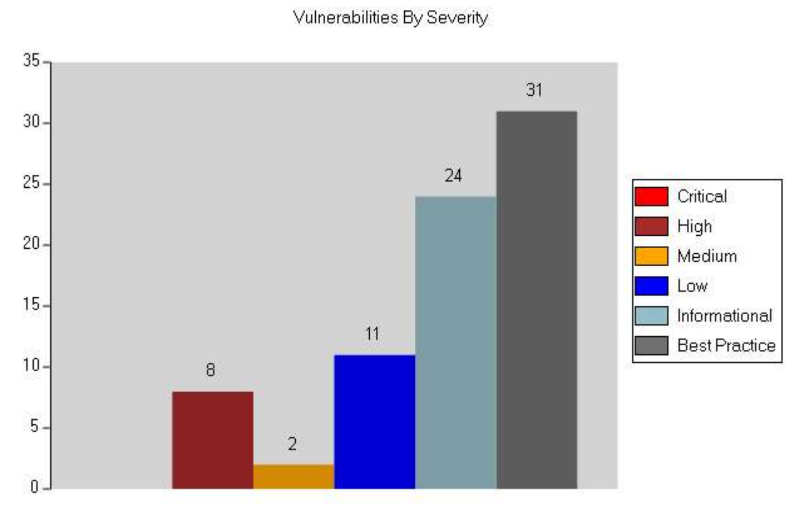
Fortify WebInspect

Vulnerability (Legacy)

Web Application Assessment Report

Scan Name:	Workflow: retest http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec		
Policy:	Standard	**Crawl Sessions:**	855
Scan Date:	1/5/2026 5:36:19 PM	**Vulnerabilities:**	21
Scan Version:	24.4.0.70	**Scan Duration:**	3 hours : 19 minutes
Scan Type:	Site	**Client:**	Custom

Server: http://172.16.21.39:7001



High **Insecure Transport**

Summary:

Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec has failed this policy. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication:

An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execution of actions not intended by the application developers.

Fix:

For Security Operations:
Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For Development:
Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For QA:
Test the application not only from the perspective of a normal user, but also from the perspective of a malicious one.

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/inicioErr.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/

High **Often Misused: Login**

Summary:

An unencrypted login form has been discovered. Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. If the login form is being served over SSL, the page that the form is being submitted to MUST be accessed over SSL. Every link/URL present on that page (not just the form action) needs to be served over HTTPS. This will prevent Man-in-the-Middle attacks on the login form. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication:

An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execution of actions not intended by the application developers.

Fix:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Reference:

Advisory:http://www.kb.cert.org/vuls/id/466433

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/
- http://172.16.21.39:7001/SisAdhereVerFirmas/inicioErr.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

High **Cross-Frame Scripting**

Summary:

A Cross-Frame Scripting (XFS) vulnerability can allow an attacker to load the vulnerable application inside an HTML iframe tag on a malicious page. The attacker could use this weakness to devise a Clickjacking attack to conduct phishing, frame sniffing, social engineering or Cross-Site Request Forgery attacks.

Clickjacking
The goal of a Clickjacking attack is to deceive the victim (user) into interacting with UI elements of the attacker's choice on the target web site without their knowledge and then executing privileged functionality on the victim's behalf. To achieve this goal, the attacker must exploit the XFS vulnerability to load the attack target inside an iframe tag, hide it using Cascading Style Sheets (CSS) and overlay the phishing content on the malicious page. By placing the UI elements on the phishing page so they overlap with those on the page targeted in the attack, the attacker can ensure that the victim must interact with the UI elements on the target page not visible to the victim.

WebInspect has detected a page which potentially handles sensitive information using an HTML form with a password input field and is missing XFS protection.

An effective frame-busting technique was not observed while loading this page inside a frame.

Execution:

Create a test page containing an HTML iframe tag whose src attribute is set to http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec. Successful framing of the target page indicates that the application is susceptibile to XFS.

Note that WebInspect will report only one instance of this check across each host within the scope of the scan. The other visible pages on the site may, however, be vulnerable to XFS as well and therefore should be protected against it with an appropriate fix.

Implication:

A Cross-Frame Scripting weakness could allow an attacker to embed the vulnerable application inside an iframe. Exploitation of this weakness could result in:

Hijacking of user events such as keystrokes
Theft of sensitive information
Execution of privileged functionality through combination with Cross-Site Request Forgery attacks

Fix:

The Content Security Policy (CSP) frame-ancestors directive obsoletes the X-Frame-Options header. Both provide for a policy-based mitigation technique against cross-frame scripting vulnerabilities. The difference is that while the X-Frame-Options technique only checks against the top-level document's location, the CSP frame-ancestors header checks for conformity from all ancestors.

If both CSP frame-ancestors and X-Frame-Options headers are present and supported, the CSP directive will prevail. WebInspect recommends using both CSP frame-ancestors and X-Frame-Options headers as CSP is not supported by Internet Explorer and many older versions of other browsers.

In addition, developers must also use client-side frame busting JavaScript as a protection against XFS. This will enable users of older browsers that do not support the X-Frame-Options header to also be protected from Clickjacking attacks.

X-Frame-Options

Developers can use this header to instruct the browser about appropriate actions to perform if their site is included inside an iframe.Developers must set the X-Frame-Options header to one of the following permitted values:

- DENY

Deny all attempts to frame the page

- SAMEORIGIN

The page can be framed by another page only if it belongs to the same origin as the page being framed

Content-Security-Policy: frame-ancestors

Developers can use the CSP header with the frame-ancestors directive, which replaces the X-Frame-Options header, to instruct the browser about appropriate actions to perform if their site is included inside an iframe. Developers can set the frame-ancestors attribute to one of the following permitted values:

- 'none'

Equivalent to "DENY" - deny all attempts to frame the page

- 'self'

Equivalent to "SAMEORIGIN" - the page can be framed by another page only if it belongs to the same origin as the page being framed

- <scheme-source>

Developers can also specify a schema such as http: or https: that can frame the page.

Reference:

Frame Busting:
Busting Frame Busting: A Study of Clickjacking Vulnerabilities on Popular Sites
OWASP: Busting Frame Busting

OWASP:
Clickjacking

Content-Security-Policy (CSP)
CSP: frame-ancestors

Specification:
Content Security Policy Level 2
X-Frame-Options IETF Draft

Server Configuration:
IIS
Apache, nginx

HP 2012 Cyber Security Report
The X-Frame-Options header - a failure to launch

File Names: ● http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

High **Often Misused: File Upload**

Summary:

Permitting users to upload files may allow attackers to inject dangerous content or malicious code to run on the server.

WebInspect has succesfully uploaded a file to the server.

WebInspect has successfully uploaded **WebInspect_11503_File_8.exe** to the server at:

http://172.16.21.39:7001/SisAdhereVerFirmas/lote/

NOTES:

- ***The uploaded file is an executable file.***
- ***The file was uploaded to the server by WebInspect as part of the Dangerous File Upload check. Please locate and remove the file from the server once testing is complete.***
- ***The check accepts three check inputs to analyze the application response. These check inputs are user configurable. The check inputs allow the check to be customized per application and help improve the check accuracy. If this result appears to be a false positive (or the scan is a suspect of false negatives for this vulnerability) please consider configuring one or more of the available check inputs to get more accurate results.***

Execution:

Browse to :
http://172.16.21.39:7001/SisAdhereVerFirmas/lote/navegarCargarArchivo.reniec
and upload a file of the reported type.

Implication:

Regardless of the language in which a program is written, the most devastating attacks often involve remote code execution, whereby an attacker succeeds in executing malicious code in the program's context. If attackers are allowed to upload files to a directory that is accessible from the Web and cause these files to be passed to a code interpreter (e.g. JSP/ASPX/PHP), then they can cause malicious code contained in these files to execute on the server.

The exact implications depend upon the nature of the files an attacker would be able to upload. Implications range from unauthorized content publishing to aid in phising attacks, all the way to full compromise of the web server.

Even if a program stores uploaded files under a directory that isn't accessible from the Web, attackers might still be able to leverage the ability to introduce malicious content into the server environment to mount other attacks. If the program is susceptible to path manipulation, command injection, or dangerous file inclusion vulnerabilities, then an attacker might upload a file with malicious content and cause the program to read or execute it by exploiting another vulnerability.

Fix:

Do not accept attachments if they can be avoided. If a program must accept attachments, then restrict the ability of an attacker to supply malicious content by only accepting the specific types of content the program expects. Most attacks that rely on uploaded content require that attackers be able to supply content of their choosing. Placing restrictions on the content the program will accept will greatly limit the range of possible attacks. Check file names, extensions, and file content to make sure they are all expected and acceptable for use by the application. Make it difficult for the attacker to determine the name and location of uploaded files. Such solutions are often program-specific and vary from storing uploaded files in a directory with a name generated from a strong random value when the program is initialized, to assigning each uploaded file a random name and tracking them with entries in a database.

Ensure that the following steps are taken to sanitize the file being received:

- Limit the types of files that can be uploaded. For instance, on an image upload page, any file other than a .jpg should be refused.
- Ensure that the web user has no control whatsoever over the name and location of the uploaded file on the server.
- Never use the name that the user assigns it.
- Never derive the filename from the web user's username or session ID.
- Do not place the file in a directory accessible by web users. It is preferable for this location to be outside of the webroot.
- Ensure that strict permissions are set on both the uploaded file and the directory it is located in.
- Do not allow execute permissions on uploaded files. If possible, deny all permission for all users but the web application user.
- Verify that the uploaded file contains appropriate content. For instance, an uploaded JPEG should have a standard JPEG file header.

Reference:

Secure file upload in PHP web applications

Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 434

OWASP Unrestricted File Upload

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/lote/navegarCargarArchivo.reniec

Medium **Poor Error Handling: Unhandled Exception**

Summary:

A Java runtime error message was found, indicating that the assessment has generated an unhandled exception in your Web application code. Unhandled exceptions are circumstances in which the application has received user input (or parameters) that it did not expect and does not know how to handle. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system. Recommendations include providing logic to handle any situation in which a parameter is not being passed or is being passed incorrectly.

Implication:

The error message may contain the location of the file in which the offending function is located. This may disclose the webroot's absolute path, as well as give the attacker the location of application include files or configuration information. It may even disclose the portion of code that failed.

Note: This vulnerability may be a false positive if the page it is flagged on is technical documentation.

Fix:

For Security Operations:

Ultimately, this problem will require a change to the application code. However, follow these recommendations to help ensure a secure Web application:

- **Use Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by using error messages

such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Use consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft an attack.
- **Proper Error Handling:** Use generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be used by an attacker when orchestrating an attack.

For Development:

This problem arises from the improper validation of characters that are accepted by the application. Any time a parameter is passed into a dynamically-generated web page, you must assume that the data could be incorrectly formatted. The application should contain sufficient logic to handle any situation in which a parameter is not being passed or is being passed incorrectly. Keep in mind how the data is being submitted, as a result of a GET or a POST. Additionally, to develop secure and stable code, treat cookies the same as parameters. The following recommendations will help ensure that you are delivering secure Web applications.

- **Stringently define the data type:** Stringently define the data type (a string, an alphanumeric character, etc.) that the application will accept. Validate input for improper characters. Adopt the philosophy of using what is good rather than what is bad. Define the allowed set of characters. For instance, if a field is to receive a number, allow that field to accept only numbers. Define the maximum and minimum data lengths that the application will accept.
- **Verify parameter is being passed:** If a parameter that is expected to be passed to a dynamic Web page is omitted, the application should provide an acceptable error message to the user. Also, never use a parameter until you have verified that it has been passed into the application.
- **Verify correct format:** Never assume that a parameter is of a valid format. This is especially true if the parameter is being passed to a SQL database. Any string that is passed directly to a database without first being checked for proper format can be a major security risk. Also, just because a parameter is normally provided by a combo box or hidden field, do not assume the format is correct. A hacker will first try to alter these parameters while attempting to break into your site.
- **Verify file names being passed in via a parameter:** If a parameter is being used to determine which file to process, never use the file name before it is verified as valid. Specifically, test for the existence of characters that indicate directory traversal, such as .../, c:\, and /.
- **Do not store critical data in hidden parameters:** Many programmers make the mistake of storing critical data in a hidden parameter or cookie. They assume that since the user doesn't see it, it's a good place to store data such as price, order number, etc. Both hidden parameters and cookies can be manipulated and returned to the server, so never assume the client returned what you sent via a hidden parameter or cookie.

For QA:

Simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site. This assessment performs both of these tasks.

Ensure that the error handling scheme is consistent and does not reveal private information about your Web application. A seemingly innocuous piece of information can provide an attacker the means to discover additional information that can be used to conduct an attack. Make the following observations:

- Do you receive the same type of error for existing and non-existing files?
- Does the error include phrases (such as "Permission Denied") that could reveal the existence of a file?

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/recursos

Medium **Cross-Frame Scripting**

Summary:

A Cross-Frame Scripting (XFS) vulnerability can allow an attacker to load the vulnerable application inside an HTML iframe tag on a malicious page. The attacker could use this weakness to devise a Clickjacking attack to conduct phishing, frame sniffing, social engineering or Cross-Site Request Forgery attacks.

Clickjacking
The goal of a Clickjacking attack is to deceive the victim (user) into interacting with UI elements of the attacker's choice on the target web site without their knowledge and then executing privileged functionality on the victim's behalf. To achieve this goal, the attacker must exploit the XFS vulnerability to load the attack target inside an iframe tag, hide it using Cascading Style Sheets (CSS) and overlay the phishing content on the malicious page. By placing the UI elements on the phishing page so they overlap with those on the page targeted in the attack, the attacker can ensure that the victim must interact with the UI elements on the target page not visible to the victim.

WebInspect has detected a response containing one or more forms that accept user input but is missing XFS protection.

An effective frame-busting technique was not observed while loading this page inside a frame.

Execution:

Create a test page containing an HTML iframe tag whose src attribute is set to http://172.16.21.39:7001/SisAdhereVerFirmas/listaExpresiones.reniec. Successful framing of the target page indicates that the application is susceptibile to XFS.

Note that WebInspect will report only one instance of this check across each host within the scope of the scan. The other visible pages on the site may, however, be vulnerable to XFS as well and therefore should be protected against it with an appropriate fix.

Implication:

A Cross-Frame Scripting weakness could allow an attacker to embed the vulnerable application inside an iframe. Exploitation of this weakness could result in:

Hijacking of user events such as keystrokes
Theft of sensitive information
Execution of privileged functionality through combination with Cross-Site Request Forgery attacks

Fix:

The Content Security Policy (CSP) frame-ancestors directive obsoletes the X-Frame-Options header. Both provide for a policy-based mitigation technique against cross-frame scripting vulnerabilities. The difference is that while the X-Frame-Options technique only checks against the top-level document's location, the CSP frame-ancestors header checks for conformity from all ancestors.

If both CSP frame-ancestors and X-Frame-Options headers are present and supported, the CSP directive will prevail. WebInspect recommends using both CSP frame-ancestors and X-Frame-Options headers as CSP is not supported by Internet Explorer and many older versions of other browsers.

In addition, developers must also use client-side frame busting JavaScript as a protection against XFS. This will enable users of older browsers that do not support the X-Frame-Options header to also be protected from Clickjacking attacks.

X-Frame-Options
Developers can use this header to instruct the browser about appropriate actions to perform if their site is included inside an iframe.Developers must set the X-Frame-Options header to one of the following permitted values:

- DENY
Deny all attempts to frame the page
- SAMEORIGIN
The page can be framed by another page only if it belongs to the same origin as the page being framed

Content-Security-Policy: frame-ancestors

Developers can use the CSP header with the frame-ancestors directive, which replaces the X-Frame-Options header, to instruct the browser about appropriate actions to perform if their site is included inside an iframe. Developers can set the frame-ancestors attribute to one of the following permitted values:

- 'none'
Equivalent to "DENY" - deny all attempts to frame the page
- 'self'
Equivalent to "SAMEORIGIN" - the page can be framed by another page only if it belongs to the same origin as the page being framed
- <scheme-source>
Developers can also specify a schema such as http: or https: that can frame the page.

Reference:

Frame Busting:
Busting Frame Busting: A Study of Clickjacking Vulnerabilities on Popular Sites
OWASP: Busting Frame Busting

OWASP:
Clickjacking

Content-Security-Policy (CSP)
CSP: frame-ancestors

Specification:
Content Security Policy Level 2
X-Frame-Options IETF Draft

Server Configuration:

IIS
Apache, nginx

HP 2012 Cyber Security Report
The X-Frame-Options header - a failure to launch

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/listaExpresiones.reniec

---

Low **Often Misused: File Upload**

Summary:

An indicator of file upload capability was found. File upload capability allows a web user to send a file from his or her computer to the webserver. If the web application that receives the file does not carefully examine it for malicious content, an attacker may be able to use file uploads to execute arbitrary commands on the server. Recommendations include adopting a strict file upload policy that prevents malicious material from being uploaded via sanitization and filtering.

Implication:

The exact implications depend upon the nature of the files an attacker would be able to upload. Implications range from unauthorized content publishing to aid in phising attacks, all the way to full compromise of the web server.

Fix:

For Security Operations:
This check is part of unknown application testing. Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions for this issue. If there is no apparent file upload capability on the page, this check may be safely ignored. You can instruct the scanner to ignore this vulnerability by right-clicking the vulnerability node on the displayed results tree and click "Ignore Vulnerability."

For QA:
This issue will need to be resolved in the production code. Notify the appropriate developer of this issue.

For Development:
Ensure that the following steps are taken to sanitize the file being received:

- Limit the types of files that can be uploaded. For instance, on an image upload page, any file other than a .jpg should be refused.
- Ensure that the web user has no control whatsoever over the name and location of the uploaded file on the server.
- Never use the name that the user assigns it.
- Never derive the filename from the web user's username or session ID.
- Do not place the file in a directory accessible by web users. It is preferable for this location to be outside of the webroot.
- Ensure that strict permissions are set on both the uploaded file and the directory it is located in.
- Do not allow execute permissions on uploaded files. If possible, deny all permission for all users but the web application user.
- Verify that the uploaded file contains appropriate content. For instance, an uploaded JPEG should have a standard JPEG file header.

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/js/jquery/jquery.form.js
- http://172.16.21.39:7001/SisAdhereVerFirmas/lote/navegarCargarArchivo.reniec

---

Low **System Information Leak: Internal IP**

Summary:

A string matching an internal/reserved IPv4 or IPv6 address range was discovered. This may disclose information about the IP addressing scheme of the internal network and can be valuable to attackers.Internal IPv4/IPv6 ranges are:
10.x.x.x
172.16.x.x through 172.31.x.x
192.168.x.x
fd00::x
If not a part of techical documentation, recommendations include removing the string from the production server.

Fix:

This issue can appear for several reasons. The most common is that the application or webserver error message discloses the IP address. This can be solved by determining where to turn off detailed error messages in the application or webserver. Another common reason is due to a comment located in the source of the webpage. This can easily be removed from the source of the page.

File Names: • http://172.16.21.39:7001/SisAdhereVerFirmas/configuracion/accionConsultarIpAcceso.reniec

Low **Web Server Misconfiguration: Server Error Message**

Summary:

A server error response was detected. The server could be experiencing errors due to a misbehaving application, a misconfiguration, or a malicious value sent during the auditing process. While error responses in and of themselves are not dangerous, per se, the error responses give attackers insight into how the application handles error conditions. Errors that can be remotely triggered by an attacker can also potentially lead to a denial of service attack or other more severe vulnerability. Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Implication:

The server has issued a 500 error response. While the body content of the error page may not expose any information about the technical error, the fact that an error occurred is confirmed by the 500 status code. Knowing whether certain inputs trigger a server error can aid or inform an attacker of potential vulnerabilities.

Fix:

For Security Operations:

Server error messages, such as "File Protected Against Access", often reveal more information than intended. For instance, an attacker who receives this message can be relatively certain that file exists, which might give him the information he needs to pursue other leads, or to perform an actual exploit. The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any server error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

Removing Detailed Error Messages

Find instructions for turning off detailed error messaging in IIS at this link:

http://support.microsoft.com/kb/294807

For Development:

From a development perspective, the best method of preventing problems from arising from server error messages is to adopt secure programming techniques that prevent problems that might arise from an attacker discovering too much information about the architecture and design of your web application. The following recommendations can be used as a basis for that.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad. Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

For QA:

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? Inconsistent methods of dealing with errors gives an attacker a very powerful way of gathering information about your web application.

Reference:

Apache:
Security Tips for Server Configuration
Protecting Confidential Documents at Your Site
Securing Apache - Access Control

Microsoft:
How to set required NTFS permissions and user rights for an IIS 5.0 Web server
Default permissions and user rights for IIS 6.0
Description of Microsoft Internet Information Services (IIS) 5.0 and 6.0 status codes

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/usr/share/zoneinfo/GMT%3ffakestp.html
- http://172.16.21.39:7001/SisAdhereVerFirmas/recursos
- http://172.16.21.39:7001/SisAdhereVerFirmas/GET

Low **Web Server Misconfiguration: Insecure Content-Type Setting**

Summary:

Missing a Content-Type header in the HTTP Response could expose the application to Cross-Site Scripting vulnerabilities via:

Content Sniffing Mismatch
Failure to explicitly specify the type of the content served by the requested resource can allow attackers to conduct Cross-Site Scripting attacks by exploiting the inconsistencies in content sniffing techniques employed by the browsers.
The Content-Type header is used by:

- The web server to dictate how the requested resource is interpreted by the user agent. In the absence of this header the browser depends on content sniffing algorithms to guess the type of content and render or interpret it accordingly.
- File upload filters to discard file types not allowed by the application. In the absence of a Content-Type header, the file upload filter relies on the file extension or the content of the file to detect and store an appropriate mime type for the uploaded file.

The lack of explicit content type specification can allow attackers to exploit the mismatch between the mime sniffing algorithm used by the browser and upload filter. By uploading files with benign extensions (like .jpg), an attacker can easily bypass the upload filter to upload files containing malicious HTML content. The browser's content sniffing algorithm will however render it as HTML based on the content of the file thus executing any malicious scripts embedded within the HTML content.

Character Set Mismatch

Character set specification is part of the Content-Type header. Absence of this specification could allow attackers to bypass input validation filters or HTML entity escape functionality and conduct Cross-Site Scripting attacks against the target application. When the character set is not specified, browsers will attempt to guess the most appropriate character set. This could result in a mismatch between the character set assumed by the application during the generation of the content and by the browser during the parsing and interpretation of the same content. An attacker can exploit this inconsistency to encode attacks using a character set that'll hide the malicious payloads from the valdiation filters and escaping mechanisms put in place by the application but at the same time will be interpreted by the browser as a valid executable entity.

Execution:

Below example scenarios demonstrate the exploitation of the weakness:

Content Sniffing Mismatch

. Attacker uploads a file with .jpg extension and no Content-Type specification. The file contains malicious HTML and JavaScript content embedded inside.

. In the absence of the Content-Type header, the application saves the uploaded file along with the mime type of the .jpg

. The attacker uses social engineering to entice the desired target into accessing the uploaded file

. Upon receiving the requested file without the Content-Type header, the target's browser assumes the content type to be HTML based on the HTML and JavaScript content inside and renders the file causing attacker's JavaScript payload to be executed.

Character Set Mismatch

0. Attacker converts the desired payload of <script>alert(document.location)</script> into UTF-7 encoded string +ADw-script+AD4-alert(document.location)+ADw-/script+AD4 and sends it to the vulnerable application.

. An application using the ISO-8859-1 character set for filtering or escaping special characters will fail to detect the the '<' and '>' characters as dangerous

. The absence of character set specification due to the missing Content-Type header will force the browser to guess the character set to use for rendering the application response containing the attacker's payload. If the browser correctly guesses the encoding as UTF-7, the injected payload will be successfully executed.

Implication:

The application fails to impose constraints on the parsing and interpretation of the response content; allowing attackers to bypass validation filters or escaping functionality and introduce malicious scripts and force the browser to execute the desired payload.

Fix:

Configure the server to send the appropriate content type and character set information for the requested resource.

Reference:

Server Configuration
Mime Types in IIS 7
Content Negotiation - Apache HTTP Server

Content Sniffing:
Mime Sniffing Standard
Content Sniffing Signatures
Secure Content Sniffing for Web Browsers [PDF]

OWASP:
OWASP Testing Guide Appendix D: Encoded Injection

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/accionConsultaReversion.reniec

Low **HTML5: Misconfigured Content Security Policy**

Summary:

Content Security Policy (CSP) is a declarative security header that allows developers to dictate which domains the site is allowed to load contents from or initiate connection to when rendered in the web browser. It provides an additional layer of security from critical vulnerabilities like cross site scripting, clickjacking, cross origin access etc. on top of input validation and having an allow list in code. Improperly configured header however, fails to provide this additional layer of security. The policy is defined with the help of fifteen directives including eight that control resource access namely -

- script-src
- img-src
- object-src
- style_src
- font-src
- media-src
- frame-src
- connect-src

Each of these takes a source list as a value specifying domains the site is allowed to access for feature covered by that directive. Developers may use wildcard * to indicate all or part of the source. None of the directives are mandatory. Browsers will either allow all sources for unlisted directive or will derive its value from optional default-src directive. Furthermore, the specification for this header has evolved over time. It was implemented as X-Content-Security-Policy in Firefox until version 23 and in IE until version 10, and was implemented as X-Webkit-CSP in Chrome until version 25. Both of the names are deprecated in favor of the now standard name Content Security Policy. Given number of directives, two deprecated alternate names, and the way multiple occurrences of same header and repeat directive in single header is treated there is a high propensity that developer might misconfigure this header.

Consider following misconfiguration scenarios:

- It can be an overly permissive policy if default-src is not set or set to wildcard and/or other directives are set to wildcard.
- Multiple instances of this header are allowed in same response. A development team and security team might both set the header but with either different policies or one may use now deprecated X-Content-Security-Poilcy or X-Webkit-CSP headers. Deprecated versions are ignored if standard Content-Security-Policy header is present.
- If a directive is repeated within the same instance of the header, all subsequent occurrences are ignored.

Following misconfigurations were detected at http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec:

Overly Permissive Content Security Policy.

- *Content-Security-Policy:default-src,script-src,object-src,font-src,style-src,img-src,frame-src,connect-src,media-src* are set to wildcard.

Execution:

Access link http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec through proxy and notice the misconfigured content security policy header in response headers.

Implication:

Content security policy header provides additional layer of security for site from client side attacks such as cross-site scripting. In absence of this header, an attacker can exploit client side vulnerabilities such as cross-site scripting, clickjacking and cross-site request forgery.

Fix:

Remove wildcard values for source-list to limit scope of cross origin access from site. Ensure canonical *Content Security Policy* name is used to specify policy. X-Content-Security-Policy and X-Webkit-CSP names are deprecated. Any references to these headers are only useful in case support for earlier browsers is desired. Presence of these headers in addition to Content-Security-Policy is said to cause unexpected behaviors on certain versions of browsers.
Content-Security-Policy version 2 standard support:

- Edge: Edge 15 - 18 supported with a nonce bug. Version 75 onwards fully supported.
- Chrome: Chrome 36-38 are missing the plugin-types, child-src, frame-ancestors, base-uri, and form-action directives. Chrome 39 is missing the plugin-types, child-src, base-uri, and form-action directives. Chrome 40 onwards fully supported.
- Firefox; Firefox 31-34 is missing the plugin-types, child-src, frame-ancestors, base-uri, and form-action directives. Firefox 35 is missing the plugin-types, child-src, frame-ancestors, and form-action directives. Firefox 36-44 is missing the plugin-types and child-src directives. Firefox 45+ is missing the plugin-types directive.

Furthermore, the report-uri directive can be configured to receive reports of attempts to violate the policy. These reports can be used as an early indication of security issues in the site as well as to optimize the policy.

Reference:

http://www.w3.org/TR/CSP/
https://owasp.org/www-community/controls/Content_Security_Policy
https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy
https://content-security-policy.com

File Names: • http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

Low **Cookie Security: Missing SameSite Attribute**

Summary:

The SameSite attribute protects cookies from Cross-Site Request Forgery (CSRF) attacks. The browser automatically appends cookies to every HTTP request made to the site that sets the cookie. Cookies might store sensitive data like session ID and authorization token or site data that is shared between different requests to the same site during a session. An attacker can perform an impersonation attack by generating a request to the authenticated site from a third-party site page loaded on the client machine because the browser automatically appended the cookie to the request.
The SameSite attribute on a cookie allows sites to control that behaviour and prevents browsers from appending the cookie to request if the request is generated from a third-party site page load. The SameSite attribute can have the following three values:

- Strict: When set to Strict, cookies are only sent along with requests upon top level navigation .
- Lax: When set to Lax, cookies are sent with top level navigation from the same host as well as GET requests originated to the host from third-party sites (for example, in iframe, link, href, and so on and the form tag with GET method only).

- None: Cookies are sent in all requests made to the site within the path and domain scope set for the cookie. Requests generated due to form submissions using the POST method are also allowed to send cookies with request.

Please note that cookies that have the SameSite attribute with the value of None must be set with the Secure attribute otherwise the browser rejects the cookies. Additionally, a few specific browser versions reject the SameSite cookie with the None value for example, Chrome versions 51 to 66, versions of the UC Browser on Android prior to version 12.13.2, versions of Safari and embedded browsers on macOS 10.14, and all browsers on iOS 12 reject cookies set with SameSite=None. A suggested workaround for this issue is to set an alternate cookie with a prefix or suffix such as *Legacy*appended to cookiename . Sites can look for this legacy cookie if it does not find a cookie that was set with SameSite=None.

Execution:

Inspect the highlighted cookie value in the HTTP response in the vulnerable session. The cookie is missing the SameSite attribute.

Implication:

Sites that set cookies without the SameSite attribute have an increased risk of CSRF attacks. Using CSRF, an attacker can impersonate a valid user and gain unauthorized access to application functionality. Furthermore, the recent versions of browsers might reject the cookie that is not set with the SameSite attribute.

Fix:

Add the SameSite attribute to all cookies. Starting February 2020, the Chrome browser made SameSite a mandatory attribute for all cookies. Any cookie without the SameSite attribute is either rejected or treated with the default behaviour, which is the same as setting the attribute value to Lax. Therefore, any cookie that must be sent regardless the origination of the request (for example, analytics cookies) should have the SameSite attribute value of none.
Furthermore, we recommend that developers continue to add traditional CSRF mitigations to the site along with SameSite attribute. Many site users might still use older browser versions to access the site. Older browser versions do not understand the SameSite attribute.

Reference:

https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-05
https://www.chromium.org/updates/same-site/incompatible-clients
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

Low

Cache Management: Web Cache Poisoning

Summary:

WebInspect has detected a web cache poisoning instance by fat GET requests. Fat GET requests are requests that contain parameters as request body. Web cache poisoning exploits the behaviour of the web server causing a malicious response to be served to legitimate requests. In this type of web cache poisoning, the cache key is derived from the original query parameters passed in the request line. However, the value of the query parameter used by the server is the duplicate value sent as part of the request body. This altered response is cached and served to the users with legitimate requests.

WebInspect has detected that the cache can be poisoned. The **accion** was passed in the request body and it is observed that the initial response differed from the later response for the same query. This indicates that the cache might be poisoned.

Execution:

To test for web cache poisoning, execute the following steps:

The cache-buster is a query parameter added to a URL for example, `cacheBuster=someRandomValue`. Add a cache-buster to the URL.
Add the query parameter as part of the request body for example `param=someRandomValue`.
Send the request and wait for the response.
Next, remove the request body and send the request.
The response received is the malicious cached response.

Implication:

If the web server is susceptible to web cache poisoning, it can lead to legitimate users being served malicious responses.

Fix:

FAT GET requests must not be accepted. Either prune the request body or do not accept such a request.

Reference:

Web Cache Poisoning

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/recursos?accion=opcionesMenu&random=0.944999190033595&ca

---

Low

HTML5: Deprecated Header

Summary:

`X-XSS-Protection` header is enabled. `X-XSS-Protection` refers to a header used by developers to control the browser's behavior and offers protection against cross-site scripting (XSS). The header was designed to filter pages to remove unsafe components or stop loading when cross-site scripting was detected. Internet Explorer 8 introduced the `X-XSS-Protection` header and it was also supported in Edge version 12-16, Chrome version 4-77, Opera version 15-64, and Safari 5-15.3. Modern browsers no longer maintain, support, or enhance implementation of this header.

Having this setting enabled does not necessarily result in a vulnerability, but in some cases such filters are leveraged to perform XSS attacks against users that would otherwise not be possible. The XSS filters in the browser use regular expressions to alter the response. For example, a regex to detect `<script>` tag would alter the tag in the response to `<sc#ipt>`. It would stop the current XSS attack but also have unintended consequences. An attacker could use this knowledge to disable legitimate scripts or client-side scripts containing security features by adding a malicious parameter. By setting the `X-XSS-Protection` to `"1; mode=block"` stops the entire page from loading but it is vulnerable to side-channel attacks [3].

This header can be set explicitly in an application server configuration or in application code. All places should be checked for insecure setting of this header. This check is reported once per host by default and can be made to report all instances by setting check input `"X-XSS-Protection_Aggressive Reporting"`

Execution:

Click the response tab of the highlighted request. You will notice the occurrence of `X-XSS-Protection` header highlighted and value is set to 1.

Implication:

Setting `X-XSS-Protection` header is ineffective for browsers that no longer support this header. Enabling `X-XSS-Protection` header might increase the risk of cross-site scripting attacks against the application if a user accesses it from an older browser that supported this feature.

Fix:

Remove the response header `X-XSS-Protection`.
OpenText recommends using `Content Security Policy (CSP)` to protect against XSS attacks.

Reference:

1 X-XSS-Protection - HTTP MDN
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection

2 IE8 XSS Filters
https://media.blackhat.com/bh-eu-10/presentations/Lindsay_Nava/BlackHat-EU-2010-Lindsay-Nava-IE8-XSS-Filters-slides.pdf

3 Abusing Chrome's XSS auditor to steal tokens PortSwigger Research
https://portswigger.net/research/abusing-chromes-xss-auditor-to-steal-tokens

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

Informational **System Information Leak: External**

Summary:

A URL or filename was found in the comments of the file.

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/listaExpresiones.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/badfile123.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/inicioErr.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

Informational **Hidden Field**

Summary:

While preventing display of information on the web page itself, the information submitted via hidden form fields is easily accessible, and could give an attacker valuable information that would prove helpful in escalating his attack methodology. Recommendations include not relying on hidden form fields as a security solution for any area of the web application that contains sensitive information or access to privileged functionality such as remote site administration functionality.

Execution:

Any attacker could bypass a hidden form field security solution by viewing the source code of that particular page.

Implication:

The greatest danger from exploitation of a hidden form field design vulnerability is that the attacker will gain information that will help in orchestrating a far more dangerous attack.

Fix:

Do not rely on hidden form fields as a method of passing sensitive information or maintaining session state. One workable bypass is to encrypt the hidden values in a form, and then decrypt them when that information is to be utilized by a database operation or a script. From a security standpoint, the best method of temporarily storing information required by different forms is to utilize a session cookie.

Whether hidden or not, if your site utilizes values submitted via a form to construct database queries, do not make the assumption that the data is non-malicious. Instead, utilize the following recommendations to sanitize user supplied input.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad.
- Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/organizacionPolitica/navegarRegistrar.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarReporteInactivos.reniec

---

Informational **Often Misused: File Upload**

Summary:

Permitting users to upload files may allow attackers to inject dangerous content or malicious code to run on the server.

WebInspect has detected file upload capabilities on this server.

As part of the Dangerous File Upload check, WebInspect may have uploaded multiple files to the server. All files uploaded as part of this test have filenames in the format WebInspect_ccccc_File_nn.extension

Where:
ccccc = WebInspect Check ID Number
nn = Unique random number

Please locate and delete these files on the server after testing is completed.

Implication:

Regardless of the language in which a program is written, the most devastating attacks often involve remote code execution, whereby an attacker succeeds in executing malicious code in the program's context. If attackers are allowed to upload files to a directory that is accessible from the Web and cause these files to be passed to a code interpreter (e.g. JSP/ASPX/PHP), then they can cause malicious code contained in these files to execute on the server.

The exact implications depend upon the nature of the files an attacker would be able to upload. Implications range from unauthorized content publishing to aid in phising attacks, all the way to full compromise of the web server.

Even if a program stores uploaded files under a directory that isn't accessible from the Web, attackers might still be able to leverage the ability to introduce malicious content into the server environment to mount other attacks. If the program is susceptible to path manipulation, command injection, or dangerous file inclusion vulnerabilities, then an attacker might upload a file with malicious content and cause the program to read or execute it by exploiting another vulnerability.

Fix:

Do not accept attachments if they can be avoided. If a program must accept attachments, then restrict the ability of an attacker to supply malicious content by only accepting the specific types of content the program expects. Most attacks that rely on uploaded content require that attackers be able to supply content of their choosing. Placing restrictions on the content the program will accept will greatly limit the range of possible attacks. Check file names, extensions, and file content to make sure they are all expected and acceptable for use by the application. Make it difficult for the attacker to determine the name and location of uploaded files. Such solutions are often program-specific and vary from storing uploaded files in a directory with a name generated from a strong random value when the program is initialized, to assigning each uploaded file a random name and tracking them with entries in a database.

Ensure that the following steps are taken to sanitize the file being received:

- Limit the types of files that can be uploaded. For instance, on an image upload page, any file other than a .jpg should be refused.
- Ensure that the web user has no control whatsoever over the name and location of the uploaded file on the server.
- Never use the name that the user assigns it.
- Never derive the filename from the web user's username or session ID.
- Do not place the file in a directory accessible by web users. It is preferable for this location to be outside of the webroot.
- Ensure that strict permissions are set on both the uploaded file and the directory it is located in.
- Do not allow execute permissions on uploaded files. If possible, deny all permission for all users but the web application user.
- Verify that the uploaded file contains appropriate content. For instance, an uploaded JPEG should have a standard JPEG file header.

Reference:

Secure file upload in PHP web applications

Standards Mapping - Common Weakness Enumeration - (CWE) CWE ID 434

OWASP Unrestricted File Upload

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/lote/navegarCargarArchivo.reniec

Informational **HTML5: Missing Content Security Policy**

Summary:

Content Security Policy (CSP) is an HTTP response security header that developers and security architects can leverage to create an allow list of domains from which the site is allowed to load resources. This header provides an in-depth security protection from critical vulnerabilities such as cross-site scripting and clickjacking. Additionally, CSP restricts execution of inline JavaScript, dynamic JavaScript code evaluation from strings, and framing of the site from external domains. While CSP is not a replacement for input validation, it can help to significantly reduce the risk of XSS from unknown weaknesses. The CSP frame-ancestors directive is equivalent to X-Frame-Options and restricts the domain that are allowed to frame the site’s content.

Execution:

Access link http://172.16.21.39:7001/SisAdhereVerFirmas through a proxy and notice the missing CSP header in the response. By default, WebInspect flags only one instance of this vulnerability per host because it is typical to set this header at the host level in a server configuration.

Perform the following steps to flag all instances of this issue:

- Create a new policy with the selection of checks that you want to include in a rescan. We recommend using the Blank or Passive policy as a base.
- Select this check and uncheck the “FlagAtHost” check input from standard description.
- Save the policy.
- Rescan with this new custom policy.

Implication:

Security architects and developers can leverage CSP to significantly reduce the risk of XSS and clickjacking attacks. CSP headers can restrict leakage of information to external domains by restricting which domains the site is allowed to load contents from when rendered in browser .

Fix:

Define a CSP policy suitable for your site. The policy can be set either with an HTTP response header or <meta /> tag.

For example:
```
Content-Security-Policy: default-src https://example.net; child-src 'none';
```
Or
```
<meta http-equiv="Content-Security-Policy" content="default-src https://cdn.example.net;
child-src 'none'; object-src 'none'">
```

Content-Security-Policy 2 is the recommended standard. Content-Security-Policy 3 is in draft. The following is a snapshot of modern browser support for the CSP header:

- Edge: Versions 15-18; supported with a nonce bug. Version 75 and later; fully supported.
- Chrome: Versions 36-38; missing the plugin-types, child-src, frame-ancestors, base-uri, and form-action directives. Version 39; missing the plugin-types, child-src, base-uri, and form-action directives. Version 40 and later; fully supported.
- Firefox: Versions 31-34; missing the plugin-types, child-src, frame-ancestors, base-uri, and form-action directives. Version 35; missing the plugin-types, child-src, frame-ancestors, and form-action directives. Versions 36-44; missing the plugin-types and child-src directives. Version 45 and later; missing the plugin-types directive.

Furthermore, the report-uri directive can be configured to receive reports of attempts to violate the policy. These reports can be used as an early indication of security issues in the site as well as to optimize the policy.

Reference:

Content Security Policy Level 3
OWASP Content Security Policy
MDN web docs
Content Security Policy (CSP) Quick Reference Guide

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas

Informational **HLI: Detected Libraries**

Summary:

Hacker Level Insights provides developers and security professionals with more context relating to the overall security posture of their application. The **jQuery UI** version **1.8.14** was detected to be in use by during this scan. While these findings do not necessarily represent a security vulnerability, it is important to note that attackers commonly perform reconnaissance of their target in an attempt to identify known weaknesses or patterns. Knowing what the hacker can see provides context which can help teams better secure their applications.
.cvc-grid {width:100%;}.cvc-box {border-radius: 15px; border: 2px solid #1a75ff; width: 100%; padding: 10px;}.cvc-cat {position: relative; display: inline-block; border-bottom: 1px dotted black;}.cvc-cat .nvd-tooltip {visibility: hidden; width: 400px; background-color: #3b96ff; color: #fff; text-align: left; border-radius: 6px; padding: 5px; position: absolute; z-index: 1;}.cvc-cat:hover .nvd-tooltip {visibility: visible;}.score-box{color:#000!important;background-color:#f1f1f1!important;width:75%;border-radius:8px}.score-box div {border-radius:8px}.score-box div div{padding-left: 10px; font-weight: bold;}.score-box > div:after,.score-box > div:before {content:""; display:table;clear:both}.sb-good{color:#fff!important;background-color:#87e547!important}.sb-warn{color:#fff!important;background-color:#fbc02d!important}.sb-bad{color:#fff!important;background-color:#ff8686!important}.nvd-tooltip:before{content:"The National Vulnerability Database (NVD) helps Fortify WebInspect HLI analysis to find information for CVEs including a brief summary by using Common Platform Enumeration(CPE) queries. The database is maintained by the National Institute of Standards and Technology (NIST)."}

Known NVD records for: cpe:2.3:a:jqueryui:jquery_ui:1.8.14

VULNERABILITY:	**FOUND VERSION:**	**FIXED VERSION:**	**SEVERITY:**
CVE-2010-5312	1.8.14	1.10.0	Medium

Cross-site scripting (XSS) vulnerability in jquery.ui.dialog.js in the Dialog widget in jQuery UI before 1.10.0 allows remote attackers to inject arbitrary web script or HTML via the title option.

VULNERABILITY:	**FOUND VERSION:**	**FIXED VERSION:**	**SEVERITY:**
CVE-2021-41182	1.8.14	1.13.0	Medium

jQuery-UI is the official jQuery user interface library. Prior to version 1.13.0, accepting the value of the `altField` option of the Datepicker widget from untrusted sources may execute untrusted code. The issue is fixed in jQuery UI 1.13.0. Any string value passed to the `altField` option is now treated as a CSS selector. A workaround is to not accept the value of the `altField` option from untrusted sources.

VULNERABILITY:	**FOUND VERSION:**	**FIXED VERSION:**	**SEVERITY:**
CVE-2021-41183	1.8.14	1.13.0	Medium

jQuery-UI is the official jQuery user interface library. Prior to version 1.13.0, accepting the value of various `*Text` options of the Datepicker widget from untrusted sources may execute untrusted code. The issue is fixed in jQuery UI 1.13.0. The values passed to various `*Text` options are now always treated as pure text, not

HTML. A workaround is to not accept the value of the `*Text` options from untrusted sources.

VULNERABILITY:	FOUND VERSION:	FIXED VERSION:	SEVERITY:
CVE-2021-41184	1.8.14	1.13.0	Medium

jQuery-UI is the official jQuery user interface library. Prior to version 1.13.0, accepting the value of the `of` option of the `.position()` util from untrusted sources may execute untrusted code. The issue is fixed in jQuery UI 1.13.0. Any string value passed to the `of` option is now treated as a CSS selector. A workaround is to not accept the value of the `of` option from untrusted sources.

VULNERABILITY:	FOUND VERSION:	FIXED VERSION:	SEVERITY:
CVE-2022-31160	1.8.14	1.13.2	Medium

jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of jQuery. Versions prior to 1.13.2 are potentially vulnerable to cross-site scripting. Initializing a checkboxradio widget on an input enclosed within a label makes that parent label contents considered as the input label. Calling `.checkboxradio( "refresh" )` on such a widget and the initial HTML contained encoded HTML entities will make them erroneously get decoded. This can lead to potentially executing JavaScript code. The bug has been patched in jQuery UI 1.13.2. To remediate the issue, someone who can change the initial HTML can wrap all the non-input contents of the `label` in a `span`.

File Names:

- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

Best Practice **Compliance Failure: Missing Privacy Policy**

Summary:

A privacy policy was not supplied by the web application within the scope of this audit. Many legislative initiatives require that organizations place a publicly accessible document within their web application that defines their website's privacy policy. As a general rule, these privacy policies must detail what information an organization collects, the purpose for collecting it, potential avenues of disclosure, and methods for addressing potential grievances.
Various laws governing privacy policies include the Gramm-Leach-Bliley Act, Health Insurance Portability and Accountability Act (HIPAA), the California Online Privacy Protection Act of 2003, European Union's Data Protection Directive and others.

Execution:

All of the web pages accessible within the scope of the scan are sampled for textual content that often constitutes a privacy policy statement. A violation is reported upon completion of the web application crawl without a successful match against any of the web pages.
Note that the privacy policy of your application could be located on another host or within a section of the site that was not configured as part of the scan. To validate, please try to access the privacy policy of your website and check to see if it was part of the scan.

Implication:

Most privacy laws are created to protect residents who are users of the website. Hence, organizations from any part of the world must adhere to these laws if they cater to customers residing in these geographical areas. Failing to do so could result in a lawsuit by the corresponding government against the organization.

Fix:

Declare a comprehensive privacy policy for the website, and ensure that it is accessible from every page that seeks personal information from users. To verify the fix, rescan the site in order to discover and audit the newly added resources.

Descriptions:
Any standard web application privacy policy should include the following components:

- A description of the intended purpose for collecting the data.
- A description of the use of the data.
- Methods for limiting the use and disclosure of the information.
- A list of the types of third parties to whom the information might be disclosed.
- Contact information for inquires and complaints.

Reference:

California Online Privacy Protection Act
http://oag.ca.gov/privacy/COPPA

National Conference of State Legislation
http://www.ncsl.org/issues-research/telecom/state-laws-related-to-internet-privacy.aspx

Gramm-Leach-Bliley Act
http://www.gpo.gov/fdsys/pkg/PLAW-106publ102/pdf/PLAW-106publ102.pdf

Health Insurance Portability and Accountability Act of 1996
https://www.cms.gov/Regulations-and-Guidance/HIPAA-Administrative-Simplification/HIPAAGenInfo/downloads/HIPAALaw.pdf

Health Insurance Portability and Accountability Act of 1996
http://ec.europa.eu/justice/policies/privacy/docs/guide/guide-ukingdom_en.pdf

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec

Best Practice **Privacy Violation: Autocomplete**

Summary:

Most recent browsers have features that will save form field content entered by users and then automatically complete form entry the next time the fields are encountered. This feature is enabled by default and could leak sensitive information since it is stored on the hard drive of the user. The risk of this issue is greatly increased if users are accessing the application from a shared environment. Recommendations include setting autocomplete to "off" on all your forms.

Reference:

Microsoft:
Autocomplete Security

File Names:
- http://172.16.21.39:7001/SisAdhereVerFirmas/login.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/proceso/navegarConsultar.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/organizacionPolitica/navegarConsultar.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/inicioErr.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/js/jquery/jquery.form.js
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporte/navegarVerificacionSemiAutomatica.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/configuracion/navegarIpAcceso.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/configuracion/navegarConfigurarReportes.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/proceso/navegarRegistrar.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporte/navegarVerificacionAutomatica.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarControlVerificaSemi.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/configuracion/navegarMotivo.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/configuracion/navegarOrigenSolcitud.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarReporteBusAdherente.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/lote/navegarCargarArchivo.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarReporteProdPeriodo.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarReporteProdUsuario.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarConsultaRegRecepcionados.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/organizacionPolitica/navegarRegistrar.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/lote/navegarRegistrar.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/verificacionAutomatica/navegarConsultar.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarReportePerito.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/programacion/navegarPendientes.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/
- http://172.16.21.39:7001/SisAdhereVerFirmas/ordenProduccion/navegarPendientesAsignar.reniec

- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarConsultaOrgVerificadas.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporte/navegarRecepcion.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarReporteGeneral.reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/reporteConsulta/navegarReporteBusRepresentante.
reniec
- http://172.16.21.39:7001/SisAdhereVerFirmas/ordenProduccion/navegarPendientesCrear.reniec