Hewlett Packard
Enterprise

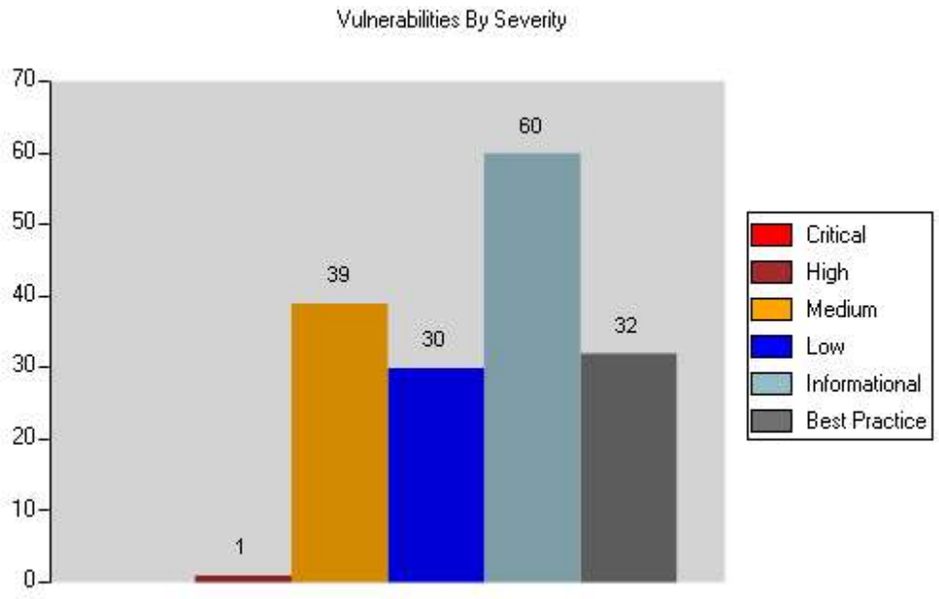
HPE Fortify WebInspect

Vulnerability (Legacy)

Web Application Assessment Report

Scan Name:	Site: https://app.formiik.com/		
Policy:		Crawl Sessions:	381
Scan Date:	1/17/2020 1:34:23 PM	Vulnerabilities:	70
Scan Version:	17.20.322.0	Scan Duration:	1 day : 3 hours
Scan Type:	Site	Client:	IE

Server: https://app.formiik.com:443



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.

Execution:

Create a test page containing an HTML iframe tag whose src attribute is set to https://app.formiik.com:443/. 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

- ALLOW-FROM origin

Developers can specify a list of trusted origins in the origin attribute. Only pages on origin are permitted to load this page inside an iframe

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

- <host-source>

Equivalent to "ALLOW-FROM" - developers can specify a list of trusted origins which maybe host name or IP address or URL scheme. Only pages on this list of trusted origin are permitted to load this page inside an iframe

- <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:
- https://app.formiik.com:443/

Medium Privacy Violation: HTTP GET

Summary:

A username was detected either in the request query string or the Set-Cookie header. A username is considered sensitive private information that should be protected. Exposing a username in query strings or cookies can also leave it exposed in

server logs, proxy logs, and network traffic inspection tools.

Execution:

Inspect the highlighted string in the request of a vulnerable session and confirm that it is indeed a username in the scanned application.

Implication:

Attackers can use the username to try to gain access to the system. Users often use the same username on multiple systems and attackers can take advantage of a hacked username and password pair to brute force credential detection on this site and might gain unauthorized access to the user's account.

Fix:

Avoid leaving login information in a query string or cookie value because an attacker could see and tamper with login values. Have a developer or security administrator examine this issue. Ensure that login information is sent via POST requests and POST parameters only over an encrypted connection, and that sensitive account information is not cached.

File Names:

- https://app.formiik.com:443/Login/Index?ReturnUrl=%2fReports%2fGetResultsEvalReportFilterView%3ftype
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView?typeStatistics=Average&UserList=1
- https://app.formiik.com:443/Login/Index?ReturnUrl=%2fReports%2fGetResultsEvalReportFilterView%3ftype
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView?typeStatistics=Sum&UserList=12345
- https://app.formiik.com:443/Login/Index?ReturnUrl=%2fReports%2fGetResultsEvalReportFilterView%3ftype
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView?typeStatistics=Sum&UserList=12345

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.

Execution:

Create a test page containing an HTML iframe tag whose src attribute is set to https://app.formiik.com:443/Operation/UpLoadFile?fielgfield=. 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

- ALLOW-FROM origin

Developers can specify a list of trusted origins in the origin attribute. Only pages on origin are permitted to load this page inside an iframe

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

- <host-source>

Equivalent to "ALLOW-FROM" - developers can specify a list of trusted origins which maybe host name or IP address or URL scheme. Only pages on this list of trusted origin are permitted to load this page inside an iframe

- <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:
- https://app.formiik.com:443/Operation/UpLoadFile?fielgfield=

Medium HTML5: Overly Permissive Message Posting Policy

Summary:

A broadcast of information to windows on the client-side using postMessage was observed. The postMessage function is one of the features of HTML5 that allows a window to send messages to another open window. The typical syntax of postMessage is *window.postMessage(message, targetOrigin, [transfer]).*

The *message* parameter contains the information to be shared and *targetOrigin* indicates the origin of the destination window. It

is recommended to explicitly specify the targetOrigin intended for a particular message. While the function permits the use of a wildcard character (*) for the targetOrigin, allowing the message to be shared to any window, we strongly recommend avoiding this practice. If a malicious user injects a new window into the javascript's execution context, then the new window will now be able to receive any information that is shared using the vulnerable postMessage construct.It is also important to validate any message that is received from another window to confirm that the sender was authorized to do so. This will avoid receiving unintended input from a malicious window.

Execution:

Inspect all messages that are transferred between windows to check the targetOrigin of the function calls. In this particular case, the given stack trace represents the execution of javascript that led to the discovery of the vulnerability.

Implication:

If the postMessage() is used to transmit sensitive information between windows, then an unauthorized window would be able to retrieve this information as well.

Fix:

All data that is shared between windows must have explicit origins to avoid eavesdropping. It is recommended to avoid using "*" for the targetOrigin in order to prevent a broadcast of the information.

Reference:

Window.postMessage API
https://developer.mozilla.org/en-US/docs/Web/API/Window.postMessage

Cross-document messaging specification
http://www.whatwg.org/specs/web-apps/current-work/multipage/web-messaging.html

Insecure Usage of new client-side primitives
http://devd.me/papers/w2sp10-primitives.pdf

File Names:

- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage

- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage
- Function:Window.postMessage

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:
- https://app.formiik.com:443/Operation/UpLoadFile?fielgfield=

Low Web Server Misconfiguration: Unprotected Directory

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

File Names:
- https://app.formiik.com:443/scripts/
- https://app.formiik.com:443/scripts/

Low Web Server Misconfiguration: Unprotected Directory

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

File Names:
- https://app.formiik.com:443/css/
- https://app.formiik.com:443/content/
- https://app.formiik.com:443/images/
- https://app.formiik.com:443/content/

- https://app.formiik.com:443/css/
- https://app.formiik.com:443/images/
- https://app.formiik.com:443/content/
- https://app.formiik.com:443/css/

Low Web Server Misconfiguration: Unprotected Directory

Summary:

Development-related directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include removing any source code directories and repositories from the production server, disabling the use of remote repositories, and ensuring that the latest patches and version updates have been performed on the version control system being used Additionally, restrict access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Execution:

Browse to https://app.formiik.com:443/message/ and inspect the content. Response should return with HTTP status code 200 and should not match target site's file not found response.

Implication:

An attacker may use the internal information obtained from the source code files to craft a precise attack against the web application. Such attacks can include, but are not limited to, SQL injection, remote file system access, malware injection and database manipulation.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, remove all source code repositories and files from the production server and do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

IIS Authentication
IIS Authentication

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

SVN
Serving websites from SVN checkout considered harmful

Subversion or CVS metadata exposure
Subversion or CVS metadata exposure

File Names:
- https://app.formiik.com:443/message/

Low Web Server Misconfiguration: Unprotected Directory

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

File Names:
- https://app.formiik.com:443/demo/
- https://app.formiik.com:443/demo/

Low Web Server Misconfiguration: Unprotected Directory

Summary:

Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

File Names:
- https://app.formiik.com:443/reports/

Low Web Server Misconfiguration: Unprotected Directory

Summary:

Logfile and/or statistic directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

File Names:
- https://app.formiik.com:443/alert/
- https://app.formiik.com:443/alert/

Low Web Server Misconfiguration: Unprotected Directory

Summary:

Directories that may be restricted to only certain users were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache
http://httpd.apache.org/docs/howto/auth.html#intro

File Names:
- https://app.formiik.com:443/login/

Low HTML5: Sensitive Information Disclosure via Client-Side Storage

Summary:

WebInspect has detected possible sensitive information being stored in the browser's storage object. It is preferable to avoid storing sensitive information in the HTML5 storage objects since these are stored on the client-side. While this option may sound attractive from a performance perspective, any information stored on the client is easily accessible and may pose a security risk if it is accessed by an unauthorized third party.

Execution:

This vulnerability is detected by inspecting the parameters that are used while calling the setItem() function on an HTML5 storage object. Ensure that the sensitivity level of the information that is saved into the storage object is benign and requires no additional protection. The parameters are inspected for possible username and password fields. Additionally, if there is any other information specific to the application that needs to be searched for it can be configured in the check input "Sensitive Parameter". In this particular case, the given stack trace represents the execution of javascript that led to the discovery of the vulnerability.

Implication:

Storing sensitive information in the localStorage or sessionStorage objects provided by HTML5 is not a secure option. While this information may not be visible to a naïve user, a technically savvy person could easily retrieve such data from a browser. If an application exhibiting this behavior is used in a publicly accessible computer, then any data stored on the client could be stolen by a malicious user.

Fix:

Avoid saving any sensitive information on the client-side. It is advisable to store sensitive information on the server and then retrieve it over a secure channel only when necessary.

Reference:

Insecure usage of new client-side primitives
http://devd.me/papers/w2sp10-primitives.pdf

OWASP HTML5 Security Cheat Sheet
https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet#Local_Storage

W3C Web Storage Specification
http://www.w3.org/TR/webstorage/

File Names:
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem
- Function:Storage.setItem

Low HTML5: Cross-Site Scripting Protection

Summary:

X-XSS-Protection HTTP response header enables developers and security architects to manage browser protection against reflected cross-site scripting. The mechanism is also known as the XSS Auditor in Chrome and the XSS filter in Internet Explorer. In modern browsers, the Content-Security-Policy header can provide better protection against XSS and setting X-XSS-Protection might be redundant. However, this header can reduce the risk of reflected XSS attacks in earlier browsers that do not support CSP.

This header can be set to one of three possible values: `0`, `1`, or `1; mode=block` . A value of `0` disables the protection. A value of `1` is the default behaviour in modern browsers that enables the protection in filter or replacement mode. For example, IE replaces JavaScript keywords such as <script> with <scr#pt> to render injected string ineffective. The value of `1; mode=block` instructs browsers to block the response from rendering in the browser. Reports of multiple exploits that leverage false positives from default behaviour that filters or replaces JavaScript injection string within the response r eturned from server. Therefore, the current recommendation is to set the header in block mode.

Execution:

Click the response tab for the highlighted request. The response header X-XSS-Protection is either missing or set to `1`.

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 unselect the check input, “FlagAtHost”,from standard description window.
- Save the policy.
- Rescan with this new custom policy.

Implication:

Attackers may leverage zero day reflective XSS against a site. If the header is not set in block mode, an attacker can use browser-specific filter bypass bugs to succeed in launching a reflected XSS against the site.

Fix:

Add a configuration setting or a line of code that adds a response header or tag to set X-XSS-Protection with the value ‘`1; mode=block`’

Reference:

Fortify Taxonomy: Software Security Errors
OWASP Secure Headers Project
CWE ID 554
Chromium Bugs

File Names:
- https://app.formiik.com:443/

Informational System Information Leak: External

Summary:

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

File Names:
- https://app.formiik.com:443/oauth/nmp?code=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6Im9yYWtleSJ9.
- https://app.formiik.com:443/Scripts/plugins/mobiikplugins/Mobiik.Helper.UI.js?v=ofv4sJJ5Hr6WhS4hfKor
- https://app.formiik.com:443/Scripts/plugins/mobiikplugins/Mobiik.Helper.UI.js
- https://app.formiik.com:443/Content/css/private.css?v=jgCmfbatEfpiy-AGRfgV5C3VGI08ZsOqipCjbGABvvE1
- https://app.formiik.com:443/Content/css/private.css
- https://app.formiik.com:443/login/
- https://app.formiik.com:443/oauth/

- https://app.formiik.com:443/Scripts/plugins/Utilities/gmaps.js
- https://app.formiik.com:443/Scripts/formiik.alert.js
- https://app.formiik.com:443/Content/css/private.css?v=jgCmfbatEfpiy-AGRfgV5C3VGI08ZsOqipCjbGABvvE1
- https://app.formiik.com:443/oauth/nmp?code=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6Im9yYWtleSJ9.
- https://app.formiik.com:443/Operation
- https://app.formiik.com:443/
- https://app.formiik.com:443/Login/badfile123
- https://app.formiik.com:443/oauth/badfile123
- https://app.formiik.com:443/demo/

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:

- https://app.formiik.com:443/
- https://app.formiik.com:443/oauth/nmp?code=eyJhbGciOiJSUzUxMiIsInR5cCI6IkpXVCIsImtpZCI6Im9yYWtleSJ9.

Informational Cache Management: Headers

Summary:

The web server sent a Vary header, which indicates that server-driven negotiation was done to determine which content should be delivered. This may indicate that different content is available based on the headers in the HTTP request. Scan configuration recommendations include viewing the HTTP response to determine what criteria is used to negotiate content, and appending custom headers and values according to the negotiate criteria being used.

Fix:

For Development:
Verify your application does not display different content based on headers, and if necessary, re-scan with appropriate headers to ensure good coverage.

For Security Operations:
Evaluate if content negotiation is truly being used, and disable if it is unnecessary. Re-scan with appropriate headers to ensure good coverage.

For QA:
This requires a server or application configuration change. Contact Security Operations for assistance with the server.

Reference:

W3C RFC 2616 Header Field Definitions
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.44

File Names:

- https://app.formiik.com:443/Content/css/site/genericForm.css?v=qIAfiFT9zuesLSp8NRyPEFUIuvGM13trfacx4
- https://app.formiik.com:443/Content/css/jquery/jquery.tagsinput.css?v=GwSIqt6USkEqjNgHQHVwY1ryYJbquo
- https://app.formiik.com:443/Content/css/filtro/tooltipster/tooltipster.css?v=08n5-9ZJ0Fof0IJHs8sQLKV
- https://app.formiik.com:443/
- https://app.formiik.com:443/Scripts/plugins/jquery/jquery-1.8.2.min.js
- https://app.formiik.com:443/Scripts/login.js
- https://app.formiik.com:443/Scripts/plugins/Utilities/ganalytics.js
- https://app.formiik.com:443/Scripts/plugins/mobiikplugins/formiik.genericsvalidations.js
- https://app.formiik.com:443/css/bootstrap.min.css
- https://app.formiik.com:443/Content/css/lg.css
- https://app.formiik.com:443/Scripts/appCues/formiik.formiikappcues.js
- https://app.formiik.com:443/Scripts/plugins/datatables/1.10.13/jquery.dataTables.min.js
- https://app.formiik.com:443/Scripts/operation/workOrderJsFromJson.js?v=OHi0Z-KR8e8RTFUmwwZhJNPKKWnN0
- https://app.formiik.com:443/Scripts/ReplyWorkorder/Widget/formiikform.widget.baserwidgetinput.js?v=T
- https://app.formiik.com:443/Scripts/Metrics/ActionsPage.js?v=lvF6CpXUCypgGCbLYHLEBxBlofveMyZZYAOYXeg
- https://app.formiik.com:443/Scripts/newRelic/interactions.js?v=B1cGVPWVwsHMxFCvWGdgKMS4Ro26oM4biEPbo
- https://app.formiik.com:443/Scripts/plugins/jquery/datepicker/i18n/jquery.ui.datepicker-en.js
- https://app.formiik.com:443/Scripts/Reports/formiik.reports.usersmodaltree.js
- https://app.formiik.com:443/Scripts/scriptsViews/workorderviewer/workorderviewer.js?v=mlZFzk_duQZVD8
- https://app.formiik.com:443/Scripts/plugins/zendesk/zenbox.js
- https://app.formiik.com:443/Scripts/plugins/tooltipster/jquery.tooltipster.min.js
- https://app.formiik.com:443/Content/css/plugins/datatables/1.10.13/jquery.dataTables.min.css
- https://app.formiik.com:443/Scripts/ReplyWorkorder/formiikform.widgetsutilities.js?v=kPsM1IcdwyC39jS

- https://app.formiik.com:443/demo/
- https://app.formiik.com:443/oauth/
- https://app.formiik.com:443/login/
- https://app.formiik.com:443/ReplyFormEdit/GetFormEditView
- https://app.formiik.com:443/Scripts/WorkOrderMaker/WorkOrderMakerPrincipal.js
- https://app.formiik.com:443/alert/
- https://app.formiik.com:443/Operation/AttachFileToWorkOrder?workorderId=
- https://app.formiik.com:443/AsyncReports/AsyncReportVisitResults?t=
- https://app.formiik.com:443/Content/css/Reports/favorite.css
- https://app.formiik.com:443/Scripts/plugins/jquery/jqgrid/i18n/grid.locale-es.js
- https://app.formiik.com:443/Scripts/Views/AsyncReports/jAsyncReportGrid.js
- https://app.formiik.com:443/Scripts/Reports/ReportGenerator/jResultsEvalReport.js
- https://app.formiik.com:443/Scripts/plugins/jquery/jqgrid/jquery.jqGrid.min.js
- https://app.formiik.com:443/Scripts/Views/Common/labels.js
- https://app.formiik.com:443/Scripts/Views/FormiikMasterPage/formiikMasterPage.js
- https://app.formiik.com:443/Scripts/scriptsViews/users/UserProfile.js
- https://app.formiik.com:443/ReplyFormEdit/GetFormEditView
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView

---

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:
- https://app.formiik.com:443/Operation/UpLoadFile?fielgfield=

Best Practice Privacy Violation: HTTP GET

Summary:

An area of the web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality utilizes query strings to pass information between pages. Information in query strings is directly visible to the end user via the browser interface, which can cause security issues. At a minimum, attackers can garner information from query strings that can be utilized in escalating their method of attack, such as information about the internal workings of the application or database column names. Successful exploitation of query string parameter vulnerabilities could lead to an attacker impersonating a legitimate user, obtaining proprietary data, or simply executing actions not intended by the application developers. Recommendations include performing server-side input validation to ensure data received from the client matches expectations.

Fix:

For Development:

The best way to ensure that attackers cannot manipulate query string parameters is to perform server-side input validation. Never implicitly trust any data returned from the client. Utilize strong data typing to ensure that numeric values are actually numeric values, for example, and that data received from a client matches what is expected.

Although not a perfect solution, as there are other methods of gathering this information, you can add a layer of protection by utilizing POST statements instead of GET for HTML form information submittal. The POST method sends form input in a data stream, not as part of the URL as with GET statements. The advantages of the POST method is that data is not visible in the browser location window and is not recorded in web server log files. Be advised, however, that POST data can still be sniffed.

For Security Operations:

Query string vulnerabilities will ultimately require a code-based solution. The best way of preventing these issues from a Security Operations perspective is to implement a "secure coding" development policy that prohibits placing potentially sensitive information or access to functionality inside query strings.

For QA:

From a QA perspective, scrutinize any query string variables displayed in the URL for information that could be potentially utilized by an attacker. Things to look for include the following:

- User Identification: Look for values that obviously represent a user, such as a social security number, a username, or something similar.
- Session Identification: Are there values that remain constant for an entire session? Values to look for include sessionid, session, sid, and s.
- Architecture Identification: Are file, directory, or pathnames best left hidden displayed in the query string?

Ensure these values cannot be easily guessed, or adjusted to impersonate a legitimate user, access sensitive information, or utilized for other malicious activity.

File Names:

- https://app.formiik.com:443/Scripts/ReplyWorkorder/Widget/formiikform.widget.passwordeditinput.js?v=
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView?typeStatistics=Sum&UserList=12345
- https://app.formiik.com:443/Login/Index?ReturnUrl=%2fReports%2fGetResultsEvalReportFilterView%3ftype
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView?typeStatistics=Sum&UserList=12345
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView?typeStatistics=Sum&UserList=12345
- https://app.formiik.com:443/Login/Index?ReturnUrl=%2fReports%2fGetResultsEvalReportFilterView%3ftype
- https://app.formiik.com:443/Login/Index?ReturnUrl=%2fReports%2fGetResultsEvalReportFilterView%3ftype

---

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:
- https://app.formiik.com:443/

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:
- https://app.formiik.com:443/Reports/GetResultsEvalReportFilterView

---

Best Practice Web Server Misconfiguration: Insecure Content-Type Setting

Summary:

The Content-Type HTTP response header or the HTML meta tag provides a mechanism for the server to specify an appropriate character encoding for the response content to be rendered in the web browser. Proper specification of the character encoding through the charset parameter in the Content-Type field reduces the likelihood of misinterpretation of the characters in the response content and ensure reliable rendering of the web page.Failure to ensure enforcement of the desired character encoding could result in client-side attacks like Cross-Site Scripting.

Execution:

Verify the character set specification on every HTTP response. Character sets can be specified in the HTTP header or in an HTML meta tag. In the case of an XML response, the character set can be specified along with the XML Declaration.

Implication:

In the absence of the character set specification, a user-agent might default to a non-standard character set, or could derive an incorrect character set based on certain characters in the response content. In some cases, both these approaches can cause the response to be incorrectly rendered. This may enable other attacks such as Cross-site Scripting.

Fix:

Ensure that a suitable character set is specified for every response generated by the web application. This can be done either by,

- Modifying the code of the web application, which would require all pages to be modified.
- Adding Content-Type header to the server configuration (**recommended**). This ensures that the header is added to all the responses with minimal development effort.

Reference:

DoD Application Security and Development STIG
http://iase.disa.mil/stigs/app_security/app_sec/app_sec.html

UTF-7 encoding used to create XSS attack
http://www.securityfocus.com/archive/1/420001

File Names:
- https://app.formiik.com:443/Content/css/lg.css
- https://app.formiik.com:443/Content/css/jquery-ui.css?v=jPevAKPvWDkRO_hpSOR4JH8QBXzxMTq5EefEEuo_es81
- https://app.formiik.com:443/Content/css/private.css?v=jgCmfbatEfpiy-AGRfgV5C3VGI08ZsOqipCjbGABvvE1
- https://app.formiik.com:443/css/bootstrap.min.css

- https://app.formiik.com:443/Content/css/filtro/tooltipster/tooltipster.css?v=08n5-9ZJ0Fof0IJHs8sQLKV
- https://app.formiik.com:443/Content/css/filtro/tooltipster/tooltipster-shadow.css?v=7Z0hcOK8vlo48Yks
- https://app.formiik.com:443/Content/css/jquery/jquery.jui_dropdown.css?v=qiRTGu-N1BAiaWzuvdQXHD3JwS8
- https://app.formiik.com:443/Content/css/generalUses.css?v=HX8O95oeRhH8VFPxCeJMaMrWqDOV_rFTvipV_i1dgp
- https://app.formiik.com:443/Content/css/jquery/select2.css?v=uC-HKGtfuTPPnXVqgIk-Ulo-YuRcInaLcTG3x3q
- https://app.formiik.com:443/Content/css/site/ficons.css?v=nSSy96wCkyjux2mwYo5mgdP_gY8xO270RVadi2zq5j
- https://app.formiik.com:443/Content/css/site/fbuttons.css?v=SsII9ExMyAzh0BMQeP59SmLoqP87qGZaaHERW4Ux
- https://app.formiik.com:443/Content/css/site/list-group.css?v=IFQGMQq4BpGVYBN8PoVF3rVeBmk9lSs5jnABX1
- https://app.formiik.com:443/Content/css/site/fpanel.css?v=qFEco_xC38YK9ZBtYi8UBsL3oUmozN_m758mqeLsYw
- https://app.formiik.com:443/Content/css/site/falert.css?v=K15qyk4oVDea__NGyTTdzf4UBf2UwDGWEm_kOUD46I
- https://app.formiik.com:443/Content/css/site/fswitch.css?v=VzOB48Lf7LBHDpxW8Qa_QyMJPojdMd-agkJp-SiiC
- https://app.formiik.com:443/Content/css/jquery/jquery.tagsinput.css?v=GwSIqt6USkEqjNgHQHVwY1ryYJbquo
- https://app.formiik.com:443/Content/css/homeOperation.css?v=bQ8KL5cTT2fcPP7nzjamF-m-UWTKuMYcmkpVEVjx
- https://app.formiik.com:443/Content/css/site/tablecontent.css?v=21M59VEo12d88yLjhEQ2yL9vGe0WLhYk4DJ0
- https://app.formiik.com:443/Content/css/site/genericForm.css?v=qIAfiFT9zuesLSp8NRyPEFUIuvGM13trfacx4
- https://app.formiik.com:443/Content/css/popbox.css?version=12
- https://app.formiik.com:443/Content/css/plugins/datatables/1.10.13/jquery.dataTables.min.css
- https://app.formiik.com:443/Content/css/themeDefault.css
- https://app.formiik.com:443/Content/css/reset.css